Doctor Appointment System With Sentiment Analysis Project Report

By

Saud Shaikh- AF04951388

Safwaan Attar - AF0495759

Index

Acknowledgement

The project "**Doctor Appointment System Sentiment Analysis "**

is the Project work carried out by

Name	Enrollment No
Saud Shaikh	**AF04951388**
Safwaan Attar	**AF0495759**

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related Topics.

We are also thankful to my family member and friends who are always there to provide support and moral boost up.

Abstract

The Doctor Appointment Management System is an advanced software solution meticulously designed to address the complexities of scheduling and managing medical appointments. This system provides an integrated platform that facilitates seamless interactions between patients and healthcare providers, aiming to improve the overall efficiency and effectiveness of appointment management within medical practices.

At its core, the system offers a user-friendly interface for patients, allowing them to easily browse through available time slots, book appointments, and make changes as needed. This reduces the manual effort required for appointment scheduling and minimizes the risk of scheduling conflicts.

For healthcare providers, the system offers robust tools to manage their schedules efficiently. Doctors can set their availability, accept or decline appointment requests, and view their daily, weekly, or monthly schedules with ease. This functionality not only streamlines the appointment booking process but also reduces the administrative workload associated with managing patient schedules.

Administrators are provided with comprehensive management features, including user authentication and profile management for doctors. They can oversee appointment histories, generate detailed reports, and ensure the smooth operation of the system. This centralization of administrative tasks supports better oversight and enhances the overall management of healthcare services.

The system is designed with a focus on security and compliance, ensuring that sensitive patient information is protected through robust encryption and secure access controls. By integrating intuitive interface, the Doctor Appointment Management System aims to foster better communication between patients and healthcare providers, ultimately contributing to improved patient care and operational efficiency.

In summary, this system is a transformative tool that addresses the challenges of appointment management in healthcare settings. It enhances the scheduling process, reduces administrative burdens, and supports a higher standard of patient care through its innovative features and user-centric design.

1. Introduction

In today's fast-paced healthcare environment, effective management of doctor appointments is crucial for delivering high-quality patient care. The Doctor Appointment Management System (DAMS) has been developed to streamline the scheduling process for both patients and healthcare providers, addressing the challenges often associated with traditional appointment booking methods.

Patients frequently encounter issues such as long wait times, scheduling conflicts, and miscommunication when trying to secure appointments. The DAMS tackles these challenges by providing a user-friendly platform that allows patients to easily book, reschedule, or cancel appointments online. This not only empowers patients with greater control over their healthcare but also reduces the administrative burden on medical staff.

For healthcare providers, the system offers an array of powerful tools designed to enhance operational efficiency. Providers can manage their schedules in real-time, track patient visits, and optimize resource allocation. This leads to more effective use of time and facilities, ensuring that patients receive timely care and that practitioners can maximize their productivity.

By facilitating seamless communication and providing easy access to essential information, the system enhances the overall patient experience.

In essence, the Doctor Appointment Management System not only addresses common pain points in the appointment process but also fosters a more efficient and responsive healthcare delivery model. By leveraging technology to improve appointment management, we aim to enhance patient satisfaction, streamline healthcare operations, and ultimately contribute to better health outcomes for the community. This project stands as a testament to our commitment to innovation in healthcare, ensuring that quality care is accessible and manageable for all stakeholders involved.

1.1 Objective of the present work

1. Simplify Appointment Scheduling: Provide a platform for patients to easily book, reschedule, and cancel medical appointments online, making the process more convenient and reducing the need for manual scheduling.
2. Boost Doctor Efficiency: Allow doctors to efficiently manage their schedules by setting their availability, viewing and adjusting appointments, and minimizing scheduling conflicts, thereby reducing administrative workload.
3. Enhance Patient Experience: Create a user-friendly interface that ensures patients can quickly and easily interact with the system to manage their appointments, improving their overall experience and satisfaction.
4. Ensure Robust Data Security: Implement strong security measures to protect sensitive patient and doctor information from unauthorized access, maintaining compliance with healthcare privacy regulations and ensuring data integrity.
5. Enable Effective Administration: Provide administrators with comprehensive tools to oversee user accounts, monitor system usage, generate reports, and manage system operations, ensuring smooth and efficient management.
6. Privacy: Doctors have to login the page with their ID and password to check the details of the patients. Other patients and any other website viewer cannot see details without doctor ID and password.
7. Maintain Accurate Data: Ensure that all appointment records, patient details, and doctor schedules are updated in real-time, providing accurate and reliable information for effective appointment management.
8. Support System Growth: Design the system to be scalable, capable of handling increased user loads and future enhancements without compromising performance or user experience.
9. Ensure User-Friendly Interface: Develop an easy-to-navigate and visually appealing interface that enhances usability for patients, doctors, and administrators, making the system intuitive and accessible for all users.

System analysis

3.1 PROBLEM DEFINITION

In the traditional healthcare system, booking a doctor's appointment is often time-consuming and inconvenient for patients. They face difficulties in knowing the doctor's availability, consultation timings, and specialization, which leads to delays and mismanagement. Manual appointment booking increases waiting times and creates errors in record-keeping. Doctors also struggle with handling large numbers of patients, cancellations, and emergency cases efficiently. There is a lack of transparency in consultation fees, availability, and proper scheduling, causing frustration for both patients and doctors. With the rise of digital technology, there is a need for an online platform that simplifies appointment booking and provides real-time information. A Doctor Appointment Website will reduce waiting times, ensure accurate scheduling, and improve communication between patients and healthcare providers. This system will save time, enhance accessibility, and increase overall satisfaction for both patients and doctors.

3.2 PRELIMINARY INVESTGATION.

Purpose

The purpose of the Doctor Appointment Website is to provide a convenient and efficient platform for patients to book, reschedule, or cancel doctor appointments online. It aims to reduce waiting times, eliminate manual errors, and improve communication between patients and healthcare providers. The system also helps doctors manage their schedules effectively, ensuring smooth operations and better patient care.

Scope

The scope of this project includes the development of a user-friendly web-based platform where patients can search for doctors by specialization, availability, and location. The system will allow patients to view consultation fees, doctor profiles, and available time slots before booking. Doctors will have access to a secure dashboard to manage their appointments, view patient details, and track records. The platform will also maintain patient history and ensure data security, making it useful for both small clinics and large hospitals.

3.3 Feasibility Study

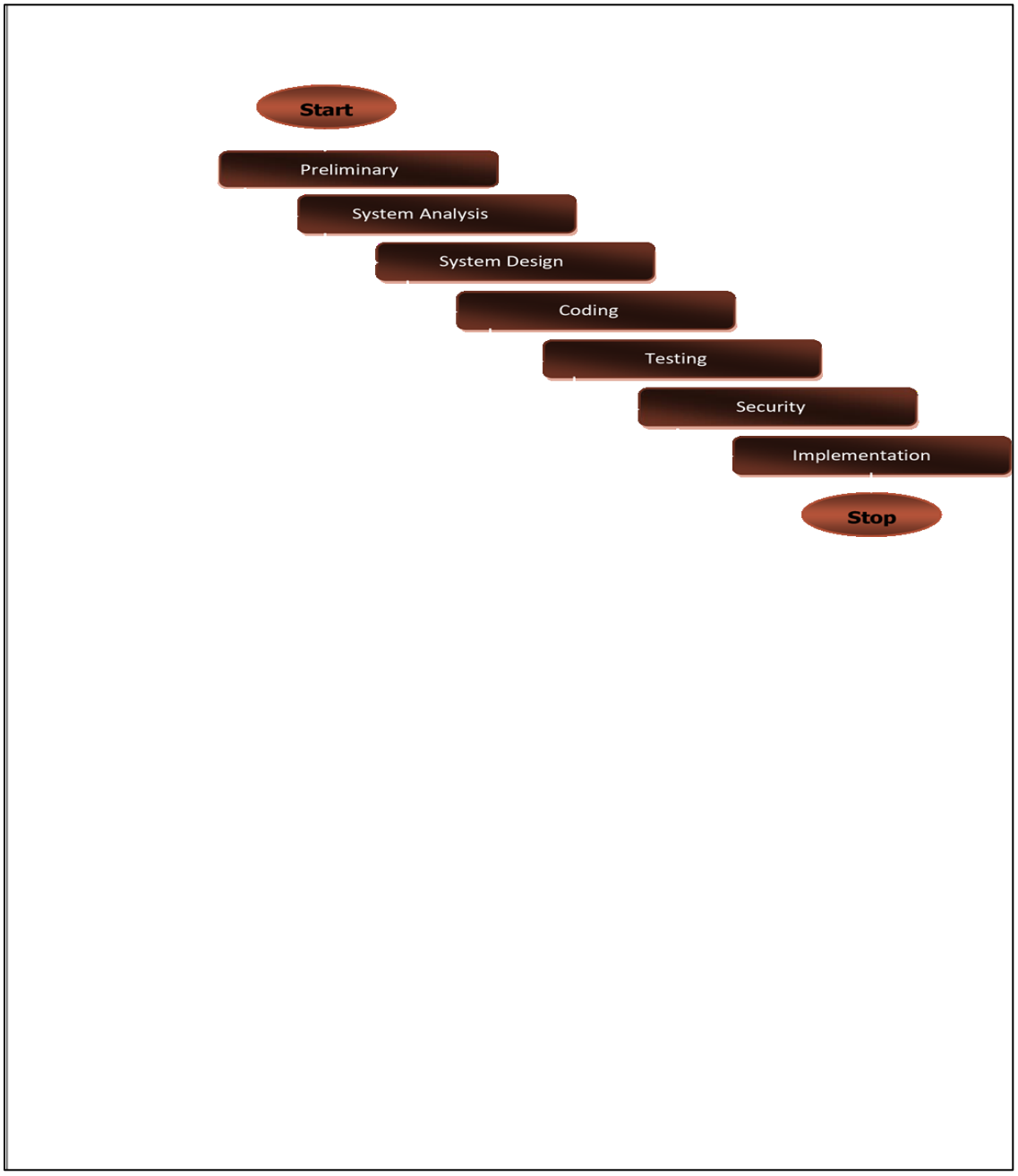
- **Operational Feasibility:** The system is user-friendly and reduces the challenges of manual appointment booking, making it easy for patients and doctors to adopt.
- **Technical Feasibility:** The website can be developed using widely available technologies such as These technologies are cost-effective and reliable.
- **Economic Feasibility:** Since the system reduces administrative costs and improves efficiency, the overall implementation is cost-effective. Development costs are minimal compared to the long-term benefits.
- **Time Feasibility:** The project can be developed and implemented within a reasonable timeframe, ensuring quick deployment and usage.

3.4 Project Planning

Project planning for apointify involves defining tasks, timelines, and resources to ensure smooth development and deployment. The process begins with requirement analysis to understand patient and doctor needs, followed by system design for architecture and database planning.. A timeline with milestones such as design completion, coding, integration, and testing will be prepared. Risks like technical issues and data security will be identified and minimized. Resources such as hardware, software, and skilled manpower will be allocated effectively. With proper planning, apointify aims to deliver a user-friendly, secure, and efficient doctor appointment platform

Phases Covered in the Plan

1.**Preliminary Investigation** – Understanding the project scope and objectives.
2.System Analysis – Identifying challenges, gathering requirements, and defining solutions.
3.System Design – Structuring modules, database design, and UI development.
4.Coding – Developing the portal using Python (Django) and integrating sentiment analysis.
5.Security – Implementing authentication, data encryption, and user privacy measures.
6.Testing – Performing unit testing, integration testing, and user acceptance testing.
7.Implementation – Deploying the final system and ensuring smooth operation.


Start
Preliminary
System Analysis
System Design
Coding
Testing
Security
Implementation
Stop

3.5 Software Requirement Specification (SRS)

The Software Requirement Specification (SRS) outlines the fundamental requirements of the *Hey Doc* Doctor Appointment Website to ensure efficient functionality, usability, and maintainability.

System Overview

The *Hey Doc* platform is designed to simplify and digitalize the doctor appointment process, providing seamless interaction between patients and healthcare providers. The system is structured into three main modules:

1. **Patient Module** – Allows patients to register, search doctors by specialization, view availability, book appointments, and manage their medical history.
2. **Doctor Module** – Provides tools for doctors to manage their schedules, approve or reject appointments, and access patient details.
3. **Admin Module** – Enables administrators to manage doctors, patients, appointments, and system-wide records.

Software Requirements

- **Frontend:** React.js, HTML, CSS, Bootstrap for responsive UI
- **Backend:** Node.js with Express.js for server-side handling
- **Database:** MySQL for storing patient, doctor, and appointment records
- **Authentication:** JSON Web Tokens (JWT) for secure login and session management
- **Web Server:** Apache or Nginx for hosting

Hardware Requirements

- **Processor:** Intel i5 or higher
- **RAM:** Minimum 8GB
- **Storage:** At least 100GB for database and records
- **Connectivity:** Reliable internet access for real-time update

3.6 Functional Requirements

1. Patient Module

Patients can:

- Register and log in securely.
- Search doctors based on specialization, location, and availability.
- Book, reschedule, or cancel appointments.
- View appointment history and medical records.

2. Doctor Module

Doctors can:

- Securely log in and manage their profiles.
- Accept, reject, or reschedule patient appointments.
- View and manage patient details for upcoming visits.
- Maintain digital records of consultations.

3. Admin Module

Admins have full control and can:

- Manage user accounts (patients and doctors).
- Monitor and manage all appointments.
- Add, edit, or remove doctor profiles.
- Ensure secure storage and integrity of patient data.

3.7 Software Engineering Paradigm

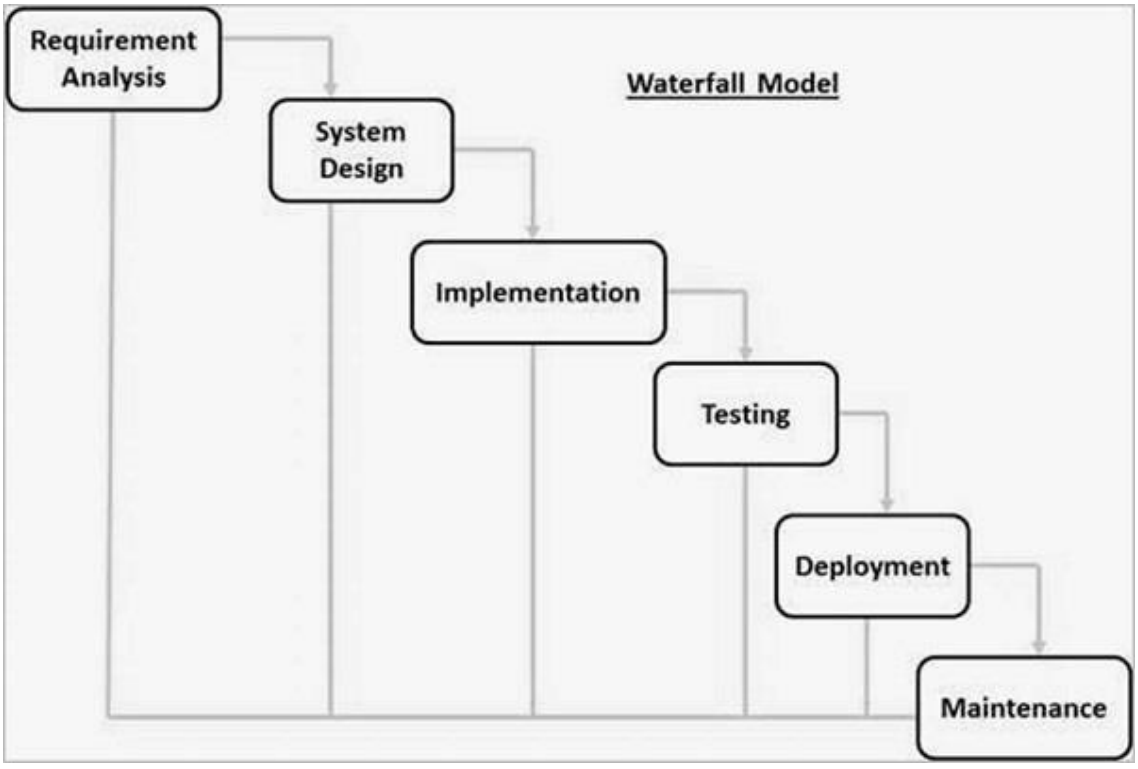
The development of the *Hey Doc* Doctor Appointment Website follows a structured approach to ensure reliability, efficiency, and maintainability. The chosen paradigm helps in defining clear stages while allowing iterative improvements.

Development model: Adapted Waterfall Model

The project follows the Waterfall Model with iterative feedback loops, ensuring smooth progression between phases.

Key Adaptations in the Waterfall Model:

1. **Structured Phase Progression** – Each phase (requirements, design, development, testing, implementation) follows a defined sequence.
2. **Iterative Refinements** – Feedback from testing allows improvements in coding and design.
3. **Defined Milestones** – Every stage is marked with clear deliverables before moving forward.
4. **Flexible Adjustments** – Overlapping is allowed when needed to improve efficiency.



Phases of Development – *Apointify*

1. **Requirement Analysis & System Study**
    - Identifying project goals, challenges, and functional specifications.
    - Gathering requirements from patients, doctors, and administrators to define core functionalities.
2. **System Design**
    - Structuring the database in MySQL for patient, doctor, and appointment records.
    - Designing the architecture with Node.js and Express.js for backend and React.js for frontend.
    - Creating user-friendly interfaces for patients, doctors, and admins.
3. **Implementation (Coding)**
    - Backend development using Node.js and Express.js for APIs and business logic.
    - Frontend development using React.js, HTML, CSS, and Bootstrap for responsive design.
    - Database integration with MySQL for secure data storage and retrieval.
    - Authentication and session handling using JSON Web Tokens (JWT).
4. **Testing & Debugging**
    - Performing unit testing, integration testing, and usability checks.
    - Debugging backend APIs and frontend modules to improve performance and reliability.
5. **Deployment & Maintenance**
    - Hosting the website on a scalable environment such as AWS, Heroku, or Nginx server.
    - Ensuring security, regular updates, and continuous feature enhancements.

DATA FLOW DIAGRAM (DFD)

Data Flow Diagram (DFD)

DFD is the abbreviation for Data Flow Diagram. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. It is a graphical tool, useful for communicating with users, managers and other personnel. It is useful for analysing existing as well as proposed system. DFDs typically composed of four main elements:

1. External entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system.

2.Process: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as "Submit payment."

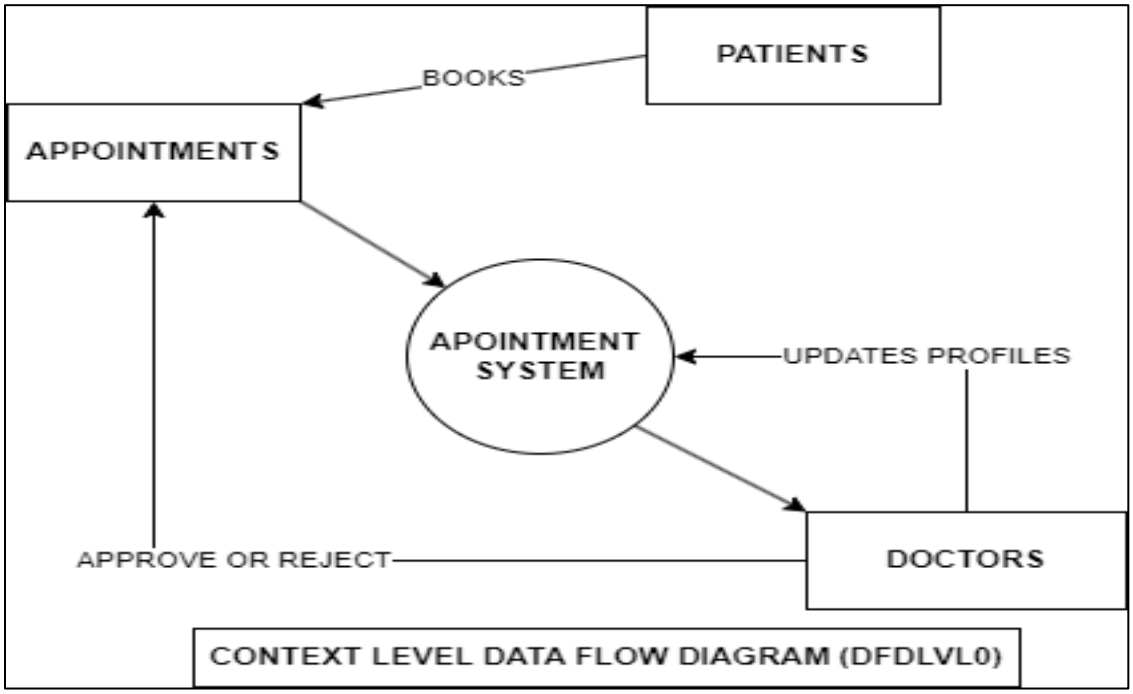
3. Data store: files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as "Orders."

4. Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labelled with a short data name, like "Billing details."

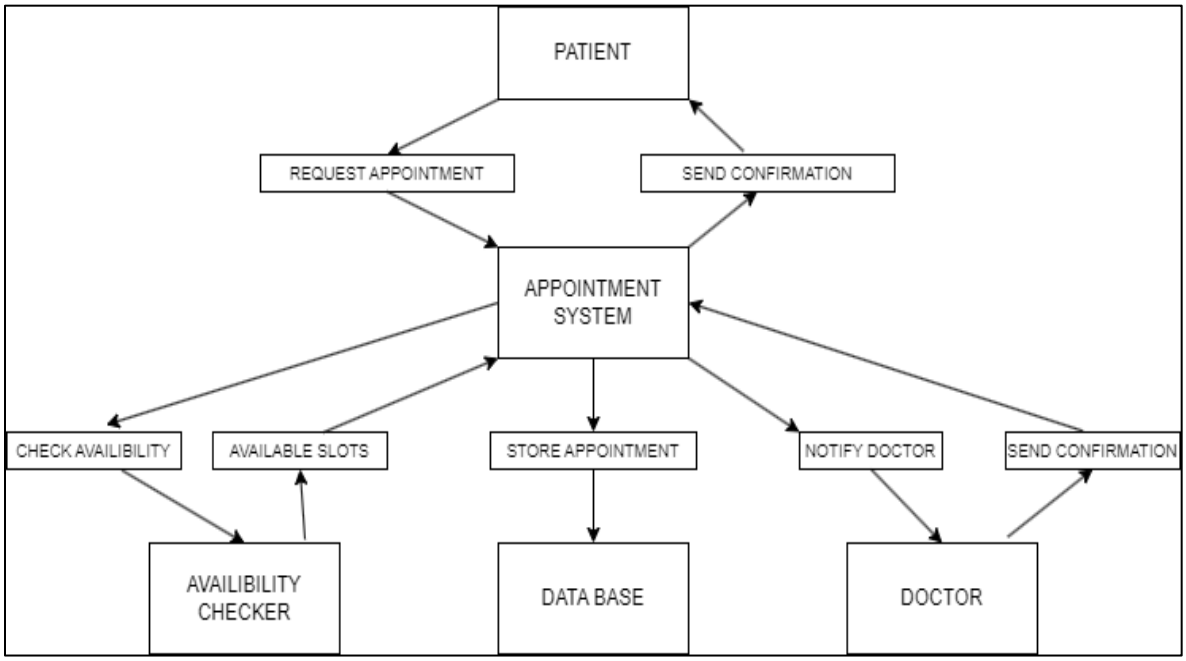
There are three types of Data Flow Diagrams (DFD):

- Context Level DFD (Level 0): This high-level diagram provides an overview of the entire system, showing the external entities interacting with the system and the main processes.
- Level 1 DFD: Level 1 DFDs break down the system into major processes or subsystems, offering a more detailed view of the main processes and their interactions.
- Level 2 DFD: These diagrams further decompose the processes identified in Level 1 DFDs into sub-processes, providing a more detailed look at how data flows within the system.

- **CONTEXT LEVEL DATA FLOW DIAGAM:**

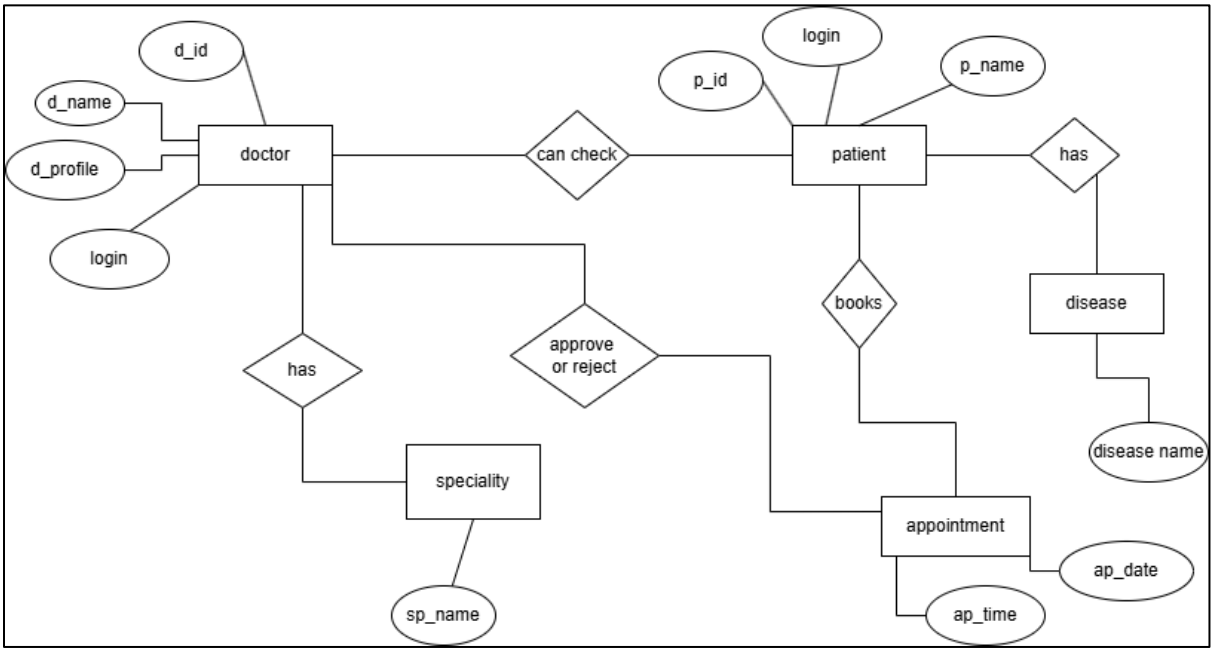


- **FIRST LEVEL DATA FLOW DIAGRAM:**



ER DIAGRAM

The ER diagram of *Apointify* represents the relationship between patients, doctors, admins, and appointments in the system. It ensures proper data flow and structured management of records in the database.



4.System design

4.1 Modules

1. Patient Module

Patients can:

- Register and log in securely.
- Search for doctors based on specialization, location, and availability.
- Book, reschedule, or cancel appointments.
- View appointment history and maintain personal medical records.

2. Doctor Module

Doctors can:

- Securely log in and manage their profiles.
- Accept, reject, or reschedule patient appointments.
- View patient details and maintain digital consultation notes.
- Manage availability, schedules, and consultation timings.

3. Admin Module

Admins have full control over the platform and can:

- Manage patient and doctor accounts.
- Monitor and update all appointments in the system.
- Add, edit, or remove doctor profiles.
- Ensure secure storage of data and maintain database integrity.
- Generate system-wide reports and analytics.

4.2 Data Structure of All Modules

The *Apointify* system uses a structured **MySQL database** for storing and managing all records. The database consists of well-defined tables for patients, doctors, admins, and appointments. Each table includes fields for unique identifiers, attributes, and relationships to ensure smooth data flow. The database allows both direct and sequential access to records by authenticated users.

Customized Tables Details

DESCRIBE DOCTORS

This table stores the doctor's information and describes it.

```
mysql> DESCRIBE doctors;
+----------------+--------------+------+-----+-------------------+-----------------------------------------------+
| Field          | Type         | Null | Key | Default           | Extra                                         |
+----------------+--------------+------+-----+-------------------+-----------------------------------------------+
| id             | int          | NO   | PRI | NULL              | auto_increment                                |
| name           | varchar(100) | NO   |     | NULL              |                                               |
| specialization | varchar(100) | NO   |     | NULL              |                                               |
| email          | varchar(150) | NO   | UNI | NULL              |                                               |
| createdAt      | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED                             |
| updatedAt      | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
+----------------+--------------+------+-----+-------------------+-----------------------------------------------+
6 rows in set (0.01 sec)

mysql>
```

DESCRIBE APPOINTMENTS

This table tell about the apppointments also mentioned many things in it.

```
mysql> DESCRIBE appointments;
+-----------+-----------------------------+------+-----+-------------------+-----------------------------------------------+
| Field     | Type                        | Null | Key | Default           | Extra                                         |
+-----------+-----------------------------+------+-----+-------------------+-----------------------------------------------+
| id        | int                         | NO   | PRI | NULL              | auto_increment                                |
| date      | date                        | NO   |     | NULL              |                                               |
| time      | varchar(50)                 | NO   |     | NULL              |                                               |
| status    | enum('Pending','Completed') | YES  |     | Pending           |                                               |
| userId    | int                         | YES  | MUL | NULL              |                                               |
| doctorId  | int                         | YES  | MUL | NULL              |                                               |
| createdAt | timestamp                   | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED                             |
| updatedAt | timestamp                   | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
+-----------+-----------------------------+------+-----+-------------------+-----------------------------------------------+
8 rows in set (0.00 sec)

mysql>
```

DESCRIBE NOTIFICATION

This table shows the notification

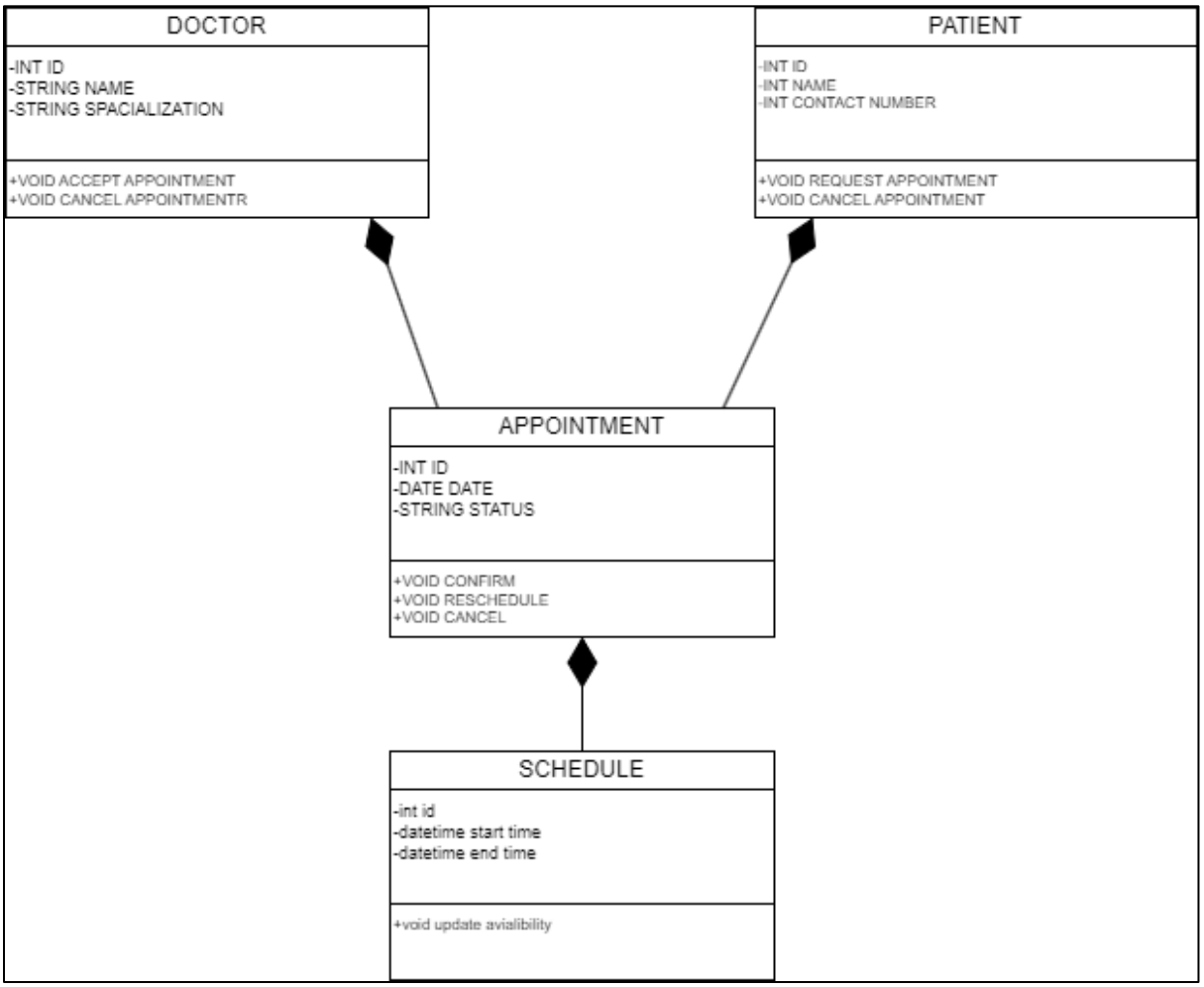
```
mysql> DESCRIBE notifications;
+-----------+-----------+------+-----+-------------------+-----------------------------------------------+
| Field     | Type      | Null | Key | Default           | Extra                                         |
+-----------+-----------+------+-----+-------------------+-----------------------------------------------+
| id        | int       | NO   | PRI | NULL              | auto_increment                                |
| content   | text      | NO   |     | NULL              |                                               |
| userId    | int       | YES  | MUL | NULL              |                                               |
| createdAt | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED                             |
| updatedAt | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
+-----------+-----------+------+-----+-------------------+-----------------------------------------------+
5 rows in set (0.01 sec)

mysql>
```

DESCRIBE USERS

This table shows how many user are there

```
mysql> DESCRIBE users;
+-----------+--------------+------+-----+-------------------+-----------------------------------------------+
| Field     | Type         | Null | Key | Default           | Extra                                         |
+-----------+--------------+------+-----+-------------------+-----------------------------------------------+
| id        | int          | NO   | PRI | NULL              | auto_increment                                |
| firstname | varchar(100) | NO   |     | NULL              |                                               |
| lastname  | varchar(100) | NO   |     | NULL              |                                               |
| email     | varchar(150) | NO   | UNI | NULL              |                                               |
| password  | varchar(255) | NO   |     | NULL              |                                               |
| createdAt | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED                             |
| updatedAt | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
| role      | varchar(50)  | YES  |     | patient           |                                               |
| age       | int          | YES  |     | NULL              |                                               |
| gender    | varchar(20)  | YES  |     | NULL              |                                               |
| mobile    | varchar(15)  | YES  |     | NULL              |                                               |
| address   | varchar(255) | YES  |     | NULL              |                                               |
| status    | varchar(20)  | YES  |     | active            |                                               |
| pic       | varchar(255) | YES  |     | NULL              |                                               |
+-----------+--------------+------+-----+-------------------+-----------------------------------------------+
14 rows in set (0.00 sec)
```

Relationship between table (Class Diagram)

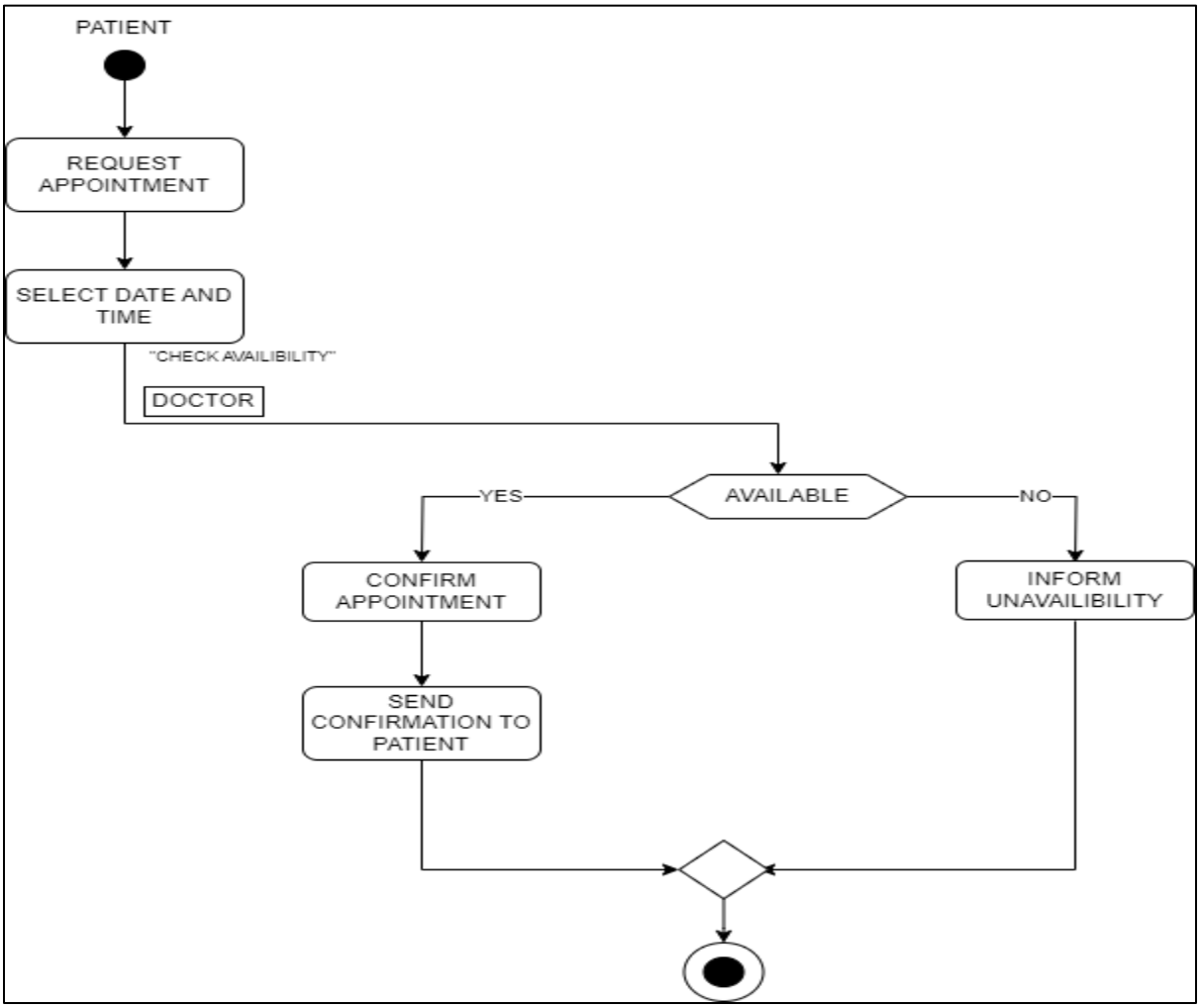

DOCTOR
-INT ID
-STRING NAME
-STRING SPACIALIZATION
+VOID ACCEPT APPOINTMENT
+VOID CANCEL APPOINTMENTR
PATIENT
-INT ID
-INT NAME
-INT CONTACT NUMBER
+VOID REQUEST APPOINTMENT
+VOID CANCEL APPOINTMENT
APPOINTMENT
-INT ID
-DATE DATE
-STRING STATUS
+VOID CONFIRM
+VOID RESCHEDULE
+VOID CANCEL
SCHEDULE
-int id
-datetime start time
-datetime end time
+void update avialibility

4.3 PROCEDURAL DESIGN:

Process logic (flowchart) of each module

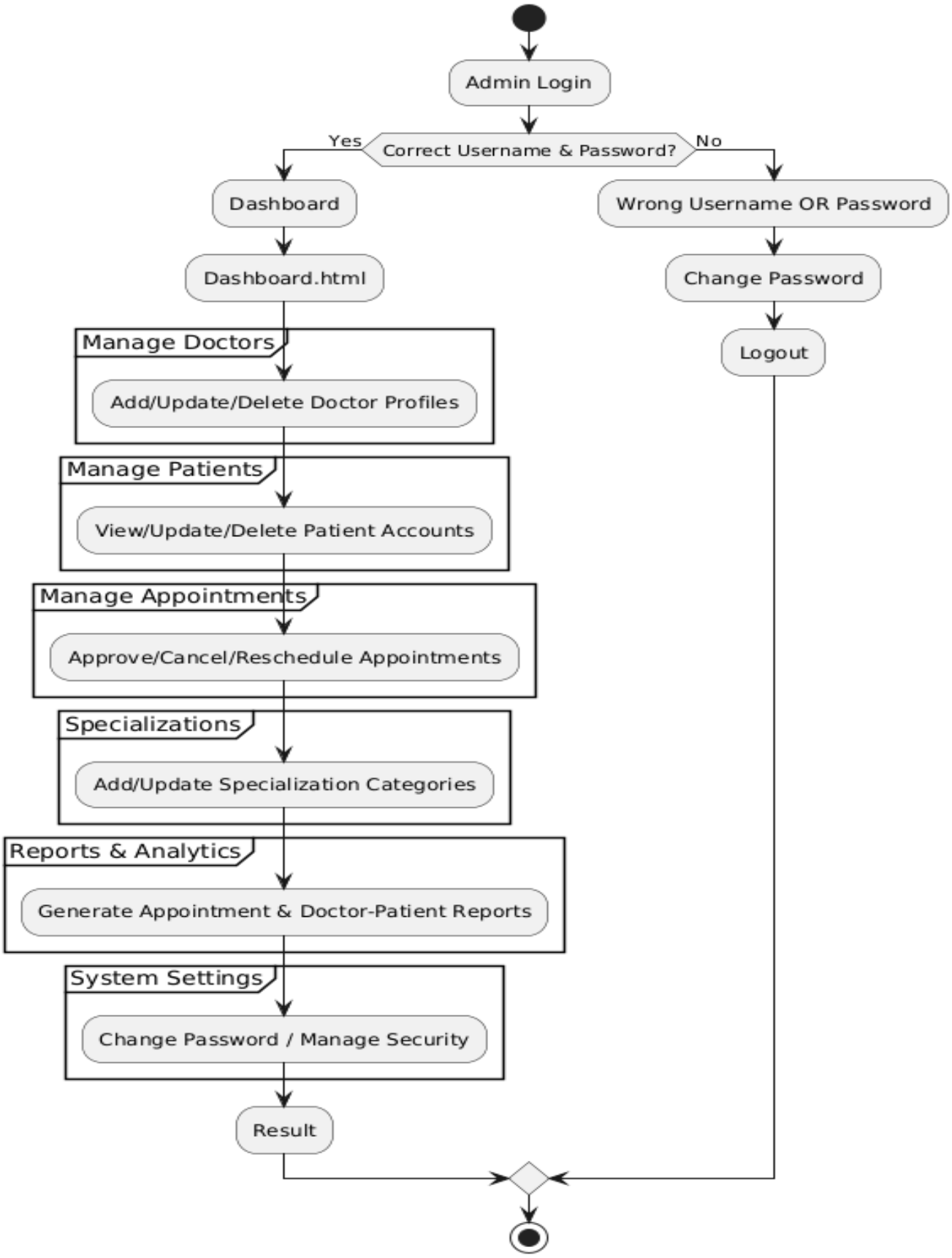
4.3.1 Patient Panel Design

In the patient panel design, the tasks for patients have been implemented to provide easy appointment booking services. On the index page, patients can register or log in to their account. After successful login, they can search for doctors based on specialization, availability, or location. Once a suitable doctor is found, the patient can book, reschedule, or cancel an appointment. Patients can also view their medical history and upcoming consultations. By selecting the desired option, patients are redirected to the respective page where they receive complete details and services. The design of the patient panel is shown in the following flowchart



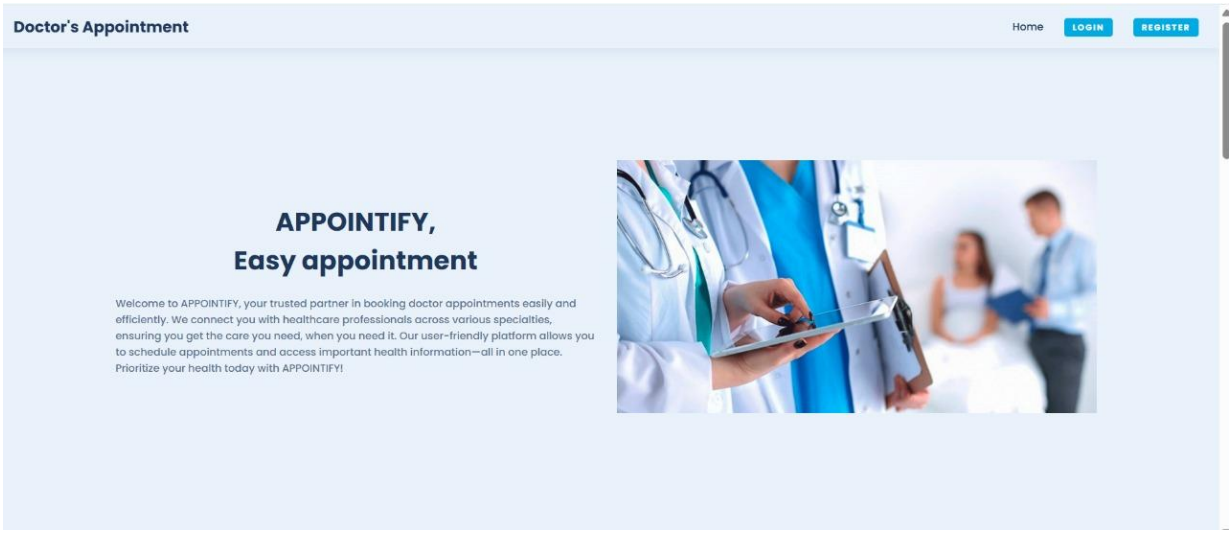
4.2.1 Admin Panel Design

We have design user login facility to manage and update all of the information. It is fully secured page. Without appropriate username and password it cannot be accessed by anyone. For admin login after giving username and password we need to click a login button , when we click login button it is not directly entered in home page , it stay in login page. Then it starts a session and set two variables called username and password. If the username and password are matched with database, it can enter in home page. It is not possible without click login button. In case if username or password are not matched with database than Invalid username or password massage is shown. We can describe the login facility in admin login by using below flow chart given below

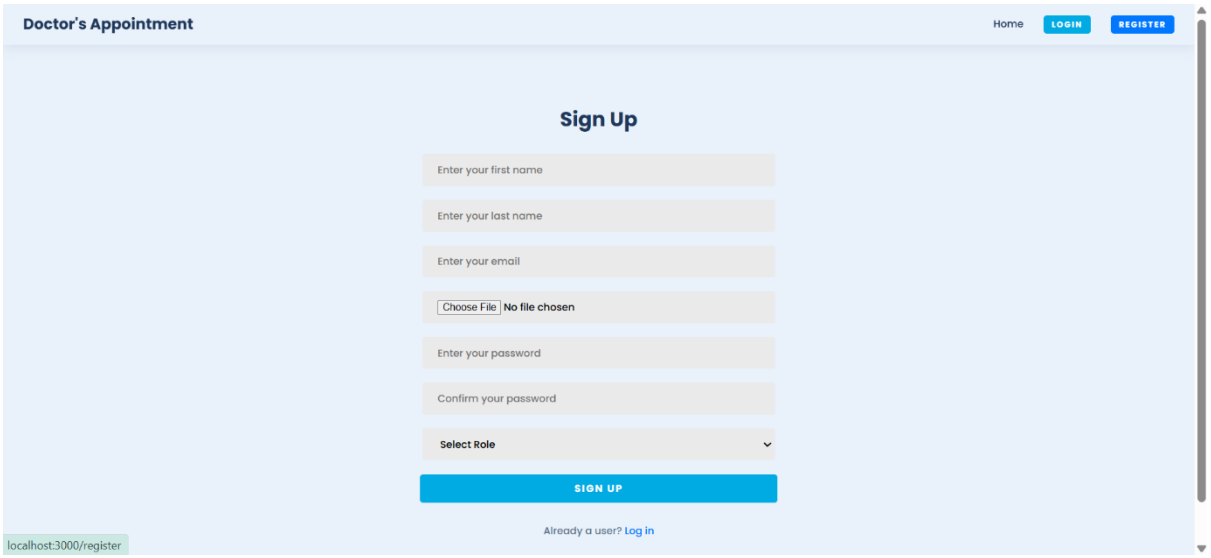

Admin Login
Correct Username & Password?
Yes
No
Dashboard
Dashboard.html
Wrong Username OR Password
Change Password
Logout
Manage Doctors
Add/Update/Delete Doctor Profiles
Manage Patients
View/Update/Delete Patient Accounts
Manage Appointments
Approve/Cancel/Reschedule Appointments
Specializations
Add/Update Specialization Categories
Reports & Analytics
Generate Appointment & Doctor-Patient Reports
System Settings
Change Password / Manage Security
Result

Screenshots

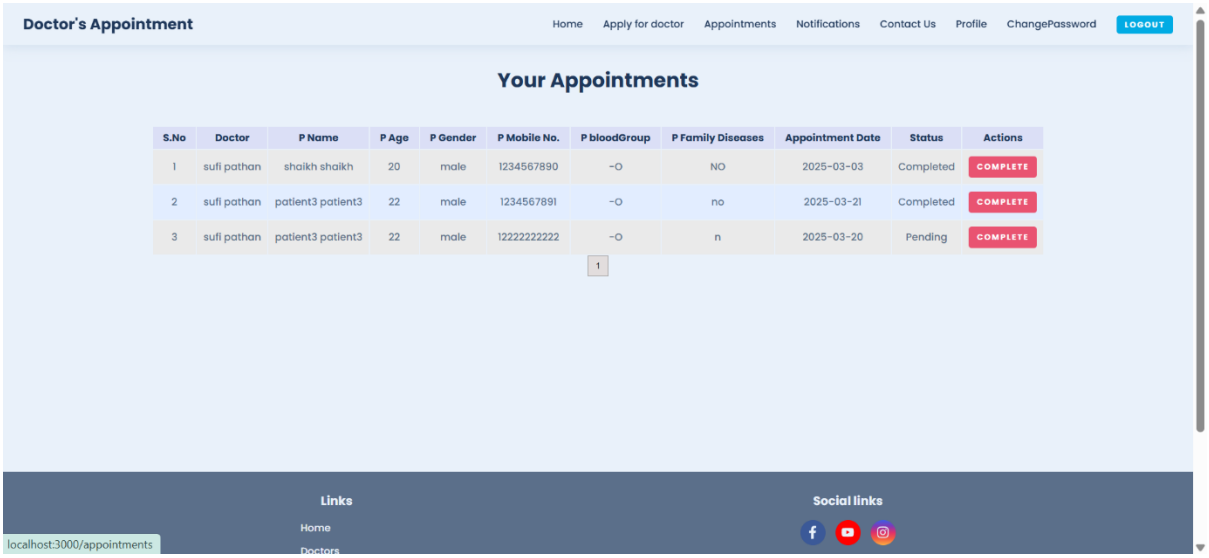
HOME PAGE :



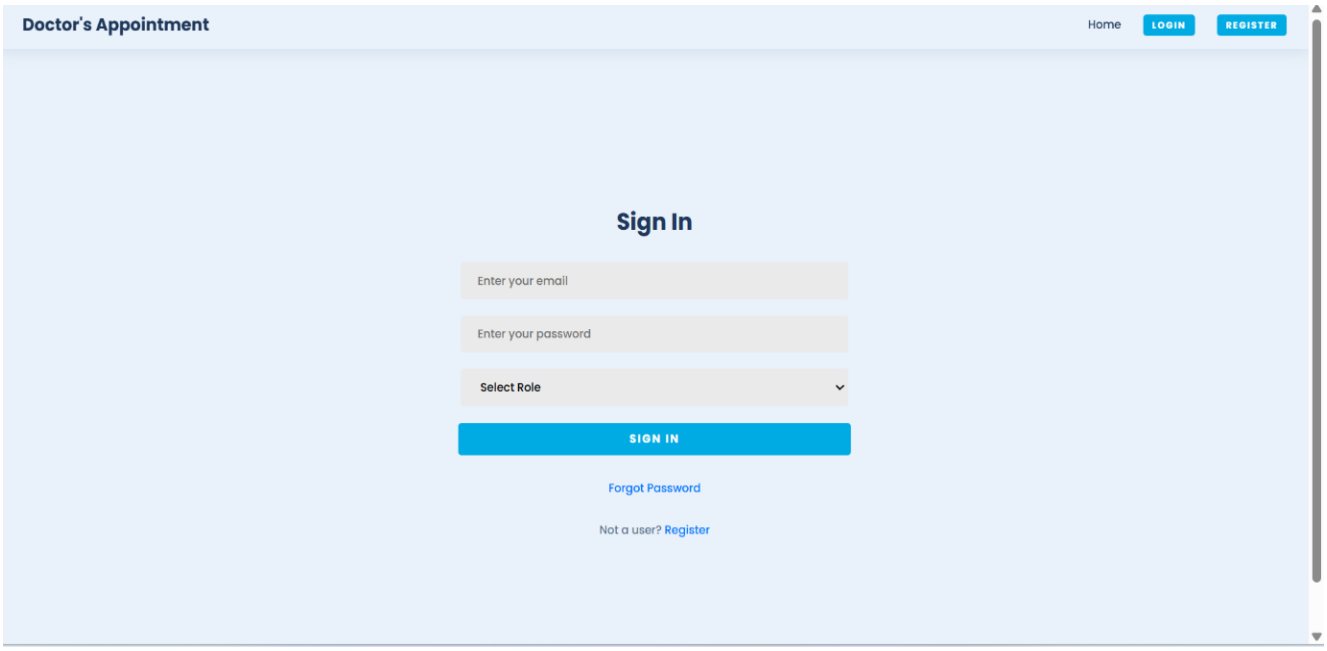
SIGN UP PAGE:



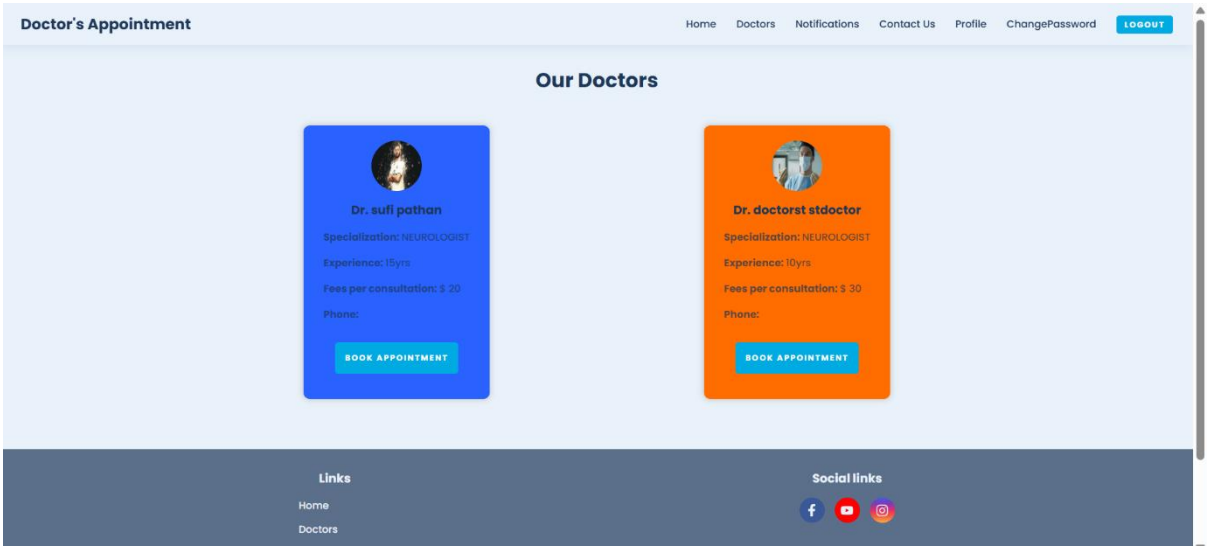
DOCTOR'S DASHBOARD:



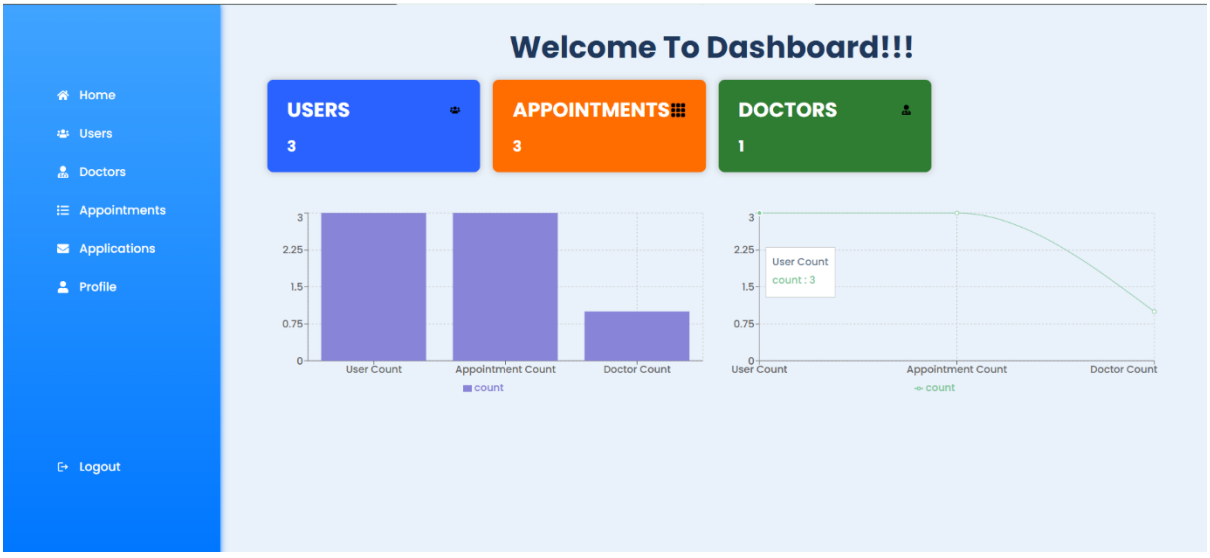
LOGIN PAGE"



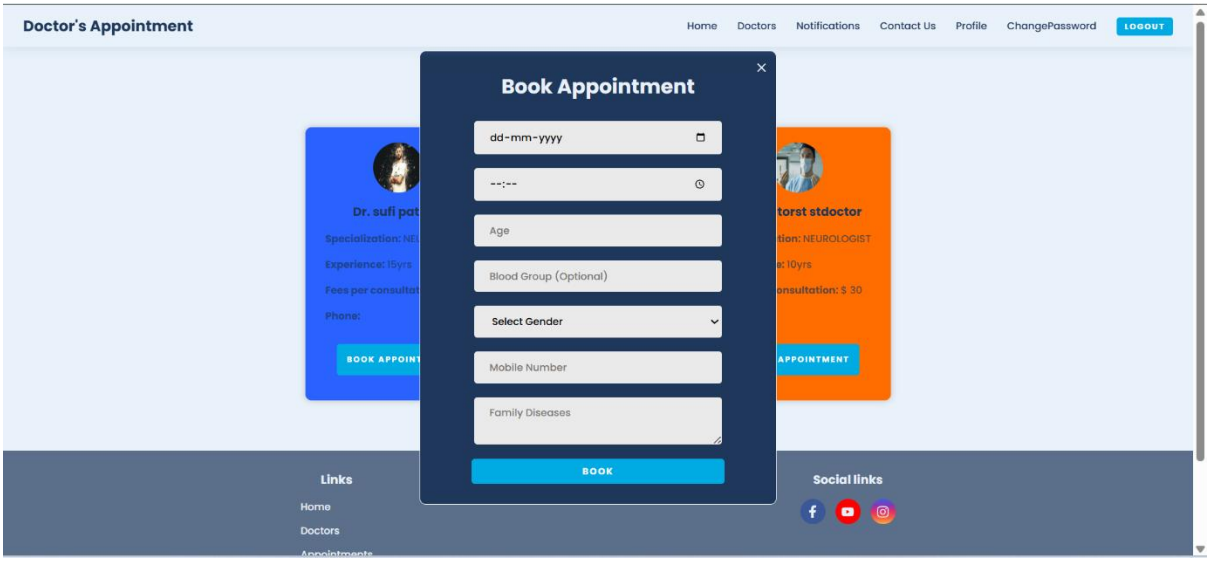
DOCTOR SELECTION PAGE:



ADMIN'S DASHBOARD:



APPOINTMENT PAGE:



Testing

 Unit Testing – Each module like appointment booking, doctor profiles, and patient login is tested individually. This ensures every small function works correctly on its own.

 Integration Testing – Combines modules such as doctor availability and appointment scheduling. It verifies smooth interaction between connected features.

 System Testing – Tests the entire *Apointify* website as a whole. Ensures all functionalities like registration, booking, and notifications meet requirements.

 Acceptance Testing – Validates that the website fulfills the expectations of patients, doctors, and admins. Confirms it is ready for real-world use.

 Performance Testing – Checks how the system behaves under heavy load, like multiple users booking appointments simultaneously. Ensures speed and stability.

 Security Testing – Ensures that patient data, medical records, and login credentials are safe. Prevents unauthorized access or data leaks.

 Usability Testing – Evaluates the ease of use for patients and doctors. Makes sure the interface is simple, clear, and user-friendly.

Coding

AboutUs.jsx

```
import React from "react";
import image from "../images/aboutimg.jpg";


const AboutUs = () => {
  return (
    <>
      <section className="container">
        <h2 className="page-heading about-heading">About Us</h2>
        <div className="about">
          <div className="hero-img">
            <img
              src={image}
              alt="hero"
            />
          </div>
          <div className="hero-content">
            <p>
          Welcome to Hey Doc, your go-to platform for selecting the right doctor and booking appointments at your convenience.

          Our website allows you to browse through a wide range of healthcare professionals across various specialties,
```

helping you find the perfect match for your medical needs. With a simple registration process, you can create

a secure patient profile using your unique ID and password, making it easy to manage appointments, track your

health history, and stay connected with your healthcare provider. At Hey Doc,

we’re committed to making healthcare more accessible, efficient, and personalized, so you can focus on what matters most—your health.

```
          </p>
        </div>
      </div>
    </section>
  </>
 );
};
import React from "react";
import image from "../images/aboutimg.jpg";

const AboutUs = () => {
 return (
  <>
    <section className="container">
      <h2 className="page-heading about-heading">About Us</h2>
      <div className="about">
        <div className="hero-img">
```

```
          <img
            src={image}
            alt="hero"
          />
        </div>
        <div className="hero-content">
          <p>
          Welcome to Hey Doc, your go-to platform for selecting the right doctor and booking appointments at your convenience.

          Our website allows you to browse through a wide range of healthcare professionals across various specialties,

          helping you find the perfect match for your medical needs. With a simple registration process, you can create

          a secure patient profile using your unique ID and password, making it easy to manage appointments, track your

          health history, and stay connected with your healthcare provider. At Hey Doc,

          we’re committed to making healthcare more accessible, efficient, and personalized, so you can focus on what matters most—your health.
          </p>
        </div>
      </div>
    </section>
  </>
 );
};
```

```
export default AboutUs;
```

```
export default AboutUs;
```

AdminApplication.jsx

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import toast from "react-hot-toast";
import Loading from "./Loading";
import { setLoading } from "../redux/reducers/rootSlice";
import { useDispatch, useSelector } from "react-redux";
import Empty from "./Empty";
import fetchData from "../helper/apiCall";
import "../styles/user.css";

axios.defaults.baseURL = process.env.REACT_APP_SERVER_DOMAIN;

const AdminApplications = () => {
  const [applications, setApplications] = useState([]);
  const dispatch = useDispatch();
  const { loading } = useSelector((state) => state.root);
```

```
const getAllApp = async (e) => {
  try {
    dispatch(setLoading(true));
    const temp = await fetchData(`/doctor/getnotdoctors`);
    setApplications(temp);
    dispatch(setLoading(false));
  } catch (error) {}
};

const acceptUser = async (userId) => {
  try {
    const confirm = window.confirm("Are you sure you want to accept?");
    if (confirm) {
      await toast.promise(
        axios.put(
          "/doctor/acceptdoctor",
          { id: userId },
          {
            headers: {
              authorization: `Bearer ${localStorage.getItem("token")}`,
            },
            data: { userId },
```

```
        }
      ),
      {
        success: "Application accepted",
        error: "Unable to accept application",
        loading: "Accepting application...",
      }
    );
    getAllApp();
   }
  } catch (error) {
   return error;
  }
};

const deleteUser = async (userId) => {
  try {
   const confirm = window.confirm("Are you sure you want to delete?");
   if (confirm) {
    await toast.promise(
      axios.put(
        "/doctor/rejectdoctor",
        { id: userId },
```

```
        {
          headers: {
            authorization: `Bearer ${localStorage.getItem("token")}`,
          },
          data: { userId },
        }
      ),
      {
        success: "Application rejected",
        error: "Unable to reject application",
        loading: "Rejecting application...",
      }
    );
    getAllApp();
   }
  } catch (error) {
   return error;
  }
};

useEffect(() => {
  getAllApp();
}, []);
```

```
return (
  <>
    {loading ? (
      <Loading />
    ) : (
      <section className="user-section">
        <h3 className="home-sub-heading">All Applications</h3>
        {applications.length > 0 ? (
          <div className="user-container">
            <table>
              <thead>
                <tr>
                  <th>S.No</th>
                  <th>Pic</th>
                  <th>First Name</th>
                  <th>Last Name</th>
                  <th>Email</th>
                  <th>Mobile No.</th>
                  <th>Experience</th>
                  <th>Specialization</th>
                  <th>Fees</th>
                  <th>Action</th>
```

```
  </tr>
</thead>
<tbody>
  {applications?.map((ele, i) => {
    return (
      <tr key={ele?._id}>
        <td>{i + 1}</td>
        <td>
          <img
            className="user-table-pic"
            src={
              ele?.userId?.pic ||
              "https://icon-library.com/images/anonymous-avatar-icon/anonymous-avatar-icon-25.jpg"
            }
            alt={ele?.userId?.firstname}
          />
        </td>
        <td>{ele?.userId?.firstname}</td>
        <td>{ele?.userId?.lastname}</td>
        <td>{ele?.userId?.email}</td>
        <td>{ele?.userId?.mobile}</td>
        <td>{ele?.experience}</td>
        <td>{ele?.specialization}</td>
```

```
        <td>{ele?.fees}</td>
        <td className="select">
         <button
           className="btn user-btn accept-btn"
           onClick={() => {
             acceptUser(ele?.userId?._id);
           }}
         >
           Accept
         </button>
         <button
           className="btn user-btn"
           onClick={() => {
             deleteUser(ele?.userId?._id);
           }}
         >
           Reject
         </button>
        </td>
      </tr>
    );
  })}
</tbody>
```

```
            </table>
          </div>
        ) : (
          <Empty />
        )}
      </section>
    )}
  </>
 );
};

export default AdminApplications;
```

Users.jsx

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import toast from "react-hot-toast";
import Loading from "./Loading";
import { setLoading } from "../redux/reducers/rootSlice";
import { useDispatch, useSelector } from "react-redux";
import Empty from "./Empty";
import fetchData from "../helper/apiCall";
```

```
axios.defaults.baseURL = process.env.REACT_APP_SERVER_DOMAIN;


const Users = () => {
  const [users, setUsers] = useState([]);
  const dispatch = useDispatch();
  const [filter, setFilter] = useState("all");
  const [searchTerm, setSearchTerm] = useState("");
  const { loading } = useSelector((state) => state.root);


  const getAllUsers = async () => {
    try {
      dispatch(setLoading(true));
      let url = "/user/getallusers";
      if (filter !== "all") {
        url += `?filter=${filter}`;
      }
      if (searchTerm.trim() !== "") {
        url += `${filter !== "all" ? "&" : "?"}search=${searchTerm}`;
      }
      const temp = await fetchData(url);
      setUsers(temp);
      dispatch(setLoading(false));
    } catch (error) {}
```

```
};

const deleteUser = async (userId) => {
  try {
    const confirm = window.confirm("Are you sure you want to delete?");
    if (confirm) {
      await toast.promise(
        axios.delete("/user/deleteuser", {
          headers: {
            authorization: `Bearer ${localStorage.getItem("token")}`,
          },
          data: { userId },
        }),
        {
          pending: "Deleting in...",
          success: "User deleted successfully",
          error: "Unable to delete user",
          loading: "Deleting user...",
        }
      );
      getAllUsers();
    }
  } catch (error) {
```

```
    return error;
  }
};

useEffect(() => {
  getAllUsers();
}, []);

const filteredUsers = users.filter((doc) => {
  if (filter === "all") {
    return true;
  } else if (filter === "firstname") {
    return doc.firstname.toLowerCase().includes(searchTerm.toLowerCase());
  } else {
    return true;
  }
});

return (
  <>
    {loading ? (
      <Loading />
    ) : (
```

```
<section className="user-section">
  <div className="ayx">
    <div className="filter">
      <label htmlFor="filter">Filter by:</label>
      <select
        id="filter"
        value={filter}
        onChange={(e) => setFilter(e.target.value)}
      >
        <option value="all">All</option>
        <option value="firstname">Name</option>

      </select>
    </div>

    <div className="search">
      <label htmlFor="search">Search:</label>
      <input
        type="text"
        className="form-input"
        id="search"
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
```

```
      placeholder="Search"
    />
  </div>
</div>
<h3 className="home-sub-heading">All Users</h3>
{users.length > 0 ? (
  <div className="user-container">
    <table>
      <thead>
        <tr>
          <th>S.No</th>
          <th>Pic</th>
          <th>First Name</th>
          <th>Last Name</th>
          <th>Email</th>
          <th>Mobile No.</th>
          <th>Age</th>
          <th>Gender</th>
          <th>Is Doctor</th>
          <th>Remove</th>
        </tr>
      </thead>
      <tbody>
```

```
{filteredUsers.map((ele, i) => (
 <tr key={ele._id}>
  <td>{i + 1}</td>
  <td>
   <img
    className="user-table-pic"
    src={ele.pic}
    alt={ele.firstname}
   />
  </td>
  <td>{ele.firstname}</td>
  <td>{ele.lastname}</td>
  <td>{ele.email}</td>
  <td>{ele.mobile}</td>
  <td>{ele.age}</td>
  <td>{ele.gender}</td>
  <td>{ele.isDoctor ? "Yes" : "No"}</td>
  <td className="select">
   <button
    className="btn user-btn"
    onClick={() => deleteUser(ele._id)}
   >
    Remove
```

```
                    </button>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      ) : (
        <Empty />
      )}
    </section>
  )}
 </>
 );
};

export default Users;
```

conn.js

```
const { MongoClient } = require("mongodb");
const mongoose = require("mongoose");
mongoose.set("strictQuery", false);
require("dotenv").config();
```

```
const client = mongoose
  .connect("mongodb://127.0.0.1:27017/Doctor-Appointment", {
    useNewUrlParser: true,
  })
  .then(() => {
    console.log("DB connected");
  })
  .catch((error) => {
    console.log("Error: ", error);

    return error;
  });

module.exports = client;
```

BookAppointment.jsx

```
import React, { useState } from "react";
import "../styles/bookappointment.css";
import axios from "axios";
import toast from "react-hot-toast";
import { IoMdClose } from "react-icons/io";

const BookAppointment = ({ setModalOpen, ele }) => {
```

```
const [formDetails, setFormDetails] = useState({
  date: "",
  time: "",
  age:"",
  bloodGroup:"",
  gender:"",
  number: "",
  familyDiseases:"",
  // prescription:"",
});

const inputChange = (e) => {
  const { name, value } = e.target;
  return setFormDetails({
    ...formDetails,
    [name]: value,
  });
};

const bookAppointment = async (e) => {
  e.preventDefault();
  try {
    await toast.promise(
```

```
axios.post(
  "/appointment/bookappointment",
  {
    doctorId: ele?.userId?._id,
    date: formDetails.date,
    time: formDetails.time,
    age: formDetails.age,
    bloodGroup: formDetails.bloodGroup,
    gender: formDetails.gender,
    number:formDetails.number,
    familyDiseases: formDetails.familyDiseases,
    // prescription: formDetails.prescription,
    doctorname: `${ele?.userId?.firstname} ${ele?.userId?.lastname}`,
  },
  {
    headers: {
      Authorization: `Bearer ${localStorage.getItem("token")}`,
    },
  }
),
{
  success: "Appointment booked successfully",
  error: "Unable to book appointment",
```

```
        loading: "Booking appointment...",
      }
    );
    setModalOpen(false);
  } catch (error) {
    return error;
  }
};

return (
  <>
    <div className="modal flex-center">
      <div className="modal__content">
        <h2 className="page-heading">Book Appointment</h2>
        <IoMdClose
          onClick={() => {
            setModalOpen(false);
          }}
          className="close-btn"
        />
        <div className="register-container flex-center book">
          <form className="register-form">
            <input
```

```
    type="date"
    name="date"
    className="form-input"
    value={formDetails.date}
    onChange={inputChange}
   />
   <input
    type="time"
    name="time"
    className="form-input"
    value={formDetails.time}
    onChange={inputChange}
   />
  <input
   type="number"
   name="age"
   placeholder="Age"
   className="form-input"
   value={formDetails.age}
   onChange={inputChange}
   required
  />
  <input
```

```
  type="text"
  name="bloodGroup"
  placeholder="Blood Group (Optional)"
  className="form-input"
  value={formDetails.bloodGroup}
  onChange={inputChange}
/>
<select
  name="gender"
  className="form-input"
  value={formDetails.gender}
  onChange={inputChange}
  required
>

  <option value="">Select Gender</option>
  <option value="male">Male</option>
  <option value="female">Female</option>
  <option value="other">Other</option>
</select>
<input
  type="number"
  name="number"
```

```
  placeholder="Mobile Number"
  className="form-input"
  value={formDetails.number}
  onChange={inputChange}
  required
/>
<textarea
  name="familyDiseases"
  placeholder="Family Diseases"
  className="form-input"
  value={formDetails.familyDiseases}
  onChange={inputChange}
></textarea>
{/* <input
  type="file"
  name="prescription"
  accept="application/pdf"
  className="form-input"
  onChange={fileChange}
/> */}

  <button
    type="submit"
```

```
              className="btn form-btn"
              onClick={bookAppointment}
            >
              book
            </button>
          </form>
        </div>
      </div>
    </div>
  </>
 );
};

export default BookAppointment;
```

Aprofile.jsx

```
import React, { useEffect, useState } from "react";
import "../styles/profile.css";
import axios from "axios";
import toast from "react-hot-toast";
import { setLoading } from "../redux/reducers/rootSlice";
import { useDispatch, useSelector } from "react-redux";
import Loading from "./Loading";
```

```
import fetchData from "../helper/apiCall";
import jwt_decode from "jwt-decode";

axios.defaults.baseURL = process.env.REACT_APP_SERVER_DOMAIN;

function Aprofile() {
  const { userId } = jwt_decode(localStorage.getItem("token"));
  const dispatch = useDispatch();
  const { loading } = useSelector((state) => state.root);
  const [file, setFile] = useState("");
  const [formDetails, setFormDetails] = useState({
    firstname: "",
    lastname: "",
    email: "",
    age: "",
    mobile: "",
    gender: "neither",
    address: "",
    password: "",
    confpassword: "",
  });

  const getUser = async () => {
```

```
  try {
    dispatch(setLoading(true));
    const temp = await fetchData(`/user/getuser/${userId}`);
    setFormDetails({
      ...temp,
      password: "",
      confpassword: "",
      mobile: temp.mobile === null ? "" : temp.mobile,
      age: temp.age === null ? "" : temp.age,
    });
    setFile(temp.pic);
    dispatch(setLoading(false));
  } catch (error) {}
};

useEffect(() => {
  getUser();
}, [dispatch]);

const inputChange = (e) => {
  const { name, value } = e.target;
  return setFormDetails({
    ...formDetails,
```

```
    [name]: value,
  });
};

const formSubmit = async (e) => {
  try {
    e.preventDefault();
    const {
      firstname,
      lastname,
      email,
      age,
      mobile,
      address,
      gender,
      password,
      confpassword,
    } = formDetails;

    if (!email) {
      return toast.error("Email should not be empty");
    } else if (firstname.length < 3) {
      return toast.error("First name must be at least 3 characters long");
```

```
} else if (lastname.length < 3) {
  return toast.error("Last name must be at least 3 characters long");
} else if (password.length < 5) {
  return toast.error("Password must be at least 5 characters long");
} else if (password !== confpassword) {
  return toast.error("Passwords do not match");
}
await toast.promise(
  axios.put(
    "/user/updateprofile",
    {
      firstname,
      lastname,
      age,
      mobile,
      address,
      gender,
      email,
      password,
    },
    {
      headers: {
        authorization: `Bearer ${localStorage.getItem("token")}`,
```

```
        },
      }
    ),
    {
      pending: "Updating profile...",
      success: "Profile updated successfully",
      error: "Unable to update profile",
      loading: "Updating profile...",
    }
  );

  setFormDetails({ ...formDetails, password: "", confpassword: "" });
 } catch (error) {
  return toast.error("Unable to update profile");
 }
};

return (
 <>

  {loading ? (
   <Loading />
  ) : (
```

```
<section className="register-section flex-center">
  <div className="profile-container flex-center">
    <h2 className="form-heading">Profile</h2>
    <img
      src={file}
      alt="profile"
      className="profile-pic"
    />
    <form
      onSubmit={formSubmit}
      className="register-form"
    >
      <div className="form-same-row">
        <input
          type="text"
          name="firstname"
          className="form-input"
          placeholder="Enter your first name"
          value={formDetails.firstname}
          onChange={inputChange}
        />
        <input
          type="text"
```

```
    name="lastname"
    className="form-input"
    placeholder="Enter your last name"
    value={formDetails.lastname}
    onChange={inputChange}
   />
  </div>
  <div className="form-same-row">
   <input
    type="email"
    name="email"
    className="form-input"
    placeholder="Enter your email"
    value={formDetails.email}
    onChange={inputChange}
   />
   <select
    name="gender"
    value={formDetails.gender}
    className="form-input"
    id="gender"
    onChange={inputChange}
   >
```

```
    <option value="neither">Prefer not to say</option>
    <option value="male">Male</option>
    <option value="female">Female</option>
  </select>
</div>
<div className="form-same-row">
  <input
    type="text"
    name="age"
    className="form-input"
    placeholder="Enter your age"
    value={formDetails.age}
    onChange={inputChange}
  />
  <input
    type="text"
    name="mobile"
    className="form-input"
    placeholder="Enter your mobile number"
    value={formDetails?.mobile}
    onChange={inputChange}
  />
</div>
```

```
<textarea
  type="text"
  name="address"
  className="form-input"
  placeholder="Enter your address"
  value={formDetails.address}
  onChange={inputChange}
  rows="2"
></textarea>
<div className="form-same-row">
  <input
    type="password"
    name="password"
    className="form-input"
    placeholder="Enter your password"
    value={formDetails.password}
    onChange={inputChange}
  />
  <input
    type="password"
    name="confpassword"
    className="form-input"
    placeholder="Confirm your password"
```

```
              value={formDetails.confpassword}
              onChange={inputChange}
            />
          </div>
          <button
            type="submit"
            className="btn form-btn"
          >
            update
          </button>
        </form>
      </div>
    </section>
  )}

  </>
 );
}

export default Aprofile;
```

6.Future Scope

Future Scope of Apointify

1. Integration of **AI-based doctor recommendation** based on patient symptoms.
2. Implementation of **video consultation** for remote healthcare access.
3. Adding **multi-language support** for diverse users.
4. Introducing **mobile app versions** for Android & iOS.
5. Integration with **digital payment gateways** for appointment fees.
6. Providing **lab test booking** alongside doctor appointments.
7. Adding **medicine ordering & delivery** within the system.
8. **Wearable device integration** (smartwatch/fitness band) for real-time health monitoring.
9. Use of **machine learning** for predicting patient health risks.

Conclusion

The development of **Apointify** provides a reliable and efficient solution for managing doctor appointments online. It bridges the gap between patients and healthcare providers by offering a user-friendly platform for booking, scheduling, and managing appointments. The system reduces the hassle of long waiting times, improves time management for doctors, and ensures better accessibility for patients. By incorporating features like appointment reminders, patient records, and an easy-to-navigate interface, the platform enhances the overall healthcare experience.

In conclusion, Apointify not only streamlines the appointment process but also contributes to the **digital transformation of healthcare services**. With further enhancements such as AI-based recommendations, telemedicine, and integration with modern technologies, the system holds great potential to evolve into a complete healthcare management solution in the future.

REFERENCES

Bibliography :

1 "Doctor Appointment System" by Dr. R. P. S. Manikandan, Aswin G, Chandru A, and Murali Prasad M (2024).

Abstract: This paper introduces a web-based tool designed to streamline the process of scheduling medical appointments, aiming to enhance healthcare access and patient experience.

2 "Literature Review on Appointment Scheduling in Hospitals Management" by Sonal G. Shelwante, Anshuli Thakare, Karishma Sakharkar, Akshta Birelliwar, and Karuna Borkar (2019).

Abstract: This literature review examines various appointment scheduling systems in hospital management, highlighting the need for efficient scheduling to improve patient satisfaction and operational efficiency.

3 "Appointify: Doctor Appointment Booking System" by M. Ayush (2024).

Abstract: This research presents the design and implementation of 'Appointify,' a system aimed at enhancing the efficiency of healthcare delivery through improved appointment scheduling.

4 "Mr. Doc: A Doctor Appointment Application System" by Shafaq Malik, Nargis Bibi, Sehrish Khan, Razia Sultana, and Sadaf Abdul Rauf (2017).

Abstract: This work proposes an Android application, 'Mr. Doc,' which provides ease and comfort to patients in scheduling appointments, aiming to resolve common issues faced during the appointment process.

Webliography :

General Healthcare & Appointment Systems

1. World Health Organization (WHO) – Guidelines on patient scheduling and digital healthcare systems.

   https://www.who.int

2. Ministry of Health and Family Welfare, India – Covers India's digital health initiatives and policies.

   https://www.mohfw.gov.in

3. Ayushman Bharat Digital Mission (ABDM) – Official Indian digital health records and telemedicine guidelines.

   https://abdm.gov.in

4. MongoDB Official Documentation – Learn database design, indexing, and queries for handling appointment scheduling efficiently.

   https://www.mongodb.com/docs/

5. Express.js Guide – Helps in setting up the backend API for booking, cancelling managing appointments.

   https://expressjs.com/en/guide/routing.html

6. React Official Documentation – Best practices for building an interactive appointment booking UI.

https://react.dev/

7. Node.js Documentation – Essential for backend operations like authentication, API handling, and connecting to MongoDB. https://nodejs.org/en/docs
8. MERN Stack Authentication (JWT & OAuth) – Secure login for patients and doctors. https://www.digitalocean.com/community/tutorial_series/node-js-jwt-authentication

Project-Specific Research

9. "Doctor Appointment System" – Research Paper by Dr. R. P. S. Manikandan & Team – Covers web-based appointment scheduling.

   https://www.ijraset.com/research-paper/doctor-appointment-system

10. Appointify: Doctor Appointment Booking System" – Discusses mobile-first appointment systems.

https://www.researchgate.net/publication/386405098_Appointify_Doctor_Appointment_Booking_System