MAIN PROJECT REPORT ON FREELANCE DOCTOR APPOINTMENT BOOKING SYSTEM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE IN

BACHELOR OF COMPUTER APPLICATIONS

OF

MAHATMA GANDHI UNIVERSITY KERALA

Submitted by

KHADEEJA BEEVI C N

REGISTER NO: 200021096319

Under the Guidance
of
Dr Julie M David

DEPARTMENT OF COMPUTER APPLICATIONS
(2020-2023)

M.E.S. COLLEGE MARAMPALLY
ALUVA -7

M.E.S. COLLEGE, MARAMPALLY
ALUVA -7

DEPARTMENT OF COMPUTER APPLICATIONS

Certificate

This to certify that the report entitled

FREELANCE DOCTOR APPOINTMENT BOOKING SYSTEM

Has been submitted by
KHADEEJA BEEVI C N
ROLL NO: 200139

In partial fulfillment of the award of the degree in

BACHELOR OF COMPUTER APPLICATION

OF

MAHATMA GANDHI UNIVERSITY

During the academic year 2022-2023

Register No: 200021096319

Project Guide Head of the Department

Submitted for the examination held on………………………………..

Examiners

1.

2.

ACKNOWLEDGEMENT

At the very outset I am very grateful to God almighty for his blessings showed upon us to complete our project. I hereby express our sincere thanks and Gratitude to Dr. Ajims P Mohammed, Principal of MES College, Marampally for his kind support. I hereby express our sincere thanks and Gratitude to Dr. Leena C Sekhar, Associate Professor and Head, Department of Computer Applications MES College, Marampally for her kind support throughout the course of the project.

I also express my sincere thanks to Dr Julie M David Assistant Professor, Department of Computer Applications MES College, project guide for their constant and sincere efforts to help bring out the best in us. Also grateful for their valuable and helpful guidance all through the course of the project.

I express our sincere thanks to our project coordinator Mr. Joseph Deril K S, Associate Professor, Department of Computer Applications for his co-operation and valuable guidance throughout the course of the project work. I also express our sincere thanks to all our faculty members of the Department of Computer Applications, MES College for their timely suggestion and encouragement.

Finally, I express my heartfelt gratitude to our parents, friends, well-wishers and all who have helped us in completing this project and making this work satisfactory.

KHADEEJA BEEVI C N

CONTENTS

1. INTRODUCTION

1.1 OVERVIEW OF THE SYSTEM

A freelance doctor appointment booking system is a digital platform that enables patients to schedule appointments with independent doctors who do not have their own clinics or work at hospitals. The system allows patients to search for available doctors based on their specialty, location, and availability, and to book appointments at their convenience.

The system benefits both doctors and patients. For doctors, it provides a platform to expand their patient base and manage their schedules more efficiently, while for patients, it offers a more convenient way to schedule appointments without having to physically visit the doctor's clinic or hospital.

The freelance doctor appointment system typically includes features such as a doctor's profile with their qualifications and experience, a booking system that allows patients to schedule and manage appointments, and a payment gateway for online transactions. Some systems may also include features such as video consultation and a patient review system to help patients make informed decisions when selecting a doctor.

The objective of a freelance doctor appointment booking system is to provide an online platform for patients to schedule appointments with independent doctors who do not have their own clinics or work at hospitals. The system aims to simplify the appointment scheduling process for both doctors and patients by providing a centralized platform for doctor-patient interactions.

2. REQUIREMENT ANALYSIS

2.1. PROBLEM DEFINITION

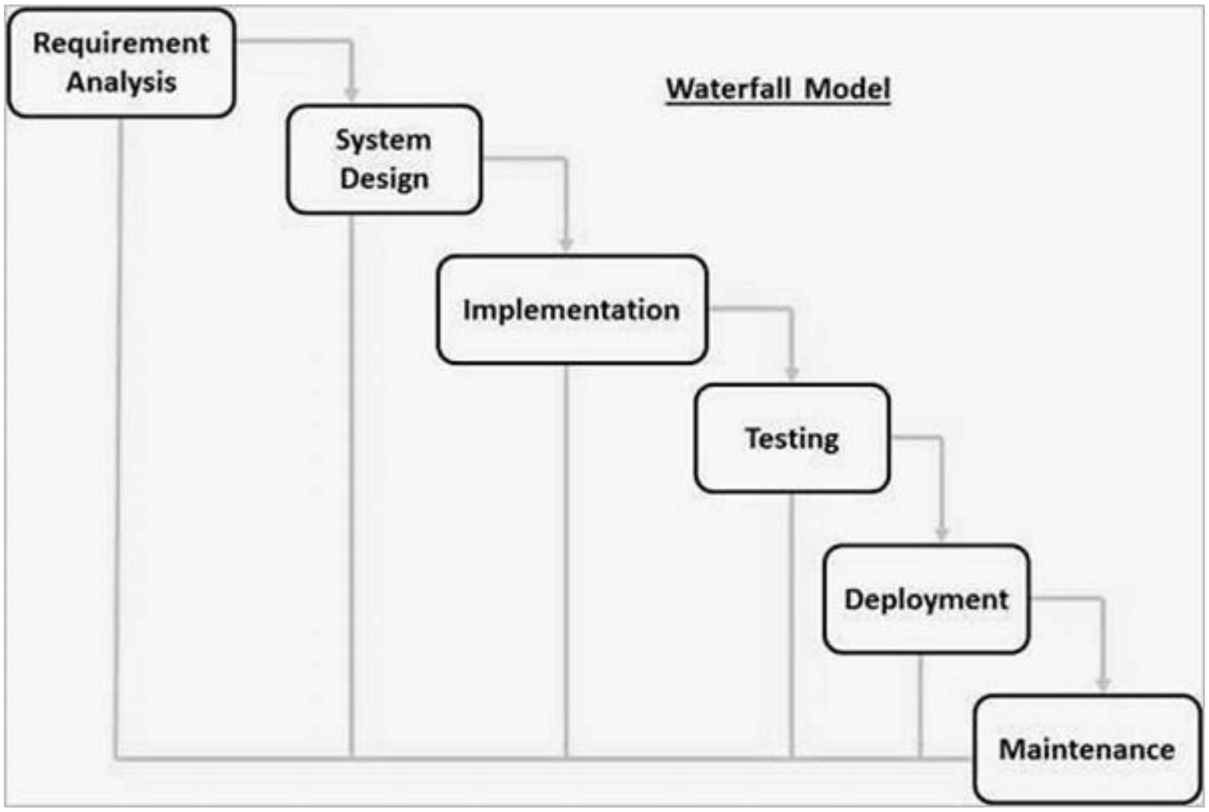
A freelance doctor appointment booking system is a software system designed to connect freelance doctors with patients who need medical attention. The system provides a platform for patients to book appointments with freelance doctors based on their availability and expertise. The problem definition of a freelance doctor appointment booking system includes the following:

- Patients need a convenient and easy way to book appointments with freelance doctors based on their availability and location.
- Freelance doctors need a platform to showcase their expertise and availability to potential patients.
- The system should be able to handle various types of medical appointments, including inperson consultations, online consultations, and follow-up appointments.
- The system should allow patients to choose from a list of freelance doctors based on their specialties and expertise.
- The system should allow freelance doctors to manage their schedules, set their availability, and manage their appointments.
- The system should provide a secure platform for patients and doctors to communicate and share medical information.
- The system should handle billing and payment processing for appointments.
- The system should provide an easy way for patients to reschedule or cancel appointments.

Overall, the goal of a freelance doctor appointment booking system is to provide a user-friendly platform for patients to connect with freelance doctors, and for freelance doctors to manage their schedules and appointments.This system can remove these drawbacks and saves time, fuel, money and energy. By developing this package we intend to address these problems with the help of the computerized system. The project is implemented using HTML and CSS as front end and Python and mysql server as backend. The waterfall model is a popular version of the systems development life cycle model for software engineering. Often considered the classic approach to the systems development life cycle, the waterfall model describes a development method that is linear and sequential. Waterfall development has distinct goals for each phase of development.

Once a phase of development is completed, the development proceeds to the next phase and there is no turning back

2.2. SELECT THE SOFTWARE DEVELOPMENT MODEL



I selected the Waterfall model as the software development model. The Waterfall approach was the first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially. The sequential phases in waterfall models are requirement gathering and analysis, system design, implementation, integration and testing, deployment of system, maintenance. The waterfall development model originates in the manufacturing and construction of industries: highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development. The sequential phases in Waterfall model are:

- Requirement Gathering and analysis **:** All possible requirements of the system to be developed like processing speed, data security, acquiring more functions etc, are captured in this phase and then documented in a requirement specification document.
- System Design **:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- Implementation **:** With inputs from system design, the system is divides as units which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
- Integration and Testing **:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- Deployment of system **:** Once the functional and non functional testing is done, the software is deployed in the customer environment or released it.
- Maintenance **:** There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for the previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

2.3. REQUIREMENT SPECIFICATION

2.3.1. Justification of Proposed system

2.3.1.1. Existing System

In an existing system patients has to go to the hospital and wait in a queue at the appointment desk to make the reservation and get an appointment. But they generally finish up waiting endlessly for very long time intervals. The patient might choose to fix an appointment, but this choice is not possible at all times and does not likely work well for all people involved in the system. People involved are as follows: The medical personnel, the hospital and the patient.

The patient longs for effortlessly convenient and accessible times.When they cannot discover a quick enough appointment they feel like waitingendlessly (Duration between scheduling and the appointment being made available).The patient also expects to be seen all of a sudden or within few minutes of their visit to the hospital or clinic (whether an appointment has been scheduled earlier or not).

Disadvantages of Existing System :-

- Technical issues: One of the main disadvantages of doctor booking appointment systems is the potential for technical issues, such as system failures, slow loading times, or difficulty accessing the platform. These issues can disrupt the patient experience and may result in missed appointments or delayed care.
- Privacy and security concerns: Doctor booking appointment systems may also raise privacy and security concerns, as patient data and personal information are stored online. Inadequate security measures can put patients' sensitive information at risk of being accessed or stolen by unauthorized parties.
- Lack of personal interaction: Doctor booking appointment systems can also limit personal interaction between patients and doctors, especially for virtual consultations. This may impact the quality of care, as doctors may not have the same level of insight into a patient's health and medical history.
- Limited availability: Doctor booking appointment systems may also have limitations in terms of available appointment times, doctors, or specialties. This can make it difficult for patients to find the care they need, especially in areas where healthcare resources are limited.

- Dependence on technology: Finally, doctor booking appointment systems can create a dependence on technology, which may not be accessible or easy to use for all patients, particularly those who are elderly or have limited access to technology.

2.3.1.2. Proposed System

A freelance doctor appointment booking system provides patients with the convenience of booking appointments online, from the comfort of their own homes. Patients can choose appointment times that work best for them, without having to worry about office hours or wait times.

Increased efficiency: A freelance doctor appointment booking system can increase efficiency for freelance doctors, allowing them to manage their schedules more effectively and reduce the amount of time spent on administrative tasks. This can lead to more time spent on patient care. Cost-effective: For patients, a freelance doctor appointment booking system can be a costeffective alternative to traditional healthcare services. Patients can often book appointments at a lower cost, and the system can help to reduce the overall cost of healthcare.

Enhanced patient experience: A freelance doctor appointment booking system can improve the patient experience by reducing wait times, providing more personalized care, and allowing patients to easily communicate with freelance doctors before and after appointments.

Flexibility: A freelance doctor appointment booking system provides patients with greater flexibility in choosing their healthcare providers. Patients can search for and choose freelance doctors based on their specific needs and preferences.

Overall, a freelance doctor appointment booking system can provide numerous benefits for both patients and freelance doctors, including improved access to healthcare, increased convenience, greater efficiency, cost-effectiveness, and enhanced patient experience..

-

2.3.2. Benefits of Proposed System

The proposed system is wholly computerized thus making it user friendlier. The system is more efficient, reliable, accurate and fast. The advantages or benefits of the proposed system are:

- The freelance doctors and doctor who does doe not have job they will get job through this system
- Payment can be done through the system
- Doctors can work from there preffered location and time
- Patients can easily apply the doctors from there home and from there nearby location.
- They can have emergency chat with doctors .

2.4. PROJECT PLANNING

The project has four months from August 1st. Considering the total available time I have prepared a plan and schedule which is given below.

SI.No	Duration	Activity
1	January 01 – January 10	Identification of need.
2	January 11 – January 15	Feasibility study
3	January 16– February 01	Analysis
4	February 02 – February 13	Design
5	February 14 – February 28	Testing
6	March 31	Implementation

2.5. PROJECT SCHEDULING

Once we have estimates of the effort and time requirement for the different phases, a schedule for the project can be prepared. Conceptually simple and effective scheduling techniques like calendar oriented charts are prepared. Progress can be represented easily by ticking off each milestone when completed. Alternatively, for each activity another bar can be drawn specifying when the activity actually started and ended, i.e., when these two milestones were achieved. Once we have estimates of the effort and time requirement for the different phases, a schedule for the project can be prepared.

Activity	Jan 01	Jan 11	Jan 16	Feb 02	Feb 14	
Identification of need.	■					
Feasibility study		■				

Analysis			■			
Design				■		
Testing					■	
Implementation						■

2.6. FEASIBILITY STUDY

A feasibility study is a high-level capsule version of the entire System analysis and Design Process. The study begins by classifying the problem definition. Feasibility is to determine if it's worth doing. Once an acceptance problem definition has been generated, the analyst develops a logical model of the system. A search for alternatives is analyzed carefully.
There are 3 parts in feasibility study:-

- Technical Feasibility
- Operational Feasibility
- Economical Feasibility

2.6.1 Technical Feasibility

This involves questions such as whether the technology needed for the system exists, how difficult it will be to build, and whether the firm has enough experience using that technology. The assessment is based on outline design of system requirements in terms of input, processes, output, fields, programs and procedures. This can be qualified in terms of volume of data, trends, frequency of updating inorder to give an introduction to the technical system. The application is the fact that it has been developed on windows 7 platform and a high configuration of 1GB RAM on Intel Pentium Dual core processor. This is technically feasible .The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the needs of the proposed system.

2.6.2 Operational Feasibility

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes. To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design

if desired operational behaviours are to be realized. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

2.6.3 Economic Feasibility

Establishing the cost-effectiveness of the proposed system i.e. if the benefits do not outweigh the costs then it is not worth going ahead. In the fast paced world today there is a great need for online social networking facilities. Thus the benefits of this project in the current scenario make it economically feasible. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/benefits analysis.

3. SOFTWARE REQUIREMENT SPECIFICATION (SRS)

3.1. INTRODUCTION

The Software Requirements Specification is designed to document and describe the agreement between the customer and the developer regarding the specification of the software product requested. Its primary purpose is to provide a clear and descriptive "statement of user requirements" that can be used as a reference in further development of the software system. This document is broken into a number of sections used to logically separate the software requirements into easily referenced parts.

The "Freelance doctor appointment booking system '' governs the registration, user login, appointment and other enquiries. The software project is initiated by the client needs.The requirement analyst has to identify the requirements by talking to these people and understanding their needs. Hence , identifying requirements necessarily involves specifying what some people have in their minds.

3.1.1. Purpose

Defining and describing the functions and specifications of the freelance doctor appointment system's primary goal of this Software Requirements Specification (SRS). This Software Requirements Specification illustrates, in clear terms, the system's primary uses and required functionality as specified by the patients. It also states the various required constraints by which the system will abide. The intended audience for this document are the development team, testing team and end users of the product.

3.1.2. Scope

The scope of a freelance doctor appointment booking system includes a range of features and functionalities that enable patients to book appointments with freelance doctors and allow freelance doctors to manage their schedules and appointments. Here are some of the key features that are typically included in the scope of a freelance doctor appointment system:

- Appointment booking: Patients should be able to easily book appointments with freelance doctors based on their availability and specialty.

- Doctor profiles: The system should provide detailed profiles for freelance doctors, including their education, experience, certifications, and specialties.

- Appointment management: Freelance doctors should be able to manage their appointments, including scheduling, rescheduling, and canceling appointments.

- Payment processing: The system should provide a secure platform for patients to pay for appointments and for freelance doctors to receive payment.

- Communication: The system should provide a secure platform for patients and freelance doctors to communicate before, during, and after appointments.

- Medical records: The system should allow freelance doctors to access and update patient medical records securely and easily.

- Feedback: The system should provide a way for patients to rate and review freelance doctors based on their experiences.

- Privacy and security: The system should ensure that all patient information is kept private and secure.

- The scope of a freelance doctor appointment booking system may vary based on the specific needs of the users and the platform's capabilities. Overall, the goal of the system is to provide a seamless and secure platform for patients and freelance doctors to connect and manage medical appointments

3.1.3. Definition, Acronyms and Abbreviation

3.1.3.1. Definition

- Admin **:** Admin manages the overall system . The admin can add ,delete , and modify the patient and doctor details. Admin give login credentials to doctors.The admin can add medical departments and receive payments from patients and give salary to doctors and view feedback from patients.

- Doctor**:** The doctor can apply and add treatments to patients and view past treatments. doctors can receive salary and add time and slots of consulting they can receive receipt .
- Patient**:** The patient can book appointments according to their preffered location and type of doctor and they want to see thye can make online payment and get receipt they can view treatment details and past details they can give feedback

3.1.3.2. Acronyms, Abbreviations

Acronyms	**Meaning**
SQL	Structured Query Language
PHP	Hypertext Preprocessor
SRS	Software Requirement Specifications
DFD	Data Flow Diagram
E-R Diagram	Entity-Relationship

3.1.4 References

- IEEE 830-1998 standard for writing SRS documents.
- K K Aggarwal, Yogesh Singh - Software Engineering,Third Edition, New Age International Publications.
- Roger S Pressman – Software Engineering: A Practitioner Approach, Sixth Edition, McGraw-Hill Higher Education.
- Learning PHP, MySQL, JavaScript, CSS: A Step by step guide to dynamic websites by Robin Nixon.
- Programming in Python 3,A Complete Introduction to the Python Languague, Second Edition by Mark Summerfield

3.1.5 Overview

This document provides a general description, including characteristics of the users of this project, the product's hardware, and the functional and data requirements of the product. The functional requirements, data requirements and constraints and assumptions made while designing the system. Also gives the user viewpoint of the product.

The developer is responsible for:

- Developing the system.
- Installing the software.
- Maintaining the system.

The SRS is divided into three major sections :

- Introduction
- Overall description
- Specific requirements

3.2. OVERALL DESCRIPTION

3.2.1. Product Perspective

The system will be developed completely independent and standalone. Each user will have to have an individual copy of the system. The freelance doctor appointment system shall be developed using client server architecture and will be compatible with Microsoft Windows operating system. Front end of the system will be developed using HTML5, CSS, javaScript, bootstrap and the back end will be developed using Python and MYSQL server.

3.2.2. Product Functions

Freelace doctor appointment booking system will allow access only to the authorized users with specific roles(system administrator, patient,doctor). Depending upon the user's role, he /she will be able to access only specific modules of the system. A summary of major functions that the freelance doctor appointment system shall perform includes:- ● Login facility for enabling only authorized access to the system.

- System administrators will be able to add, modify, delete view profiles.
- The User will be able to add, modify details and,cancel bookings .
- The user have quick chat with doctors
- The doctors can add time and slot of consulting
- Admin,User,doctors can sign in/sign out.

3.2.3. User Characteristics

Admin

- Controls the entire system.
- Have access to entire features of the system.
- Ability to monitor the entire system.
- Can edit, modify, delete and doctors.
- Add medical departments for doctors for applying ▪
- Accept or reject doctor application
- View past treatments of patients by patient id
- View booking details by dates
- View feedbacks of patients
- Accept payments and generate receipt
- Add salary to doctor

Doctors

- Have only limited access to the system.

- Can apply
- Change password and manage profile
- View booking details by dates
- View past treatments of patients by patient id
- Accept appointment of patients
- Emergency chat with patients
- Add treatments of patients
- Accept salary from admin

Patient

- Have only limited access to the system.
- Can register
- Change password and manage profile
- Book doctors by department,time,slots
- View booking history
- Cancel bookings
- Search doctors by name,department or locality
- View doctor details
- Make payment
- Download reciept

3.2.4. Constraints

- The system must be user friendly.

- Designed in Python using MYSQL.

3.2.5. Assumptions and Dependencies

Users can just retrieve the connection details and offers from the database. The user can add new connection details by changing some limited portions of the database. Administrators will have entire access throughout the database. System should have proper user authentication. The assumptions are that the coding should be error free, the system should be user friendly so that the users can easily access data, the system should have more storage capacity and provide fast access to database, the system should save time unlike the existing system, the user must provide correct user name and password to enter to the system.

3.3. Specific Requirements

3.3.1. External Interfaces

- Account Registration : The registration function shall allow users to create secure accounts The account will track the user’s name, phone number,address, username and password and other contact details .
- Account Login : The account login function shall allow account members to enter their username and password. Once verified, users will be able to access.
- Search : The search function shall offer users the ability to search for doctors and departments and locality
- Payment:The patient can pay fees and get the receipt online and admin can pay salary to the doctors and receive reciepts
- Bookings : The patients can book doctors according to their department and locality and they can cancel booking anytime and doctor need to add time and slot for booking
- Account Logout : The account logout function shall allow account members to exit their account for security purposes.
- Administrator Control Panel : This control panel will allow administrators to add,edit or remove the details of doctors,patients.

3.3.2. Functional Requirements

Functional requirements are the specific features and functionalities that a freelance doctor appointment booking system must have in order to fulfill its intended purpose. Here are some of the key functional requirements of a freelance doctor appointment booking system:

- Registration and profile creation: Patients and freelance doctors should be able to create profiles with basic information such as name, contact information, specialty, and availability.
- Appointment booking: Patients should be able to search for available freelance doctors based on their specialty, location, availability, and other criteria, and book appointments online.
- Appointment management: Freelance doctors should be able to manage their schedules, including adding, modifying, and canceling appointments.
- Communication: Patients and freelance doctors should be able to communicate with each other through a secure messaging system within the platform.
- Medical records management: Freelance doctors should be able to access and manage patient medical records, including creating new records, updating existing ones, and viewing medical history.
- Billing and payment processing: The system should be able to handle payments for appointments and other services, including processing payments securely and issuing invoices.
- Feedback: Patients should be able to rate and review freelance doctors, and these ratings and reviews should be visible to other patients.
- Security and privacy: The system should be designed with strong security and privacy features to ensure that all patient data and communications are secure and protected.
- Reporting and analytics: The system should provide reports and analytics to help freelance doctors and administrators manage appointments, track performance, and identify areas for improvement.

These functional requirements are essential for a freelance doctor appointment booking system to be effective and user-friendly, and they should be designed with the needs of both patients and freelance doctors in mind..

3.3.3. Performance Requirements

- Should run on windows Xp/7/8/8.1
- Reasonable response time.
- Reports should be generated within a reasonable time.
- Searching should be done within minimum time.

3.3.4. Logical Database Requirements

- System should have been installed with MYSQL as the back end.
- Create different tables for patient details, connection details, fee payment and these tables should be linked using primary key and foreign keys.

Insert valid data to the created tables.

3.3.5. Design Constraints

Design constraints are those constraints that are imposed on the design solution. These constraints are typically imposed by the customer, by the development organization, or by external regulations. The constraints may be imposed on the hardware, software, data, operational procedures, interfaces, or any other part of the system. Design constraints can have a significant impact on the design and should be validated prior to imposing them on the solution. A straightforward approach to address design constraints is to categorize the type of constraints (e.g., hardware, software, procedure, and algorithm), identify the specific constraints for each category, and capture them as system requirements in the Requirements package along with the corresponding rationale. The design constraints are then integrated into the physical architecture.

3.3.6 Software System Attributes

- Reliability :

  The software should not have any reliability issues. The software will be thoroughly tested and any issues resolved.

- Availability :

  The software will execute as a standalone system so as long as the machine is running, the program will be available. The key to maintaining availability will be by ensuring a connection to the database server is available. Failure to connect to the database will make data unavailable.

- Security :

  This software is intended to communicate over an internal network; therefore security is of little concern. The user will have to enter the username and password so the program can connect to the database server. The username and password will not be stored because encryption of such information is outside the scope of the project.

- Maintainability :

  The software will be composed of various modules decreasing the complexity

- Portability :

  As stated previously, this software will only run under the Windows OS. The setup file, setup.info, can be copied to multiple machines so that each program does not have to be set up separately.

3.3.7. Organizing the Specific Requirements

- Functional Hierarchy

In this system the overall functionality is organized by Data flow diagrams and E-R diagrams. Based on these diagrams, data relationships and dependencies are found and a functional hierarchy is made for organizing the specific requirements.

4. SOFTWARE AND HARDWARE REQUIREMENT

4.1 Software Specifications

- Operating system – Windows 7 or higher
- Front-end - HTML, CSS ● Server script - Python
- Back-end - MySQL 4.2

4.2 Hardware Specifications

- Processor - Intel Pentium Dual Core
- RAM - 4GB
- Hard disk - 10G

5. SYSTEM DESIGN

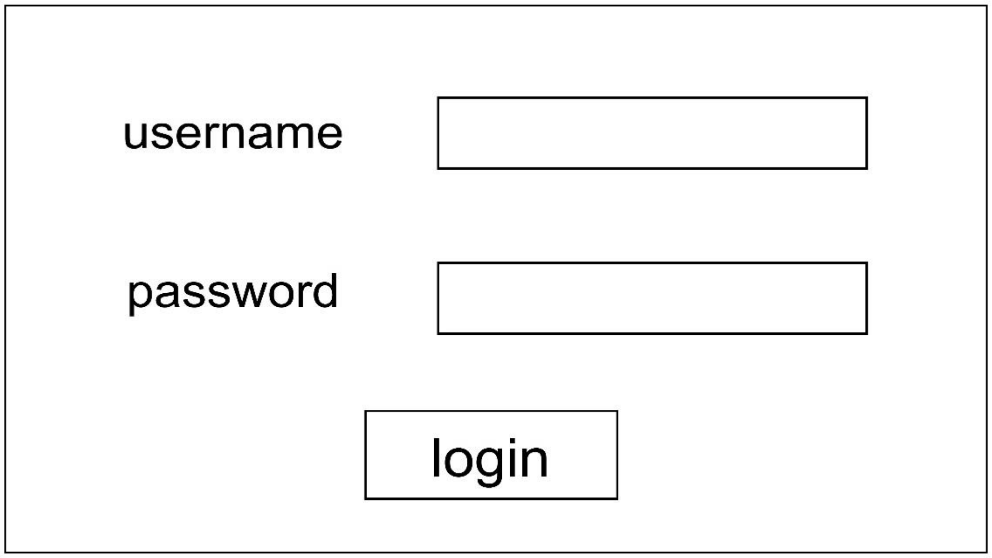
5.1. INTRODUCTION

Hardware and software requirements for the installation and smooth functioning of this project could be configured based on the requirements needed by the component of the operating environment that works as frontend system here we suggest minimum configuration for the both hardware and software components.

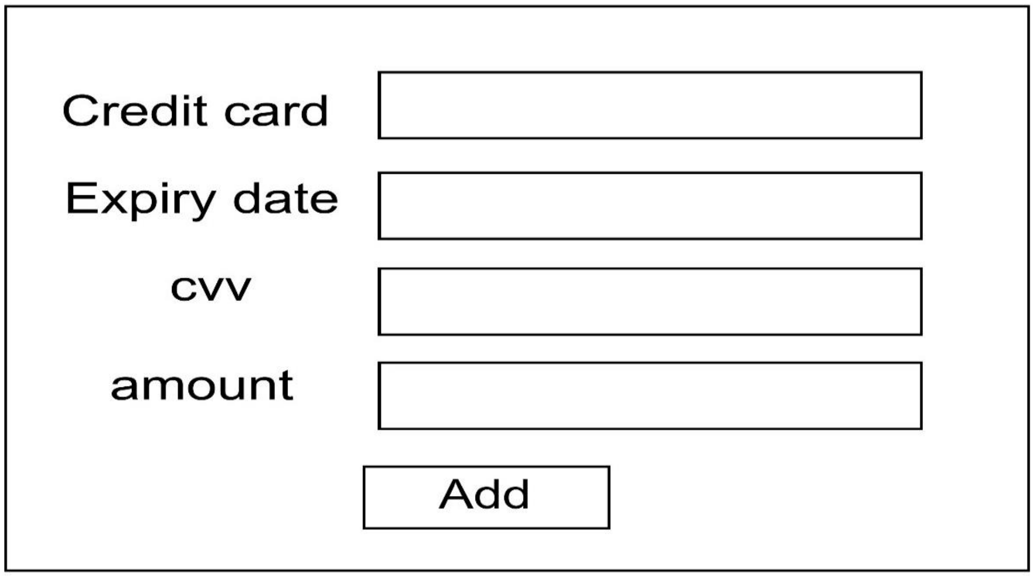
The system can produce real-time reports faster and easier compared to the manual process. The system has a search module that allows the user to easily look for an item and view its details in less amount of time. WD management will provide an efficient service to give the management satisfaction that enables to transact fast, secure and reliable.

5.2. INPUT DESIGN

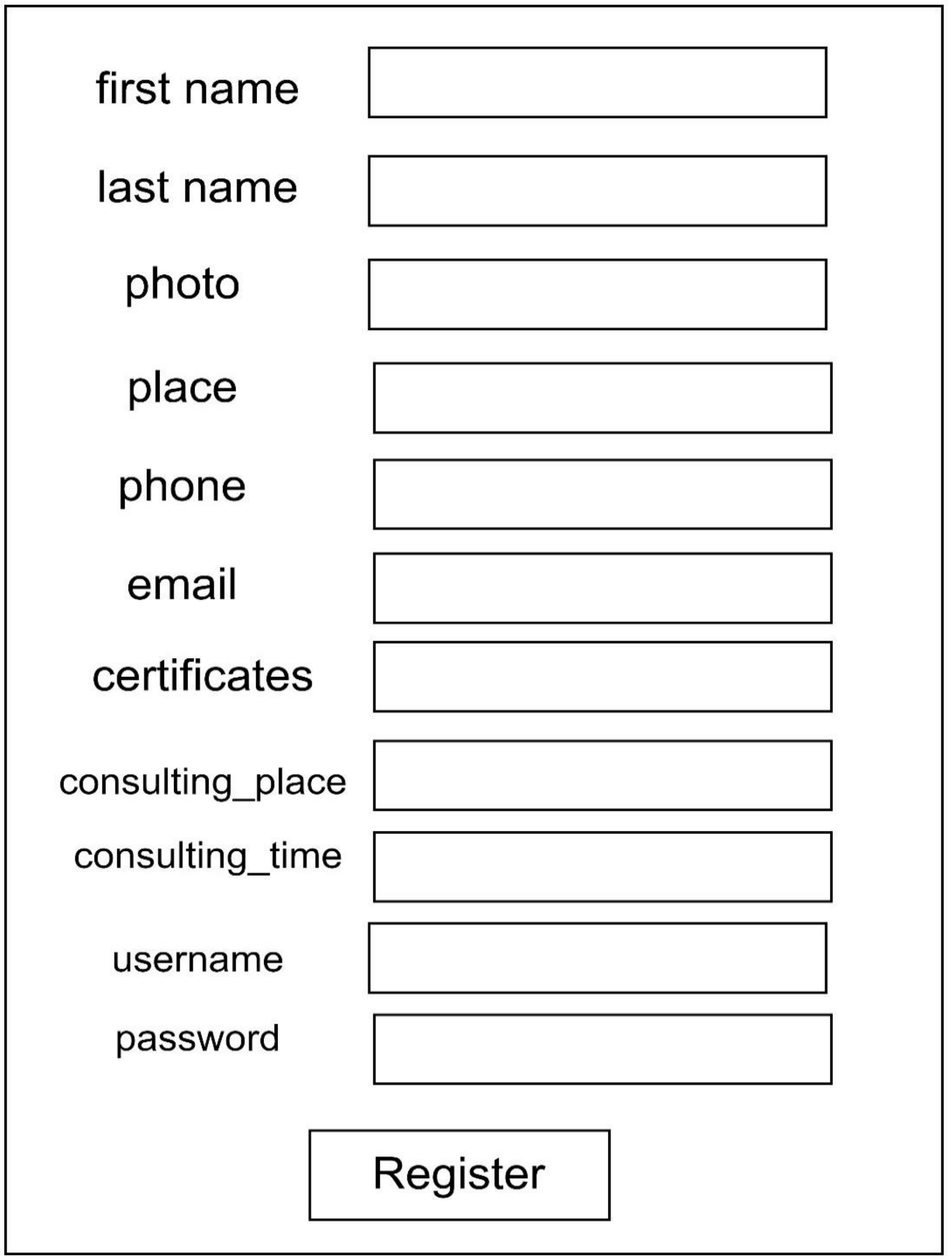
- **login**



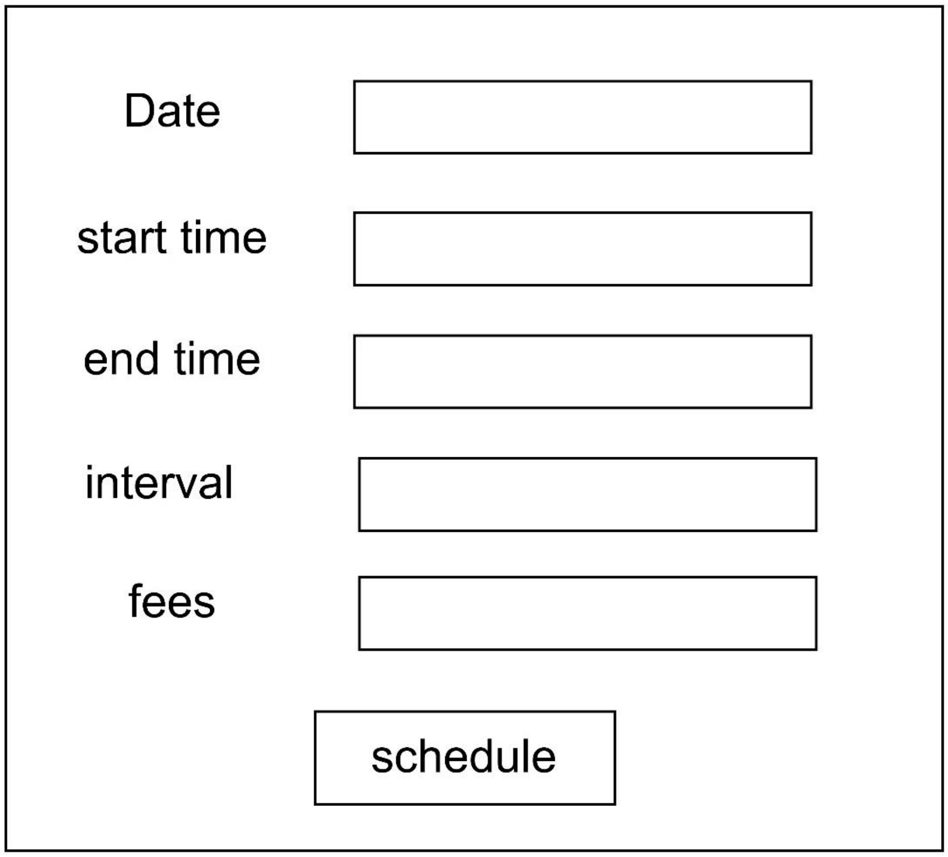
- **payment**



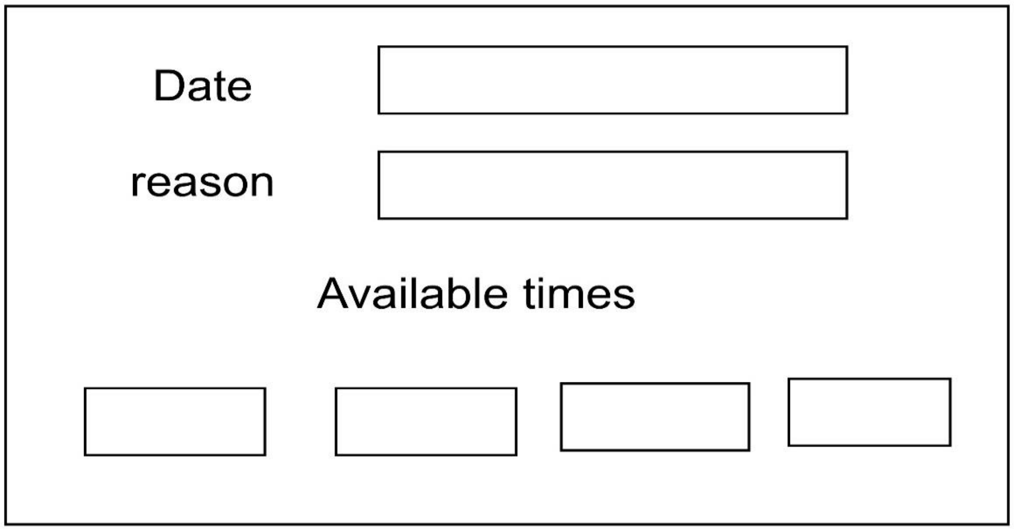
- **application**



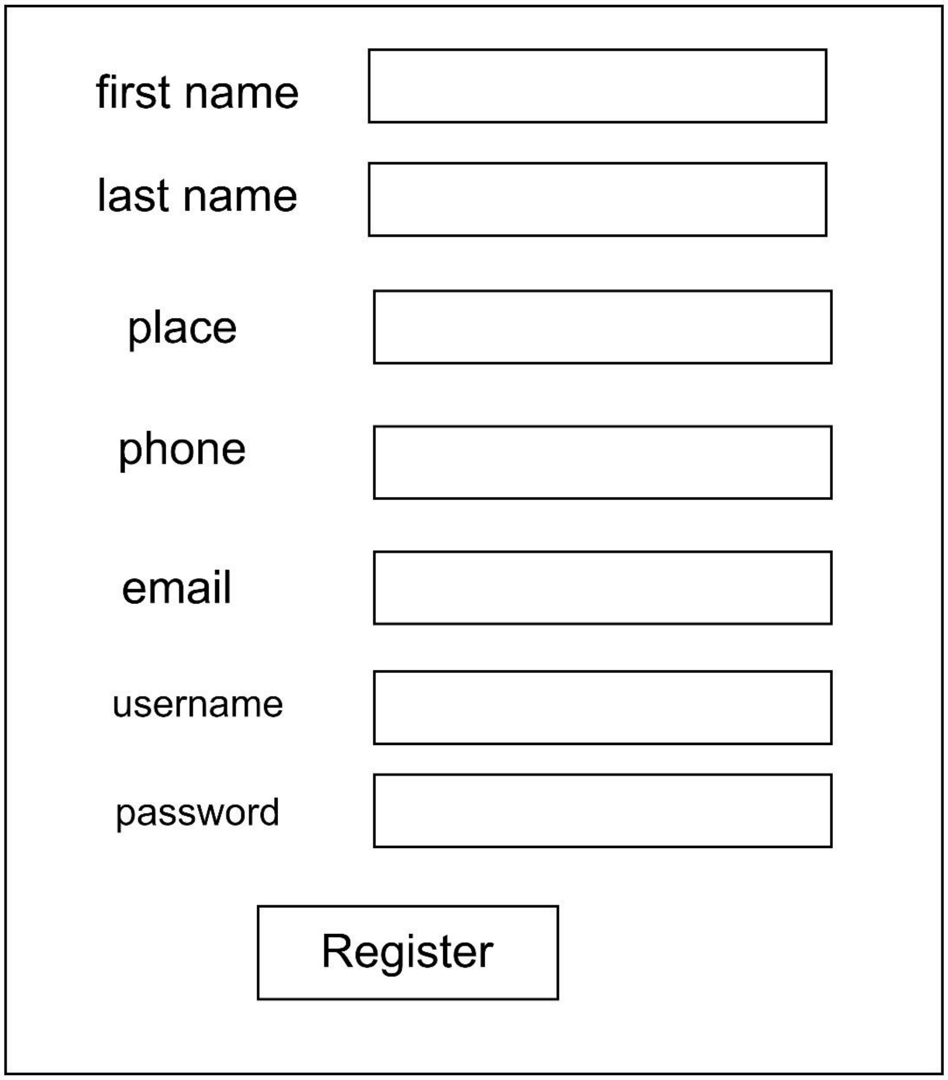
- **schedule**



- **booking**

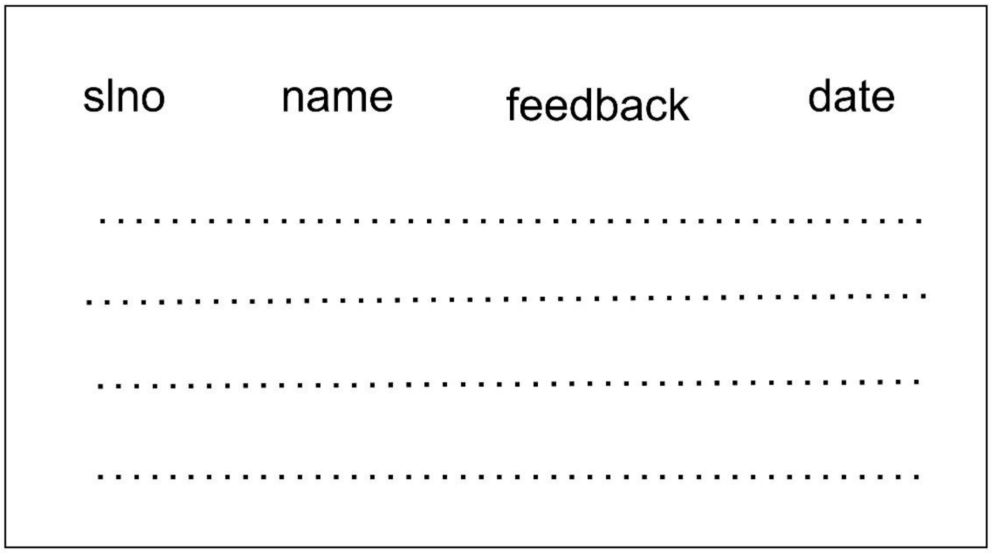


- **patient registration**

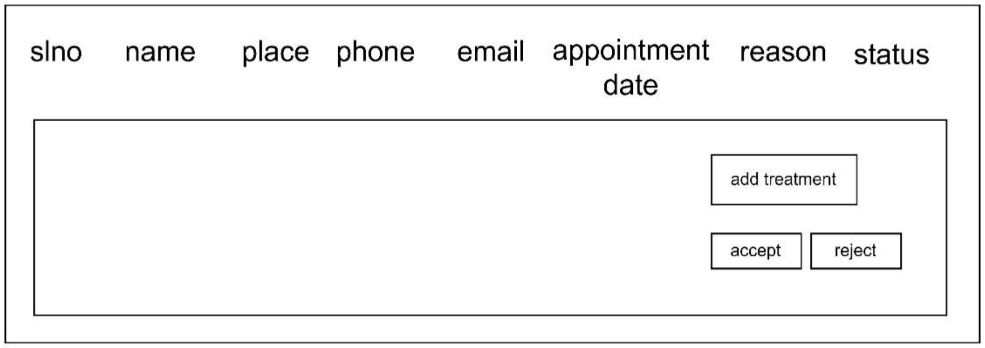


5.3. OUTPUT DESIGN

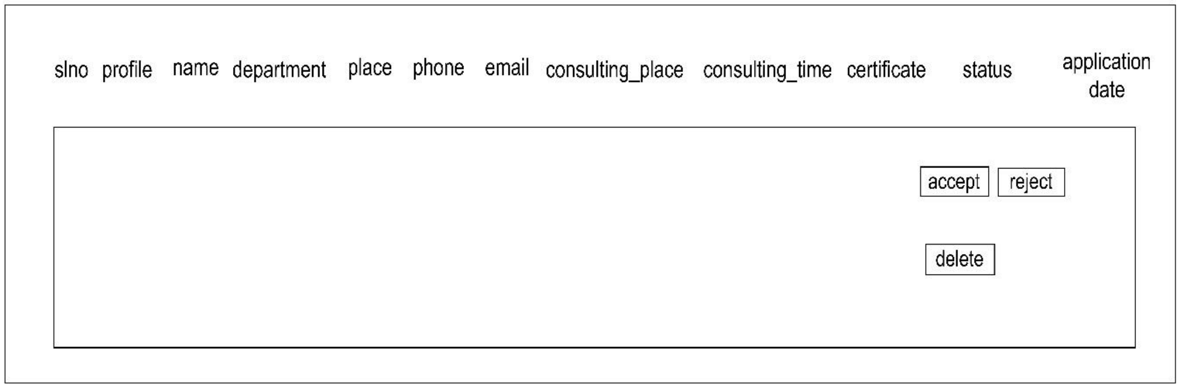
- **feedback**



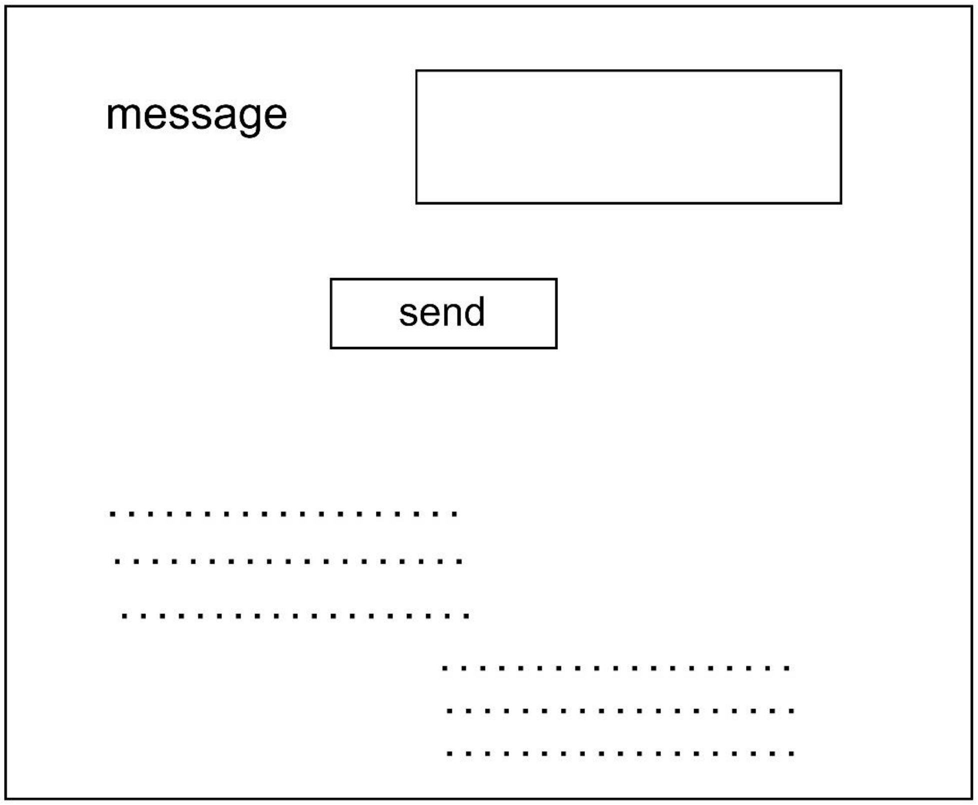
- **bookings**



- **doctor applicaton**



- **message**



- **patient**

slno	name	place	phone	email

- **fees**

slno	name	phone	email	amount	date

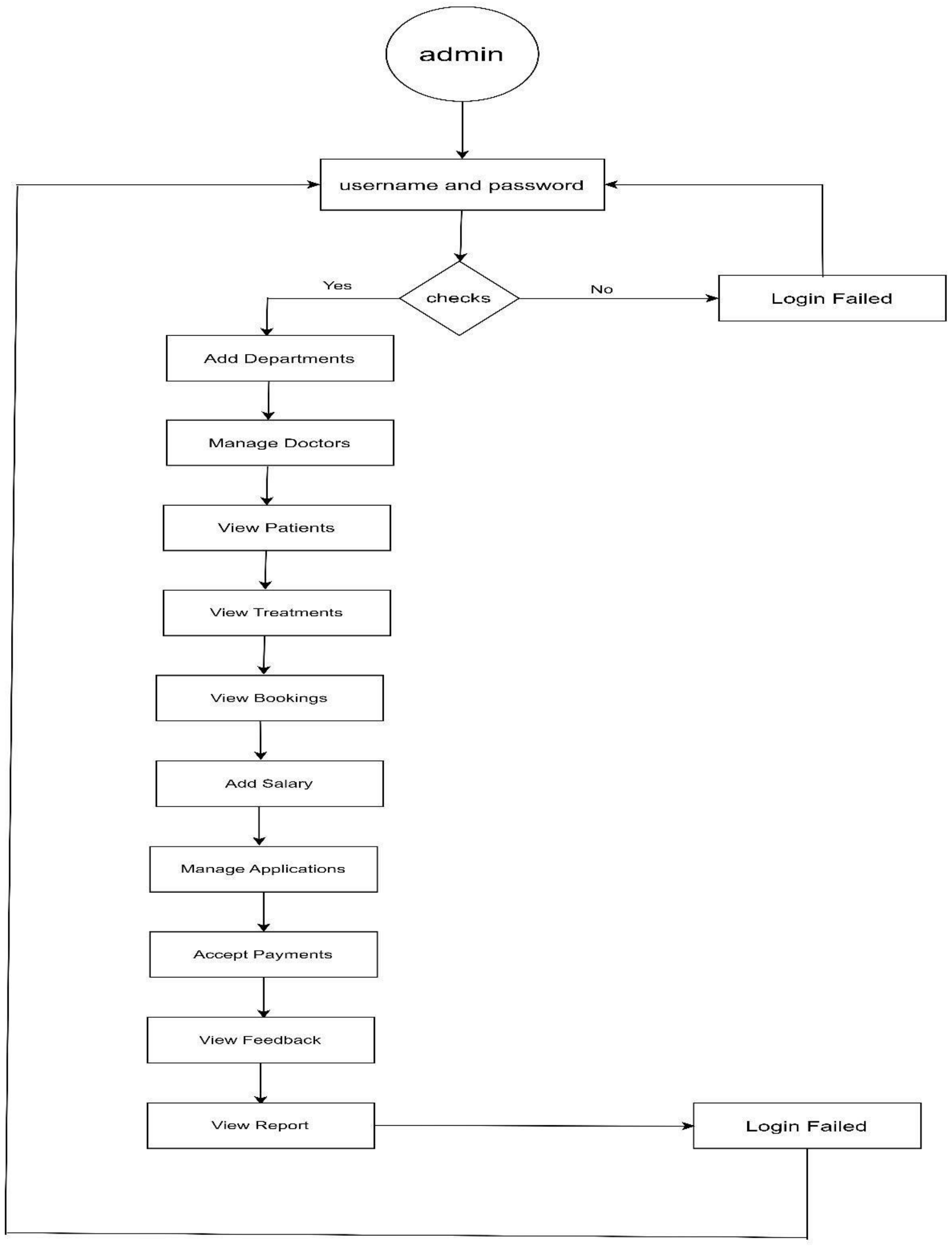
5.4. NUMBER OF MODULES AND THIER DESCRIPTION

The Freelance Doctor apoointment booking system have three modules. They are:

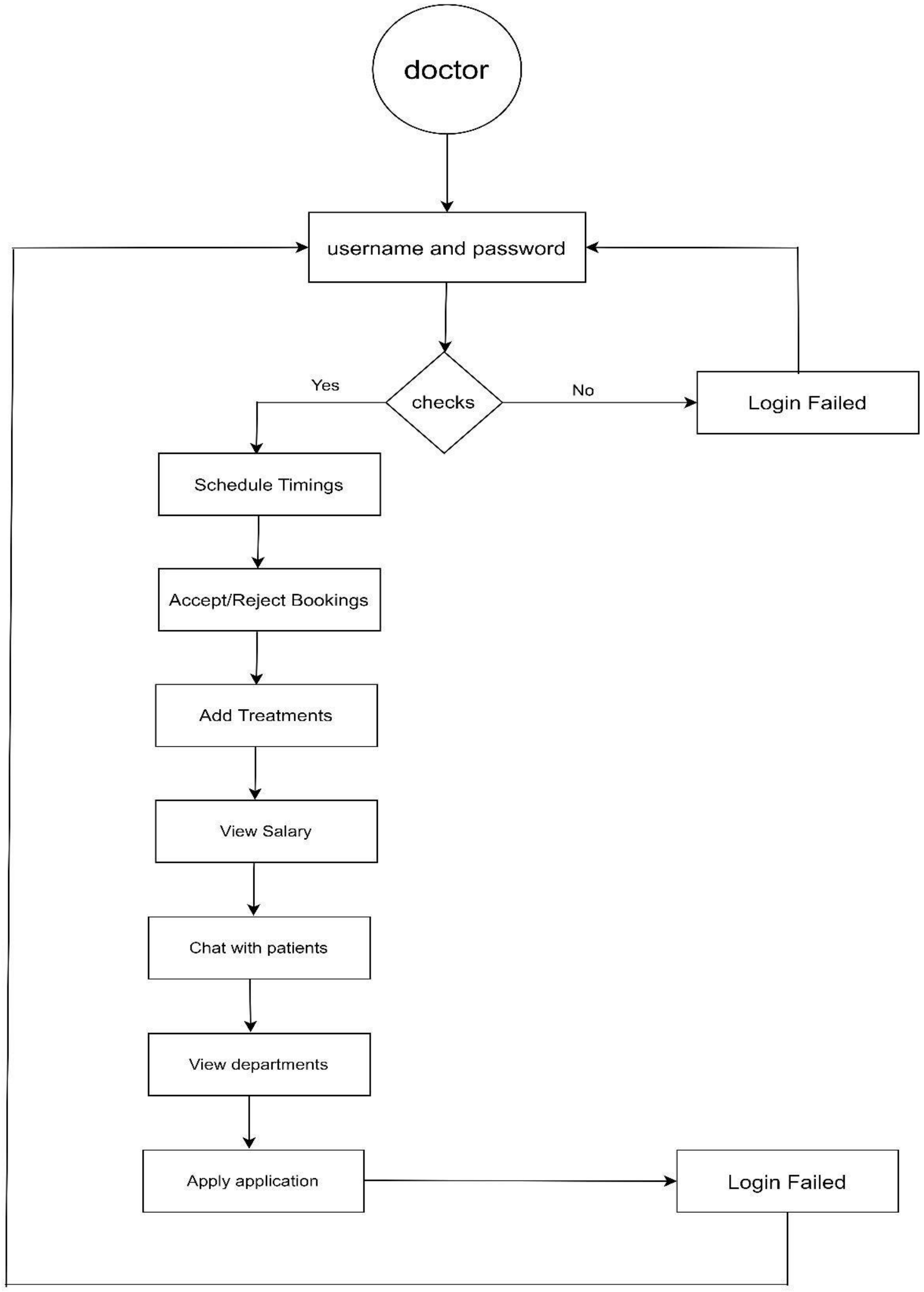
- Admin manages the overall system . The admin can add ,delete , and modify the patient and doctor details. Admin give login credentials to doctors.The admin can add medical departments and receive payments from patients and give salary to doctors and view feedback from patients.
- Doctor**:** The doctor can apply and add treatments to patients and view past treatments. doctors can receive salary and add time and slots of consulting they can receive receipt .
- Patient**:** The patient can book appointments according to their preffered location and type of doctor and they want to see thye can make online payment and get receipt they can view treatment details and past details they can give feedback

5.5. FUNCTIONAL DIAGRAM

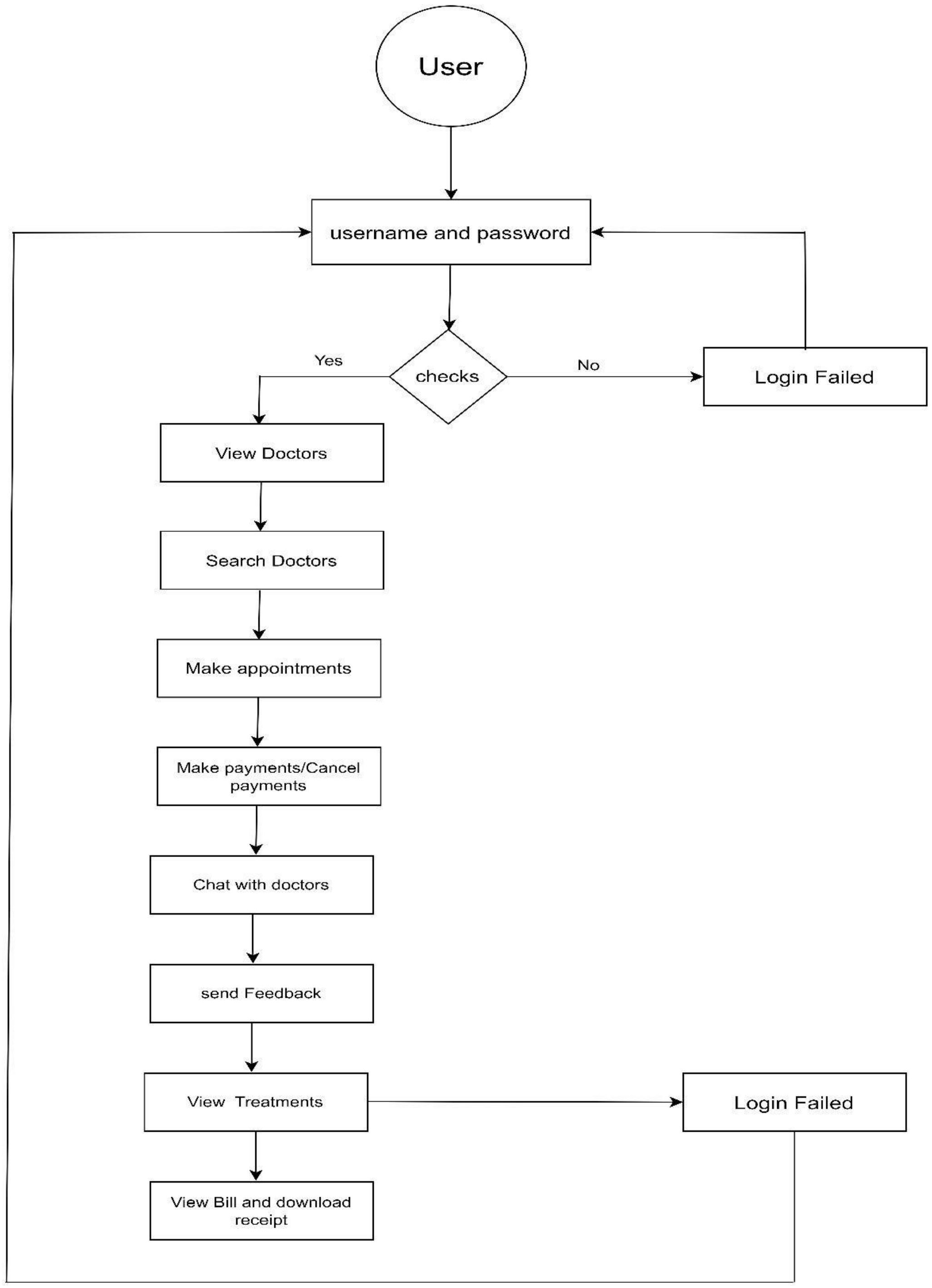
Admin



Doctor


doctor
username and password
checks
Yes
No
Login Failed
Schedule Timings
Accept/Reject Bookings
Add Treatments
View Salary
Chat with patients
View departments
Apply application
Login Failed

Patient


User
username and password
checks
Yes
No
Login Failed
View Doctors
Search Doctors
Make appointments
Make payments/Cancel payments
Chat with doctors
send Feedback
View Treatments
Login Failed
View Bill and download receipt

5.6. DATABASE DESIGN

Login table

Field name	Datatype	Constraints	Description
login_id	Int(11)	Primary key	Login id of login
username	Varchar(100)	null	Username for login
password	Varchar(100)	null	Password for login
usertype	Varchar(100)	null	Usertype of user

Booking table

Field name	Datatype	Constraints	Description
booking_id	Int(11)	Primary key	Id of booking
schedule_id	Int(11)	Null	Schedule id of booking
user_id	Int(11)	Null	User id of user
Adate	Varchar(100)	Null	Date of booking
atime	Varchar(100)	Null	Time of booking
reason	Varchar(100)	Null	Reason for booking
status	Varchar(15)	Null	Status of booking

Chat table

Field name	Datatype	Constraints	Description
chat_id	Int(11)	Primary key	Id of chat
sender_id	Int(11)	Null	Id of sender
reciever_id	Int(11)	Null	Id of reciever
chat	Varchar(100)	Null	Chat description

Department table

Field name	Datatype	Constraints	Description
department_id	Int(11)	Primary key	Id of department
department	Varchar(100)	Null	Name of department
descriptiom	Varchar(100)	Null	Description of department

Table doctor

Field name	Datatype	Constraints	Description
doctor_id	Int(11)	Primary key	Id of doctor
login_id	Int(11)	Null	Id of login
dept_id	Int(11)	Null	Id of department
fname	Varchar(100)	Null	firstname of doctor
lname	Varchar(100)	Null	Lastname of doctor
place	Varchar(100)	Null	Place of doctor
phone	Varchar(100)	Null	Phone of doctor
email	Varchar(100)	Null	Email of doctor
gender	Varchar(100)	Null	Gender of doctor
consulting_place	Varchar(100)	Null	Place of consulting
consulting_time	Varchar(100)	Null	Time of consulting
certificate	Varchar(100)	Null	Certificate of doctor
status	Varchar(100)	Null	Status of application
date	Varchar(100)	Null	Date of application
image	Varchar(100)	Null	Image of doctor

Table feedback

Field name	Datatype	Constraints	Description
feedback_id	Int(11)	Primary key	Id of feedback
user_id	Int(11)	Null	Id of user
feedback	Varchar(100)	Null	Feedback
date	Varchar(100)	Null	Date of feedback

Table payment

Field name	Datatype	Constraints	Description
payment_id	Int(11)	Primary key	Id of payment
booking_id	Int(11)	Null	Id of booking
date	Varchar(100)	Null	Date of booking

Table salary

Field name	Datatype	Constraints	Description
salary_id	Int(11)	Primary key	Id of salary
doctor_id	Int(11)	Null	Id of doctor
amount	Varchar(100)	Null	Amount of salary
date	Varchar(100)	Null	Date of salary
month	Varchar(100)	Null	Month of salary

Table schedule

Field name	Datatype	Constraints	Description
schedule_id	Int(11)	Primary key	Id of schedule
doctor_id	Int(11)	Null	Id of doctor
available_date	Varchar(100)	Null	Date for scheduling
starting_time	Varchar(100)	Null	Starting time of schedule
ending_time	Varchar(100)	Null	Ending time of schedule
interval	Varchar(100)	Null	Interval of schedule
fees	Varchar(100)	Null	Fees of consulting

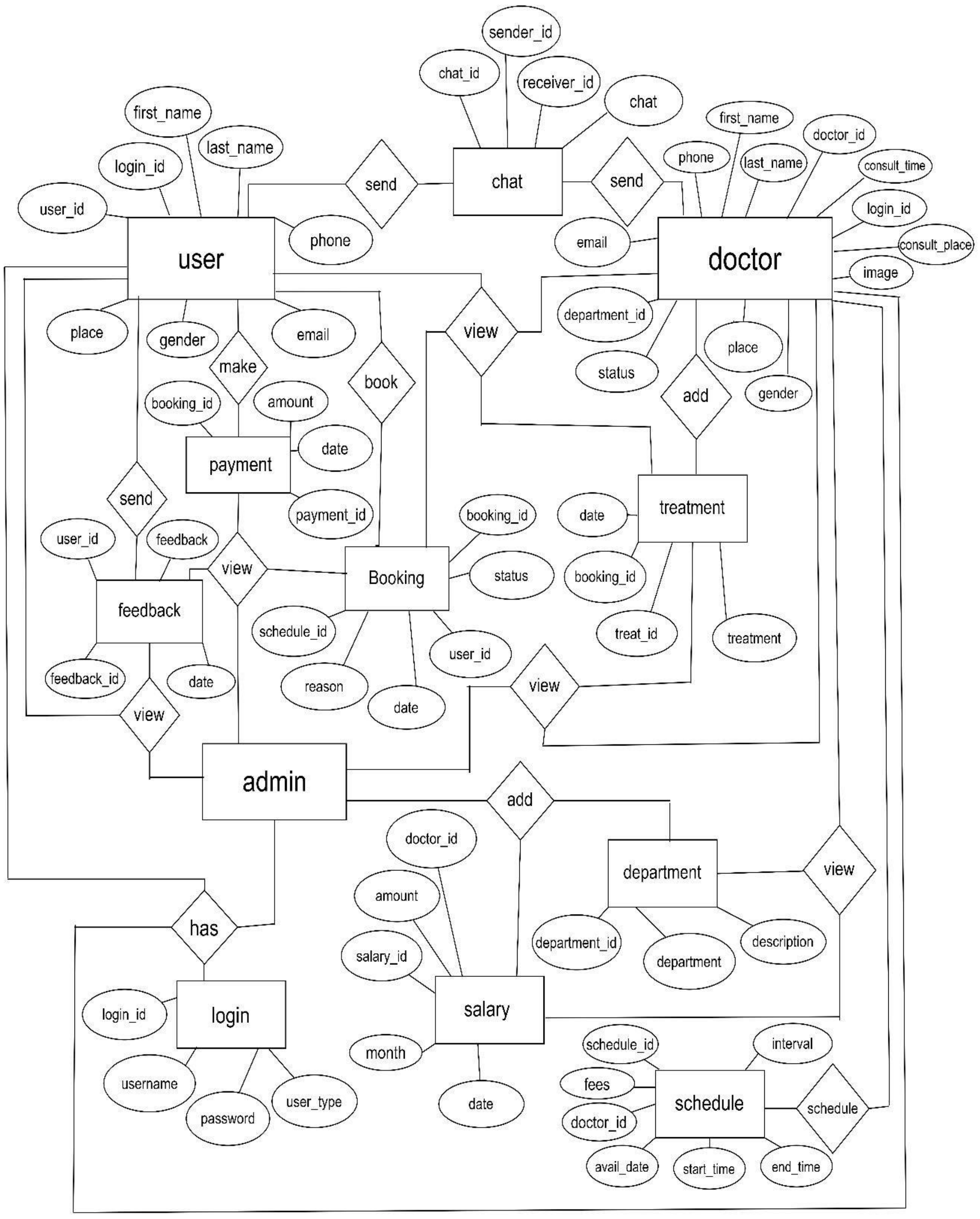
Table treatment

Field name	Datatype	Constraints	Description
treatment_id	Int(11)	Primary key	Id of treatment
treatment	Varchar(100)	Null	Decription of treatment
booking_id	Int(11)	Null	Id of booking
date	Varchar(100)	Null	Date of booking

Table user

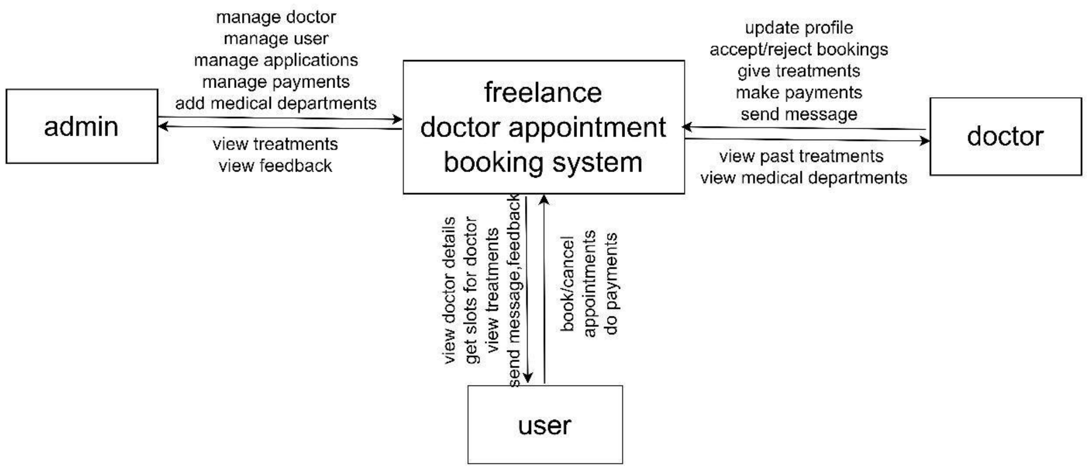
Field name	Datatype	Constraints	Description
user_id	Int(11)	Primary key	Id of user
login_id	Int(11)	Null	Id of login
fname	Varchar(100)	Null	Firstname of user
lname	Varchar(100)	Null	Lastname of user
place	Varchar(100)	Null	Place of user
phone	Varchar(100)	Null	Phone of user
email	Varchar(100)	Null	Email of user

5.7. E R DIAGRAM

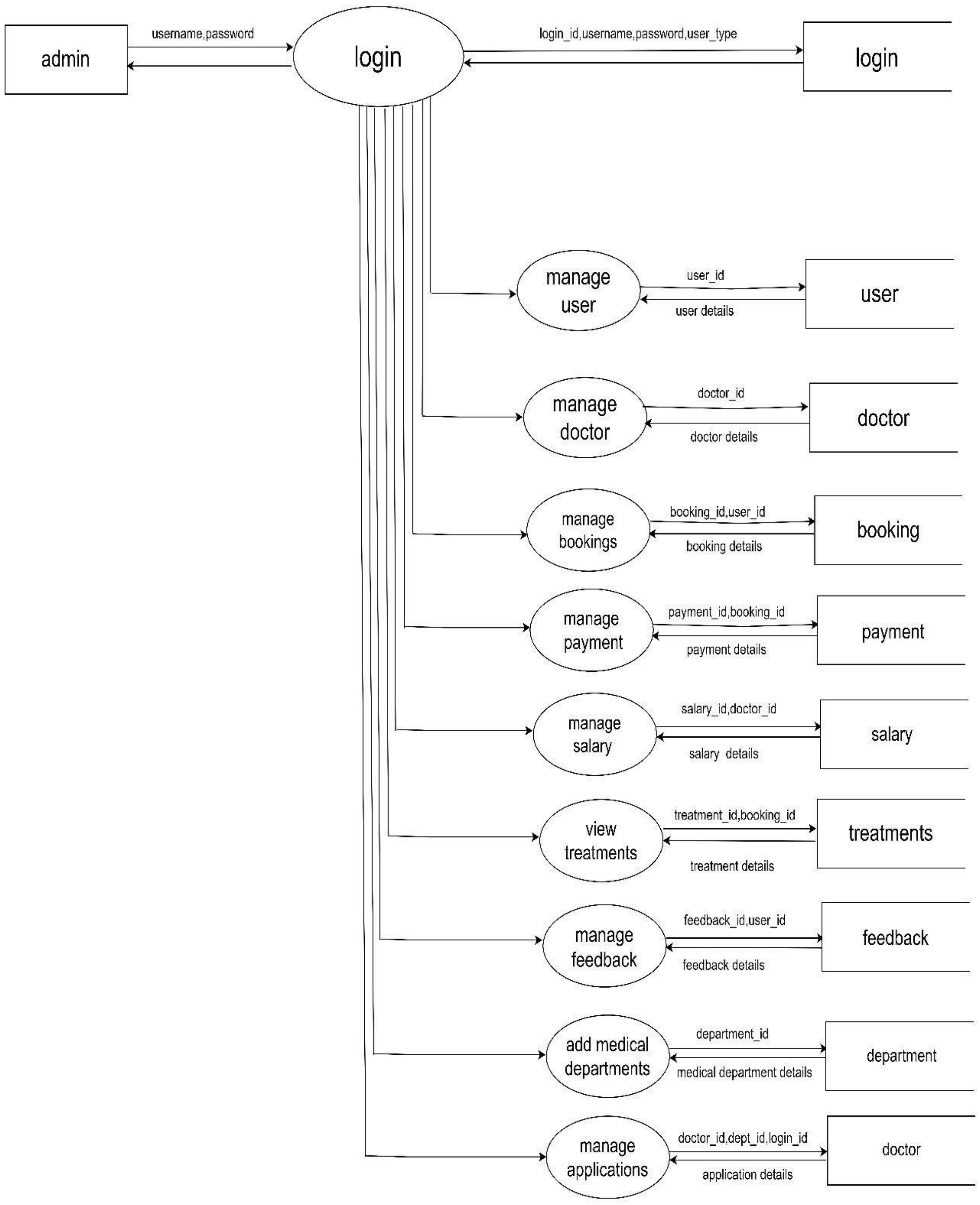


5.8. DATA FLOW DIAGRAM

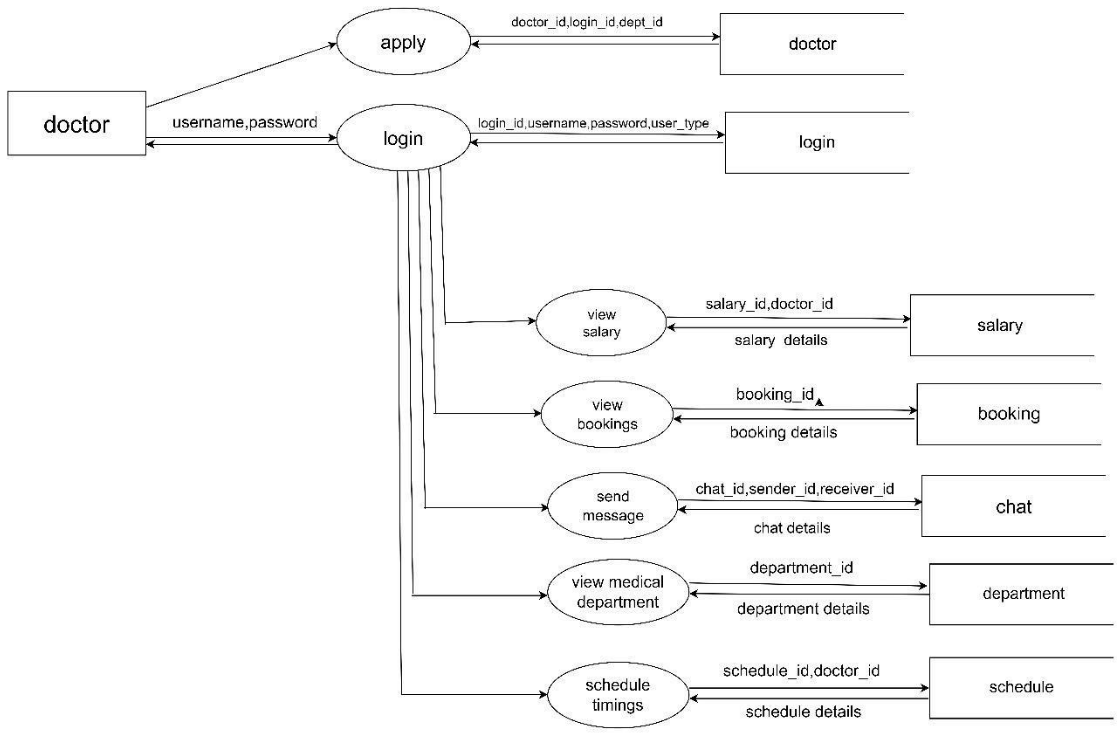
Level 0



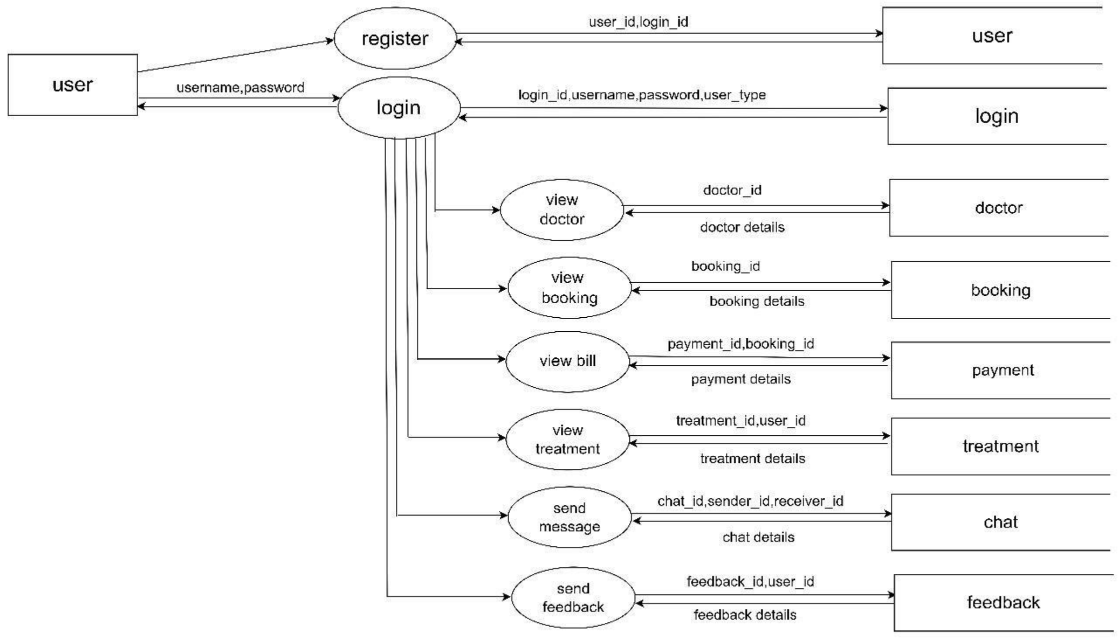
Level 1 admin



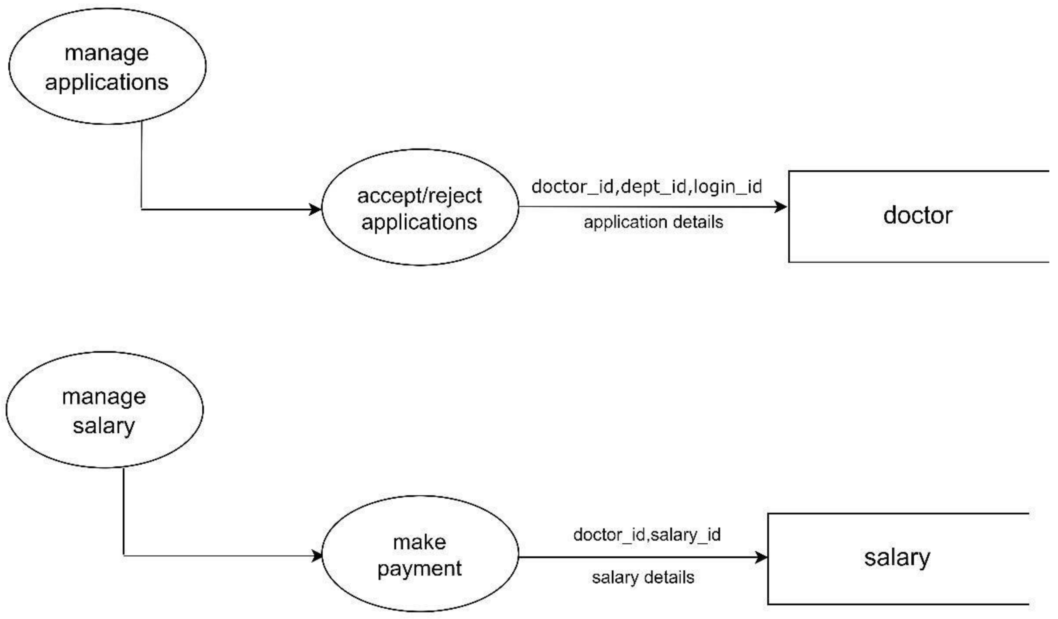
Level 1 doctor



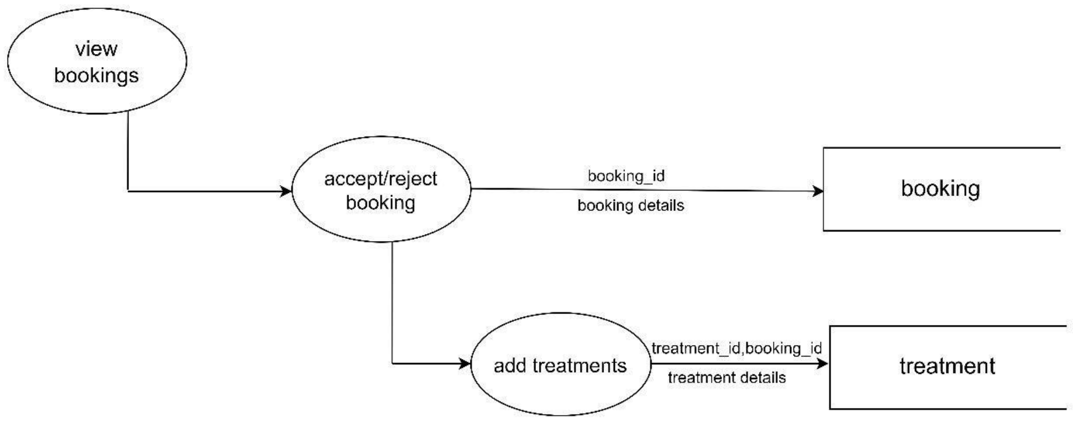
Level 1 patient



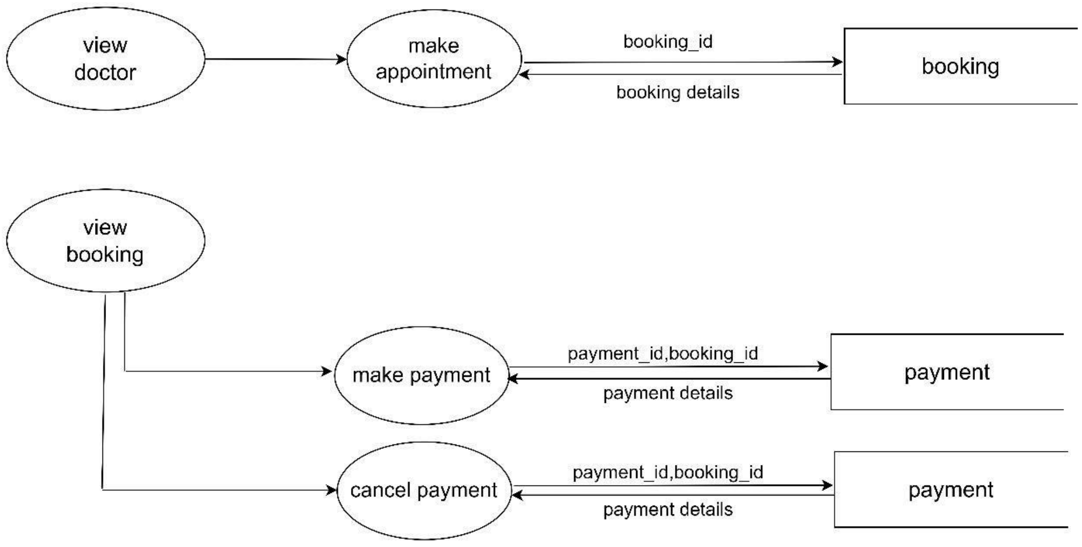
Level 2 admin



Level 2 doctor



Level 2 patient



6. SYSTEM DEVELOPMENT

6.1. PROCESS DESCRIPTION

Proposed system is fully computerized one. With the help of proposed system we can overcome all the limitations of the existing system. The information about products can be retrieved very quickly with a button click when compared to earlier system. It provides a fast and efficient service to both doctors and user. Proposed system is low time consuming, gives accurate results, reliability can improve with the help of security.

6.2. PSEUDO CODE

Admin.py

```
from flask import * from database
import *
admin=Blueprint('admin',__name__
)

@admin.route('/admin_home') def
admin_home():
return render_template('admin_home.html')

@admin.route('/admin_manage_department',methods=['get','post']) def
admin_manage_department():
data={}
q="select * from department"
data['tr']=select(q) if 'submit'
in request.form:
fname=request.form['fname']
des=request.form['des']

q="insert into department values(null,'%s','%s')"%(fname,des)
insert(q) print(q)
return redirect(url_for('admin.admin_manage_department'))

if 'action' in request.args:
```

```
action=request.args['action'] id=request.args['id']

else:
action=None

if action=="delete":
q="delete from department where department_id='%s'"%(id) delete(q)
return redirect(url_for('admin.admin_manage_department'))

if action=="update":
q="select * from department where department_id='%s'"%(id)
res=select(q) data['up']=res



if 'update' in request.form:
fname=request.form['fname']
des=request.form['des']
q="update   department   set    department='%s',description='%s'    where
department_id='%s'"%(fname,des,id) update(q) flash("updated")
return redirect(url_for('admin.admin_manage_department'))



return render_template("admin_manage_department.html",data=data)

@admin.route('/admin_view_request',methods=['get','post']) def
admin_view_request():
data={}
q="SELECT * FROM doctor inner join request using(doctor_id) "
res=select(q) data['tr']=res
```

```
if 'action' in request.args:
action=request.args['action']
id=request.args['id']

else:
action=None

if action=="accept":
q="update request set status='accept' where request_id='%s'"%(id)
update(q) flash('updated successfully')
return redirect(url_for('admin.admin_view_request'))

if action=="reject":
q="update login set status='reject'   where request_id='%s'"%(id)
update(q) flash('updated successfully')
return redirect(url_for('admin.admin_view_request'))

return render_template('admin_view_request.html',data=data)



@admin.route('/admin_view_patient',methods=['get','post'])
def admin_view_patient(): data={} q="SELECT * FROM user
" res=select(q) data['tr']=res

return render_template('admin_view_patient.html',data=data)

@admin.route('/admin_view_doctor',methods=['get','post']) def
admin_view_doctor():
data={}
q="SELECT * FROM doctor inner join login using(login_id) inner join department on
department.department_id=doctor.dept_id " res=select(q) data['tr']=res
```

```
if 'action' in request.args:
action=request.args['action']
id=request.args['id']

else:
action=None

if action=="accept":
q="update login set usertype='doctor' where login_id='%s'"%(id) update(q)
q="update doctor set status='active' where login_id='%s'"%(id) update(q)
flash('updated successfully')
return redirect(url_for('admin.admin_view_doctor'))

if action=="reject":
q="update login set usertype='reject'  where login_id='%s'"%(id) update(q)
q="update doctor set status='inactive'  where login_id='%s'"%(id) update(q)
flash('updated successfully')
return redirect(url_for('admin.admin_view_doctor'))

if action=="delete":
q="delete from doctor  where login_id='%s'"%(id) delete(q)
flash('deleted successfully')
return redirect(url_for('admin.admin_view_doctor'))

return render_template('admin_view_doctor.html',data=data)



@admin.route('/admin_view_booking',methods=['get','post']) def
admin_view_booking():
data={}
q="SELECT * FROM booking INNER JOIN USER USING(user_id) INNER JOIN `payment`
USING(booking_id)" res=select(q) data['tr']=res
```

```
return render_template('admin_view_booking.html',data=data)

@admin.route('/admin_view_payment',methods=['get','post']) def
admin_view_payment():
data={}
q="SELECT * FROM payment inner join booking using(booking_id) inner join user using(user_id)
"
res=select(q) data['tr']=res

return render_template('admin_view_payment.html',data=data)



@admin.route('/admin_view_treatment',methods=['get','post']) def
admin_view_treatment():
data={}
q="SELECT * FROM treatment INNER JOIN booking USING(booking_id) INNER JOIN USER
USING(user_id)  " res=select(q) data['tr']=res

return render_template('admin_view_treatment.html',data=data)

@admin.route('/admin_view_feedback',methods=['get','post']) def
admin_view_feedback():
data={}
q="SELECT * FROM feedback inner join user using(user_id)  "
res=select(q) data['tr']=res

return render_template('admin_view_feedback.html',data=data)
@admin.route('/admin_add_doc_salary',methods=['get','post'])
def admin_add_doc_salary(): data={}
q="select * from department"
print(q) res=select(q)
data['sals']=res
```

```
if 'next' in request.form: dept=request.form['dept']
return redirect(url_for('admin.select_doctor',dept=dept))

q="SELECT * FROM `salary` INNER JOIN `doctor` USING(doctor_id) INNER JOIN `department` ON
`department`.`department_id`=`doctor`.`dept_id`"
res=select(q) data['sal']=res
return render_template('admin_add_doc_salary.html',data=data)
@admin.route('/select_doctor',methods=['get','post'])
def select_doctor():
data={} dept=request.args['dept']

q="select * from doctor where dept_id='%s'"%(dept)
res=select(q) data['doc']=res

if 'pay' in request.form:
doc=request.form['doc']
sal=request.form['sal']
mon=request.form['mon']
q="select * from salary where month='%s' and doctor_id='%s'"%(mon,doc)
res=select(q) if res:
flash("Already Paid") else:
return redirect(url_for('admin.pay_salary',doc=doc,sal=sal,mon=mon)) return
render_template('select_doctor.html',data=data)




@admin.route('/pay_salary',methods=['get','post']) def
pay_salary():
data={} doc=request.args['doc']
sal=request.args['sal']
mon=request.args['mon']
data['amt']=sal
```

```
if 'submit' in request.form:
q="insert into salary values(null,'%s','%s',curdate(),'%s')"%(doc,sal,mon) insert(q)
flash("Paid successfully")
return redirect(url_for('admin.admin_add_doc_salary')) return
redirect(url_for('admin.pay_salary',doc=doc,sal=sal))


return render_template('pay_salary.html',data=data)



@admin.route('/admin_managereport',methods=['post','get']) def
admin_managereport():
data={} if "sale" in
request.form:
daily=request.form['daily'] if
request.form['monthly']=="":
monthly="" else:
monthly=request.form['monthly']+'%'
print(monthly) doc=request.form['doctor']
user=request.form['user']
q="SELECT *,doctor.fname AS doctor,`user`.fname AS `USER` FROM booking INNER JOIN `schedule`
USING (schedule_id) inner join payment using(booking_id)INNER JOIN treatment
USING(booking_id)  INNER JOIN `user` USING(user_id) INNER JOIN doctor USING(doctor_id)
INNER JOIN department ON department.department_id=doctor.dept_id where (`doctor`.fname like
'%s')or (`USER`.fname like '%s') or (`treatment`.date like '%s'  ) or (`treatment`.date like '%s' )
"%(doc,user,daily,monthly)
res=select(q) print(q)
data['report']=res
session['res']=res
r=session['res'] else:
```

```
q="SELECT *,doctor.fname AS doctor,`user`.fname AS `USER` FROM booking INNER JOIN `schedule`
USING (schedule_id)INNER JOIN treatment USING(booking_id)  INNER JOIN `user` USING(user_id)
INNER JOIN doctor USING(doctor_id) INNER JOIN department ON
department.department_id=doctor.dept_id" res=select(q) data['report']=res

return render_template('admin_manage_report.html',data=data)
```

Doctor.py

```
from flask import * from database
import *
doctor=Blueprint('doctor',__name__
)

@doctor.route('/doctor_home') def
doctor_home():
        return render_template('doctor_home.html')


@doctor.route('/doctor_view_booking',methods=['get','post']) def
doctor_view_booking():
        data={}
 q="select *,booking.status as st from booking inner join schedule using (schedule_id) inner join
user using (user_id)  where doctor_id='%s' "%(session['did'])  res=select(q)  data['tr']=res

        if 'action' in request.args:
                action=request.args['action']
                id=request.args['id']
           else:
                action=None

    if action=="accept":          q="update    booking    set    status='accept'    where
booking_id='%s'"%(id)
```

```
                update(q)
                flash('updated successfully')
                return redirect(url_for('doctor.doctor_view_booking'))

    if action=="reject":          q="update   booking   set   status='reject'        where
booking_id='%s'"%(id)
                update(q)
                flash('updated successfully')
                return redirect(url_for('doctor.doctor_view_booking'))

        return render_template('doctor_view_booking.html',data=data)

@doctor.route('/doctor_add_treatment',methods=['get','post']) def
doctor_add_treatment():
        data={}
    id=request.args['id']    q="select
* from treatment"
    data['tr']=select(q)     if
'submit' in request.form:
                fname=request.form['fname']



          q="insert into treatment values(null,'%s','%s',curdate())"%(fname,id)
          insert(q)          print(q)
                return redirect(url_for('doctor.doctor_add_treatment',id=id))
        if 'action' in request.args:
          action=request.args['action']   id=request.args['id']


        else:
                action=None

    if action=="delete":          q="delete      from      department      where
department_id='%s'"%(id)
```

```
                    delete(q)
                    return redirect(url_for('doctor.doctor_add_treatment'))



            return render_template("doctor_add_treatment.html",data=data)



@doctor.route('/doctor_view_salary',methods=['get','post']) def
doctor_view_salary():
            data={}
            did=session['did']

    q="select       *       from       salary       where       doctor_id='%s'"%(did)
    data['tr']=select(q)
            return render_template("doctor_view_salary.html",data=data)

@doctor.route('/doctor_send_application',methods=['get','post']) def
doctor_send_application():
            data={}
            q="select * from application"
            data['tr']=select(q)       if
            'submit' in request.form:
                    date=request.form['date']
              place=request.args['palce']     time=request.form['time']
              certi=request.form['img']

                    q="insert                              into                              treatment
values(null,'%s','%s','%s','%s','%s',curdate())"%(session['did'],date,place,time,certi)
              insert(q)
    print(q)
                    return redirect(url_for('doctor.doctor_send_application'))

            if 'action' in request.args:
                action=request.args['action']
        id=request.args['id']

            else:
                    action=None

        if action=="delete":            q="delete       from       department       where
department_id='%s'"%(id)
                    delete(q)
                    return redirect(url_for('doctor.doctor_send_application'))



            return render_template("doctor_send_application.html",data=data)



@doctor.route('/doctor_view_profile',methods=['get','post']) def
doctor_view_profile():
            data={}
            did=session['did']
            q="SELECT * FROM doctor inner join login using(login_id) where
            doctor_id='%s'"%(did) res=select(q) data['tr']=res
            if 'action' in request.args:
                action=request.args['action']    id=request.args['id']
                        else:
                    action=None

        if action=="update":            q="select   *   from   doctor      where
doctor_id='%s'"%(id)
                res=select(q)
        data['updated']=res



            if 'update' in request.form:
                fname=request.form['fname']
        lname=request.form['lname']
        place=request.form['place']
        phone=request.form['phone']
        email=request.form['email']                 #
gender=request.form['gender']
                        q="update                                   doctor                                   set
fname='%s',lname='%s',place='%s',phone='%s',email='%s'                                   where
doctor_id='%s'"%(fname,lname,place,phone,email,id)
        update(q)
                        flash("updated successfully")

                        return redirect(url_for('doctor.doctor_view_profile'))

                return render_template("doctor_view_profile.html",data=data)



@doctor.route('/doctor_chat')
def psycho_chat():
                data={}
        q="select * from user "
        res=select(q)    data['tr']=res

                return render_template("doctor_chat.html",data=data)



@doctor.route('/doctor_message',methods=['get','post']) def
doctor_message():
                data={}
        receiver_id=request.args['receiver_id']
        sender_id=session['lid']
        data['psycho']=sender_id

 q="select * from chat where (sender_id='%s' and receiver_id='%s') or (receiver_id='%s' and
sender_id='%s')"%(sender_id,receiver_id,sender_id,receiver_id)
```

```
        print(q)
        res1=select(q)
        data['msg']=res1


            if 'submit' in request.form:
                        msg=request.form['message']
                        q="insert into chat values(null,'%s','%s','%s')"%(sender_id,receiver_id,msg)
                        insert(q)
              return redirect(url_for('doctor.doctor_message',receiver_id=receiver_id))      return
render_template("doctor_message.html",data=data)
@doctor.route('/doctor_schedule',methods=['get','post']) def doctor_schedule():
            data={}
            did=session['did']

        from datetime import date,datetime
        today=date.today()      data['today']=today


        q3="select * from schedule where doctor_id='%s'"%(did)
        res=select(q3)              data['sch']=res



            if 'submit' in request.form:
              dt=request.form['date']
        st=request.form['st']                et=request.form['et']
        inte=request.form['inte']
        fee=request.form['fee']



                        dss = dt
```

```
                # Split the string into a list using comma as separator
        new_dss = dss.split(",")

                # Join the list elements with a comma separator and print       print(",".join(new_dss))
                for i in new_dss:
        print(i)

                        q="select          *          from     schedule          where  available_date='%s'
        and doctor_id='%s'"%(dt,did)
                        res=select(q)
        if res:
                                        return     HttpResponse("<script>alert('Choose     Another
Date....!!');window.location='/doctor_schedule';</script>")
                        else:
                            q="insert          into       schedule
values(null,'%s','%s','%s','%s','%s','%s')"%(did,i,st,et,inte,fee)
                                        insert(q)
                                        #          return          HttpResponse("<script>alert('Added
Successfully....!!');window.location='/doctor_schedule_time';</script>")
        return render_template("doctor_schedule.html",data=data)
```

User.py

```
from flask import * from
database import *
user=Blueprint('user',__name__)

@user.route('/user_home') def
user_home():
     return render_template('user_home.html')
```

```
@user.route('/user_view_profile',methods=['get','post']) def
user_view_profile():
        data={}

        uid=session['uid']
        q="SELECT * FROM user inner join login using(login_id) where user_id='%s'"%(uid)
    res=select(q)    data['tr']=res

        if 'action' in request.args:
          action=request.args['action']
    id=request.args['id']
                else:
               action=None

    if action=="update":           q="select  *  from  user    where
user_id='%s'"%(id)
          res=select(q)
    data['updated']=res


        if 'update' in request.form:
          fname=request.form['fname']
    lname=request.form['lname']
    place=request.form['place']
    phone=request.form['phone']
    email=request.form['email']           #
gender=request.form['gender']
    q="update user set
fname='%s',lname='%s',place='%s',phone='%s',email='%s'                    where
user_id='%s'"%(fname,lname,place,phone,email,id)
               update(q)
               flash("updated successfully")
```

```
            return redirect(url_for('user.user_view_profile')) return
        render_template('user_view_profile.html',data=data)
        @user.route('/user_view_doctor',methods=['get','post']) def
        user_view_doctor():
    data={}          if    "search"    in
request.form:
                    p=request.form['product']+'%'
                    q="SELECT * FROM doctor INNER JOIN `department` on
department.department_id=doctor.dept_id  WHERE fname LIKE '%s' and status='active'"%(p)
            res=select(q)
    data['tr']=res



    else:


                    q="SELECT * FROM doctor INNER JOIN `department` on
department.department_id=doctor.dept_id where status='active'"
            res=select(q)
    data['tr']=res              print(q)
            print(res)

        return render_template('user_view_doctor.html',data=data)






@user.route('/user_select_timing',methods=['get','post']) def
user_select_timing():
        from datetime import date,datetime,timedelta

        data={}
        uid=session['uid']
        id=request.args['id']
        date=request.args['date']
        data['date']=date


        q="SELECT * FROM `schedule` where schedule_id='%s'"%(id)
        r=select(q)       if r:
                          print(q)
                          data['con']=r



                x = r[0]['starting_time']
        y = r[0]['ending_time']
        ttt=r[0]['interval']
        ttime=int(ttt)
        hour_and_minute=x
                          date_time_obj = datetime.strptime(x, '%H:%M')
  s=[]            while    hour_and_minute<y:                if
hour_and_minute<y:
                                    date_time_obj += timedelta(minutes=ttime)
        hour_and_minute = date_time_obj.strftime("%H:%M")
                                                  print(hour_and_minute)

                                                  s.append(hour_and_minute)


        else:                       break
        data['s']=s       print(s)

             if 'time' in request.form:
                date=request.form['date']
        reason=request.form['reason']
        time=request.form['date']
```

```
                q="select * from booking where adate='%s' and atime='%s'"%(date,time)
            res=select(q)
    if res:
                flash("Please Choose Another Time")            else:
                    q="insert into booking
values(null,'%s','%s','%s','%s','%s','%s')"%(id,session['uid'],date,time,reason,'pending')
                insert(q)
    print(q)
                flash("Booked Successfully.....!!!")
    return redirect(url_for('user.user_view_booking'))
        return render_template('user_select_timing.html',data=data)

@user.route('/user_view_treatment',methods=['get','post']) def
user_view_treatment():
        data={}
        uid=session['uid']
    q="SELECT * FROM treatment INNER JOIN booking USING(booking_id)  INNER JOIN `schedule`
USING(schedule_id) INNER JOIN DOCTOR USING(doctor_id) inner join login using(login_id) where
user_id='%s'"%(uid)
    res=select(q)    data['tr']=res

    return render_template('user_view_treatment.html',data=data)
@user.route('/user_make_appoinment',methods=['get','post']) def user_make_appoinment():
        data={}
        id=request.args['id']



    q="select * from schedule where doctor_id='%s'"%(id)
    res=select(q)    data['sch']=res           print(res)



        # q="select * from booking where user_id='%s'"%(session['uid'])
        # data['bok']=select(q)
```

```
            # if 'submit' in request.form:
            #           det=request.form['detail']
        #       id=request.args['id']    #
        q="insert into booking
values(null,'%s','%s','%s','pending',curdate())"%(id,session['uid'],det)
            #           insert(q)
            #           print(q)
            #           return redirect(url_for('user.user_view_doctor'))

            # if 'action' in request.args:
            #           action=request.args['action']
            #           id=request.args['id']

            #          else:          #
                action=None

            # if action=="delete":
            #           q="delete from booking where booking_id='%s'"%(id)
            #           delete(q)
            #           return redirect(url_for('user.user_view_doctor'))

            return render_template('user_make_appoinment.html',data=data)

@user.route('/user_view_booking',methods=['get','post']) def
user_view_booking():
            data={}
            q="SELECT *,booking.status AS b_st FROM booking INNER JOIN SCHEDULE USING
(schedule_id) INNER JOIN doctor USING (doctor_id) INNER JOIN department ON
department.department_id=doctor.dept_id WHERE user_id='%s'"%(session['uid'])
```

```
        print(q)
        res=select(q)    print(res)
        data['bok']=res

            if 'action' in request.args:
                action=request.args['action']
        id=request.args['id']

            else:
                        action=None

        if action=="delete":                  q="delete      from      booking      where
booking_id='%s'"%(id)
                        delete(q)
                flash("refunded your amount..!")                      return
redirect(url_for('user.user_view_doctor'))

            return render_template('user_view_booking.html',data=data)
            @user.route('/user_make_payment',methods=['get','post']) def user_make_payment():
            data={}
        amt=request.args['amt']
        data['amt']=amt



            if 'submit' in request.form:
                det=request.form['amt']
        exp=request.form['exp']
        card=request.form['card']
        cvv=request.form['cvv']                        id=request.args['id']
                        q="insert into payment values(null,'%s',curdate(),'%s')"%(id,det)
                        insert(q)
                        q="update booking set status='paid' where booking_id='%s'"%(id)
                        update(q)
```

```
                    flash("paid successfully")
                    print(q)
                    return redirect(url_for('user.user_view_doctor'))

          return render_template('user_make_payment.html',data=data)

@user.route('/viewinvoice') def
viewinvoice():
  data={}
  from datetime import date,datetime
today=date.today()    data['today']=today
omid=request.args['id']
  q="SELECT *,payment.date as p_date FROM booking INNER JOIN payment USING (booking_id)
INNER JOIN `schedule` USING(schedule_id) INNER JOIN doctor USING(doctor_id) INNER JOIN
department ON doctor.dept_id=department.department_id WHERE user_id='%s' and
booking_id='%s'"%(session['uid'],omid)    print(q)    data['pay']=select(q)
  return render_template("bill.html",data=data)



@user.route('/user_chat') def
psycho_chat():
          data={}
     q="select * from doctor where status='active'"
     res=select(q)   data['tr']=res

          return render_template("user_chat.html",data=data)


@user.route('/user_message',methods=['get','post']) def
user_message():
          data={}
```

```
        id=request.args['id']
        uid=session['lid']
        data['user']=uid

        q="select * from chat where (sender_id='%s' and receiver_id='%s') or (receiver_id='%s' or
sender_id='%s')"%(uid,id,uid,id)     res1=select(q)              data['msg']=res1           if 'submit' in
request.form:
                        msg=request.form['message']
                        q="insert into chat values(null,'%s','%s','%s')"%(uid,id,msg)
                        insert(q)
                        flash("send message..")
                return redirect(url_for('user.user_message',id=id))     return
render_template("user_message.html",data=data)



@user.route('/patient_add_feedback',methods=['get','post']) def
patient_add_feedback():
            data={}
            uid=session['uid']
        q="select * from feedback where user_id='%s'"%(session['uid'])         data['feed']=select(q)

            if 'submit' in request.form:
                        feed=request.form['feed']
                        q="insert into feedback values(null,'%s','%s',curdate())"%(uid,feed)
                insert(q)
        print(q)
                        flash("send successfully..")
                        return redirect(url_for('user.patient_add_feedback'))
        return render_template('patient_add_feedback.html',data=data)
```

Main.py

```
from flask import Flask from
public import public from
admin import admin from
user import user from
doctor import doctor



app=Flask(__name__) app.secret_key="helooo"
app.register_blueprint(public)
app.register_blueprint(admin,url_prefix='/admin')
app.register_blueprint(user,url_prefix='/user')
app.register_blueprint(doctor,url_prefix='/doctor')



app.run(debug=True,port=5476
```

Public.py

```
from flask import * from database
import * import uuid
public=Blueprint('public',__name__)
```

```
@public.route('/') def
home():
return render_template('home.html')

@public.route('/login',methods=['get','post'])
def login(): session.clear() if "submit" in
request.form: u=request.form['uname']
p=request.form['password']
q="select * from login where username='%s' and password='%s'"%(u,p)
print(q) res=select(q) if res:
session['lid']=res[0]['login_id'] if
res[0]['usertype']=="admin":
flash("Logging in")
return redirect(url_for("admin.admin_home"))

elif res[0]['usertype']=="doctor":
q="select * from doctor where login_id='%s'"%(session['lid'])
res1=select(q) if res1:
session['did']=res1[0]['doctor_id']
print(session['did']) flash("Logging
in")
return redirect(url_for("doctor.doctor_home"))

elif res[0]['usertype']=="user":
q="select * from user where login_id='%s'"%(session['lid'])
res1=select(q) if res1:
session['uid']=res1[0]['user_id']
print(session['uid']) flash("Logging
in")
return redirect(url_for("user.user_home"))

# elif res[0]['usertype']=="seller":
#           q="select * from seller where login_id='%s'"%(res[0]['login_id'])
#     res1=select(q)          #
      if res1:
#                             session['sid']=res1[0]['seller_id']
#                             print(session['sid'])
#                             flash("Logging in")
#                             return redirect(url_for("rescue.rescue_home"))




# elif res[0]['type']=="customer":
#     q="select * from customer inner join login using (username) where username='%s' and
status='0'"%(u)
#             res=select(q)
#             if res:
#                             flash('inactive')
#             else:


#                             q="select * from customer where username='%s'"%(lid)
#                             res=select(q)
#                             if res:
#                                             session['customer_id']=res[0]['customer_id']
#                                             cid=session['customer_id']
#                             return redirect(url_for('customer.customer_home'))

else:
flash("Registration Under Process") flash("You
are Not Registered")

return render_template('login.html')
```

```
@public.route('/reg',methods=['get','post'])
def reg(): data={}
dept=request.args['dept']

if 'submit' in request.form:

fname=request.form['fname']
lname=request.form['lname']
place=request.form['place']
phone=request.form['phone']
email=request.form['email']
gen=request.form['gender']
pl=request.form['pl']
time=request.form['time']
i=request.files['img']
path='static/'+str(uuid.uuid4())+i.filename i.save(path)
img=request.files['image']
src='static/'+str(uuid.uuid4())+i.filename img.save(src)



uname=request.form['username'] pas=request.form['password']
q="insert into login values(null,'%s','%s','pending')"%(uname,pas)
id=insert(q) print(q)
q="insert                                         into                                         doctor
values(null,'%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','active',curdate(),'%s')"%(id,dept,
fname,lname,place,phone,email,gen,place,time,path,src) insert(q)
flash("Registered successfully..!")

return redirect(url_for('public.login'))

return render_template("reg.html",data=data)
```

```
@public.route('/userreg',methods=['get','post']) def
userreg():


if 'submit' in request.form:

fname=request.form['fname'] lname=request.form['lname']

place=request.form['place']
phone=request.form['phone']
email=request.form['email']

uname=request.form['username'] pas=request.form['password']
q="insert into login values(null,'%s','%s','user')"%(uname,pas)
id=insert(q) print(q)
q="insert                                   into                                   user
values(null,'%s','%s','%s','%s','%s','%s')"%(id,fname,lname,place,phone,email) insert(q)
flash("Registered successfully..!")

return redirect(url_for('public.login'))
return render_template("userreg.html")



@public.route('/View_depts') def
View_depts():
data={} q="SELECT * FROM
department " res=select(q)
data['dept']=res
return render_template('View_depts.html',data=data)
```

7. VALIDATION CHECKS

The validation phase reveals the failures and the buds in the developed system. It will become to known about the practical difficulties the system faces when the operated in the true environment. Validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- Has the user filled in all required fields?
- Has the user entered a valid email?
- Has the user entered text in a numeric field?

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server. Most often, the purpose of validation is to ensure correct user input. Validation can be defined by many different methods, and deployed in many different ways. Serverside validation is performed by a web server, after input has been sent to the server. Client-side validation is performed by a web browser, before input is sent to a web server

8. SYSTEM IMPLEMENTATION

8.1 Testing

Testing focuses on the logical internals of the software, ensuring that all the statements have been tested on the functional external, that is, conducting tests using various test data to detect errors and ensure that defined input will produce actual results that agree with required results. It is the major quality control measure used during software development. It uncovers the errors introduced:

- During Coding
- During other previous phases like: Requirement Analysis and Designing Levels of

Testing

1. Unit Testing
2. Integration Testing
3. Validation Testing
4. Output Testing

Unit Testing

Different modules are tested against the specifications produced during the design for the modules. Unit testing is essentially for verification of the code produced during the coding phase. Its main goal is to test the internal logic of the modules, typically done by the programmer of the module. Main focus in this testing is testing the code.

Integration Testing

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration plan to those aggregates, and delivers as its output. The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items.

Output Testing

No system could be useful if it does not produce the required output in the specific format. Output testing is performed to ensure the correctness of the output and its format. The output generated or displayed by the system is tested asking the users about the format required by them.

Validation Testing

In software project management, software testing, and software engineering, validation is the process of checking that a software system meets specifications and that it fulfills its intended purpose. The errors which are uncovered during the integration testing are corrected during this phase.

8.2 SYSTEM IMPLEMENTATION

Implementation is the stage in the project where theoretical design is turned into a working system and is giving confidence on the new system for the users which will work efficiently and effectively. It involves careful planning, investigation of the current system and its constraints on implementations, design of methods to achieve the changeover, and evaluation of changeover methods. Apart from planning, major tasks for preparing the implementation are education and training of users. The major complex system being implemented the more evolved will be the system analysis and the design effort required just for implementation. An implementation coordination committee based on policies of individual organizations has been appointed. The implementation process begins with preparing a plan for implementation of the system. According to this plan the activities are to carry out discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system.

Implementation is the final and important phase. The most critical stage in achieving a successful new system and in giving the users confidence in the new system and in giving the users confidence that the new system will work and be effective. The system can be implemented only after thorough testing is done and if found to work according to the specification.

8.3 SECURITY

In any organization data is the most important element and the main issue related to it is the security of those valuable data. One of the major areas in the development process of a system is providing security to all its data in an efficient way. In my work, as it is for a Wholesale distribution management it is tightly protected by an authentication session password system. Only the administrator can access the entire system. The database server is equipped with an efficient password security system. So the entire system is provided with tight security.

9. FUTURE SCOPE OF THE PROJECT

The future scope of a freelance doctor appointment booking system is vast, and there are many potential areas for growth and improvement in the coming years. Here are some of the potential future developments and features that could be added to a freelance doctor appointment system:

- Integration with wearable technology: As wearable technology continues to advance, it may be possible to integrate these devices with a freelance doctor appointment system, allowing doctors to monitor patient health in real-time and provide more personalized and effective care.
- Telemedicine: Telemedicine is becoming increasingly popular as a way to provide remote medical care. In the future, freelance doctor appointment systems may include more advanced telemedicine capabilities, such as video consultations and remote monitoring.
- Artificial intelligence (AI) and machine learning (ML): AI and ML have the potential to revolutionize healthcare, and freelance doctor appointment systems could benefit from these technologies. For example, AI could be used to match patients with the most suitable freelance doctors based on their symptoms and medical history.
- Blockchain technology: Blockchain technology has the potential to improve security and privacy in healthcare. Freelance doctor appointment systems could use blockchain to store and manage patient data securely and transparently.
- Virtual reality (VR): VR technology could be used to create immersive virtual environments for medical consultations and procedures, allowing patients to feel more comfortable and relaxed during their appointments.
- Predictive analytics: Predictive analytics could be used to identify potential health risks and allow freelance doctors to take preventative measures before a patient's condition worsens.
- Collaborative care: In the future, freelance doctor appointment systems could include more collaboration and coordination between freelance doctors and other healthcare providers, such as hospitals, clinics, and pharmacies.

Overall, the future scope of a freelance doctor appointment booking system is vast and full of exciting possibilities. As technology continues to advance, these systems will become more sophisticated, secure, and effective, providing patients with better access to healthcare and freelance doctors with more opportunities to provide high-quality care.

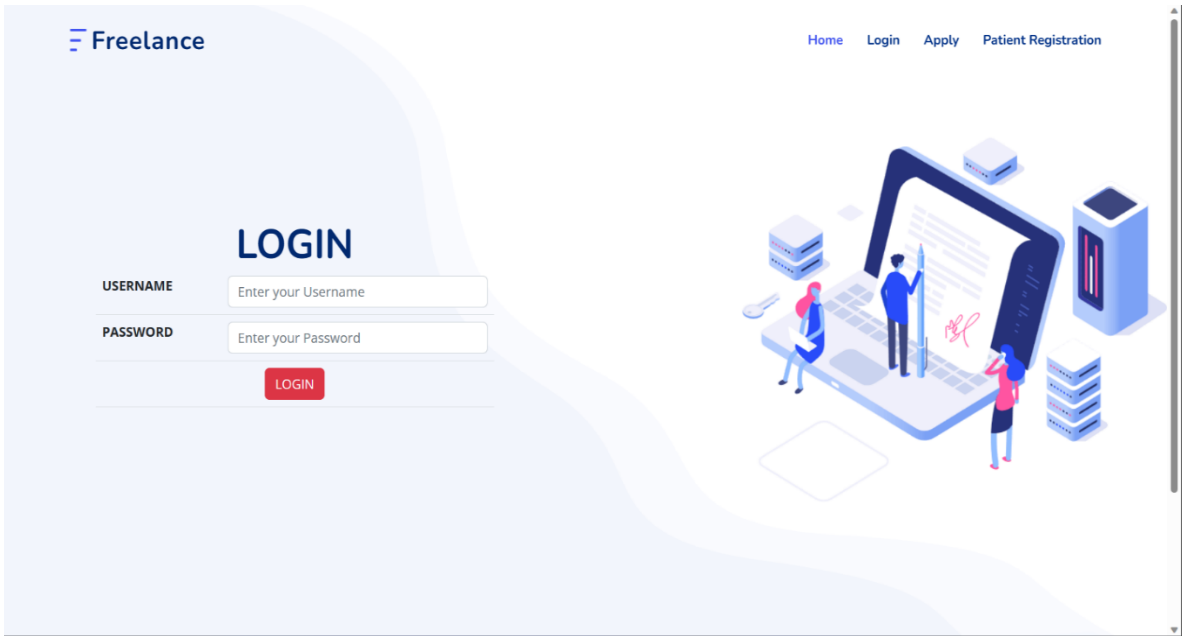
10. CONCLUSION

The Project title “ Freelance doctor appointment system ” is wholly computerized thus making it user friendly. The system is more efficient , reliable, accurate and fast. It is a solution for problems of wholesale firms .The project has a very vast scope in future.It helps to reduce manual work. In the future the system can be automated using AI without admin. As the internet has become an inevitable part of everyday lives and convenience of booking doctors from anywhere provides an enhancement of this project. In future the Administrator of the web site can be given more functionality, like looking at a specific users profile, This currently designed for a particular individual can be made collaborative by including various other clinics or hospitals.

11. APPENDIX

11.1. SAMPLE INPUTS

- **login**

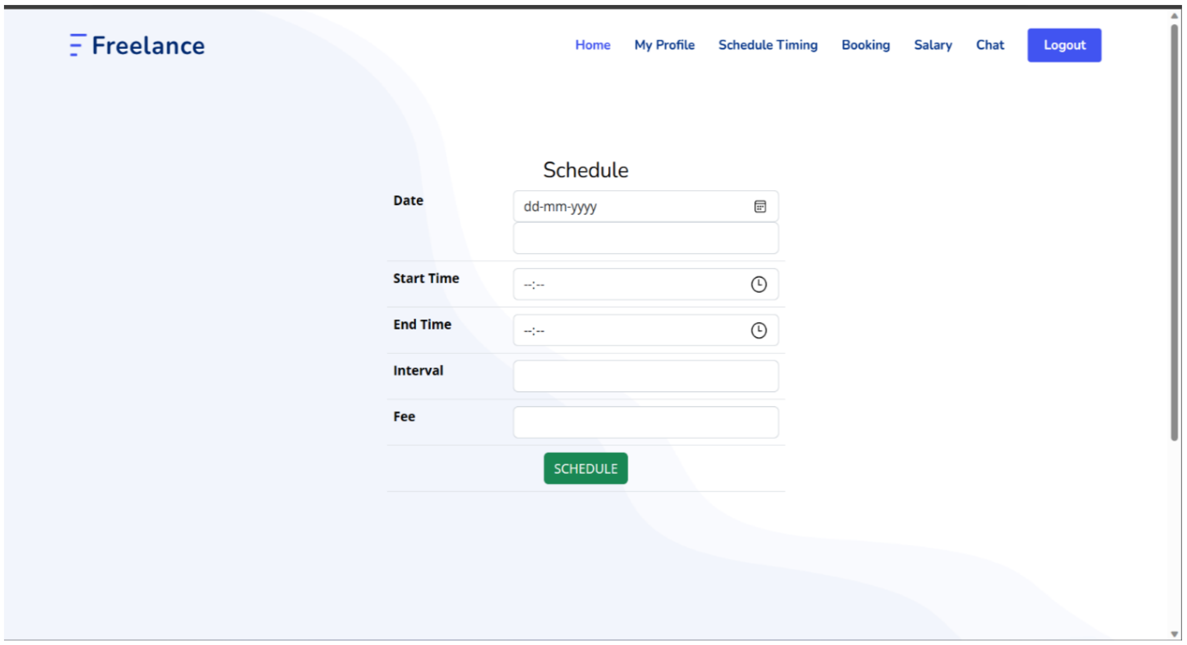


LOGIN

USERNAME	Enter your Username
PASSWORD	Enter your Password

LOGIN

- **schedule**

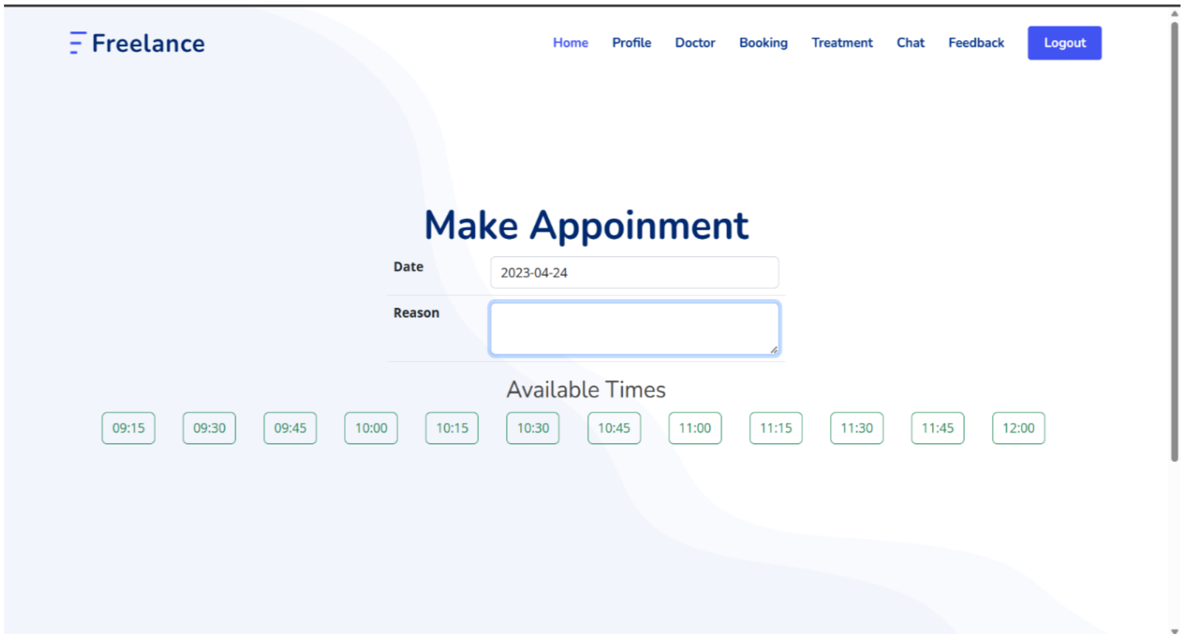


Schedule

Date	dd-mm-yyyy
Start Time	--:--
End Time	--:--
Interval	
Fee	

SCHEDULE

- **make appointment**



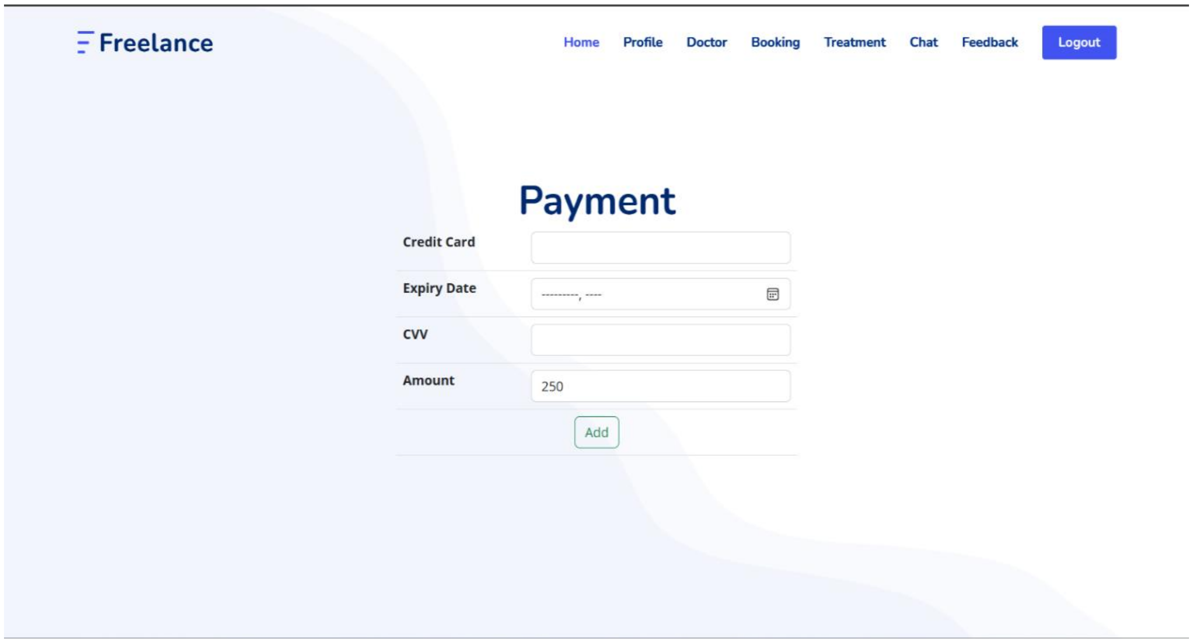
Make Appoinment

Date	2023-04-24
Reason	

Available Times

09:15 09:30 09:45 10:00 10:15 10:30 10:45 11:00 11:15 11:30 11:45 12:00

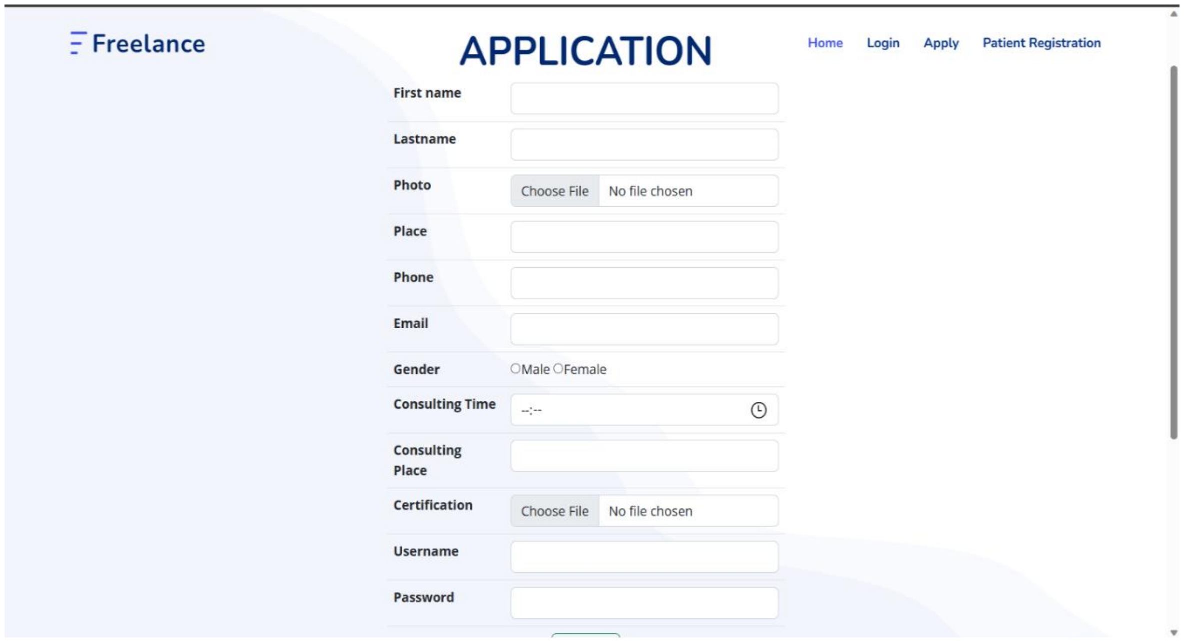
- **payment**



Payment

Credit Card	
Expiry Date	---------, ----
CVV	
Amount	250
	Add

- **doctor application**



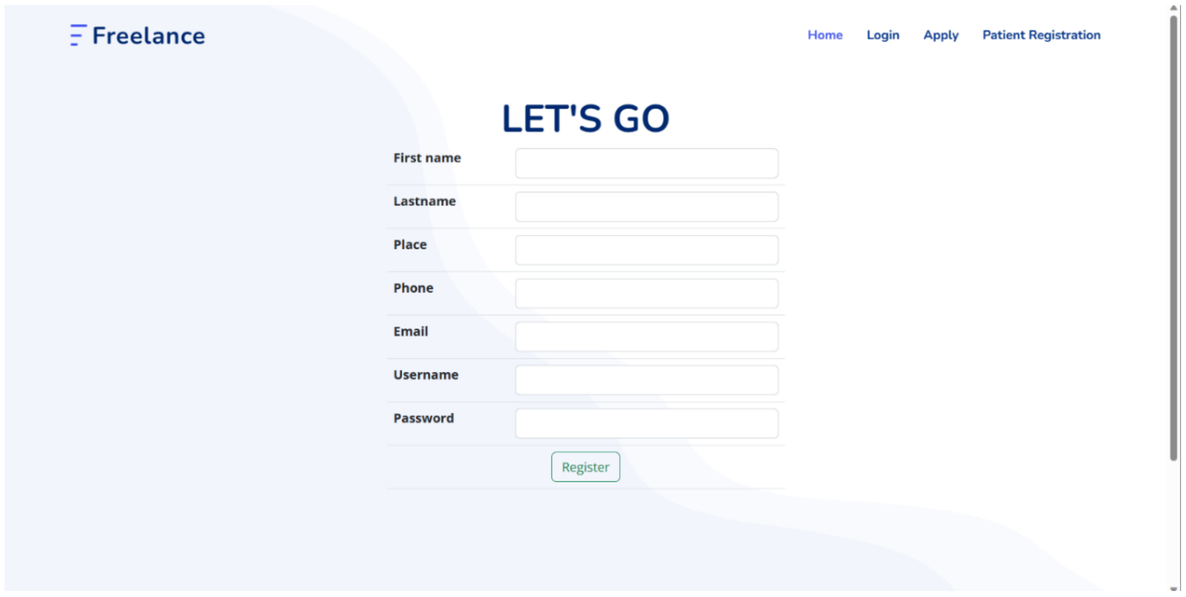
Freelance

Home Login Apply Patient Registration

APPLICATION

First name	
Lastname	
Photo	Choose File No file chosen
Place	
Phone	
Email	
Gender	○Male ○Female
Consulting Time	--:--
Consulting Place	
Certification	Choose File No file chosen
Username	
Password	

- **patient registration**



Freelance

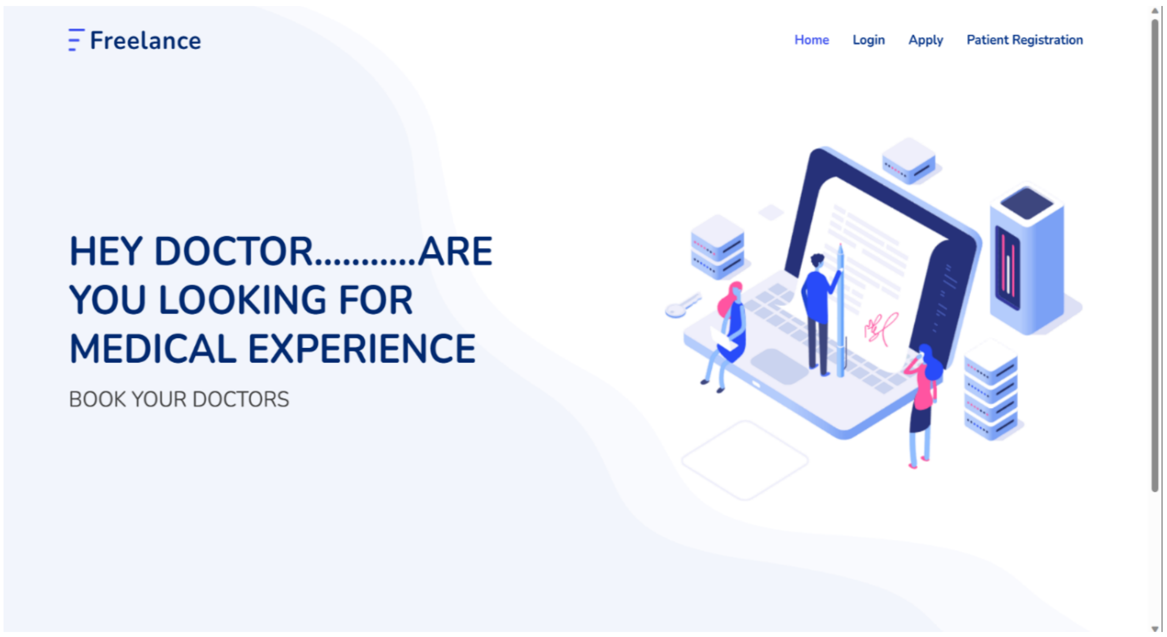
Home Login Apply Patient Registration

LET'S GO

First name	
Lastname	
Place	
Phone	
Email	
Username	
Password	

Register

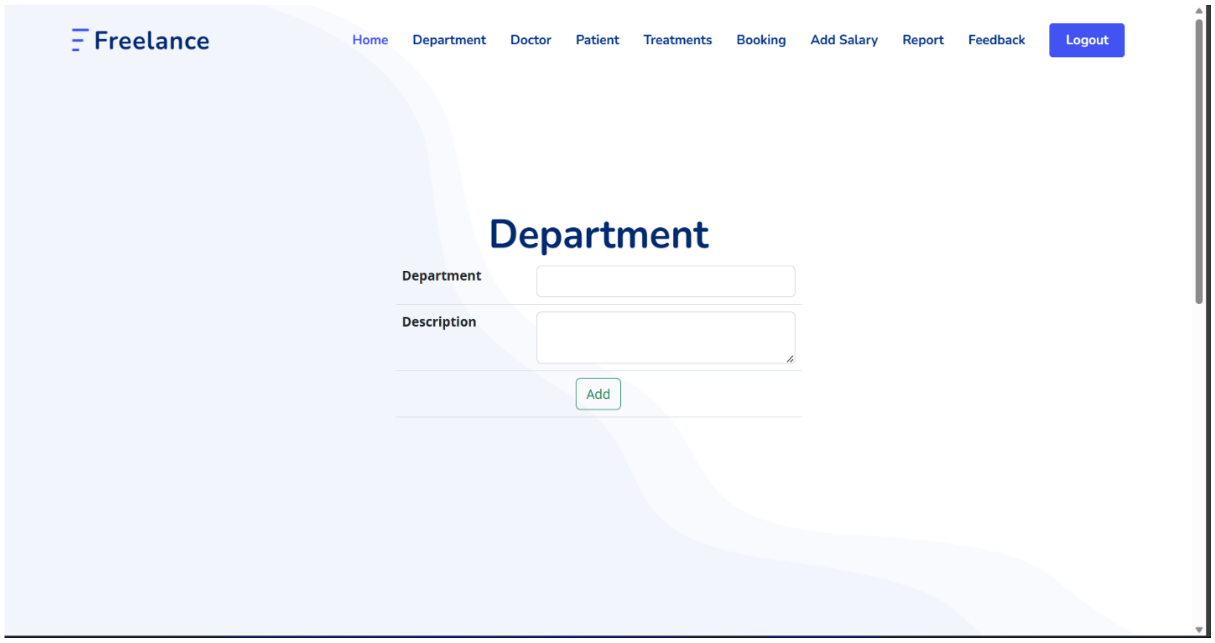
- **index**



HEY DOCTOR...........ARE YOU LOOKING FOR MEDICAL EXPERIENCE

BOOK YOUR DOCTORS

- **add department**

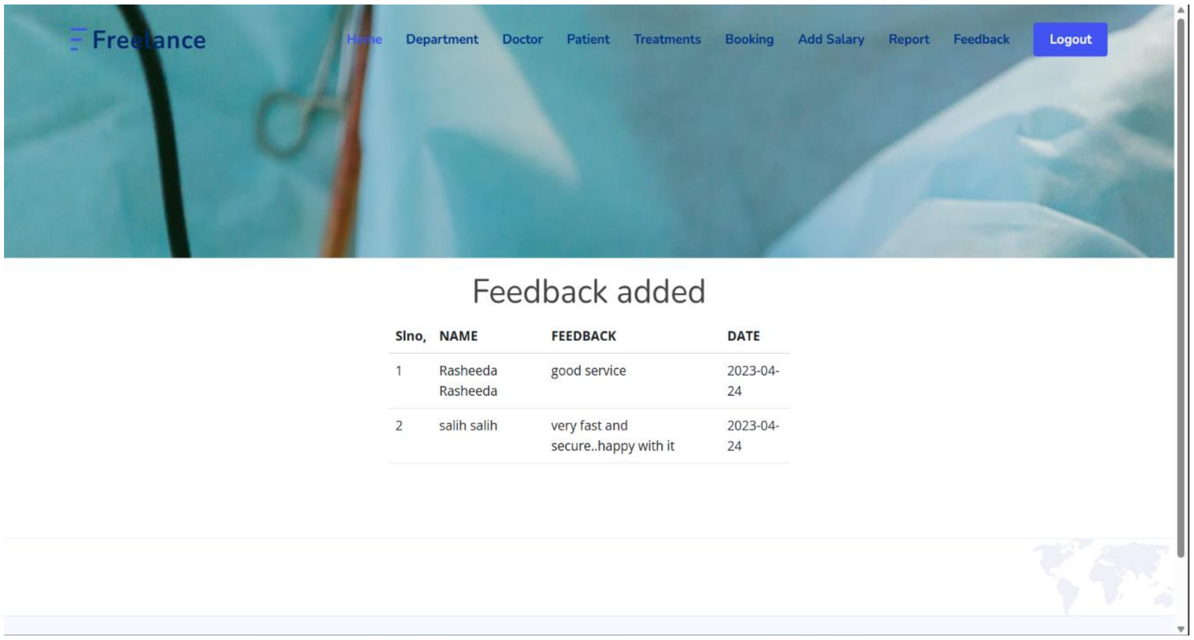


Department

Department	
Description	
	Add

11.2. SAMPLE OUTPUTS AND REPORTS

- **view feedback**

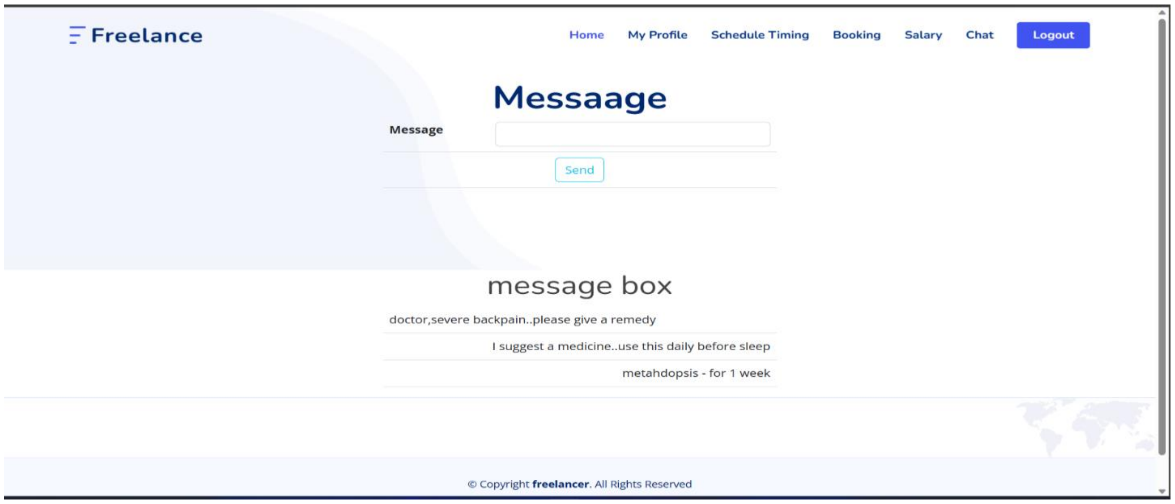


Freelance
Home Department Doctor Patient Treatments Booking Add Salary Report Feedback Logout

Feedback added

Slno,	NAME	FEEDBACK	DATE
1	Rasheeda Rasheeda	good service	2023-04-24
2	salih salih	very fast and secure..happy with it	2023-04-24

- **view message**



Messaage

Message

message box

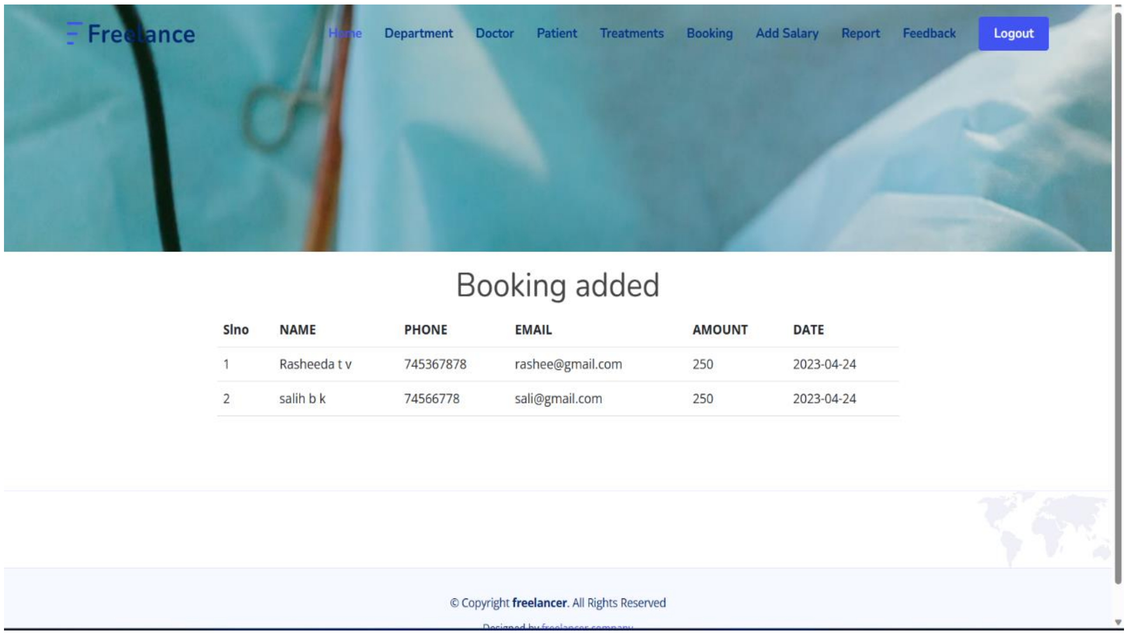
doctor,severe backpain..please give a remedy

I suggest a medicine..use this daily before sleep

metahdopsis - for 1 week

© Copyright freelancer. All Rights Reserved

- **view bookings**

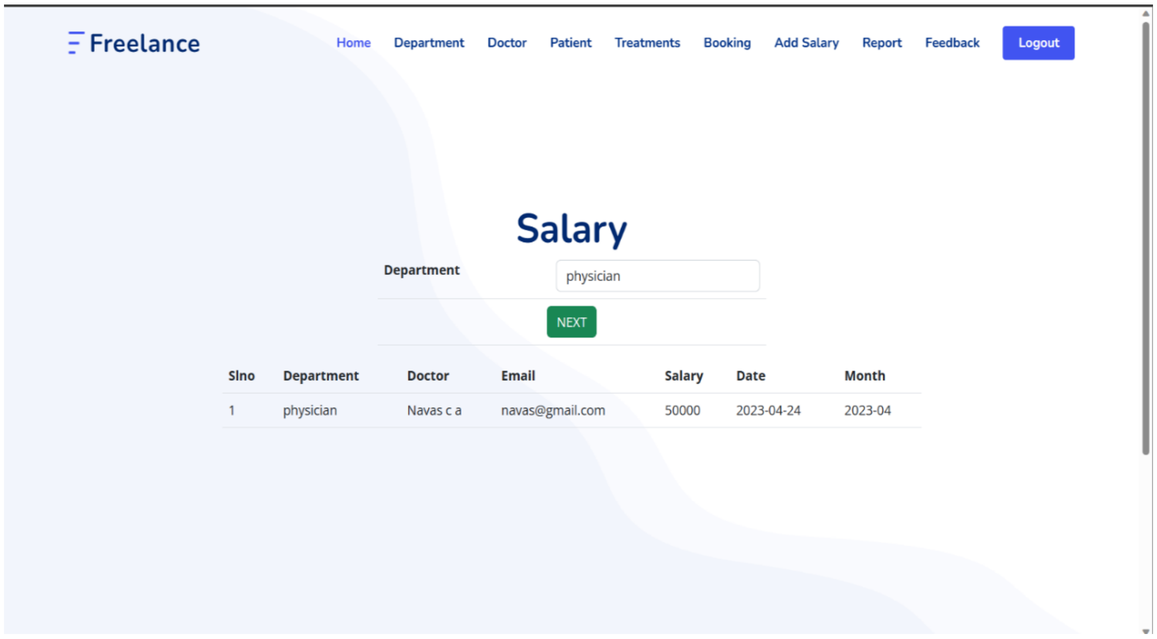

Freelance
Home Department Doctor Patient Treatments Booking Add Salary Report Feedback Logout

Booking added

Slno	NAME	PHONE	EMAIL	AMOUNT	DATE
1	Rasheeda t v	745367878	rashee@gmail.com	250	2023-04-24
2	salih b k	74566778	sali@gmail.com	250	2023-04-24

© Copyright **freelancer**. All Rights Reserved

- **salary report**

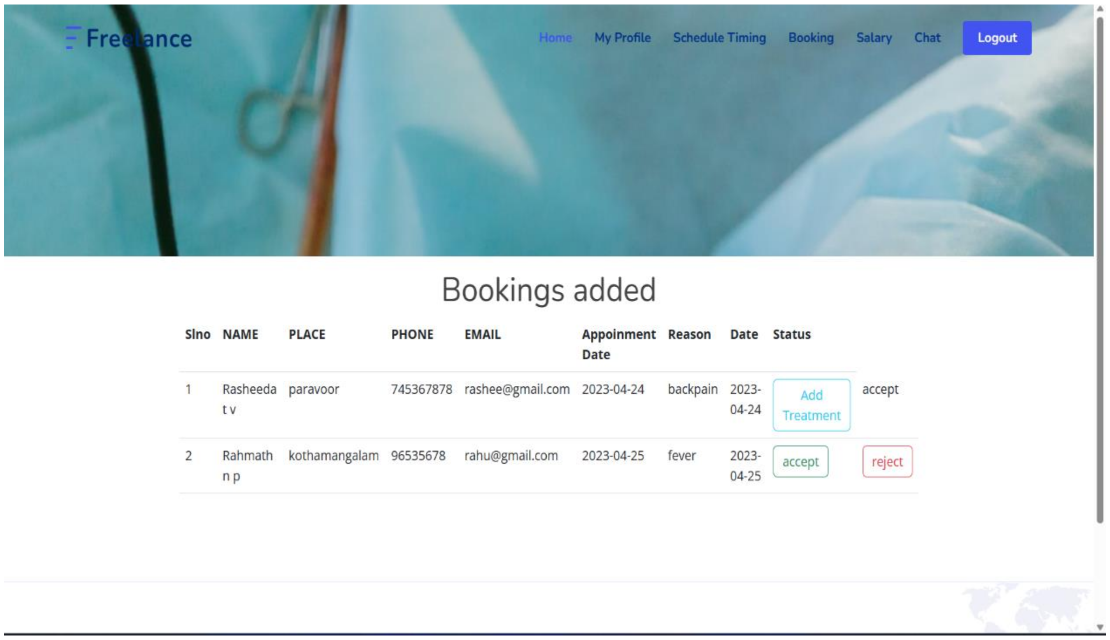

Freelance
Home Department Doctor Patient Treatments Booking Add Salary Report Feedback Logout
NEXT

Salary

Department: physician

Slno	Department	Doctor	Email	Salary	Date	Month
1	physician	Navas c a	navas@gmail.com	50000	2023-04-24	2023-04

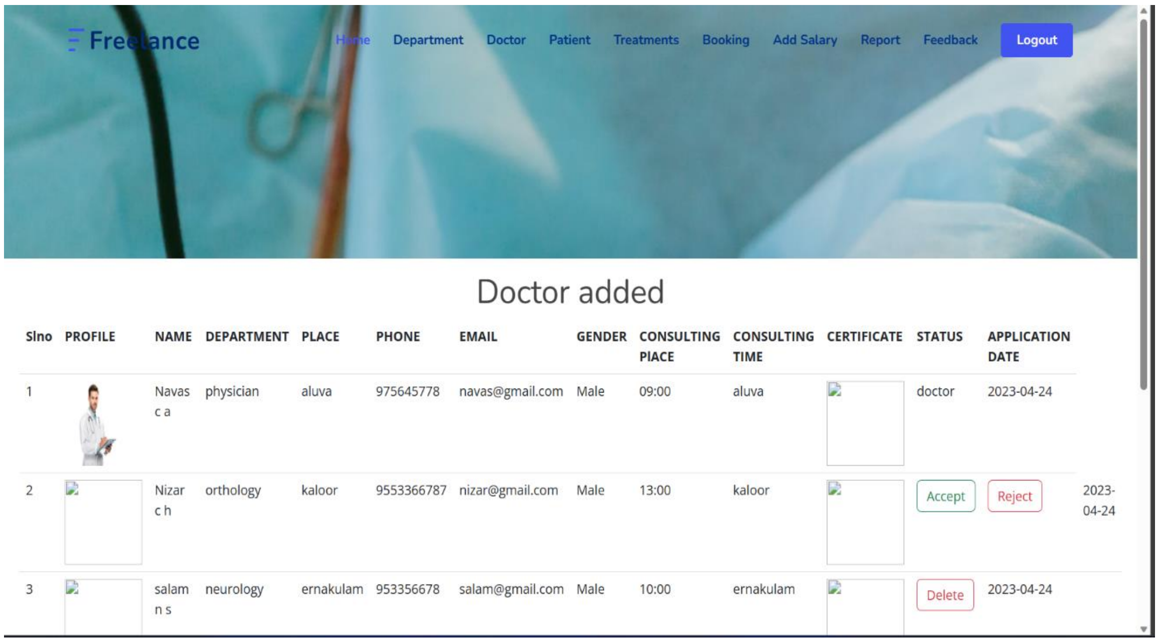
- **manage bookings**



Bookings added

Slno	NAME	PLACE	PHONE	EMAIL	Appoinment Date	Reason	Date	Status	
1	Rasheeda t v	paravoor	745367878	rashee@gmail.com	2023-04-24	backpain	2023-04-24	Add Treatment	accept
2	Rahmath n p	kothamangalam	96535678	rahu@gmail.com	2023-04-25	fever	2023-04-25	accept	reject

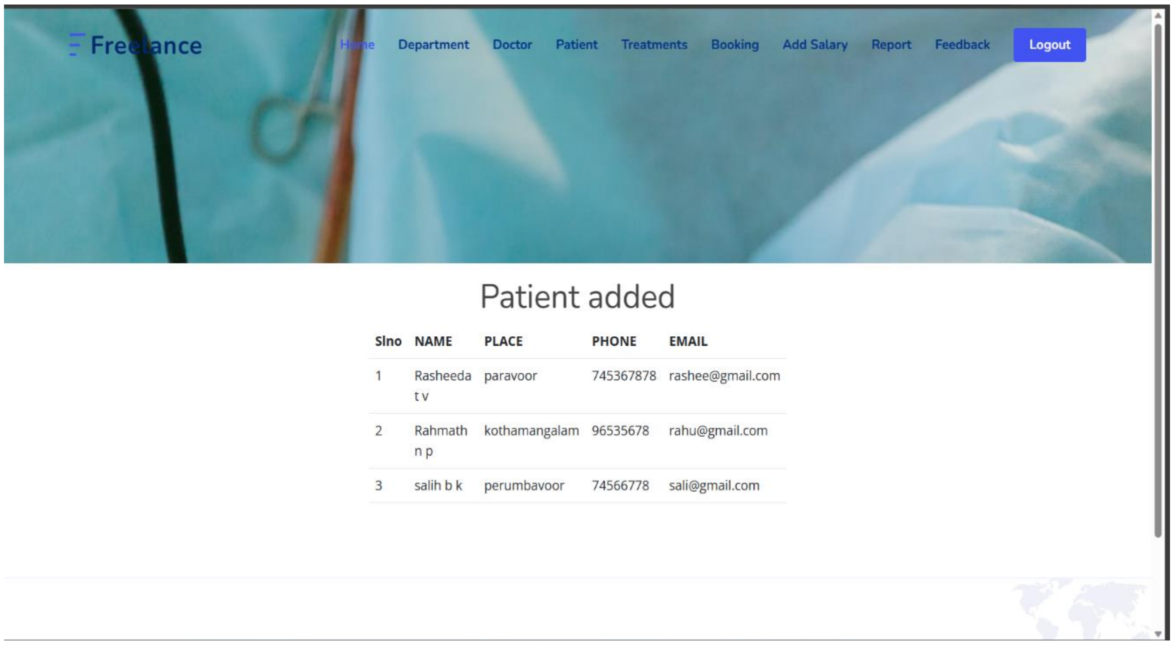
- **view doctors**



Doctor added

Slno	PROFILE	NAME	DEPARTMENT	PLACE	PHONE	EMAIL	GENDER	CONSULTING PlACE	CONSULTING TIME	CERTIFICATE	STATUS	APPLICATION DATE	
1		Navas c a	physician	aluva	975645778	navas@gmail.com	Male	09:00	aluva		doctor	2023-04-24	
2		Nizar c h	orthology	kaloor	9553366787	nizar@gmail.com	Male	13:00	kaloor		Accept	Reject	2023-04-24
3		salam n s	neurology	ernakulam	953356678	salam@gmail.com	Male	10:00	ernakulam		Delete	2023-04-24	

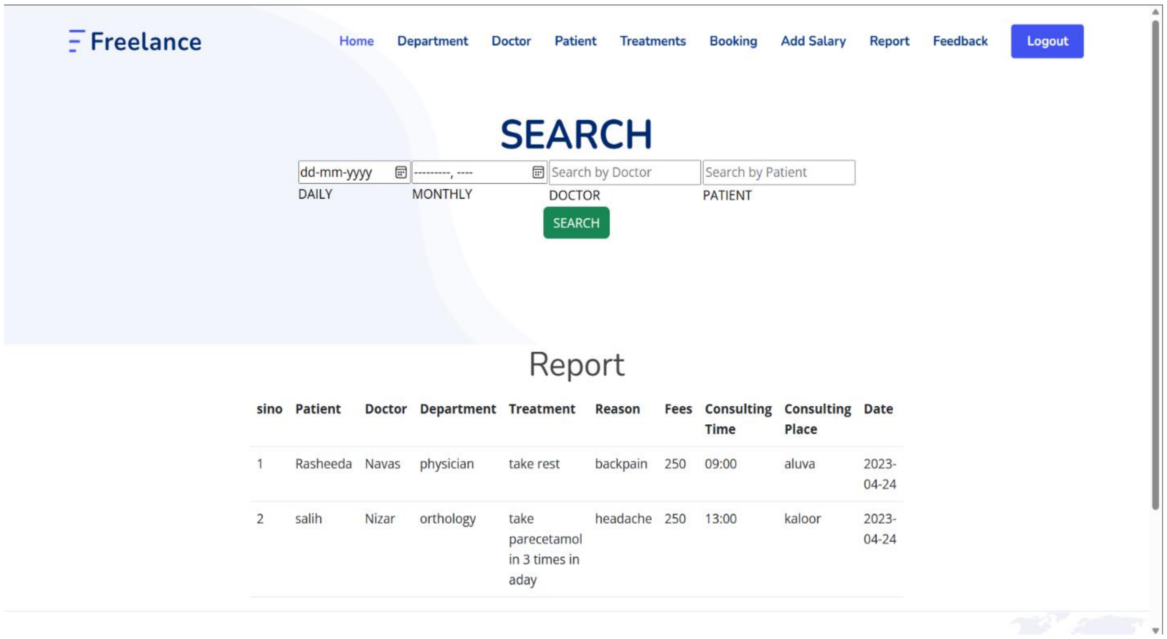
- **view patients**



Patient added

Slno	NAME	PLACE	PHONE	EMAIL
1	Rasheeda t v	paravoor	745367878	rashee@gmail.com
2	Rahmath n p	kothamangalam	96535678	rahu@gmail.com
3	salih b k	perumbavoor	74566778	sali@gmail.com

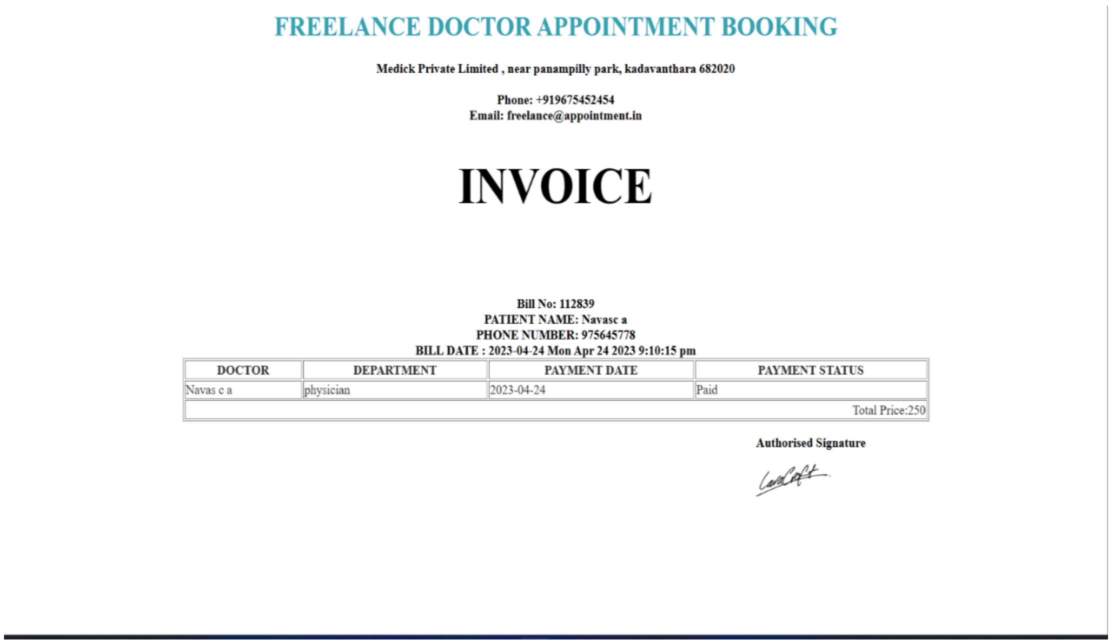
- **report**



Report

sino	Patient	Doctor	Department	Treatment	Reason	Fees	Consulting Time	Consulting Place	Date
1	Rasheeda	Navas	physician	take rest	backpain	250	09:00	aluva	2023-04-24
2	salih	Nizar	orthology	take parecetamol in 3 times in aday	headache	250	13:00	kaloor	2023-04-24

- **bill report**



FREELANCE DOCTOR APPOINTMENT BOOKING

Medick Private Limited , near panampilly park, kadavanthara 682020

Phone: +919675452454
Email: freelance@appointment.in

INVOICE

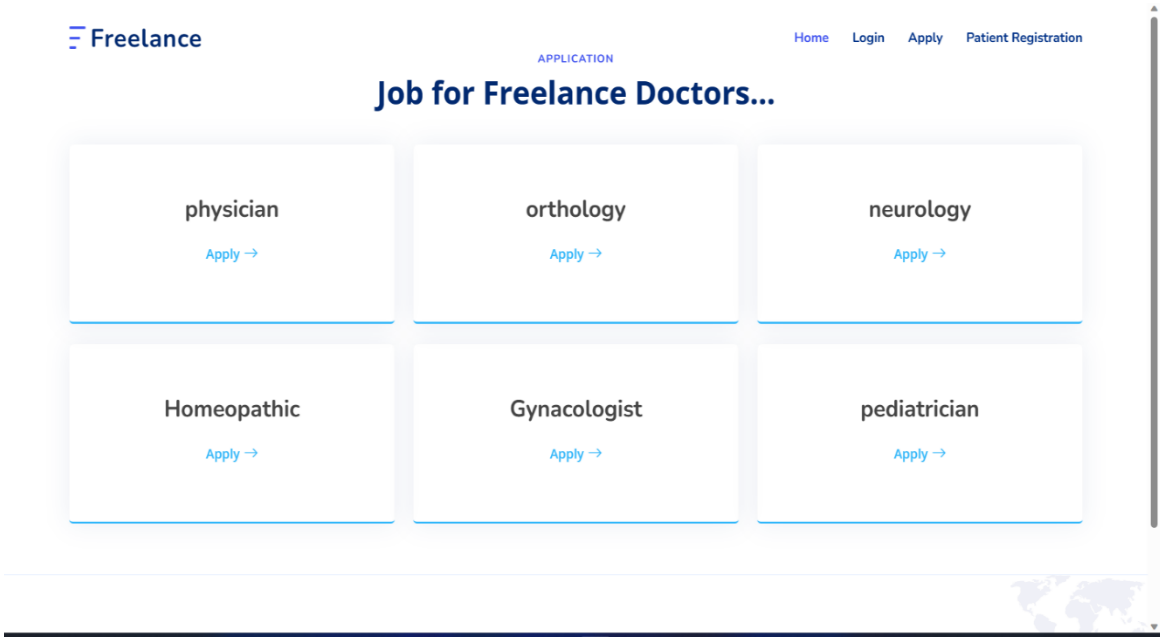
Bill No: 112839
PATIENT NAME: Navasc a
PHONE NUMBER: 975645778
BILL DATE : 2023-04-24 Mon Apr 24 2023 9:10:15 pm

DOCTOR	DEPARTMENT	PAYMENT DATE	PAYMENT STATUS
Navas c a	physician	2023-04-24	Paid
			Total Price:250

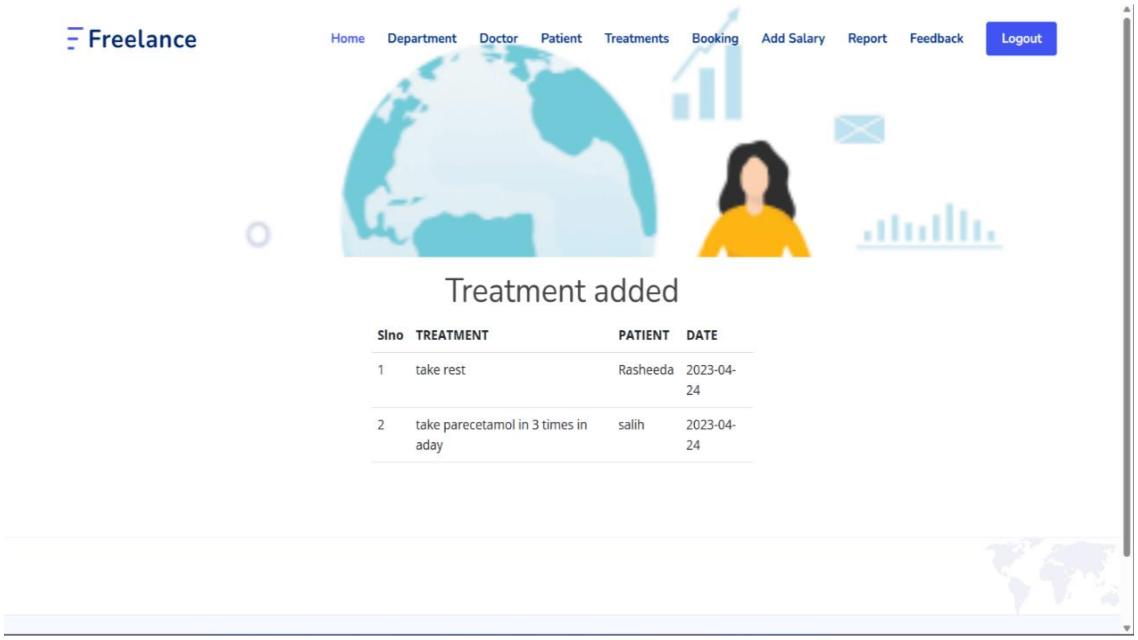
Authorised Signature

[illegible]

- **apply**



- **treatment report**



Treatment added

Slno	TREATMENT	PATIENT	DATE
1	take rest	Rasheeda	2023-04-24
2	take parecetamol in 3 times in aday	salih	2023-04-24

12. BIBLIOGRAPHY

- Pressman, Roger. Software Engineering: A Practitioner's Approach Sixth Edition. McGrawHill Higher Education, 2009.
- Singh,Yogesh and K.K Aggarwal. Software Engineering Third Edition.New Age International Publications, 2007.
- Learning PHP, MySQL, JavaScript, CSS: A Step by step guide to dynamic websites by Robin Nixon.
- Programming in Python 3,A Complete Introduction to the Python Languague, Second Edition by Mark Summerfield
- **Electronic Materials :**
- **Internet**
- Pressman , Roger . Software engineering resources . 1997 . Rs Pressman & Associates , Inc. 21 Dec 2021 . <http://www.rspa.com>
- Stack Overflow - Where Developers Learn , Share , and Build <https://stackoverflow.com>

  • W3Schools Online Web Tutorials< https://www.w3schools.com>