

◀◀◀ A Project Report On ▶▶▶

“Online Pathology Booking System”

Submitted To

Panskura Banamali College (Autonomous)

For the partial

Fulfilment of the Requirement for the Award,
Degree of Bachelor of Computer Science (BSC)



Submitted By

Pratik Mukherjee

ID: 2019UG012226

Roll: 12161022 No. 251032

Reg. No: 2019PBC00198 of year: 2019-2020

Guided By

SK MD Habibullah

Department of Computer Science

Panskura Banamali College (Autonomous)

Panskura, P.O.: Panskura R.S., Dist.: Purba Medinipur, PIN: 721102



Panskura Banamali College

(AUTONOMOUS College 2018- 2019 to 2027-2028)

Largest Rural Based, NAAC Re-accredited 'A' Grade (2016-2021)

&

DIST-FIST (Govt. of India) Sponsored College

Website: www.panskurabanamalicollege.org Email: principal.pbc@gmail.com

Department of Computer Science

Project Report On

“Online Pathology Booking System”

Self-Declaration Certificate

This is to certify that the Project Proposal entitled “**Online Pathology Booking System**” submitted by **Pratik Mukherjee** is an Authentic work carried out for the partial fulfilment of the requirements for the award of the certificate of project by Panskura Banamali College, Purba Medinipur under the guidance **SK MD Habibullah**. The matter embodied in this project work has not been submitted earlier for award of any certificate to the best of our knowledge and belief.

Date:

.....
Pratik Mukherjee



Panskura Banamali College

(AUTONOMOUS College 2018- 2019 to 2027-2028)

Largest Rural Based, NAAC Re-accredited 'A' Grade (2016-2021)

&

DIST-FIST (Govt. of India) Sponsored College

Website: www.panskurabanamalicollege.org Email: principal.pbc@gmail.com

Department of Computer Science

Project Report On

“Online Pathology Booking System”

Certificate by Guide

This is to certify that this project entitled “Online Pathology Booking System” submitted by **Pratik Mukherjee** as a partial fulfilment of the certificate of project by Panskura Banamali College done by him is an authentic work carried out under my guidance and best of our knowledge and belief.

Date:

.....
Sk. MD. Habibullah

Project Guide

Panskura Banamali College



Panskura Banamali College

(AUTONOMOUS College 2018- 2019 to 2027-2028)

Largest Rural Based, NAAC Re-accredited 'A' Grade (2016-2021)

&

DIST-FIST (Govt. of India) Sponsored College

Website: www.panskurabanamalicollege.org Email: principal.pbc@gmail.com

Department of Computer Science

Project Report On

“Online Pathology Booking System”

Certificate of Approval

This is certifying that this proposal Project, entitled “Online Pathology Booking System” is record of bona-fide work, carried out by under my supervision and guidance.

In my opinion, the report in its presents form is in partial fulfilment of all the requirements, as specified by the Panskura Banamali College. In, fact it has attained the standard, necessary for submission. To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Date:

.....
Sk. MD. Habibullah

Panskura Banamali College

H.O.D. of Computer Science

Abstract

The Online Pathology booking System is designed for Any Pathology Lab to replace their existing manual, paper-based system. The new system is to control the following information; patient information, room availability, staff and operating and patient invoices. These services are to be provided in an efficient, cost effective manner, with the goal of reducing the time and resources currently required for such tasks.

A significant part of the operation of any Pathology Lab involves the acquisition, management and timely retrieval of great volumes of information. This information typically involves; patient personal information and medical history, staff information, staff scheduling and various facilities waiting lists. All of this information must be managed in an efficient and cost wise fashion so that an institution's resources may be effectively utilized PLMS will automate the management of the Pathology Lab making it more efficient and error free. It aims at standardizing data, consolidating data ensuring data integrity and reducing inconsistencies.

Acknowledgement

Any work small or big is always a tough ask for a sole individual to get it correct working as per the requirements. It was also not any different. I would first thank our project centre **Panskura Banamali college** where I was trained with the updated technologies of MySQL architecture and the all the techniques that is required for the fulfilment of this project.

I would like to thank my main project instructor **Mosh Hamedani** who taught me with his professional Django skills from the very basics to the advanced courses throughout this project by his short & concise video lectures.

I would also like to thank my project guide **SK MD Habibullah** who had rendered his valuable guidance throughout this project.

I like to thank my **parents** to help me doing this uphill task and supporting me in times of difficulties and need.

I also thank my **College Management**, Faculties & all other staffs for giving us the opportunity to learn and deploy our skills in the field of **online project**.

Thanking You

With best Regards

Pratik Mukherjee

Student Declaration

I hereby declare that this project report entitled **“ONLINE PATHOLOGY BOOKING SYSTEM”** has been prepared by me under the guidance Of **SK MD Habibullah**, in our **Computer Science** department of **Panskura Banamali College**, In partial fulfilment of the requirement of the award of degree B.sc in computer science(Hons.) by Panskura Banamali College Under **Vidyasagar University**.

I would like to declare that this is the result of my own effort and that it has not been satisfied to any College or to other boards for the award of degree.

Prepared By

Pratik Mukherjee

My Responsibilities

To give **Online Pathology Booking System** its form that it represents, it was my honest effort for nearly three months. It is because of my entire effort that **Online Pathology Booking System** has its presentable form. Starting from the scratch, the entire project has been divided into various modules and in every module the coding & testing is done manually by me.

Pratik Mukherjee: - I was responsible for the whole design of the project. The system can be opened for any anonymous or registered user. At first the Department section, Doctor's list, Test's list, Test's Collection, Query services and Image gallery is open for any people. They can view those and if decides to place an order, check-up appointment or wants to give a review then they must have to login to there account or register as a new user. Anyone can also subscribe to our newsletter with their Name & Email-ID.

SK MD Habibullah: - He was responsible for the planning of this project. The system first checks for valid ID and Password of the Administrator. The Administrator can Insert, Update or Delete customer Information and tests details.

A user who send request for test order or book for doctor check-up, Administrator can accept the request. He has the major responsibility for maintaining the store.

Index

1. Introduction	11
2. Review of Existing System	12
3. Problems with Existing System	13
4. System Analysis	14
5. About the Project	15
6. Hardware & Software Requirement	16
7. Tools & Techniques	17
8. Problem Statement	18
9. Feasibility Study	19
10. Requirement Analysis	20
11. Software Requirement Specifications	21
12. Project Planning	22
13. Project Scheduling	23
14. Software Engineering Tools	25
15. Iterative Waterfall Model	26
16. Project Description	28
17. System Design	29
a. Use Case Diagram	32
b. Data Flow Diagram	35
i. Context Level DFD	35
ii. Level 1 DFD for Admin	36
iii. Level 2 for Admin	37
iv. Level 1 for Users	38
18. Database Schema	38
19. E-R Diagram	46
20. System Flow Chart	47
21. System Coding	50
a. Admin Page	50

b. Database Models	54
c. Root URLs	58
d. Store App URLs	59
e. Core App URLs	59
f. Client Views	60
g. Login Restriction in some Endpoints	64
22. Output Screen Shots	65
a. Home Page	66
b. Registration Form	66
c. Login Page	67
d. Orders Page	67
e. Doctors Page	68
f. Tests Page	69
g. Admin Home	70
h. Orders List Page	71
i. Check-ups Page	73
23. Software Testing	73
24. System Maintenance	76
25. Advantages of Proposed System	77
26. Applications	77
27. Result Analysis	78
28. Challenges	78
29. Future Enhancements	79
30. Conclusion	80
31. Bibliography	81

Introduction

Documenting the assembly, maintenance use and troubleshooting of a system as complex as a voting system is a difficult task. A single system might combine proprietary hardware, custom operating systems and software, commodity personal computers, and pen and paper. Describing these highly varied components is one function of voting system documentation. But it is not only voting system technology that is heterogeneous; so are voting systems users, and the environments in which the systems are used. Election officials, poll workers, and voters often need guidance from documents in order to administer or use a voting system. In addition, voting system certification—the process by which a system is approved for use in state— demands evidence that a voting system complies with one or more sets of standards or guidelines.

Documentation provides the details that certification bodies need to evaluate the System. Taken as a whole, voting system documentation must explain the system in several ways to meet the needs of these multiple audiences. As the modern organizations are automated and computers are working as per the instructions, it becomes essential for the coordination of human beings, commodity and computers in a modern organization. Many big cities where the life is busy needs the transaction of the goods within few minutes of time. So, this online information recorded by the distributor helps him to complete this task within the time.

The administrators and all the others can communicate with the system through this project, thus facilitating effective implementation and monitoring of various activities of the voting Software.

Review of Existing System

Functions ➔

- Registering Forms.
- All info's regarding registration First Name, Last Name, email, phone, etc details are filled by the user and then are stored in the records.
- Separate databases were kept at separate sites.
- Usage of database for each & every records was very high.
- Maintenance of record takes very much time. Because everything was manual so maintenance was difficult.

Study Material ➔

- The study material is all of the python programming books available in the website.
- All details were not available.

Chat & Query ➔

- To maintain an admin to chat and query section about users.
- The details are filled in database in the user records.

Help ➔

- This section provides all helps related to this website.

Problems with Existing System

- Communication involves a lot of database work. The system was not a computer-based application as a result communication among the five functions involved a lot of database work, that is, in case the user forgot the user ID while filling the complaint then there is no response suddenly in the website about on admin in the help desk.
- Being completely manual there is always a possibility of manual mistakes in proceeding with the system.
- Large amount of data was stored but it will be hanged system.
- There was no web server available in website in domain name of the website, it is available to show only as a local host.

System Analysis

Existing System ➔

Pathology Labs currently use a manual system for the management and maintenance of critical information. The current system requires numerous paper forms, with data stores spread throughout the pathology lab management infrastructure. Often information (on forms) is incomplete, or does not follow management standards. Forms are often lost in transit between departments requiring a comprehensive auditing process to ensure that no vital information is lost. Multiple copies of the same information exist in the Pathology Lab and may lead to inconsistencies in data in various data stores.

Proposed System ➔

The Pathology Lab Management System (PHMS) is designed for Any Pathology Lab to replace their existing manual, paper-based system. The new system is to control the following information; patient information, staff and patient invoices. These services are to be provided in an efficient, cost effective manner, with the goal of reducing the time and resources currently required for such tasks.

About the Project

The project **Online Pathology Booking System** is a web-based application based on the basis of MySQL architecture & Django Framework. Here we can send a request from the client side to the server side and the server satisfies the request from the client side.

Being an application based on the client-server architecture this project has the front end developed in HTML, CSS with bit of JavaScript & Python and also has the backend developed in **Rest Framework** on top of **Django**.

It is divided into five parts mainly →

Searching: - Here is two types of searching one is tests and available doctors searching with their qualification, department and time schedule. Other is searching on any field including users for administrators.

Registering: - Here a user can register themselves and create a user & password by give required credentials. But for the admin section there is one **Superuser** who can manage the whole site, he has to hire an admin and create an admin user with restricted or unrestricted accessibilities.

Login: - There are two types of login that it provides with namely **Admin & User**. The user can login from there registered **username** or **email** and **password**. The Administrator or Manager login enters to the inner part of the project. Once a person logs in with administrator credentials, he/she can update, delete and insert any type of database activity with any module. This login has been kept under tight scanner and high surveillance with history saving.

Requesting: - A user can send test order, book for a check-up, view reports, update their own personal info's and also can make payment. But all these activities are restricted to registered users only. Here at first user have to login and then only they can access these endpoints.

Hardware and Software Requirement

Hardware Requirements ➔

The software developed will run on any Client /Server environment.

The Minimum Specification of **Server** are:

In my Pathology System the hardware I have used:

- More than 320 GB hard disk
- 2 GB or more RAM
- Intel i3 (2.1 GHz) processor
- Colour LCD (17inch)
- 52x CD-ROM Driver
- 12/24 DAT Driver

The Minimum specification of **Clients** are:

- 80 GB Hard Disk
- 512MB RAM
- Intel Pentium-4 (2.1 GHz) processor
- Colour monitor (15 inch)
- 52x CD-ROM Driver

Software Requirements ➔

In my Pathology Booking system the software I have used for designing tools:

- **Database:** MySQL
- **Platform:** Django
- **Language:** Python
- **Operating System:** Windows 10 Home & Kali Linux.

Tools and Techniques

❖ Database

As the word implies, we know this is a logical space for storing your valuable data with very minimum-security risks. Generally, a database server is consisted of several databases; we reference them by their name for appropriate operations in our system. In our system we have given the database name as 'lifecell'.

❖ MySQL Version 8.0

It is a powerful open-source relational database management system. As with other relational databases MySQL stores data in tables made up of rows and columns. We can define, manipulate, control and query data using structured query language, more commonly known as SQL.

❖ PyCharm 2021.2.3 Professional Edition

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for python developers, tightly integrated to create a convenient environment for productive python. Here in professional version **PyCharm** comes with Python core, web and data Science development including code assistance, refactoring, visual debugging, deployment, supports popular web frameworks like Django, Flask, Pyramid and it also has a wide range of database support.

❖ DataGrip 2021.2.3 Professional Edition

DataGrip is a database management environment for developers. It is designed to query, create, and manage databases. Databases can work locally, on a server, or in the cloud. Supports MySQL, PostgreSQL, Microsoft SQL Server, Oracle, and more. It just needs a JDBC drive driver to connect the database and we can start working.

Problem Statement

As per the norms of system development lifecycle is concerned. The very first step in designing a system is defining the problem. A system is a collection of entities, which work together in synergy to achieve a predefined goal and objectives.

The definition phase of the system answers the question, what does the need and wants from the new system? The definition phase can be called the requirement analysis or logical design phase.

We can never skip the definition phase. New system is always be evaluated, first and foremost, on whether or not it fulfilled the business requirement regardless of how impressive the technological solution be.

The initial activity of definition phase is to outline the business requirements. The foundation of this activity was established in the study phase when the system improvement objectives were identified. This activity translates those objectives into inputs, outputs, and process that will be specified in greater deal during subsequent activities.

The problem produced a need to change the existing system. Some of these shortcomings are being discussed below:

- Manual work is tedious & slow.
- Manual work is more prone to errors.
- It is difficult to keep details of heavy records.
- Any task may require lot of manual intervention, which results in big loss of time.
- Paper records also require huge space to store them safety.
- Main problem related to security, i.e. records are not secured.
- The problem of personnel is big i.e. extra Staff always was a set back to the organization.

Feasibility Study

Feasibility study is conducted once the problem is clearly understood. Feasibility study is a high-level capsule version of the entire system analysis and design process. The objective is to determine quickly at a minimum expense how to solve a problem. The purpose of feasibility is not to solve the problem but to determine if the problem is worth solving. The system has been tested for feasibility study in the following points:

❖ Technical Feasibility:

The project entitles “Online Pathology Management System” technically feasible because of below mentioned feature. The project was developed in Python with Graphical User Interface.

It provides the high level of reliability, availability and compatibility. All these makes Python an appropriate language for this project. Thus, in the existing software Python is a powerful language with wide range of versatility.

❖ Economical Feasibility:

The computerized system will help in automate the selection leading the profits and details of the organisation. With this software, the machine and manpower utilisation are expected to grow up by 80-90% approximately. The costs incurred of not creating the system are set to be great, because precious time can be wanted by manually.

❖ Operational Feasibility:

In this project the management will know the details of each project where he may be presented and the data will be maintained as decentralised and if any inquires for that particular contract can be known as per the requirements and necessities.

Requirement Analysis

The Requirement Management deals with analysing, developing, maintaining, documenting, and verifying patient's requirements. The patient's requirements need to be tracked throughout the project life cycle to ensure that the final product meets all the requirements. Requirement Outputs, such as Software Requirement Specifications or any other document may be produced.

The analyst (or a team) determines the requirements of the patient or end user. A variety of techniques may be used to study the requirements such as, interviews, evaluation of similar products or projects, discussions etc.

The requirements are analysed to ensure that they are feasible and appropriate to implement in software, clearly stated, consistent with each other, testable, and complete. To analyse the requirements various techniques may be used e.g. Data Flow Diagrams, functional decomposition, object-oriented decomposition, simulations, modelling, prototyping, etc. Issues affecting the requirement analysis are identified and resolved. Each of the outputs will be peer-reviewed as per review Procedure and approved at one or more points as decided by the Project Manager at the time of Project Planning.

Software Requirement Specifications

The Software Requirements Specification is produced at the culmination of the analysis task. SRS is a document that completely describes what the proposed software should do without describing how the software will do it. The basic limitation for this is that the user need keeps changing as environment in which the system was to function changes with time. This leads to a request for requirement changes even after the requirement phase is done and the SRS is produced.

The functions and performance allocated to software as part of the system engineering are referred by:

- Establishing a complete information description of the system.
- A detailed functional description.
- An indication of performance requirements and design constraints.
- A representation of system behaviour.
- Appropriate validation criteria & other info's related to requirements.

Project Planning

Project life cycle has three stages: -

Project Initiation:

Development team prepares the project plans and finalizes the outcome of each phase. In this stage team also prepares the comprehensive list of tasks involved in each phase and the project assigns responsibilities to the members, depending on their skills.

Project Execution:

In this stage the team develops the project. This stage is consisting of following phases: -

- Requirement analysis.
- Low level design.
- High level design.
- Construction.
- Testing
- Acceptance.

Project Completion:

In this stage, the team has to update the site in a regular basis. Each new test item has to be added by the administrator as according to the needs and demands. This stage is very important for freshness of this site.

When any updating or upgradation is required for the website, the developers or maintenance team make the website up to date.

There are lots of requirements after the project completion. As this website is dynamic website in which lots of changes are required.

Project Scheduling

The Prototype Model:

For better development of a project it is very essential to select a relevant model. **Prototype model** can be one of the best options for such a purpose.

Because the system is complicated and large and there is no existing system. Computerised prototyping is an attractive idea. In this situation letting the client test the prototype provides the variable inputs, which help in determining the requirements of the system. It is also an effective method of demonstrating the feasibility of a certain approach.

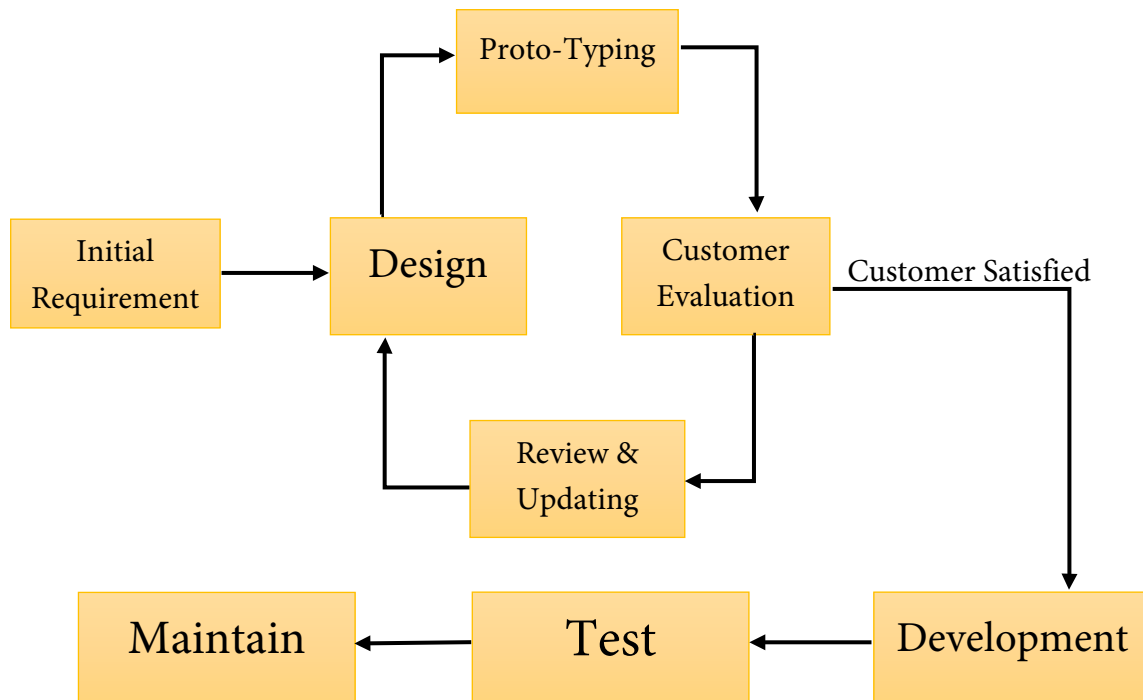


Figure-1: - **Prototype Model**

Reasons Behind Using this Prototype Model:

- It helps to reduce the cost and time.
- It improves communication with the best client as after every prototype there is an interactive session with client.
- It provides an early detection of errors.
- It allows the developers to have a grater control over the problem.
- Because the system is complicated and large and there is no existing system. Computerised prototyping is such an attractive idea. In this situation letting the client test the prototype provides of the system. It is also an effective method of demonstrating the feasibility of a certain approach.

Software Engineering Tools

To develop the application “**Online Pathology Booking System**” which will enable user to use the website of the agency to perform various tasks online networking branches, we do follow the linear sequential model, which suggest a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support. The waterfall model encompasses the following phrases: -

The classic life cycle or the **Iterative Waterfall Model**, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support. System engineering and analysis encompasses requirements gathering at the system level with a small amount of top-level design and analysis. Information gathering encompasses requirements gathering at the strategic business level and at the business area level.

Iterative Waterfall Model

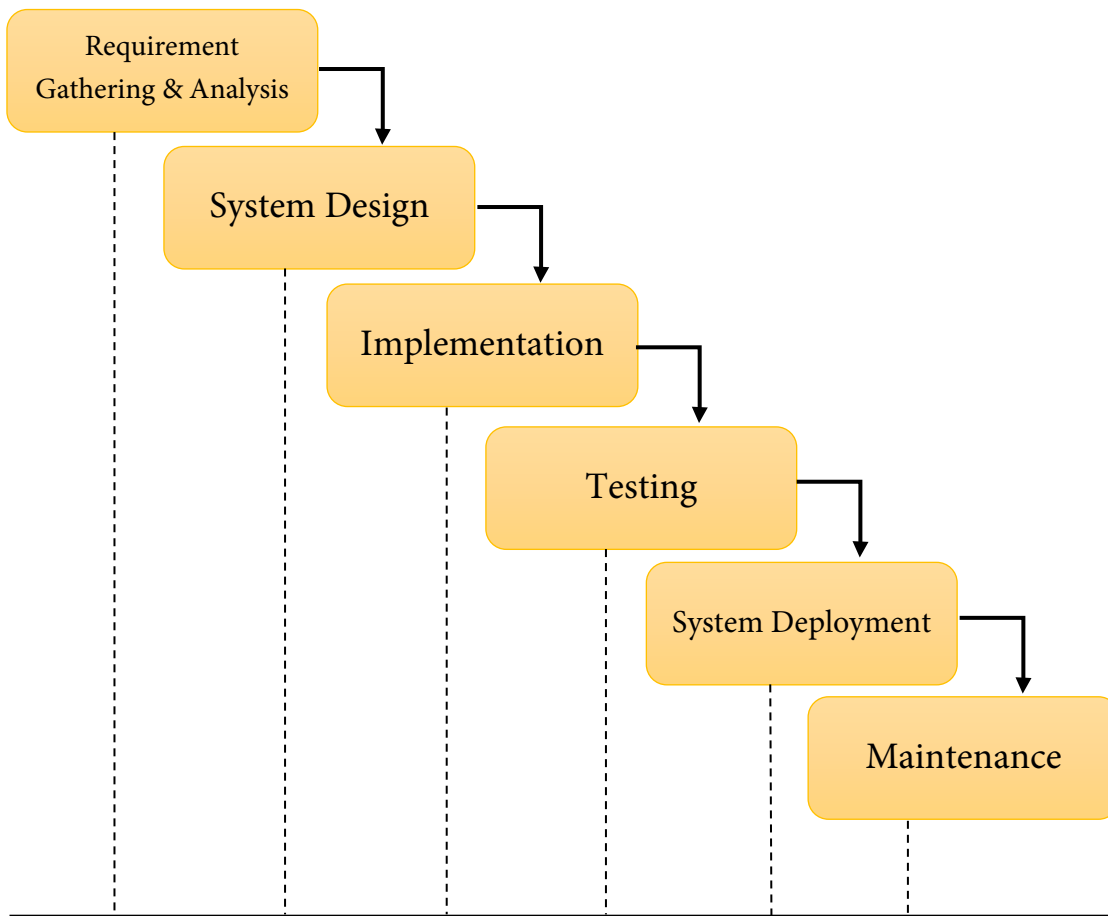


Figure-2: Iterative Waterfall Model.

We had pointed out in the previous section that in a practical software development project, the classical waterfall model is hard to use. We had branded the classical waterfall model as an idealistic model. In this context, the iterative waterfall model can be thought of as incorporating the necessary change to the classical waterfall model to make it usable in practical software development projects.

The feedback path introduced by the iterative waterfall model are shown in above figure. The feedback paths allow the correcting errors committed by the programmers during some phase, as when these are detected in a later phase. For example, if during the test phase a design error is identified, then the feedback path allows the design to be reworked and the changes to be reflected on the design documents and all other subsequent documents.

Feasibility Study:

The main aim of the feasibility study activity is to determine whether it would be financially and technically feasible to develop the project. All the technical feasibilities are used in our project and the economical feasibility is very low here in Online Pathology Booking System.

Requirement Analysis and Specification:

The aim of this phase is to understand the exact requirement of the user and to document them properly. This project Online Pathology Booking System is consisted of two distinct activities, namely *requirement gathering & analysis*, and *requirement specification*.

Design:

The goal of our Online Pathology Booking System's design phase is to transform the requirements specified from a user into a structure for implementation in the Python programming language, used in this project.

Coding and Unit Testing:

Here the main job is to translate the software design into source code. When we start the unit testing of our project, the functional modules have been tested separately so that, overall error could be reduced.

Integration and System Testing:

When all the modules are coded and tested then we start the integration of our project. We make a paper plan to integrate the different modules to get finalised product.

Maintenance:

This is very important in development of software. Here we can correct those errors which are not found in the development phase and also can improve the implementation of software and enhance the functions of our project.

Project Description

- ✓ It is an efficient user-friendly GUI web-design for professional software.
- ✓ It is an example of web application using **PyCharm 2021.2.3 Professional** with **MySQL 8.0**
- ✓ There are two types of users – Administrator & User.
- ✓ The administrator has full access over the front-end and back-end.
- ✓ The user is provided only with an abstract view of the selected interfaces.
- ✓ The home page can view by both users, it contains various links to various interfaces.
- ✓ Here the user can give a review.
- ✓ Can query to admin about their question.
- ✓ Can give a complain for a test to rectify the problem.
- ✓ Random user can also subscribe to our newsletter to get notified about new tests or some new changes in site.
- ✓ The administrator can login only through username or password but the user can login using username or email and corresponding password.
- ✓ The administrator has the authority to insert, update, or delete the details of particular hardware components, if required which gets reflected to the database.
- ✓ The administrator can also change passwords which reflects in database.

System Design

Design is the first step in development phase for any techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization.

Once the software requirements have been analysed and specified the software design involves three technical activities – design, coding, implementation and testing that are required to build and verify the software.

The design activities are of main importance in this phase, because in this activity, decisions ultimately affecting the success of the software implementation and its ease of maintenance are made. These decisions have the final bearing upon reliability and maintainability of the system. Design is the only way to accurately translate the customer's requirements into finished software or a system.

Design is the place where quality is fostered in development. Software design is a process through which requirements are translated into a representation of software. Software design is conducted in two steps. Preliminary design is concerned with the transformation of requirements into data.

❖ UML Diagrams:

UML stands for Unified Modelling Language. UML is a language for specifying, visualising and documenting the system. This is the step while developing any product after analysis. The goal from this is to produce a model of the entities involved in the project which later need to build. The representation of the entities that are to be used in the product being developed need to be designed.

There are various kinds methods in software design.

Use Case Diagram:

Use case diagrams model behaviours within a system and help the developers understand what the user requires. The stick man represents what's called an actor.

Use case diagrams can be useful for getting an overall view of the system and clarifying who can do and more importantly what they can't do. Use case diagrams consist of use cases and actors.

- The purpose is to show the interactions between the use case and actor.
- To represent the system requirements from user's perspective.
- An actor could be the end-user of the system or an external system.

What is use-case diagram?

A use-case diagram is a dynamic or behaviour diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services and functions that the system needs to perform. In this context, a system is something being developed or operated, such as a website. The actors are people or entities operating under defined rules within the system.

Why make use-case diagrams?

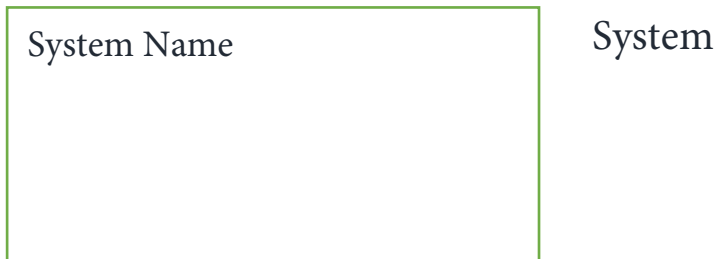
Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities.

They also help in identifying internal or external factors that may influence the system and should be taken into consideration.

They provide a good high-level analysis from outside the system. Use case diagrams specify how the system interacts with actors without worrying about the details of how that functionality is implemented.

Basic diagram symbols & notations?**System:**

Draw your system's boundaries using a rectangle that contains use-cases. Place actors outside the system's boundaries.

**Use Case:**

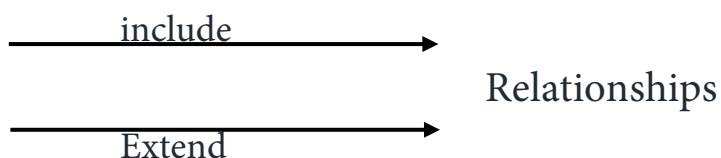
Draw use cases using ovals with verbs that the systems functions.

**Actors:**

Actors are the users of a system. When one system is the actor of another system label the actor system with the actor stereotype.

**Relationships:**

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labelled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.



Use Case Diagram

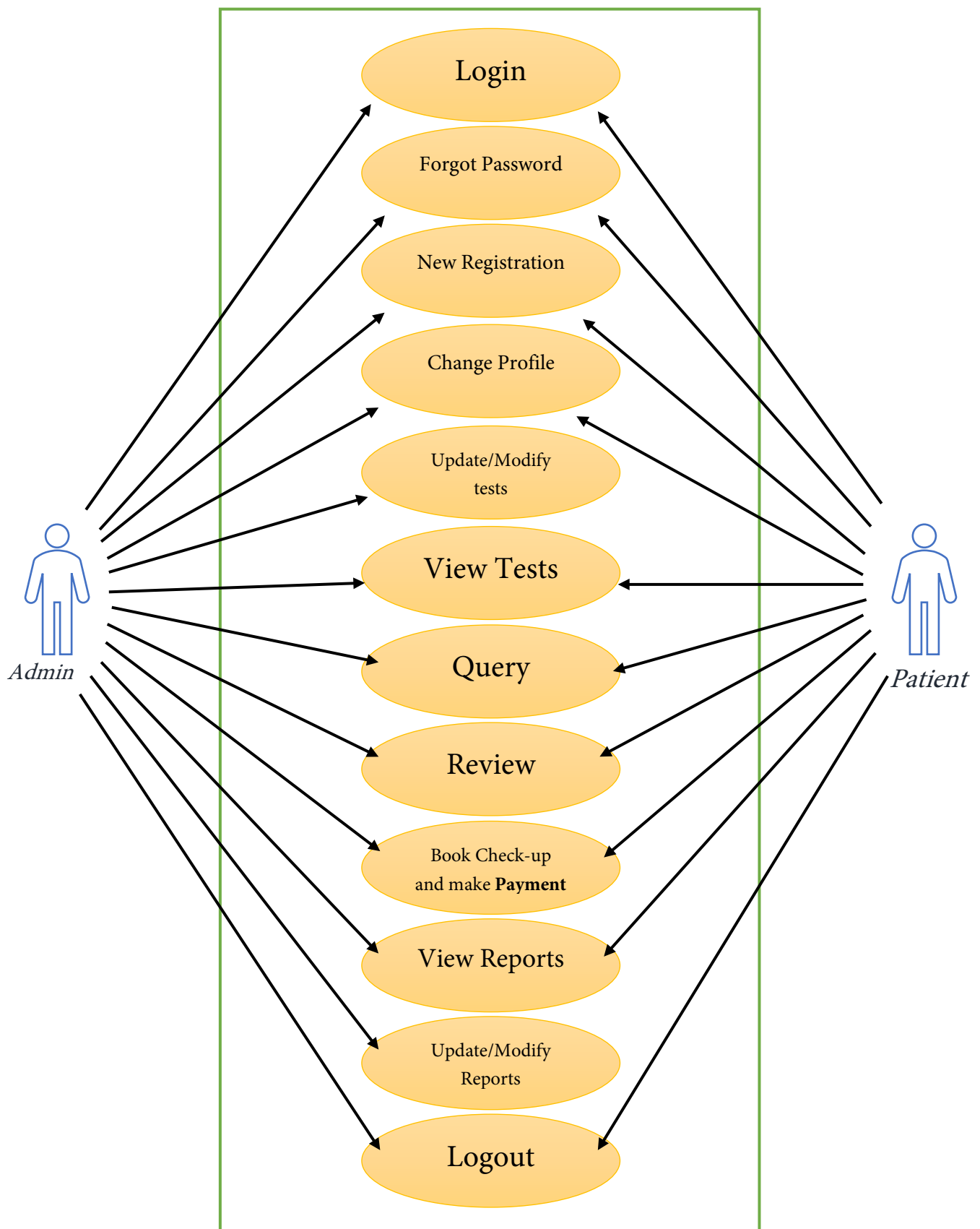


Figure-3: Use Case Diagram for **Online Pathology Booking System**

Data Flow Diagram (DFD):

A DFD is a graphical tool used to describe and analyses the movement of the data through a system depicting the flow of data, store of data, source or destination of data and the processes that respond to changes in data. When the processes is described logically and independently of the physically components associated with the system. In contrast, physically data flow diagrams show the actual implementations and the movement of data between people, department and workstations.

The DFD is one of the most important tool sets by the system analyst to model system components namely:

- System Process.
- Any external entities that interact with the system (source or destination of data).
- Data store.
- The information flows in the system.

Rules for DFD:

- Fix the scope of the system by means of context diagrams.
- Organize the DFD so that the main sequence stays in action.
- Reads from left to right and top to bottom.
- Identify all inputs and outputs.
- Identify and label each process internal to the system with rounded circles.
- A process is required for all the data transformation and transfers.
- Don't indicate hardware and ignore control information.
- Names of processes should accurately convey the internal workings.
- External sources and destination should be indicated with squares.
- Number each occurrence of repeated external entities.

Symbols used in DFD:

Processes:

It is a piece of work performed on data i.e. it shows what a system does.



Data Store:

It is a place where data is held between processes, known as a file.



External Entities:

Represents the people, programs, organizations or other entities. Interact with the system by supplying input or using output.



Data Flow:

Shows the direction of the data in Data Flow Diagram.



Data Flow Diagram

Context Level DFD

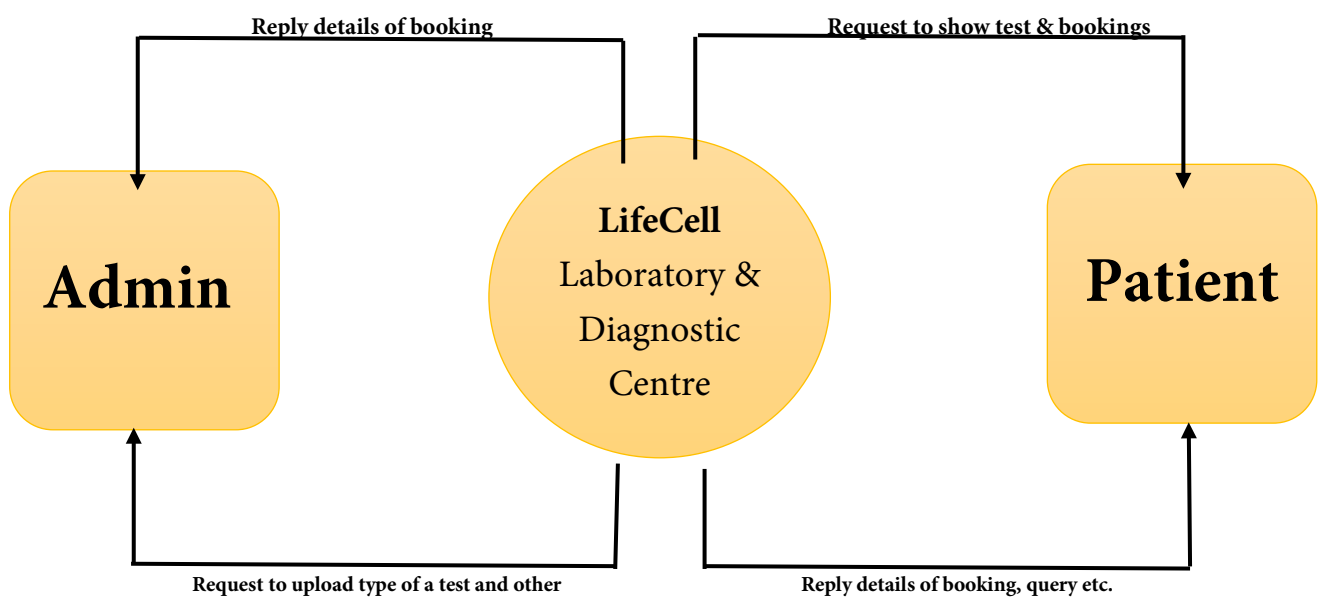


Figure-4: Context Level DFD

Level 1 DFD for Admin

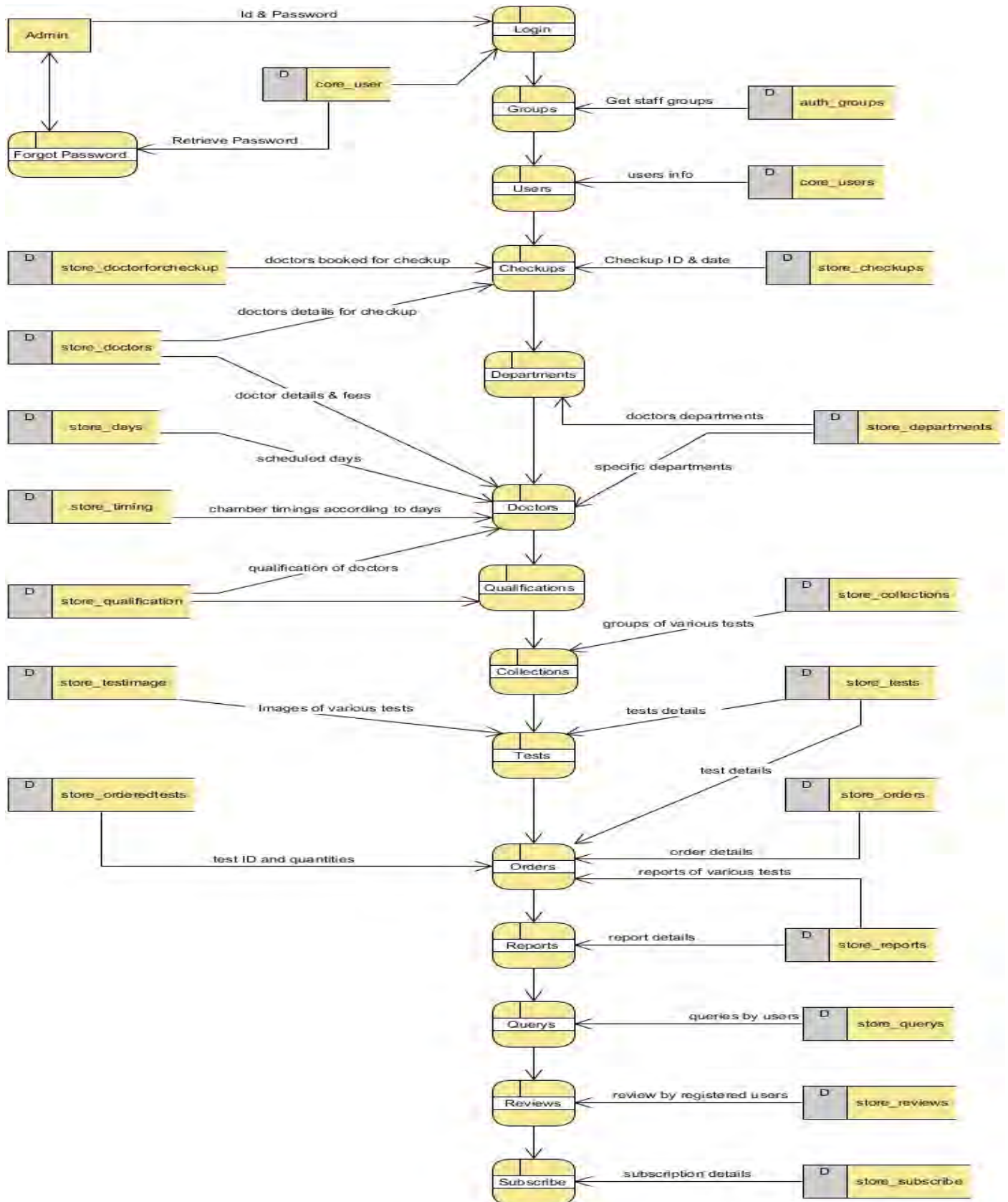


Figure-5: Level 1 DFD for Admin

Level 2 for Admin

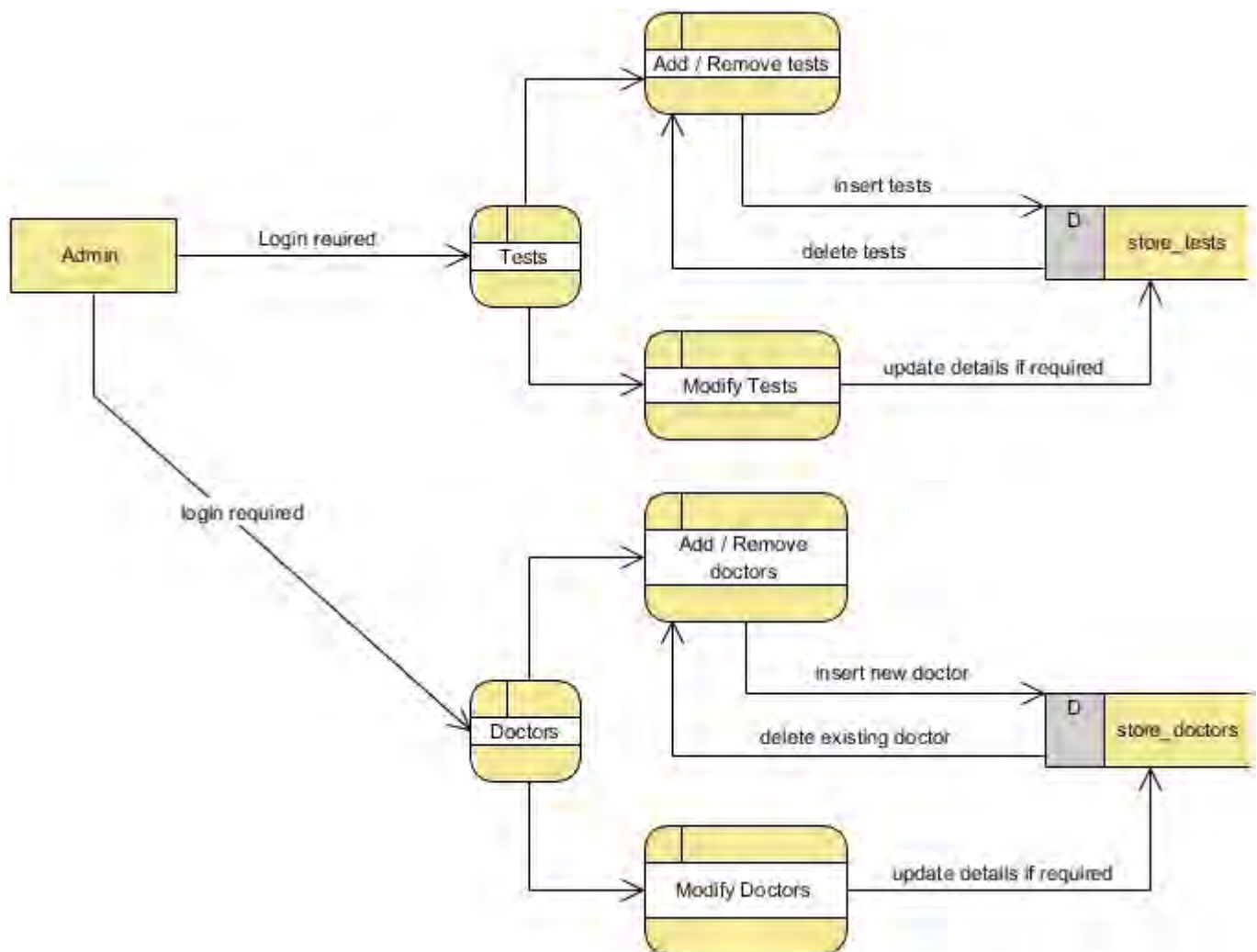


Figure-6: Level 2 DFD for Admin

Level 1 for Users

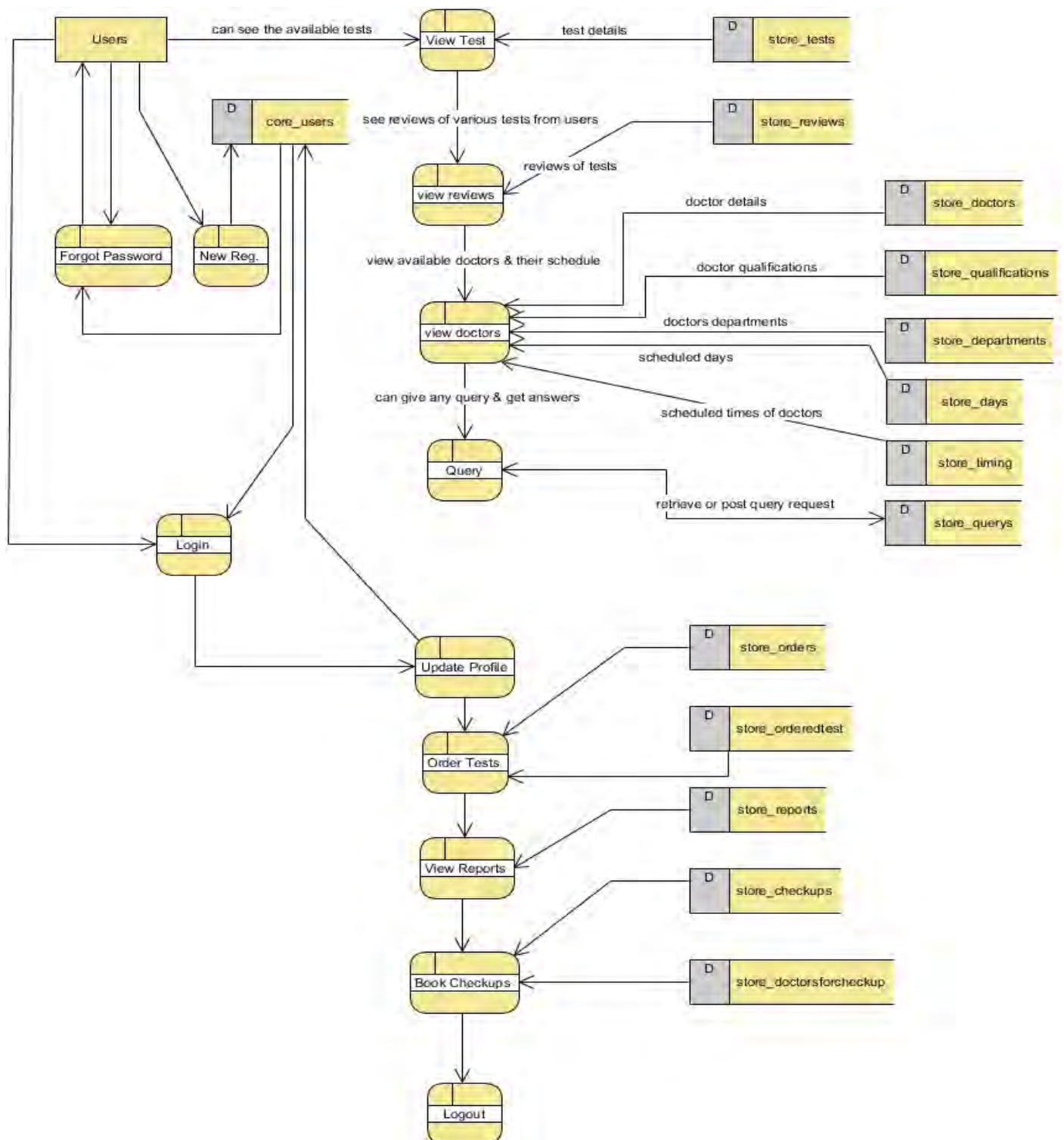


Figure-7: Level 1 DFD for Users

Database Schema

Table:

In our system we store the necessary information's on such several tables, which are as follows

Admin & Users → core_user:

```
create table core_user
(
    id                bigint auto_increment
        primary key,
    password          varchar(128) not null,
    last_login        datetime(6)  null,
    is_superuser      tinyint(1)   not null,
    username          varchar(150) not null,
    is_staff          tinyint(1)   not null,
    is_active         tinyint(1)   not null,
    date_joined       datetime(6)  not null,
    first_name        varchar(150) null,
    last_name         varchar(150) null,
    email             varchar(254) not null,
    address           varchar(300) null,
    phone             varchar(10)  null,
    sex               varchar(1)   null,
    age              varchar(5)    null,
    birth_date        date         null,
    doctor_name       varchar(255) null,
    password_store    varchar(250) null,
    constraint email  unique (email),
    constraint username unique (username)
);
```

Check-up → store_checkup:

```

create table store_checkup
(
    id          bigint auto_increment
    |          primary key,
    booked_at   date          not null,
    payment_status varchar(1) not null,
    user_id     bigint        not null,
    constraint store_checkup_user_id_801ea102_fk_core_user_id
    |          foreign key (user_id) references core_user (id)
);

```

Collections → store_collection:

```

create table store_collection
(
    id          bigint auto_increment
    |          primary key,
    title       varchar(255) not null,
    featured_test_id bigint    null,
    constraint title
    |          unique (title),
    constraint store_collection_featured_test_id_00e72463_fk_store_test_id
    |          foreign key (featured_test_id) references store_test (id)
);

```

Week Days → store_day:

```

create table store_day
(
    id  bigint auto_increment
    |  primary key,
    name varchar(20) not null
);

```


Departments ➔ store_department:

```
create table store_department
(
    id      bigint auto_increment
        primary key,
    title varchar(255) not null,
    constraint title
        unique (title)
);
```

Doctors ➔ store_doctor:

```
create table store_doctor
(
    id          bigint auto_increment
        primary key,
    first_name  varchar(255) not null,
    last_name   varchar(255) not null,
    email       varchar(254) not null,
    phone       varchar(10)  not null,
    fees        decimal(8, 2) not null,
    last_update datetime(6)  not null,
    address     longtext     not null,
    image       varchar(100) null,
    qualification_id bigint   not null,
    department_id bigint     not null,
    constraint email
        unique (email),
    constraint store_doctor_department_id_0b50ac9f_fk_store_department_id
        foreign key (department_id) references store_department (id),
    constraint store_doctor_qualification_id_7ed85668_fk_store_qualification_id
        foreign key (qualification_id) references store_qualification (id)
);
```

Doctors for Check-up → store_doctorforcheckup:

```

create table store_doctorforcheckup
(
    id          bigint auto_increment
        primary key,
    doctor_fees decimal(6) not null,
    checkup_id  bigint      not null,
    doctor_id    bigint      not null,
    constraint store_doctorforcheckup_checkup_id_f2b2360c_fk_store_checkup_id
        foreign key (checkup_id) references store_checkup (id),
    constraint store_doctorforcheckup_doctor_id_69a7a69f_fk_store_doctor_id
        foreign key (doctor_id) references store_doctor (id)
);

```

Orders → store_order:

```

create table store_order
(
    id          bigint auto_increment
        primary key,
    placed_at   date      not null,
    payment_status varchar(1) not null,
    user_id     bigint      not null,
    constraint store_order_user_id_ae5f7a5f_fk_core_user_id
        foreign key (user_id) references core_user (id)
);

```

Ordered Test → store_orderedtest:

```

create table store_orderedtest
(
    id          bigint auto_increment
        primary key,
    quantity    int         not null,
    unit_price  decimal(6, 2) not null,
    order_id    bigint       not null,
    test_id     bigint       not null,
    constraint store_orderedtest_order_id_39e5ed6b_uniq
        unique (order_id),
    constraint store_orderedtest_order_id_39e5ed6b_fk_store_order_id
        foreign key (order_id) references store_order (id),
    constraint store_orderedtest_test_id_1590ad7e_fk_store_test_id
        foreign key (test_id) references store_test (id)
);

```

Qualification→ store_qualification:

```
create table store_qualification
(
    id      bigint auto_increment
    |      primary key,
    title   varchar(255) not null,
    name    varchar(300) not null,
    constraint store_qualification_name_f1d2607e_uniq
    |      unique (name),
    constraint title
    |      unique (title)
);
```

Query→ store_query:

```
create table store_query
(
    id      bigint auto_increment
    |      primary key,
    name     varchar(255) not null,
    phone    varchar(10)  not null,
    date     date         not null,
    question longtext     not null,
    answer   longtext     null
);
```

Report→ store_report:

```
create table store_report
(
    id      bigint auto_increment
    |      primary key,
    detail   longtext not null,
    order_id bigint   not null,
    user_id  bigint   not null,
    date     date     not null,
    test_id  bigint   not null,
    constraint store_report_order_id_558c6075_fk_store_order_id
    |      foreign key (order_id) references store_order (id),
    constraint store_report_test_id_2efeb413_fk_store_test_id
    |      foreign key (test_id) references store_test (id),
    constraint store_report_user_id_ed9ee0f4_fk_core_user_id
    |      foreign key (user_id) references core_user (id)
);
```


Review: store_review:

```

create table store_review
(
    id          bigint auto_increment
    |          primary key,
    description longtext not null,
    date        date      not null,
    test_id     bigint     not null,
    user_id     bigint     not null,
    constraint store_review_test_id_d799563a_fk_store_test_id
    |          foreign key (test_id) references store_test (id),
    constraint store_review_user_id_cc54d86d_fk_core_user_id
    |          foreign key (user_id) references core_user (id)
);

```

Subscribe➔ store_subscribe:

```

create table store_subscribe
(
    id      bigint auto_increment
    |      primary key,
    name    varchar(150) not null,
    email   varchar(254) not null,
    date    date          not null,
    constraint store_subscribe_email_72256c8f_uniq
    |          unique (email)
);

```

Timing➔ store_timing:

```

create table store_timing
(
    id          bigint auto_increment
    |          primary key,
    start       time(6) not null,
    end         time(6) not null,
    day_id      bigint   not null,
    doctor_id   bigint   not null,
    constraint store_timing_day_id_ae402b06_fk_store_day_id
    |          foreign key (day_id) references store_day (id),
    constraint store_timing_doctor_id_5aea1cc8_fk_store_doctor_id
    |          foreign key (doctor_id) references store_doctor (id)
);

```

Tests→ store_tests:

```

create table store_test
(
    id          bigint auto_increment
    |          primary key,
    title       varchar(255) not null,
    slug        varchar(50)  not null,
    code        varchar(255) not null,
    description  longtext    null,
    unit_price   decimal(8, 2) not null,
    last_update  datetime(6) not null,
    collection_id bigint      not null,
    constraint code
    |          unique (code),
    constraint store_test_collection_id_741ccd28_fk_store_collection_id
    |          foreign key (collection_id) references store_collection (id)
);

create index store_test_slug_3a09bbc0
    on store_test (slug);

```

Test Images→ store_testimage:

```

create table store_testimage
(
    id          bigint auto_increment
    |          primary key,
    image       varchar(100) not null,
    test_id     bigint       not null,
    constraint store_testimage_test_id_d546fa9e_fk_store_test_id
    |          foreign key (test_id) references store_test (id)
);

```

E-R Diagram

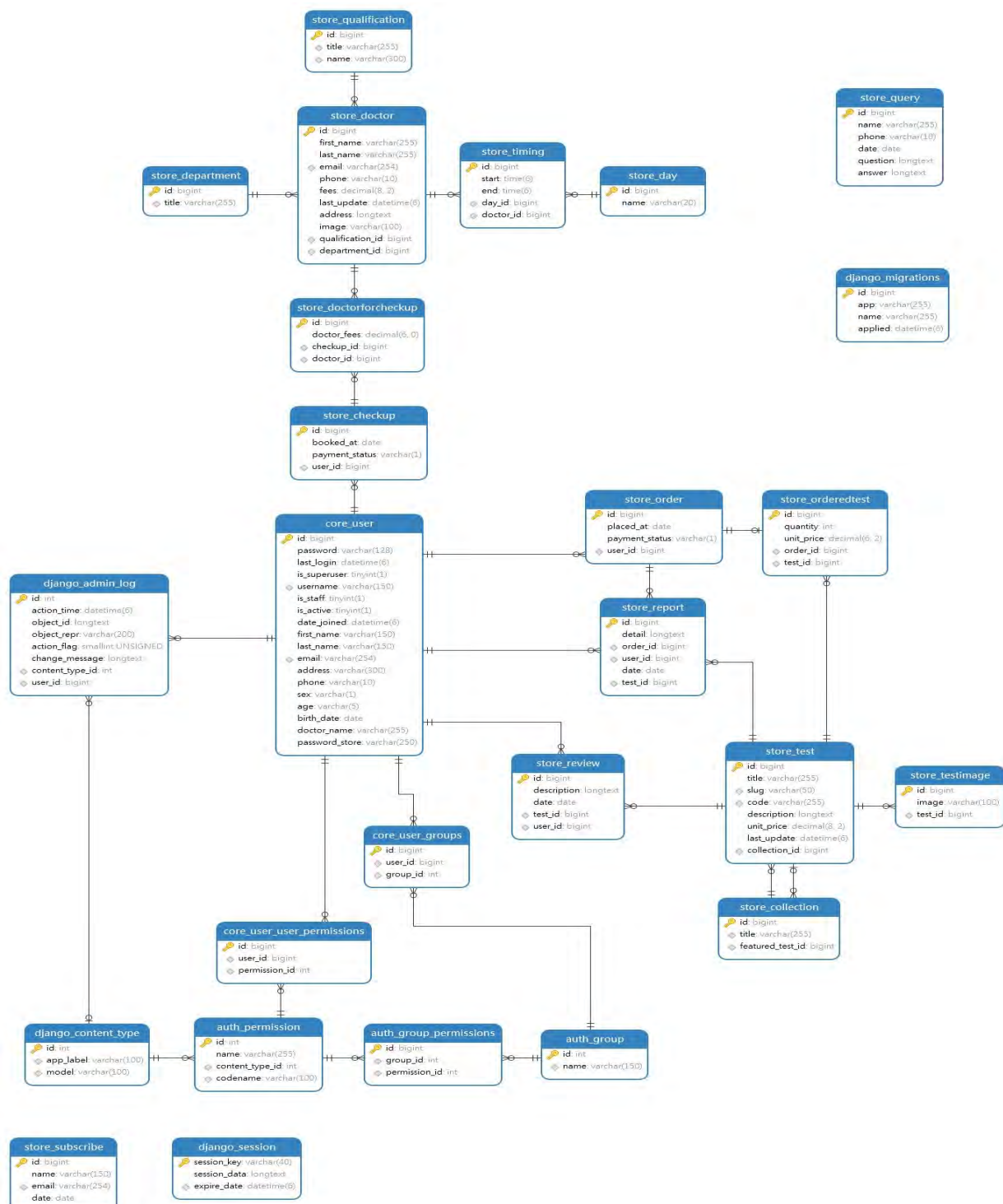
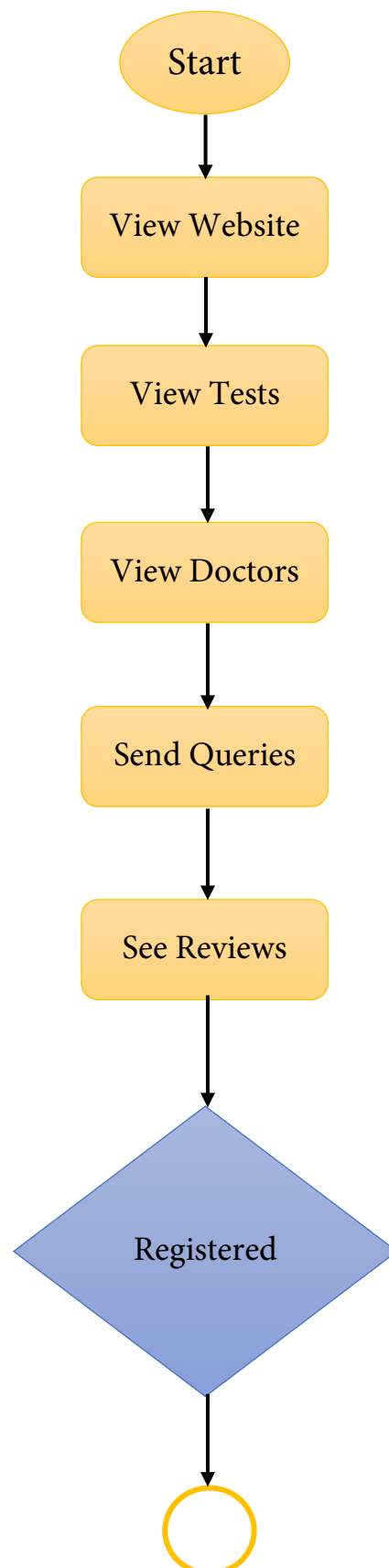
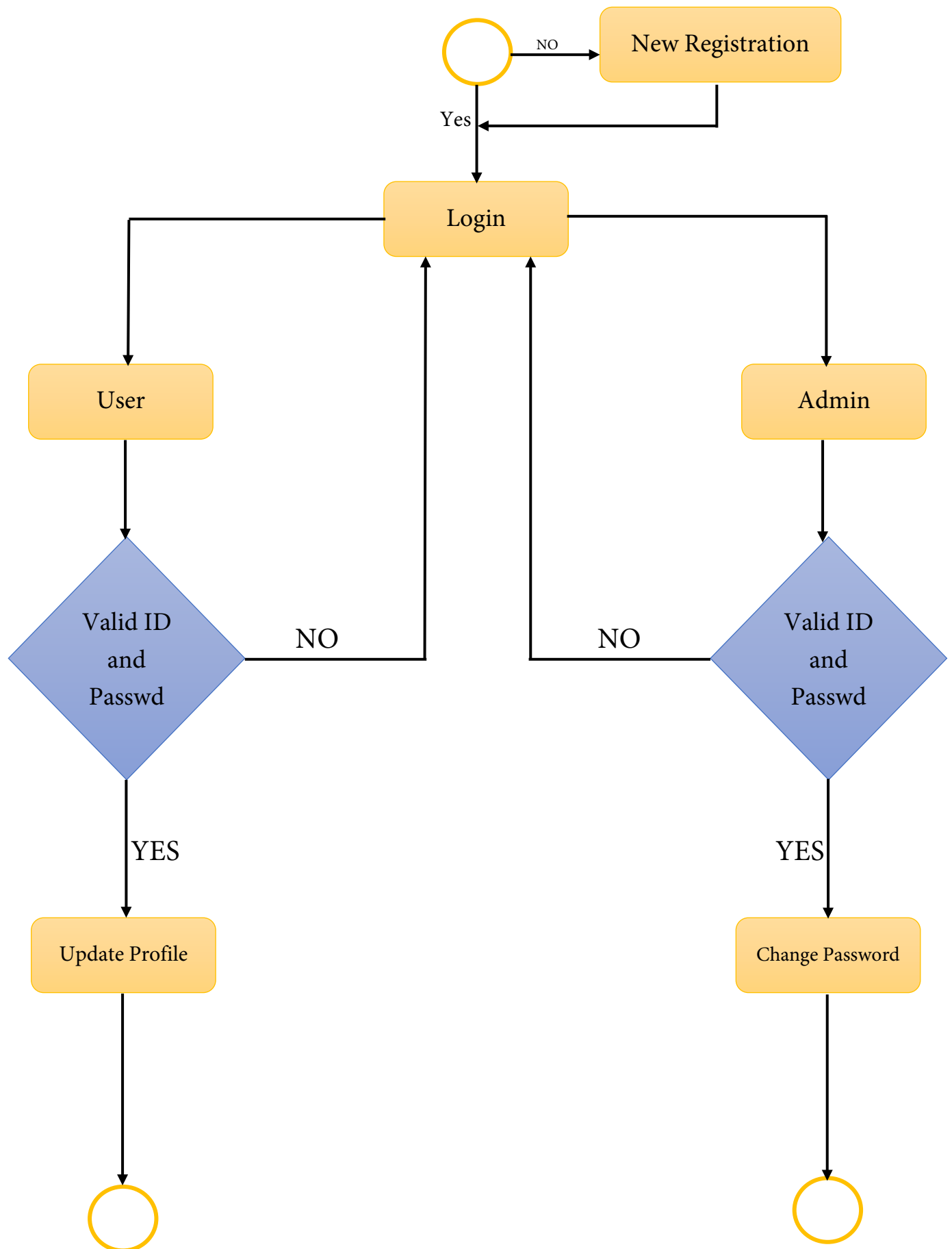
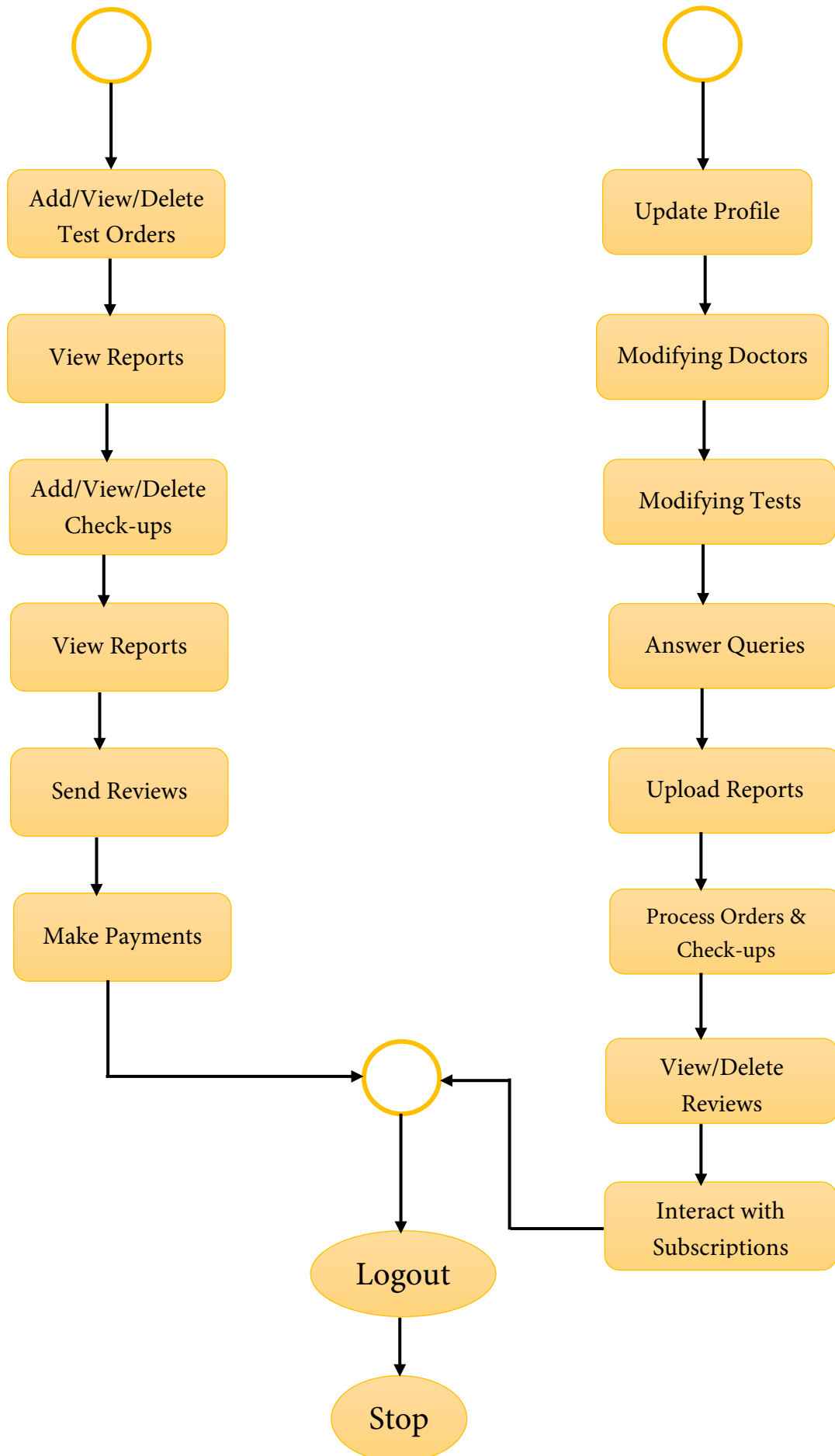


Figure-8: E-R Diagram

System Flow Chart







System Coding

Admin Page

```

from django.contrib import admin
from django.db.models import Count
from django.urls import reverse
from django.utils.html import format_html
from django.utils.http import urlencode

from . import models

@admin.register(models.Qualification)
class QualificationAdmin(admin.ModelAdmin):
    list_display = ['title', 'doctors_count', 'name']
    list_per_page = 10
    search_fields = ['title', 'name']

    def doctors_count(self, qualification: models.Qualification):
        url = (
            reverse('admin:store_doctor_changelist')
            + '?'
            + urlencode({
                'qualification_id': str(qualification.id)
            })
        )
        return format_html('<a href={}>{} Doctors</a>', url,
            qualification.doctors_count)

    def get_queryset(self, request):
        return super(QualificationAdmin, self).get_queryset(request).annotate(
            doctors_count=Count('doctor')
        )

@admin.register(models.Department)
class DepartmentAdmin(admin.ModelAdmin):
    list_display = ['title', 'doctors_count']
    list_per_page = 10
    search_fields = ['title']

    def doctors_count(self, speciality: models.Department):
        url = (
            reverse('admin:store_doctor_changelist')
            + '?'
            + urlencode({
                'department_id': str(speciality.id)
            })
        )
        return format_html('<a href={}>{} Doctors</a>', url, speciality.doctors_count)

    def get_queryset(self, request):
        return super(DepartmentAdmin, self).get_queryset(request).annotate(
            doctors_count=Count('doctor')
        )

```

```

def image_thumbnail(self, profile: models.Doctor):
    if profile.image.name != '':
        return format_html(
            f"<a href='{profile.image.url}'>"
            f"<img src='{profile.image.url}' "
            f"style='object-fit: cover; width: 125px; height: 125px;' "
            f"/>"
            f"</a>"
        )
    else:
        return ''

class TimingInline(admin.TabularInline):
    model = models.Timing
    min_num = 1
    extra = 0

@admin.register(models.Doctor)
class DoctorAdmin(admin.ModelAdmin):
    autocomplete_fields = ['qualification', 'department']
    list_display = ['first_name', 'last_name', 'qualification', 'department', 'fees',
                    'image_thumbnail']
    list_editable = ['fees']
    list_filter = ['department', 'qualification', 'last_update']
    list_per_page = 10
    inlines = [TimingInline]
    search_fields = ['first_name__istartswith', 'last_name__istartswith']

    def image_thumbnail(self, doctor: models.Doctor):
        if doctor.image.name != '':
            return format_html(
                f"<a href='{doctor.image.url}'>"
                f"<img src='{doctor.image.url}' "
                f"style='object-fit: cover; width: 125px; height: 125px;' "
                f"/>"
                f"</a>"
            )
        else:
            return ''

@admin.register(models.Collection)
class CollectionAdmin(admin.ModelAdmin):
    list_display = ['title', 'tests_count']
    list_per_page = 10
    search_fields = ['title']

    @admin.display(ordering='tests_count')
    def tests_count(self, collection: models.Collection):
        # reverse('admin:app_model_page')
        # app ---> what app are you working on
        # model ---> what is the target model
        # page ---> what is the target page, called 'changelist'
        url = (
            reverse('admin:store_test_changelist')
            + '?'
            + urlencode({
                'collection_id': str(collection.id)
            })
        )

```

```

    )
    return format_html('<a href={}>{} Tests</a>', url, collection.tests_count)

def get_queryset(self, request):
    return super(CollectionAdmin, self).get_queryset(request).annotate(
        tests_count=Count('test')
    )

class TestImageInline(admin.TabularInline):
    model = models.TestImage
    max_num = 3
    extra = 1
    readonly_fields = ['thumbnail']

    def thumbnail(self, test_image: models.TestImage):
        if test_image.image.name != '':
            return format_html(f"<img src='{test_image.image.url}' class='thumbnail'
/>")
        return ''

@admin.register(models.Test)
class TestAdmin(admin.ModelAdmin):
    autocomplete_fields = ['collection']
    prepopulated_fields = {
        'slug': ['title', 'code']
    }
    list_display = ['title', 'code', 'collection_title', 'unit_price']
    list_editable = ['unit_price']
    list_filter = ['collection', 'last_update']
    list_select_related = ['collection']
    list_per_page = 10
    inlines = [TestImageInline]
    search_fields = ['title__startswith']

    @admin.display(ordering='collection_title')
    def collection_title(self, test: models.Test):
        return test.collection.title

    class Media:
        css = {
            'all': ['store/styles.css']
        }

class OrderedTestsInline(admin.StackedInline):
    model = models.OrderedTest
    autocomplete_fields = ['test']
    min_num = 1
    max_num = 10
    extra = 0

@admin.register(models.Order)
class OrderAdmin(admin.ModelAdmin):
    autocomplete_fields = ['user']
    inlines = [OrderedTestsInline]
    search_fields = ['id']
    list_display = ['id', 'placed_at', 'payment_status', 'user', 'report_status']
    # 'ordered_tests_count']

```

```

def report_status(self, order: models.Order):
    try:
        order.reports.get(order_id=order.id)
        return 'Created'
    except models.Report.DoesNotExist:
        url = reverse('admin:store_report_add')
        return format_html('<a href="{}">Create!</a>', url)

# @admin.display(ordering='ordered_tests_count')
# def ordered_tests_count(self, order: models.Order):
#     return format_html('<a>{} Tests</a>', order.ordered_tests_count)

def get_queryset(self, request):
    return super(OrderAdmin, self).get_queryset(request).\
        select_related('user',
            'tests').annotate(ordered_tests_count=Count('tests'))

class DoctorForCheckupInline(admin.TabularInline):
    model = models.DoctorForCheckup
    autocomplete_fields = ['doctor']
    min_num = 1
    max_num = 10
    extra = 0

@admin.register(models.Checkup)
class CheckupAdmin(admin.ModelAdmin):
    autocomplete_fields = ['user']
    inlines = [DoctorForCheckupInline]
    list_display = ['id', 'user', 'booked_at', 'payment_status', 'doctors_for_checkup']

    @admin.display(ordering='doctors_for_checkup')
    def doctors_for_checkup(self, checkup: models.Checkup):
        return format_html('<a>{} Doctors</a>', checkup.doctors_for_checkup)

    def get_queryset(self, request):
        return super(CheckupAdmin, self).get_queryset(request).annotate(
            doctors_for_checkup=Count('doctors')
        )

@admin.register(models.Review)
class ReviewAdmin(admin.ModelAdmin):
    autocomplete_fields = ['user', 'test']
    ordering = ['-date']
    list_display = ['user', 'test', 'description', 'date']

@admin.register(models.Query)
class QueryAdmin(admin.ModelAdmin):
    list_display = ['name', 'date', 'phone', 'question', 'answer']
    list_editable = ['answer']
    list_per_page = 30
    ordering = ['-date']

@admin.register(models.Subscribe)
class SubscribeAdmin(admin.ModelAdmin):
    list_display = ['name', 'email', 'date']

```

```

@admin.register(models.Report)
class ReportAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        obj.test = obj.order.tests.test
        obj.user = obj.order.user
        super(ReportAdmin, self).save_model(request, obj, form, change)

    def name(self, report: models.Report):
        return report.user

    autocomplete_fields = ['order']
    list_display = ['id', 'order', 'name', 'detail']
    readonly_fields = ['test', 'user']

```

Database Models

```

import time
from datetime import datetime

from django.contrib import admin
from django.db import models
from django.core.validators import MinValueValidator, MinLengthValidator
from django.conf import settings

class Department(models.Model):
    title = models.CharField(max_length=255, unique=True)

    def __str__(self):
        return self.title

class Qualification(models.Model):
    name = models.CharField(max_length=300, unique=True)
    title = models.CharField(max_length=255, unique=True)

    def __str__(self):
        return f"{self.title} [ {self.name} ]"

class Doctor(models.Model):
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    email = models.EmailField(unique=True)
    phone = models.CharField(
        max_length=10,
        validators=[
            MinLengthValidator(10)
        ]
    )
    qualification = models.ForeignKey(Qualification, on_delete=models.PROTECT)
    department = models.ForeignKey(Department, on_delete=models.PROTECT)
    fees = models.DecimalField(
        max_digits=8,
        decimal_places=2,
        validators=[

```

```

        MinValueValidator(1),
    ]
)
last_update = models.DateTimeField(auto_now=True)
address = models.TextField()
image = models.ImageField(
    upload_to='store/images',
    null=True,
    blank=True
)

def __str__(self):
    return f"{self.first_name} {self.last_name}"

def name(self):
    return f"{self.first_name} {self.last_name}"

class Meta:
    ordering = ['first_name', 'last_name']

class Day(models.Model):
    name = models.CharField(max_length=20)

    def __str__(self):
        return self.name

class Timing(models.Model):
    start = models.TimeField()
    end = models.TimeField()
    day = models.ForeignKey(Day, on_delete=models.CASCADE)
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='timings')

    def convert_to_12_hour(self, value, formating_24="%H:%M:%S"):
        formating_12 = "%I:%M %p"
        value = str(value)
        time24 = datetime.strptime(value, formating_24)
        time12 = time24.strftime(formating_12)
        return time12

    def __str__(self):
        return f"{self.convert_to_12_hour(self.start)} to
{self.convert_to_12_hour(self.end)}"

class Collection(models.Model):
    title = models.CharField(max_length=255, unique=True)
    featured_test = models.ForeignKey(
        'Test',
        on_delete=models.SET_NULL,
        null=True,
        related_name='+',
        blank=True
    )

    def __str__(self):
        return self.title

class Meta:
    ordering = ['title']

```

```

class Test(models.Model):
    title = models.CharField(max_length=255)
    slug = models.SlugField()
    code = models.CharField(max_length=255, unique=True)
    description = models.TextField(null=True, blank=True)
    unit_price = models.DecimalField(
        max_digits=8,
        decimal_places=2,
        validators=[
            MinValueValidator(1),
        ]
    )
    last_update = models.DateTimeField(auto_now=True)
    collection = models.ForeignKey(Collection, on_delete=models.PROTECT)

    def __str__(self):
        return f"{self.title} [{self.code}]"

    class Meta:
        ordering = ['title']

class TestImage(models.Model):
    test = models.ForeignKey(Test, on_delete=models.CASCADE, related_name='images')
    image = models.ImageField(upload_to='store/images')

class Checkup(models.Model):
    PAYMENT_STATUS_PENDING = 'P'
    PAYMENT_STATUS_COMPLETE = 'C'
    PAYMENT_STATUS_FAILED = 'F'
    PAYMENT_STATUS_CHOICE = [
        (PAYMENT_STATUS_PENDING, 'Pending'),
        (PAYMENT_STATUS_COMPLETE, 'Complete'),
        (PAYMENT_STATUS_FAILED, 'Failed')
    ]

    booked_at = models.DateField(auto_now=True)
    payment_status = models.CharField(
        max_length=1,
        choices=PAYMENT_STATUS_CHOICE,
        default=PAYMENT_STATUS_PENDING
    )
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.PROTECT)

class DoctorForCheckup(models.Model):
    checkup = models.ForeignKey(Checkup, on_delete=models.PROTECT,
        related_name='doctors')
    doctor = models.ForeignKey(Doctor, on_delete=models.PROTECT)
    doctor_fees = models.DecimalField(
        max_digits=6,
        decimal_places=0,
        validators=[
            MinValueValidator(1),
        ]
    )

class Order(models.Model):

```



```

PAYMENT_STATUS_PENDING = 'P'
PAYMENT_STATUS_COMPLETE = 'C'
PAYMENT_STATUS_FAILED = 'F'
PAYMENT_STATUS_CHOICE = [
    (PAYMENT_STATUS_PENDING, 'Pending'),
    (PAYMENT_STATUS_COMPLETE, 'Complete'),
    (PAYMENT_STATUS_FAILED, 'Failed')
]

placed_at = models.DateField(auto_now_add=True)
payment_status = models.CharField(
    max_length=1,
    choices=PAYMENT_STATUS_CHOICE,
    default=PAYMENT_STATUS_PENDING
)
user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.PROTECT)

def __str__(self):
    return f"Order ID {self.id}"

class Meta:
    permissions = [
        ('cancel_order', 'Can Cancel Order')
    ]

class OrderedTest(models.Model):
    DEFAULT_QUANTITY = 1

    order = models.OneToOneField(Order, on_delete=models.PROTECT, related_name='tests')
    test = models.ForeignKey(Test, on_delete=models.PROTECT)
    quantity = models.IntegerField(
        default=DEFAULT_QUANTITY,
        validators=[
            MinValueValidator(1),
        ]
    )
    unit_price = models.DecimalField(
        max_digits=6,
        decimal_places=2,
        validators=[
            MinValueValidator(1),
        ]
    )

    def __str__(self):
        return self.test.title

class Review(models.Model):
    test = models.ForeignKey(Test, on_delete=models.CASCADE, related_name='reviews')
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.PROTECT)
    description = models.TextField()
    date = models.DateField(auto_now_add=True)

class Query(models.Model):
    name = models.CharField(max_length=255)
    phone = models.CharField(
        max_length=10,
        validators=[
            MinLengthValidator(10)

```

```

    ]
)
date = models.DateField(auto_now_add=True)
question = models.TextField()
answer = models.TextField(null=True, blank=True)

class Subscribe(models.Model):
    name = models.CharField(max_length=150)
    email = models.EmailField(unique=True)
    date = models.DateField(auto_now_add=True)

    def __str__(self):
        return f"{self.email}"

class Report(models.Model):
    order = models.ForeignKey(Order, on_delete=models.PROTECT, related_name='reports')
    test = models.ForeignKey(Test, on_delete=models.PROTECT)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.PROTECT)
    detail = models.TextField()
    date = models.DateField(auto_now_add=True)

```

Root URLs

"""E_Pathology URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
<https://docs.djangoproject.com/en/4.0/topics/http/urls/>

Examples:

Function views

1. Add an import: `from my_app import views`
2. Add a URL to `urlpatterns`: `path('', views.home, name='home')`

Class-based views

1. Add an import: `from other_app.views import Home`
2. Add a URL to `urlpatterns`: `path('', Home.as_view(), name='home')`

Including another URLconf

1. Import the `include()` function: `from django.urls import include, path`
2. Add a URL to `urlpatterns`: `path('blog/', include('blog.urls'))`

"""

```

from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

```

```

from . import views

```

```

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home_page, name='home'),
    path('auth/', include('core.urls')),
    path('subscribe', views.subscription_page, name='subscribe'),
    path('store/', include('store.urls')),
    path('__debug__/', include('debug_toolbar.urls')),
    # path('auth/', include('django.contrib.auth.urls')),
    # path('auth/', include('djoser.urls.jwt')),

```

```
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Store App URLs

```
from django.contrib.auth.decorators import login_required
from django.urls import path, include
from rest_framework_nested import routers
from . import views

app_name = 'store'

router = routers.DefaultRouter()
router.register('collections', views.CollectionViewSet, basename='collections')
router.register('departments', views.DepartmentViewSet, basename='departments')
router.register('tests', views.TestViewSet, basename='tests')
router.register('doctors', views.DoctorViewSet, basename='doctors')
router.register('orders', views.OrderViewSet, basename='orders')
router.register('checkups', views.CheckupViewSet, basename='checkups')
router.register('querys', views.QueryViewSet, basename='querys')
router.register('reviews', views.ReviewViewSet, basename='reviews')
router.register('reports', views.ReportViewSet, basename='reports')

# tests_router = routers.NestedDefaultRouter(router, 'tests', lookup='test')
# tests_router.register('images', views.TestImageViewSet, basename='test-images')

urlpatterns = [

] + router.urls
```

Core App URLs

```
from django.contrib import admin
from django.urls import path, include, reverse
from rest_framework_nested import routers

from . import views

app_name = 'core'

router = routers.DefaultRouter()
router.register('profile', views.UserProfileViewSet, basename='profile')

urlpatterns = [
    path('login/', views.login_user, name='login'),
    path('logout/', views.logout_user, name='logout'),
    path('register/', views.register_user, name='register'),
    path('forgot_password/', views.forgot_password, name='forgot_password'),
] + router.urls
```

Client Views

```

from django.contrib.auth.decorators import login_required
from django.utils.decorators import method_decorator
from django.contrib.auth.mixins import LoginRequiredMixin
from django.db.models import Count
from django.http import HttpRequest, HttpResponse
from django.shortcuts import render, redirect
from django.urls import reverse
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.decorators import action

from rest_framework.filters import SearchFilter, OrderingFilter
from rest_framework.response import Response
from rest_framework.viewsets import ModelViewSet
from rest_framework.renderers import AdminRenderer, JSONRenderer, TemplateHTMLRenderer

from rest_framework.permissions import IsAuthenticated, AllowAny,
IsAuthenticatedOrReadOnly

from .filters import TestFilter, QueryFilter, OrderFilter, ReviewFilter, DoctorFilter,
ReportFilter
from .models import Test, OrderedTest, Doctor, Collection, Order, Checkup, Query,
Review, TestImage, Report, Department
from .mixins import LoginRequiredMixin
from .pagination import DefaultPagination
from .permissions import IsAdminOrReadOnly
from .serializers import TestSerializer, DoctorSerializer, CollectionSerializer,
OrderSerializer, \
    AddOrderedTestSerializer, CheckupSerializer, AddDoctorForCheckupSerializer, \
    AddQuerySerializer, QuerySerializer, ReviewSerializer, AddReviewSerializer, \
    TestImageSerializer, AddTestImageSerializer, ReportSerializer, DepartmentSerializer

class DepartmentViewSet(ModelViewSet):
    renderer_classes = [AdminRenderer]
    serializer_class = DepartmentSerializer
    permission_classes = [IsAdminOrReadOnly]

    def get_queryset(self):
        return Department.objects.\
            annotate(doctors_count=Count('doctor')).\
            prefetch_related('doctor_set').all()

    def destroy(self, request, *args, **kwargs):
        if Doctor.objects.filter(department_id=kwargs['pk']).count() > 0:
            return Response({
                'error': 'This Collection can not be deleted as it has one or more
Doctors.'
            })
        return super(DepartmentViewSet, self).destroy(request, *args, **kwargs)

class CollectionViewSet(ModelViewSet):
    renderer_classes = [AdminRenderer]
    serializer_class = CollectionSerializer
    permission_classes = [IsAdminOrReadOnly]

    def get_queryset(self):
        return Collection.objects.\

```

```

        annotate(tests_count=Count('test')).\
        prefetch_related('test_set').all()

def destroy(self, request, *args, **kwargs):
    if Test.objects.filter(collection_id=kwargs['pk']).count() > 0:
        return Response({
            'error': 'This collection can not be deleted, as it has 1 or more
Tests.'
        })
    return super(CollectionViewSet, self).destroy(request, *args, **kwargs)

class TestViewSet(ModelViewSet):
    renderer_classes = [AdminRenderer]
    queryset =
Test.objects.select_related('collection').prefetch_related('images').all()
    serializer_class = TestSerializer
    filter_backends = [DjangoFilterBackend, SearchFilter, OrderingFilter]
    pagination_class = DefaultPagination
    filterset_class = TestFilter
    search_fields = ['title', 'description']
    ordering_fields = ['unit_price']
    permission_classes = [IsAdminOrReadOnly]

    def get_serializer_context(self):
        return {
            'request': self.request,
            'user_id': self.request.user.id
        }

    def destroy(self, request, *args, **kwargs):
        if OrderedTest.objects.filter(test_id=kwargs['pk']).count() > 0:
            return Response({
                'error': 'This test can not be deleted, as it has 1 or more orders.'
            })
        return super(TestViewSet, self).destroy(request, *args, **kwargs)

class DoctorViewSet(ModelViewSet):
    renderer_classes = [AdminRenderer]
    queryset = Doctor.objects.select_related('department',
'qualification').prefetch_related('timings').all()
    serializer_class = DoctorSerializer
    filter_backends = [DjangoFilterBackend]
    filterset_class = DoctorFilter
    permission_classes = [IsAdminOrReadOnly]

    def get_serializer_context(self):
        return {
            'request': self.request
        }

    @action(detail=True, renderer_classes=[TemplateHTMLRenderer])
    def schedule(self, request: HttpRequest, **kwargs):
        doctor = Doctor.objects.get(pk=kwargs['pk'])

        return render(
            request,
            'doctor_schedule.html',
            {
                'name': doctor.name(),
                'schedule': doctor.timings.all()
            }
        )

```

```

    }
)

class OrderViewSet(LoginRequiredMixin, ModelViewSet):
    http_method_names = ['get', 'post', 'delete', 'put']
    renderer_classes = [AdminRenderer]
    filter_backends = [DjangoFilterBackend, OrderingFilter]
    filterset_class = OrderFilter
    ordering_fields = ['placed_at']
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        return Order.objects.select_related('user').\
            prefetch_related('tests__test').\
            filter(user_id=self.request.user.id).\
            order_by('-placed_at', '-payment_status')

    def get_serializer_class(self):
        if self.request.method == 'POST':
            return AddOrderedTestSerializer
        return OrderSerializer

    def get_serializer_context(self):
        return {
            'user_id': self.request.user.id,
            'user': self.request.user,
        }

    @action(detail=True, renderer_classes=[TemplateHTMLRenderer])
    def payment(self, request: HttpRequest, **kwargs):
        order = Order.objects.get(pk=kwargs['pk'])
        order.payment_status = order.PAYMENT_STATUS_COMPLETE
        order.save()
        total_payable = order.tests.unit_price
        return render(
            request,
            'order_payment_msg.html',
            {
                'value': total_payable,
                'id': kwargs['pk'],
                'order': reverse('store:orders-list')
            }
        )

class CheckupViewSet(LoginRequiredMixin, ModelViewSet):
    renderer_classes = [AdminRenderer]
    permission_classes = [IsAuthenticated]

    def get_serializer_class(self):
        if self.request.method in ['POST']:
            return AddDoctorForCheckupSerializer
        return CheckupSerializer

    def get_serializer_context(self):
        return {
            'user_id': self.request.user.id
        }

    def get_queryset(self):
        return Checkup.objects.select_related('user').\

```

```

        prefetch_related('doctors__doctor').\
        filter(user_id=self.request.user.id).\
        order_by('-booked_at', '-payment_status')

    @action(detail=True, renderer_classes=[JSONRenderer])
    def payment(self, request: HttpRequest, **kwargs):
        checkup = Checkup.objects.get(pk=kwargs['pk'])
        checkup.payment_status = checkup.PAYMENT_STATUS_COMPLETE
        checkup.save()
        total_payable = sum(item.doctor_fees for item in checkup.doctors.all())
        return render(
            request,
            'checkup_payment_msg.html',
            {
                'value': total_payable,
                'id': kwargs['pk'],
                'checkup': reverse('store:checkups-list')
            }
        )

class QueryViewSet(ModelViewSet):
    renderer_classes = [AdminRenderer]
    http_method_names = ['get', 'post', 'delete']
    filter_backends = [DjangoFilterBackend]
    filterset_class = QueryFilter
    queryset = Query.objects.all().order_by('-date')
    permission_classes = [AllowAny]

    def get_serializer_class(self):
        if self.request.method == 'POST':
            return AddQuerySerializer
        return QuerySerializer

class ReviewViewSet(ModelViewSet):
    filter_backends = [DjangoFilterBackend]
    filterset_class = ReviewFilter
    renderer_classes = [AdminRenderer]
    queryset = Review.objects.select_related('user', 'test').all().order_by('-date')
    permission_classes = [IsAuthenticatedOrReadOnly]

    def get_serializer_context(self):
        return {
            'user_id': self.request.user.id
        }

    def get_serializer_class(self):
        if self.request.method == 'POST':
            return AddReviewSerializer
        return ReviewSerializer

class TestImageViewSet(LoginRequiredMixin, ModelViewSet):
    renderer_classes = [AdminRenderer]
    permission_classes = [IsAdminOrReadOnly]

    def get_serializer_class(self):
        if self.request.method == 'GET':
            return TestImageSerializer
        return AddTestImageSerializer

```

```
def get_serializer_context(self):
    return {
        'test_id': self.kwargs['test_pk'],
    }

def get_queryset(self):
    return TestImage.objects.filter(test_id=self.kwargs['test_pk'])

class ReportViewSet(LoginRequiredMixin, ModelViewSet):
    renderer_classes = [AdminRenderer]
    permission_classes = [IsAdminOrReadOnly]
    serializer_class = ReportSerializer
    filter_backends = [DjangoFilterBackend]
    filterset_class = ReportFilter

    def get_queryset(self):
        return Report.objects.filter(user_id=self.request.user.id).order_by('-date')
```

Login Restriction in some Endpoints

```
from django.contrib.auth.mixins import LoginRequiredMixin as BaseLoginRequiredMixin
from django.urls import reverse
```

```
class LoginRequiredMixin(BaseLoginRequiredMixin):
    def get_login_url(self):
        return str(reverse('core:login'))
```


Output Screen Shots

Home Page

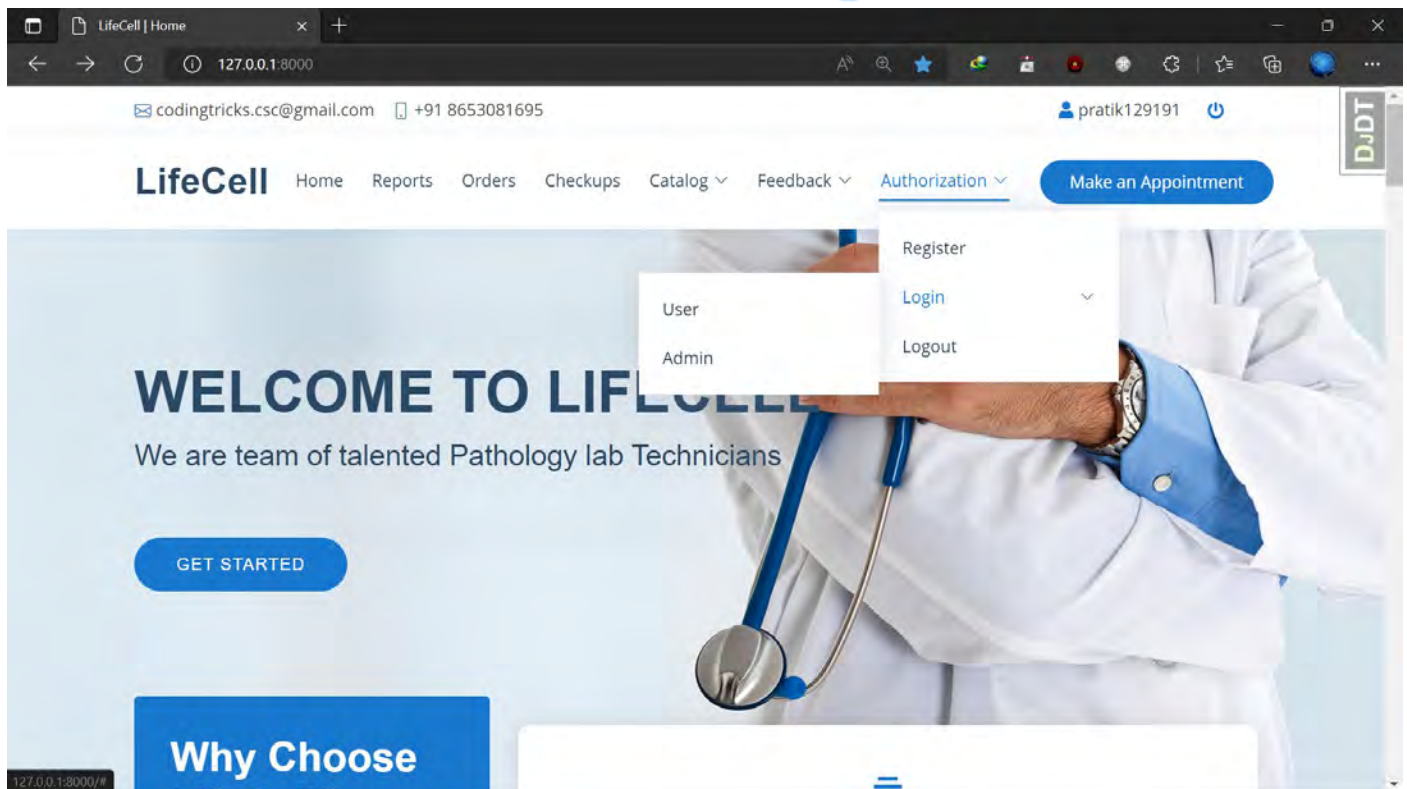


Figure-9: Home Screen Upper Part

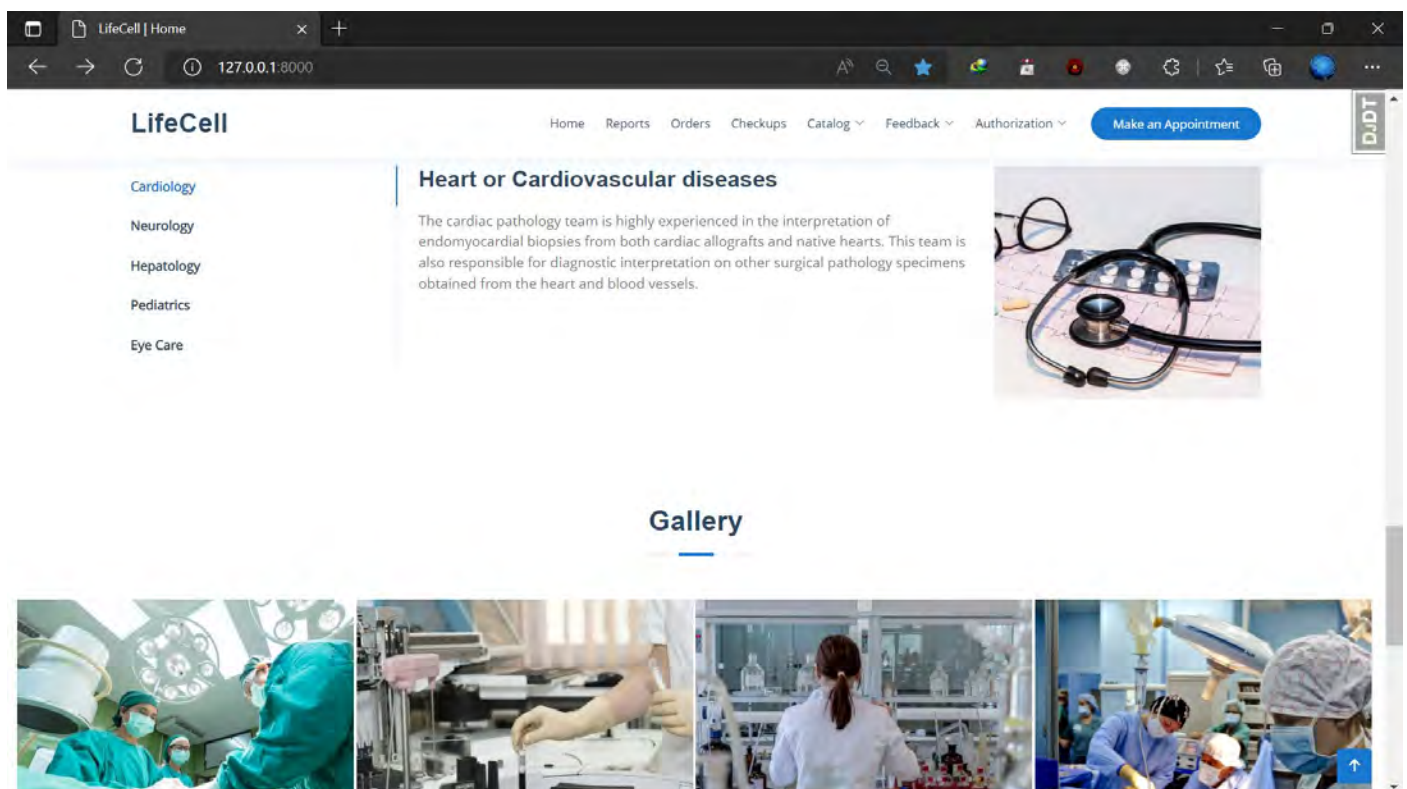
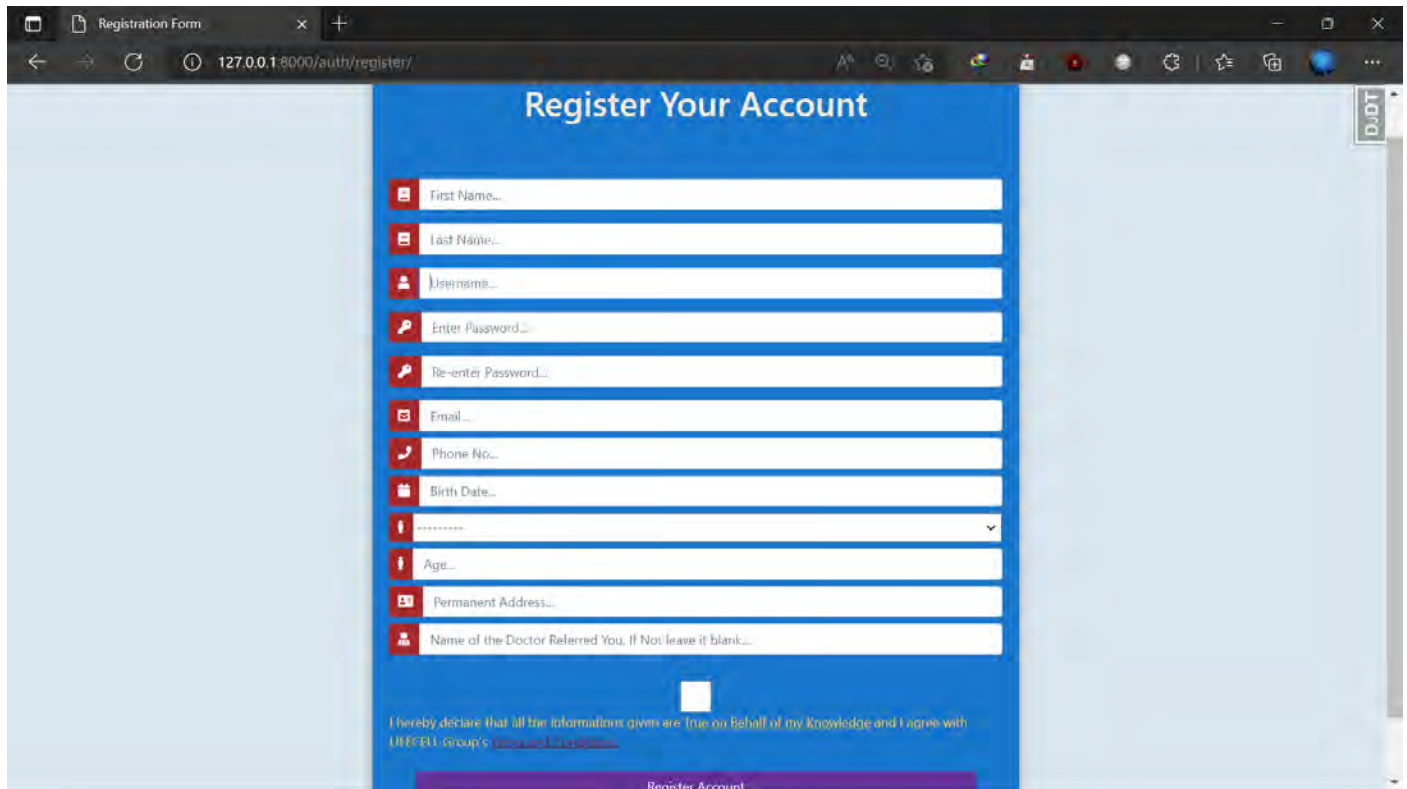


Figure-10: Home Screen Lower Part

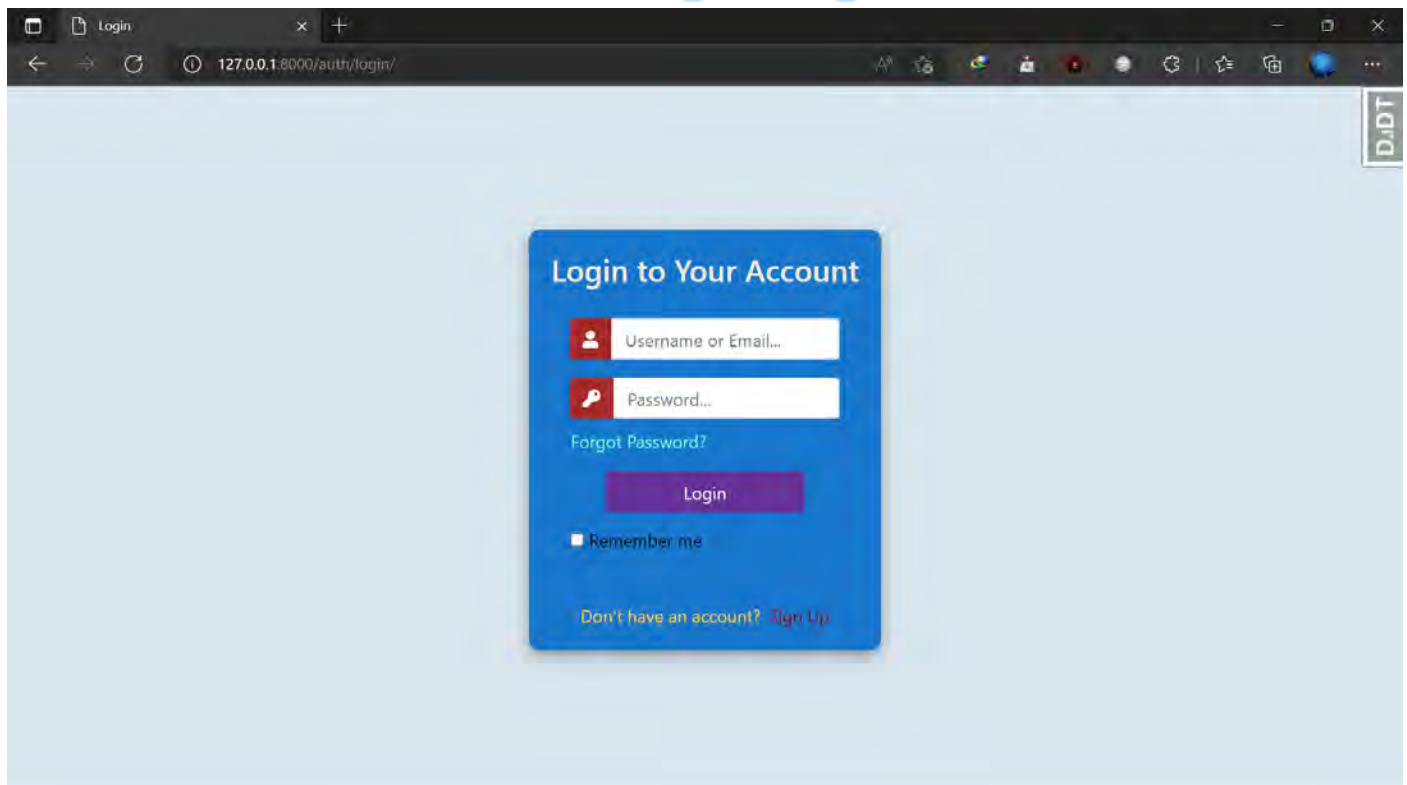
Registration Form



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/auth/register/". The page title is "Registration Form". The main content area has a blue header with the text "Register Your Account". Below the header, there is a form with the following fields: First Name..., Last Name..., Username..., Enter Password..., Re-enter Password..., Email..., Phone No..., Birth Date..., a dropdown menu, Age..., Permanent Address..., and Name of the Doctor Referred You, If Not leave it blank... At the bottom of the form, there is a checkbox and a text area for a declaration: "I hereby declare that all the informations given are true on behalf of my Knowledge and I agree with UPPCL Group's [Privacy and Conditions](#)". A purple button labeled "Register Account" is at the bottom of the form.

Figure-10: Registration Page

Login Page



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/auth/login/". The page title is "Login". The main content area has a blue header with the text "Login to Your Account". Below the header, there is a form with the following fields: Username or Email..., Password..., and a "Forgot Password?" link. A purple button labeled "Login" is below the password field. Below the login button, there is a checkbox labeled "Remember me". At the bottom of the form, there is a link: "Don't have an account? [Sign Up](#)".

Figure-11: Login Page

Orders Page

Order List

Id	Placed_at	Tests	Total_payable	Reports	Payment_status
(7)	2022-06-20	id: 7 test: id: 23, title: Full blood count	580.00	Pending!	Click to Pay!
(6)	2022-06-20	id: 8 test: id: 13, title: Billirubin & Amylase	145.00	View Report!	Payment Successful
(5)	2022-06-17	id: 5 test: id: 15, title: Sugar	230.00	Pending!	Click to Pay!

Figure-11: Orders Page

Doctors Page

Doctor List

Name	Qualification	Department	Availability	Fees	Checkup	Image
Agniva Maiti	MD	Nephrology	See Schedule	630.00	Book Checkup!	
Arohi Roy	DOM	Neurology	See Schedule	570.00	Book Checkup!	
Avradeep Sharma	MBBS	Hepatology	See Schedule	470.00	Book Checkup!	

Figure 12: Doctor-list Page.

Tests Page

Test List

Title	Code	Unit_price	Collection	Reviews	Order
Billirubin & Amylase	BAT	145.00	Chemical	See Reviews!	⚡ Place Order! ➔
Calcium	CACL	110.00	Chemical	See Reviews!	⚡ Place Order! ➔
Chloride	CHLRD	140.00	Chemical	See Reviews!	⚡ Place Order! ➔
Differential White Blood Count	DWBC	310.00	Haematology	See Reviews!	⚡ Place Order! ➔
Full blood count	FBC	580.00	General	See Reviews!	⚡ Place Order! ➔
Gene expression tests	GES	358.00	Genetic	See Reviews!	⚡ Place Order! ➔
Gene panel	GPST	520.00	Genetic	See Reviews!	⚡ Place Order! ➔
Glucose	GLC	125.00	Chemical	See Reviews!	⚡ Place Order! ➔
HB TC DC and ESR Test	HTET	780.00	General	See Reviews!	⚡ Place Order! ➔
Hematocrit Red Blood Cell Volume	HCT	380.00	Haematology	See Reviews!	⚡ Place Order! ➔

Figure 13: Tests Page

Filters

Field filters

Collection:

Unit price is greater than:

Unit price is less than:

Search

Ordering

unit_price - ascending

unit_price - descending

Figure 14: Tests Page 'Filter' Button.

Query Page

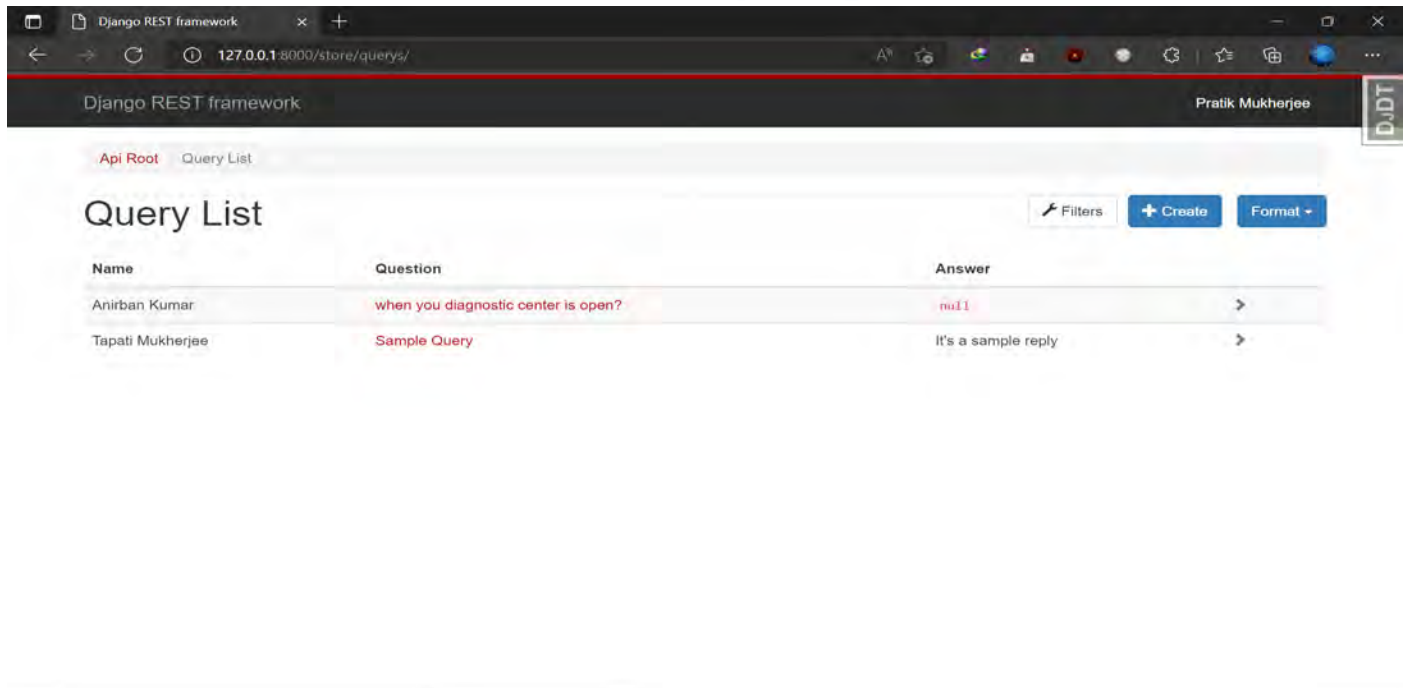


Figure 15: Query Page

Collections Page

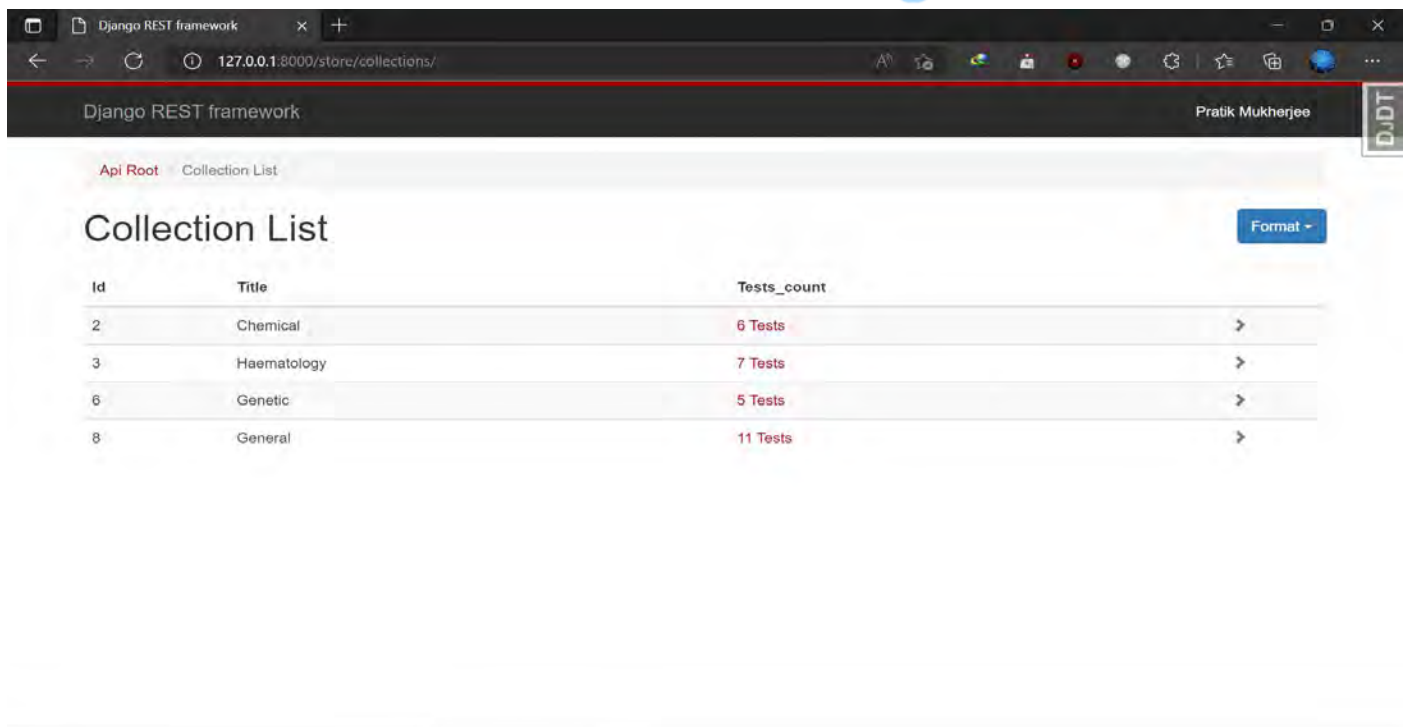


Figure 16: Collections Page

Admin Home

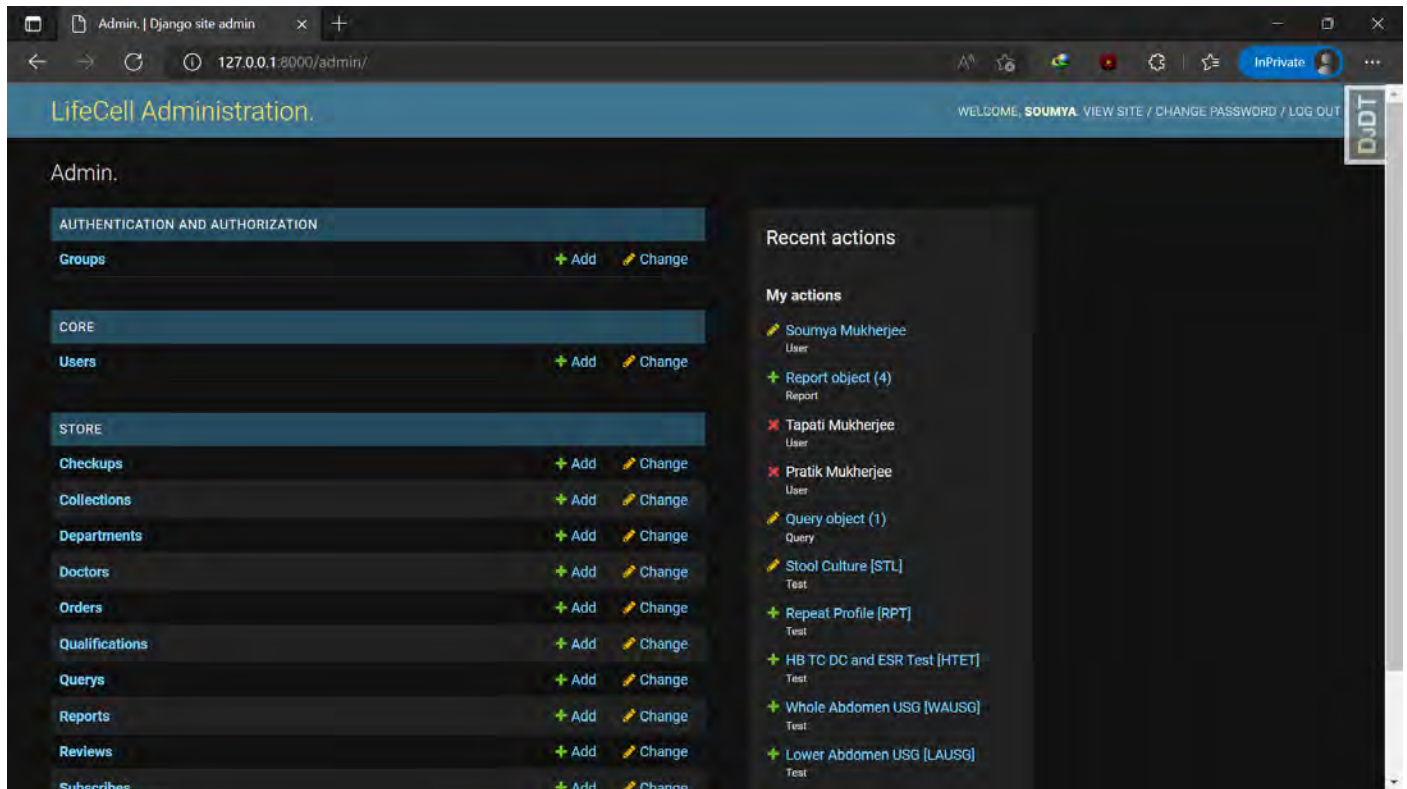


Figure 17: Admin Home Page

Orders List Page

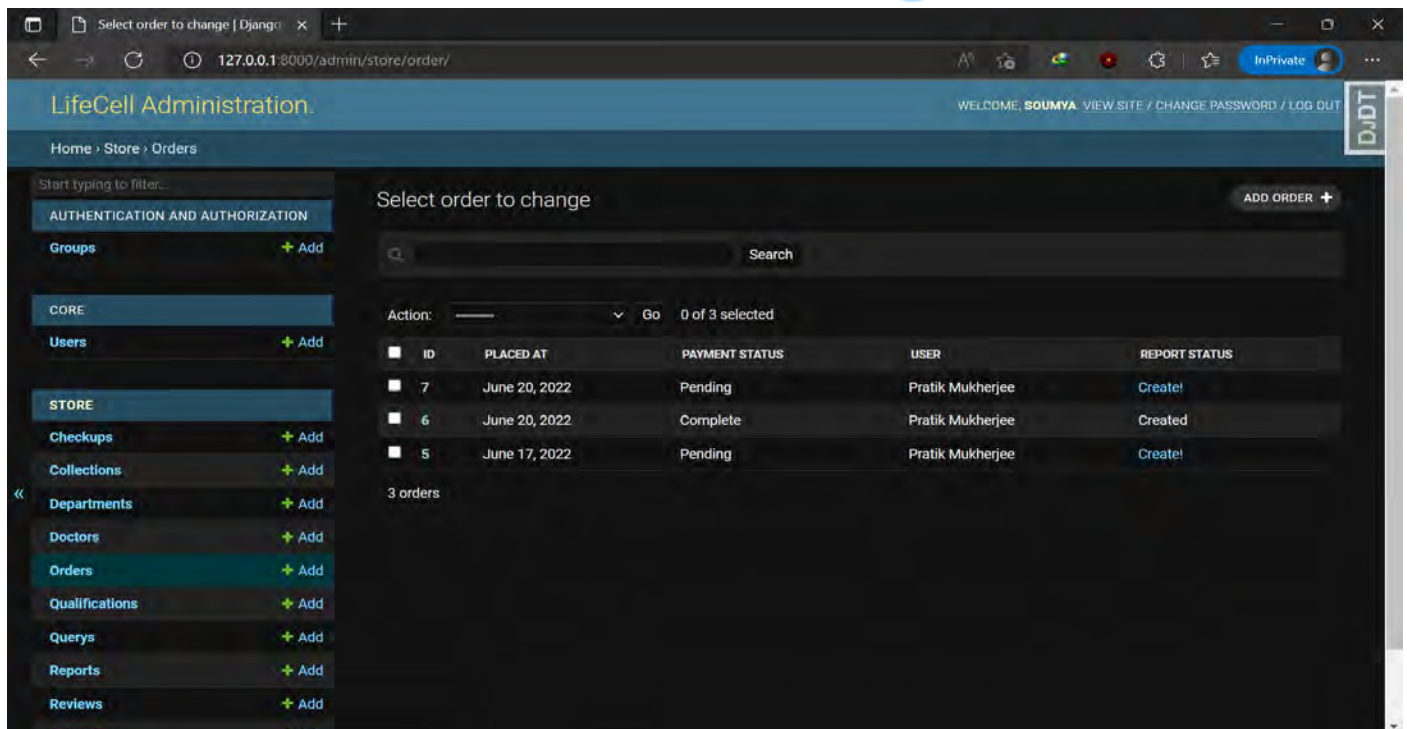


Figure 18: Orders List Page

Queries Page

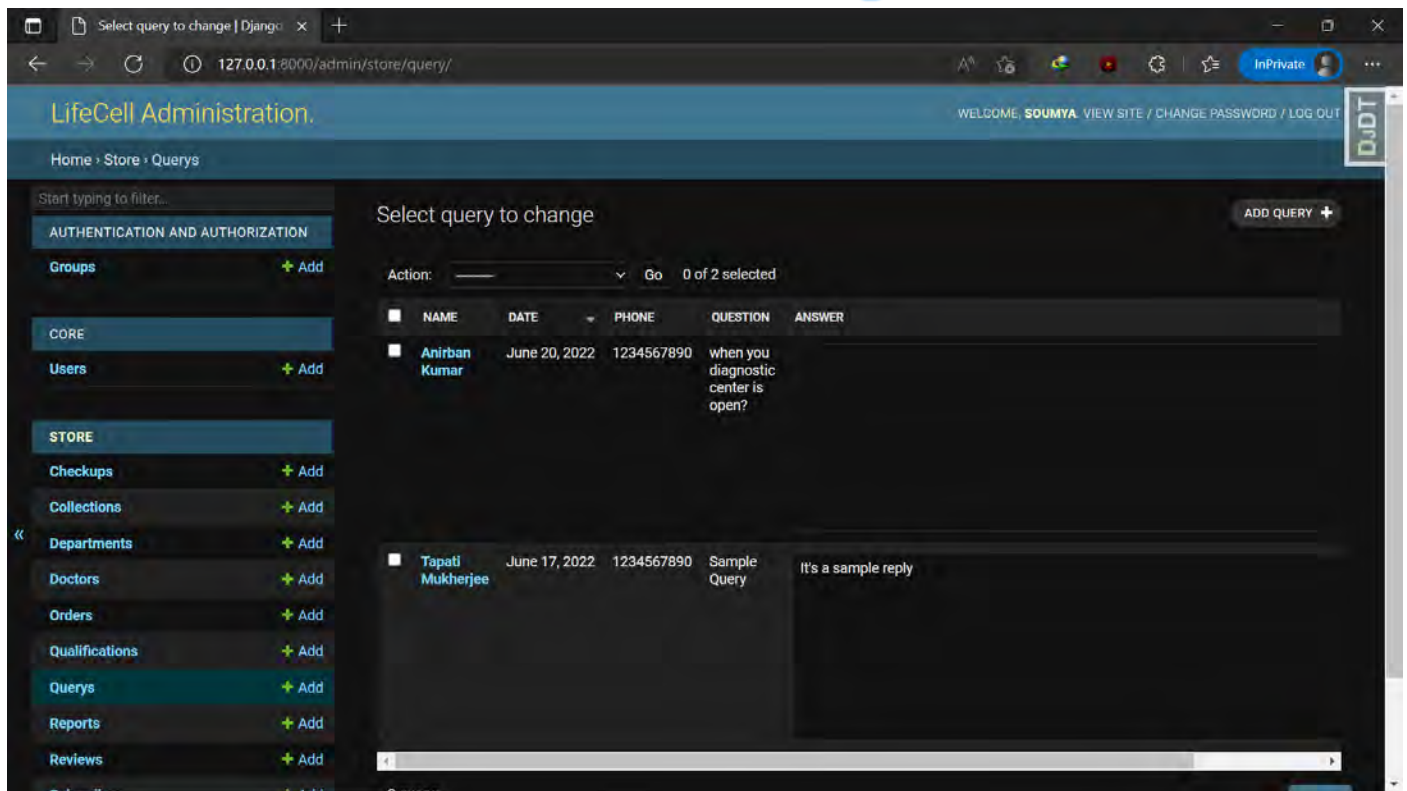


Figure 19: Queries Page for Admin

Tests List

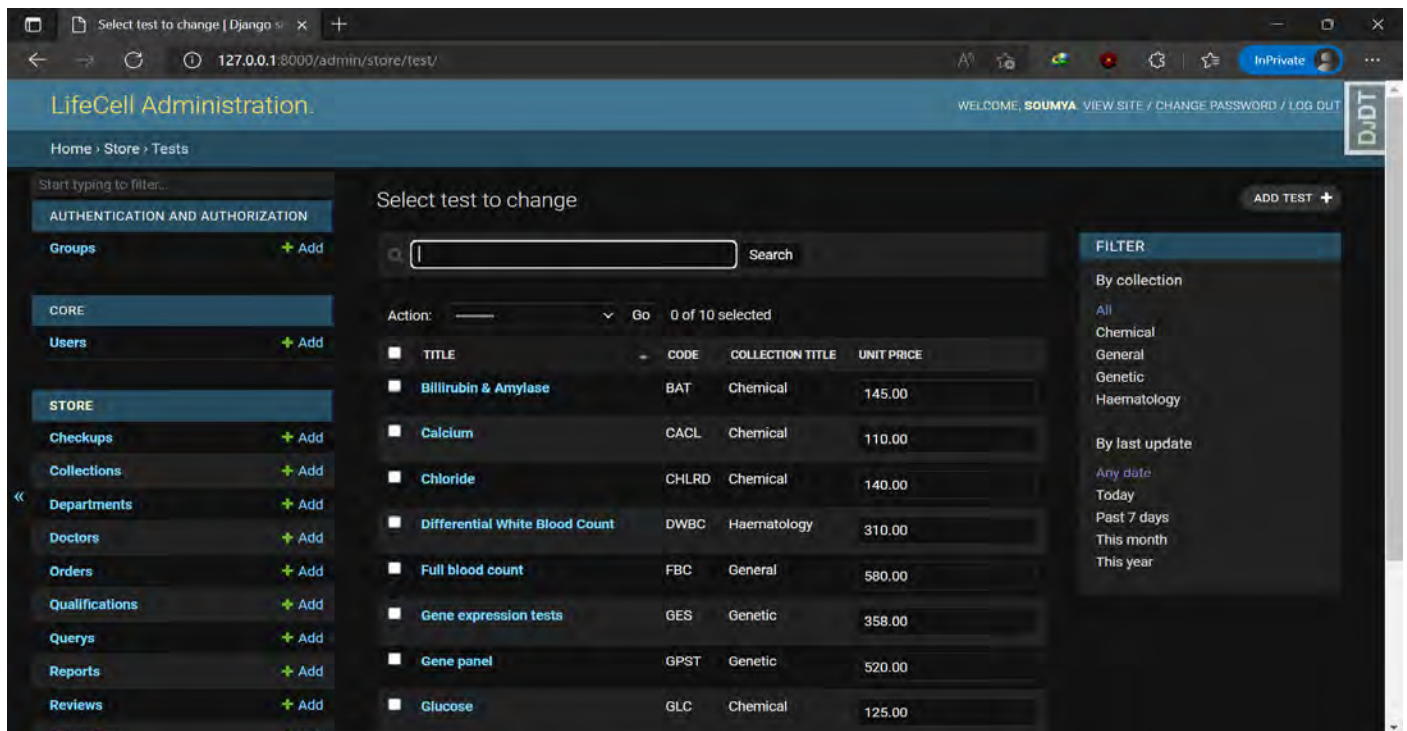


Figure 20: Tests List for Admin

Doctors List

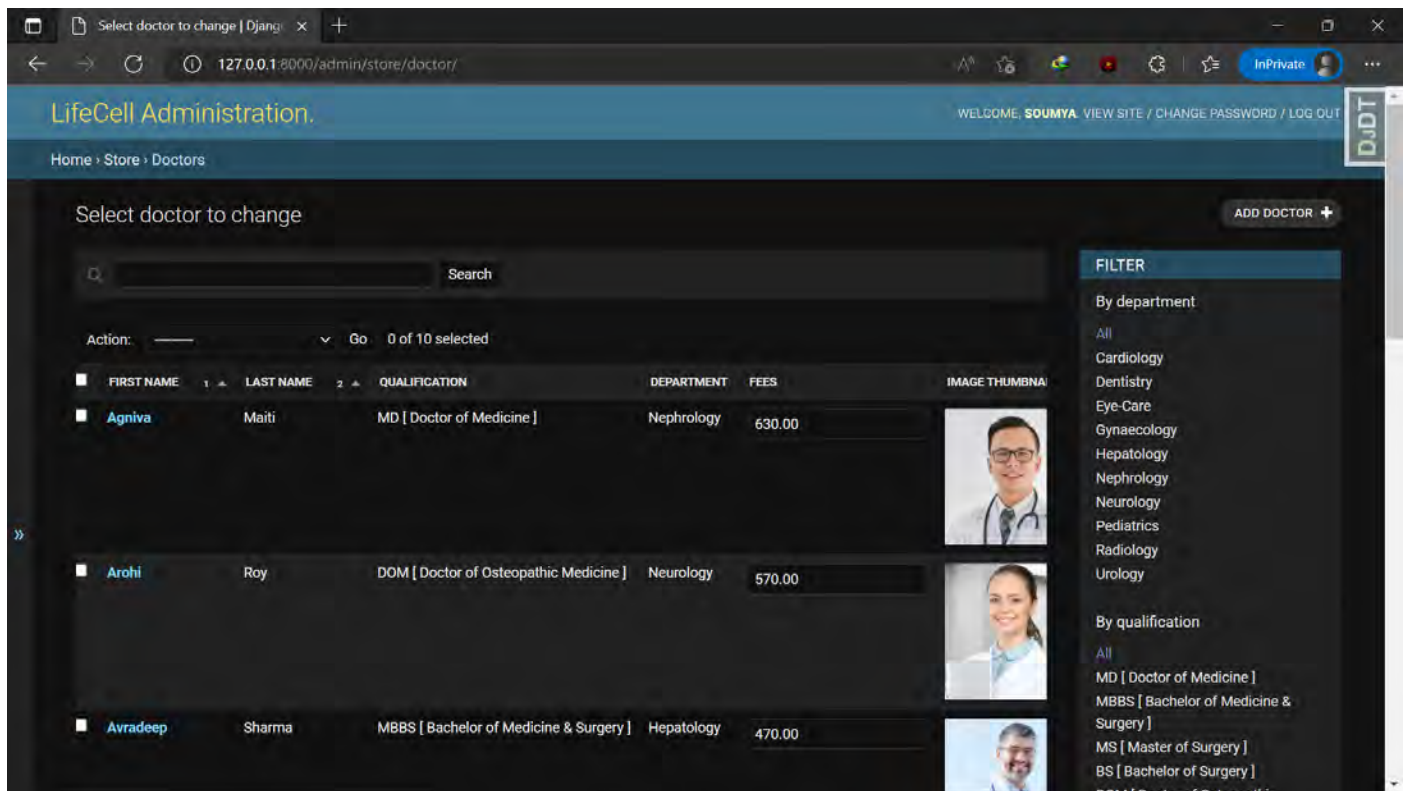


Figure 21: Doctors List for Admin

Check-ups Page

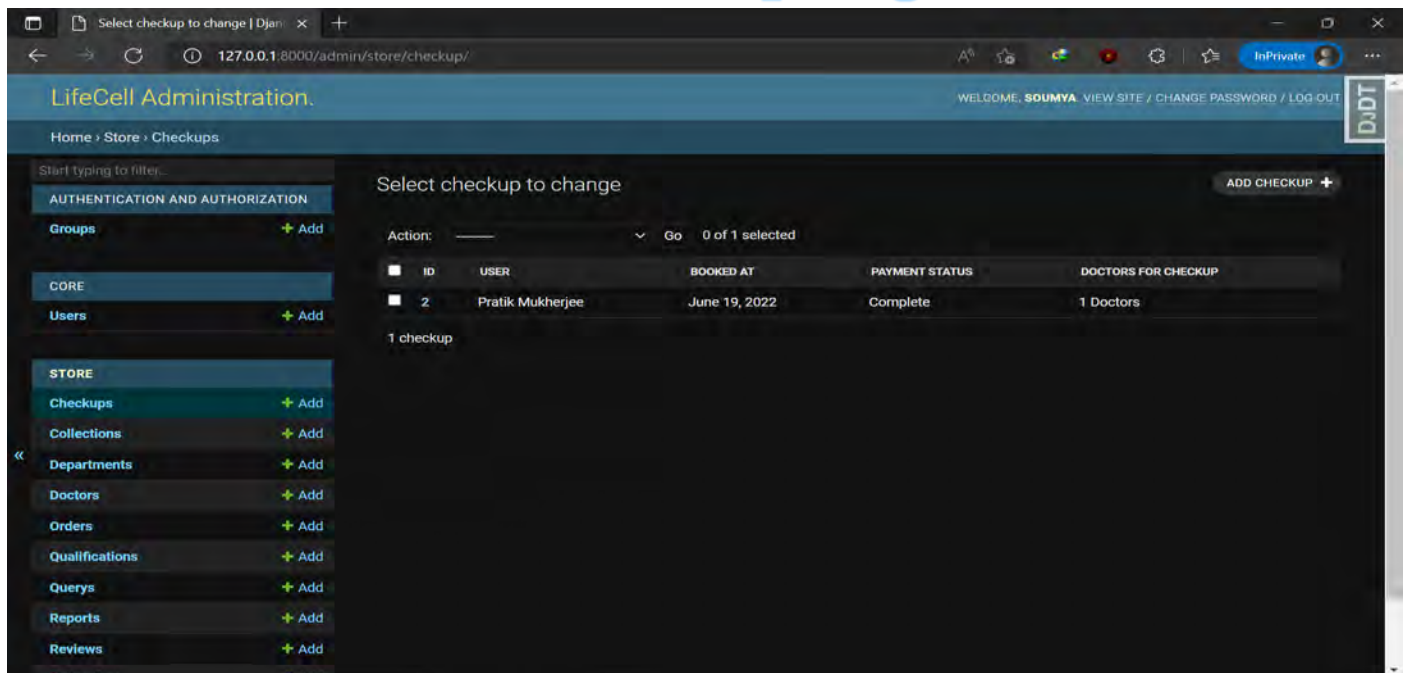


Figure 32: Check-ups Page for Admin

Software Testing

The testing activities are done in all phases of life cycle in an iterative software development model. However, the emphasis on testing activities varies in different phases. The testing process focuses on the logical intervals of the software ensuring that all statements have been tested and on functional interval is conducting tests to uncover errors & ensure that defined input will produce actual result that agree with the required results. Program level testing, module level testing integrated & carried out. A good test case is one that has a high probability of finding an undiscovered error. There are some testing objectives, such as:

- It is a process of executing a program with the intense of finding error in a program.
- A good test case is one that has a probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an undiscovered error

Testing Principles:

- All tests should be traceable to end user requirements
- Tests should be planned long before testing begins.
- Testing should begin on a small scale and progress towards testing in large.
- Exhaustive testing is not possible
- To be most effective testing should be conducted by an independent third-party

Types of Testing:

The primary objective for test case design is to derive a set of tests that has the highest livelihood for uncovering defects in software. To accomplish this objective two different categories of test case design techniques are available, like:

White Box Testing:

White box sometimes called “Glass box testing” is a test case design uses the control structure of the procedural design to drive test case. Using white box testing methods, the following tests were made on the system”

1. All independent paths within a module have been exercised once. In our system, ensuring that case was selected and executed checked all case structures. The bugs that were prevailing in some part of the code where fixed
2. All logical decisions were checked for the truth and falsity of the values.

Black Box Testing:

Black box testing focuses on the functional requirements of the software. This is black box testing enables the software engineering to derive a set of input conditions that will fully exercise all functional requirements for a program. Black box testing is not an alternative to white box testing rather it is complementary approach that is likely to uncover a different class of errors that white box methods like:

- 1) Interface errors
- 2) Performance in data structure
- 3) Performance errors
- 4) Initializing and terminating errors

Testing Plan:

Initial test plan address system test planning, and over the elaboration, design & execution phases this plan is updated to cater other testing requirements of these phases. The test plan must contain the following:

- Scope of testing.
- Methodology to be used for testing.
- Types of tests to be carried out.
- Resources & system requirements.
- Identification of various forms to be used to record test case & results.

System Maintenance

The maintenance of existing software can account for over 60 percent of all effort expended by the development organization, and the percentage continues to rise as more software's are produced. Uninitiated readers may ask why so much maintenance is required and much effort is expended. Much of the software we depend today is on average 10 to 15 years old. Even when these programs were created using the best design and coding techniques known at the time, they were created when program size and storage space were principle concerns. They were then migrated to new platforms, adjusted for changes in machine and operating system technology and enhanced to meet user needs all without enough regard to overall architecture.

The result is the poorly designed structures, poor coding, poor logic and poor documentation of the software systems we are now called on to keep running. We may define maintenance by describing four activities that are under taken after the program is released for use.

We defined for maintenance activities: corrective maintenance, adaptive maintenance, perfective maintenance or enhancement and preventive maintenance or reengineering. Only about 20% of all maintenance work is spent in fixing mistake. The remaining 80% are spent adopting existing systems to change in their external environment, making enhancements requested by the users, and reengineering an application for future use. When maintenance is considered to encompass all of these activities, it is easy to see why you absorb so much effort.

Advantages of Proposed System

- Reduces cost and improves efficiency of overall patient's data management.
- Integrates with referral commission reporting.
- Easy information sharing.
- It has a flexible reporting system.
- A centralized patient reporting system is also there, which helps in managing reports with less overhead and accessible by staff members.
- Flexibility to accommodate new requirements improves quality and compliance.
- Partially challenged patients can order tests or book check-ups by their own.

Applications

- The main aim of this project is to help patients to make their tests on their own. They should be able to register themselves to the diagnostic centre and get their tests done.
- It can be applied by any pathology and diagnostic centres to get their maintenance job done so easily.

Result Analysis

Sl. No.	Previous System	Current System
1.	Here most of the tasks was pen-paper based. So, there was a lot of overhead to get the maintenance job done.	Here the total concept is computerised and database based. So, the task becomes so easy to maintain.
2.	Previously there were many staff to handle this farm, so the customers could get their facilities.	In this new proposed system, we don't need any extra staff to facilitate the customers.
3.	In the previous system, patients could not register themselves by their own.	Here patients can register themselves by their own.
4.	Previously, customers have to come to the centre to collect their reports.	Patients can get their reports from the website just by logging in to their registered account.
5.	Here the service was in a limited area.	But now, due to cloud hosting of the website the service spread into a wide range of area.

Challenges

- In remote area this service may be unreachable due to lack of support system.
- There were a lot of issues to build this project as technical support was not always handy.
- As this is my first project so I have to find the suitable process to get rid of my errors over and over again.
- As I went through this project, I faced a major problem on the sequence of categories I have to do to make this fully responsible website.

Future Enhancements

This application avoids the manual work and the problems concern with it. It is an easy way to obtain the information regarding the various travel services that are present in our System.

Well I and my team member have worked hard in order to present an improved website better than the existing one's regarding the information about the various activities. Still, we found out that the project can be done in a better way. Primarily, in this system patient login and then go to reception. By using this patient will send request for consulting the doctor. Reception will set the date for doctor appointments. After that doctor see his appointments and see the patients, surgeries also done.

The next enhancement is, we will develop online services. That mean, if patient have any problems, he can send his problem to the doctor through internet from his home then doctor will send reply to him. In this patient have some login name and password.

Conclusion

The package was designed in such a way that future modifications can be done easily. The following conclusion can be deduced from the development of the project.

- Automation of the entire system improves the efficiency
- It provides a friendly graphical user interface which proves to be better when compared to the existing system.
- It gives appropriate access to the authorized users depending on their permissions.
- It effectively overcomes the delay in communications.
- Updating of information becomes so easier.
- System security, data security and reliability are the striking features.
- The System has adequate scope for modification in future if it is necessary.

Bibliography

The following sites were referred during the analysis & execution of this project.

- <https://codewithmosh.com/> (Dated on: 22-06-2022)
- <https://www.djangoproject.com/> (Dated on: 22-06-2022)
- <https://github.com/django/django> (Dated on: 22-06-2022)
- <https://www.w3schools.com/django/> (Dated on: 22-06-2022)
- <https://www.geeksforgeeks.org/django-tutorial/> (Dated on: 22-06-2022)
- <https://docs.djangoproject.com/en/4.0/intro/whatsnext/> (Dated on: 22-06-2022)

The End