

CHAPTER 1

INTRODUCTION

1.1 ABOUT THE PROJECT

In today's digital era, healthcare services are increasingly transitioning to online platforms to provide better convenience and accessibility. A doctor's appointment platform simplifies the process of connecting patients with healthcare providers by enabling features like appointment scheduling, profile management, and streamlined communication. By using the **MERN stack** (MongoDB, Express.js, React, and Node.js), such a platform can be developed efficiently, ensuring high performance and scalability. The MERN stack provides an excellent foundation for developing this platform due to its modular and cohesive architecture:

MongoDB is ideal for storing complex data such as user profiles, doctor availability, and appointment records.

Express.js simplifies backend development by handling routing and API creation.

React ensures a dynamic and responsive user interface, allowing for real-time updates and seamless interactions.

Node.js powers the server-side logic, enabling efficient handling of user requests and business operations.

This project aims to create a full-stack web application that offers a seamless experience for patients, doctors, and administrators. Patients can search for medical professionals, book consultations, and view their appointment history, while doctors can manage their availability and schedules. Additionally, administrators can oversee users and system

activities to maintain smooth operations. The platform integrates essential features like secure user authentication, appointment management, and optional payment processing, ensuring a complete healthcare scheduling solution. The choice of the MERN stack ensures a responsive and interactive interface, a robust backend, and a scalable database, making it ideal for such an application. In addition to core features, the platform can incorporate advanced functionalities such as online payment processing for appointments, integration with third-party services like telehealth platforms, and personalized recommendations based on patient preferences. By adopting such enhancements, the platform can cater to a wide range of user needs and establish itself as an indispensable tool in modern healthcare.

1.2 EXISTING SYSTEM

The current methods for managing doctor appointments vary significantly, from manual approaches to more advanced digital solutions. Many smaller clinics still rely on traditional methods like scheduling appointments via phone or in person. These methods are inefficient, prone to human error, and often result in issues like double bookings or missed appointments. Some healthcare providers have adopted outdated digital systems, which offer basic functionalities but often lack real-time updates, scalability, and advanced features such as automated notifications or online payment options. In contrast, larger hospitals often implement complex Hospital Management Systems (HMS) that handle various tasks, including appointment scheduling, billing, and patient records. However, these systems tend to be costly, difficult to implement, and not particularly user-friendly for patients.

Meanwhile, platforms like Practo, Zocdoc, and Healthgrades provide online appointment booking along with features such as doctor searches, reviews, and real-time availability. While these platforms enhance convenience, they often come with subscription fees that may be unaffordable for smaller practices. Additionally, these systems are typically not customizable, limiting healthcare providers' ability to tailor them to specific requirements.

Another limitation of many existing systems is the lack of robust communication tools, such as appointment reminders or direct messaging, which can lead to missed appointments and a less satisfactory user experience. A system built using the MERN stack could address these gaps by offering secure authentication, real-time scheduling, seamless communication, and integration with advanced features such as telemedicine and online payments, improving the overall efficiency and accessibility of healthcare services.

1.3 DRAWBACKS OF EXISTING SYSTEM

- ❖ Inefficient Booking Processes
- ❖ Lack of Real-Time Functionality
- ❖ Poor Communication and Engagement
- ❖ High Costs and Limited Accessibility
- ❖ Security and Modern Feature Gaps

1.4 PROPOSED SYSTEM

To overcome the drawbacks in the existing system, the proposed system is designed. The proposed system is entirely web-based. Through the web-based application, people can file grievances under this suggested method and monitor their progress. In addition to enabling direct messaging between patients and doctors for improved communication, the system dynamically updates availability and sends real-time notifications and reminders via email or SMS for upcoming appointments or cancellations. Patients can view real-time doctor availability and book appointments instantly.

1.5 ADVANTAGES OF PROPOSED SYSTEM

- ❖ Improved Efficiency
- ❖ Enhanced Patient Experience
- ❖ Cost-Effectiveness
- ❖ Security and Compliance

CHAPTER 2

SYSTEM ANALYSIS

2.1 IDENTIFICATION OF NEED

For small clinics or healthcare providers in particular, a website for doctor's appointments is essential for improving the process of scheduling medical consultations. By enabling online appointment booking, real-time doctor availability viewing, and automatic reminders, it removes the inefficiencies associated with manual scheduling. In addition to saving staff and patient time, this also lowers scheduling errors. Furthermore, it makes healthcare services more accessible and raises patient satisfaction by offering 24/7 appointment scheduling access.

2.2 FEASIBILITY STUDY

2.2.1 Operational Feasibility

Operational feasibility is necessary as it ensures the project developed is a successful one. Tests have been carried out to ensure the operational feasibility of the system. The proposed system works efficiently and displays the information very quickly. The various aspects of the operational feasibility are used to measure the urgency and the acceptability of the solution that has been proposed.

2.2.2 Technical Feasibility

This research is used to evaluate the system's technical feasibility, or requirements. This might put a strain on existing technological resources as a result, the customer may face unreasonable expectations. Because only little or no changes are needed to utilise this approach, the designed system should have a low requirement.

2.2.3 Economical Feasibility

This research is being conducted to determine the system's economic impact on the organization. The amount of money the corporation may invest in system research and development is restricted. It is necessary to justify spending. As a result, the constructed system was also under budget, which was possible because the majority of the technologies employed were publicly accessible. Only the customized items have to be bought.

2.3 SOFTWARE REQUIREMENTS SPECIFICATION

System requirements is a statement that identifies the functionality that is needed by a system in order to satisfy the customer's requirements. System requirements are the most efficient way to address user needs while lowering implementation costs.

- ❖ Hardware Requirements
- ❖ Software Requirements

2.3.1 Hardware Requirements

The hardware for the system is selected considering the factors such as CPU processing speed, memory access, peripheral channel access speed, printed speed; seek time & relational data of hard disk and communication speed etc. Below is the minimum hardware requirement of the project.

System	: I5 Processor
Hard disk	: 1 TB
RAM	: 8 GB
Monitor	: 15 VGA color

2.3.2 Software Requirements

The software for the project is selected considering the factors such as working front end environment, flexibility in the coding language, database knowledge of enhances in backend technology etc.

OPERATING SYSTEM : WINDOWS 10

FRONT END : React.js, CSS

BACK END : Mongo DB, Node.js, Redux, Express

FRONT END

React.js

React is an open-source JavaScript library for building user interfaces. It was developed by Facebook and is now maintained by Facebook and a community of individual developers and companies. React is widely used for creating dynamic and interactive web applications.

Component-based: React is centred around the concept of components. Developers create self-contained, reusable UI components that can be composed to build complex user interfaces. This promotes modularity and code reusability.

Server-Side Rendering (SSR): React can be used for server-side rendering, which improves performance, search engine optimization (SEO), and initial page load times.

Facebook and Instagram: React was initially developed by Facebook for use in its web applications, and it's also used by Instagram. This real-world usage demonstrates its robustness and scalability.

CSS

CSS (Cascading Style Sheets) is a vital component of modern web development, enabling the styling and layout of HTML elements within web pages. With CSS, developers can define the visual appearance of elements, including colours, fonts, spacing, and positioning, creating engaging and visually appealing user interfaces. One of CSS's key features is its use of selectors, which allow

developers to target specific HTML elements or groups of elements and apply styling rules to them. This level of specificity enables precise control over the presentation of content, enhancing the overall user experience.

CSS promotes the separation of content from design, facilitating cleaner and more maintainable code. By keeping styling instructions separate from HTML markup, developers can easily update the visual presentation without altering the underlying content structure. CSS plays a crucial role in creating responsive web designs that adapt seamlessly to different screen sizes and devices. Media queries in CSS enable developers to apply different styles based on factors such as screen width, height, and orientation, ensuring optimal viewing experiences across various platforms.

Furthermore, CSS preprocessors like Sass and Less extend CSS's capabilities by introducing features such as variables, mixing, and functions, which enhance code organization, reusability, and maintainability. Overall, CSS is an essential tool for creating modern, accessible, and user-friendly websites, empowering developers to craft compelling and immersive web experiences for users worldwide.

BACKEND AND DATABASE

MongoDB

MongoDB is a popular and versatile NoSQL database system that has gained widespread adoption in recent years. It stands out for its flexible, document-oriented data model, making it an excellent choice for applications that require dynamic, rapidly changing, or unstructured data. MongoDB uses a JSON-like format called BSON to store data, allowing for the easy storage and retrieval of complex data structures. Its ability to horizontally scale across multiple servers makes it a scalable and high-performance solution for handling large volumes of data and high-traffic applications.

One of MongoDB's notable features is its support for automatic sharding, which enables data distribution across multiple servers, improving both data distribution and fault tolerance. Additionally, MongoDB is known for its strong query capabilities, including support for geospatial queries, text search, and aggregation. Developers often find MongoDB appealing due to its use of familiar programming languages like JavaScript and its integration with a variety of programming frameworks and platforms.

However, MongoDB's schema-less design can sometimes lead to challenges in maintaining data consistency and integrity, which requires careful consideration during application development. Overall, MongoDB has become a favoured choice for modern, data-intensive applications, where its scalability, flexibility, and support for various data types make it a valuable addition to the database landscape

Node.js

Node.js is a powerful, open-source JavaScript runtime environment built on Chrome's V8 JavaScript engine. It allows developers to run JavaScript code outside the browser, enabling server-side scripting for web applications. Node.js utilizes an event-driven, non-blocking I/O model, making it highly efficient and scalable for handling concurrent connections and real-time applications. One of Node.js's key strengths is its asynchronous, event-driven architecture, which enables developers to build fast and scalable network applications. This model allows Node.js to handle numerous connections simultaneously without blocking execution, making it ideal for building highly responsive and efficient servers.

Node.js has a rich ecosystem of packages available through npm (Node Package Manager), which is the largest ecosystem of open-source libraries in the world. This extensive collection of modules and libraries simplifies development by providing ready-made solutions for various tasks, from web frameworks like Express.js to database drivers like MongoDB and MySQL.

Node.js is well-suited for building modern web applications, APIs, and microservices due to its lightweight and modular nature. It's also popular for developing real-time applications such as chat applications, gaming servers, and IoT (Internet of Things) applications. Node.js's ability to share code between the client and server-side applications simplifies development workflows and promotes code reuse. Overall, Node.js has revolutionized server-side development, empowering developers to build high-performance, scalable, and real-time applications using JavaScript across the entire stack.

Express

Express.js is a minimalist and flexible Node.js web application framework used for building web and mobile applications. It provides a robust set of features, including middleware, routing, and template engine integration, making it a popular choice among developers. Express.js revolves around middleware functions that handle HTTP requests and responses. Middleware functions have access to the request and response objects and can perform tasks such as parsing request bodies, authenticating users, and error handling. This modular approach allows developers to create reusable and composable components, improving code organization and maintainability.

Routing is another essential aspect of Express.js, enabling developers to define routes for handling different HTTP methods and URL patterns. Route handlers execute logic based on the incoming requests, allowing developers to implement RESTful APIs, serve static files, and render dynamic content using template engines like EJS or Pug.

Express.js also supports error handling, static file serving, and integration with third-party middleware through its middleware ecosystem. Its lightweight nature and extensive documentation make it an ideal choice for building scalable and efficient web applications. Express.js provides a solid foundation for building web servers and APIs in Node.js, offering developers the flexibility and tools needed to create modern and performant applications.

CHAPTER 3

SYSTEM DESIGN

3.1 MODULE DESCRIPTION

The Healthcare Booking Hub is a web application that enables patients to search for doctors, book appointments, and manage appointments. It allows doctors to manage their availability and appointments, and administrators to oversee the system.

ADMIN MODULE:

- Login
- Appointment Management
- Doctor Manipulation
- Doctors List
- Log Out

3.2 Admin Module Description

Login:

The admin logs into the system using a secure username and password. Multi-factor authentication ensures authorized access. The system tracks login sessions for enhanced security and accountability.

Appointment Management:

Enables admins to oversee all appointments, including pending, confirmed, and cancelled ones. Admins can reschedule or cancel appointments in case of conflicts or user requests.

Doctor Manipulation:

Admins can add new doctors, update profile details, or deactivate accounts. They ensure doctors' credentials are verified before approval. This module allows for managing schedules and availability as needed.

Doctors List:

Displays a comprehensive list of all registered doctors, along with their profiles. Includes information such as name, speciality, contact information, and availability. Offers search and filter options for convenient navigation and access.

Logout:

Ensures a secure log-out process to protect admin access. Session details are cleared upon logging out to prevent unauthorized use. Encourages admins to log out after use for data safety.

USER MODULE:

- Register
- User Profile Manipulation
- Doctors Page

- Filter Speciality
- Related Doctors
- Book Appointments
- User Appointment Page
- Logout

Register:

Users can sign up by providing personal information such as their name, contact information, and password. Validates inputs to guarantee data accuracy and keeps credentials secure.

User Profile Manipulation:

Users can change their profile information, including contact information, password, and preferences. Ensures that data is updated in real time and securely saved in the database. Includes possibilities for profile photo uploads and notification settings.

Doctors Page:

Displays a list of all registered doctors, including their specialities and availability. Users can explore and select doctors for consultations. Allows users to access detailed doctor profiles.

Filter Speciality:

Users can filter doctors based on specialization, availability, and consultation. Makes it easier to discover the proper doctor for the user's specific needs. Provides easy access to key options.

Related Doctors:

Suggests doctors based on the user's search criteria or previous selections. Recommends possibilities based on specialization. Improves user convenience with intelligent suggestions.

Book Appointments:

Allows you to arrange appointments by selecting the doctor, day, time, and type of consultation. Users can include medical documents or notes when booking. It sends confirmation and reminders via alerts.

User Appointment Page:

Users can browse, manage, and track their upcoming and past appointments. Provides status information such as pending, verified, and done. Includes rescheduling and cancellation options

Logout:

Ensures secure log-out to end user sessions and clear sensitive data. Protects accounts by preventing unauthorized access after the session ends. Encourages safe practices for enhanced security.

DOCTOR MODULE:

- Login
- Patients Appointments Management
- Doctors Profile Manipulation
- Logout

Login:

Doctors log in securely using their credentials, which include multi-factor authentication for enhanced security. Provides access to their personalised dashboard. Session tracking prevents unauthorised access.

Patients Appointments Management:

Doctors can view, confirm, reschedule, and cancel patients' appointments. Provides a well-organised calendar for easy scheduling management. Sends automated updates to patients regarding any changes.

Doctors Profile Manipulation:

Allows doctors to update their profiles, which include specialisation, contact information, and availability. Ensures patients receive accurate information. Changes are logged in real time and securely saved.

Logout:

Allows doctors to log out safely, with session data erased to prevent misuse. Protects sensitive information and promotes secure access practices. Ends active sessions safely.

PAYMENT MODULE:

- Payment Method

Payment Method:

Enables a secure log-out process to terminate user sessions. Clears session data to keep user accounts secure from unauthorised access. Encourages safe browsing practices by prompting users to log out after each use.

3.3 ACTIVITY DIAGRAM:

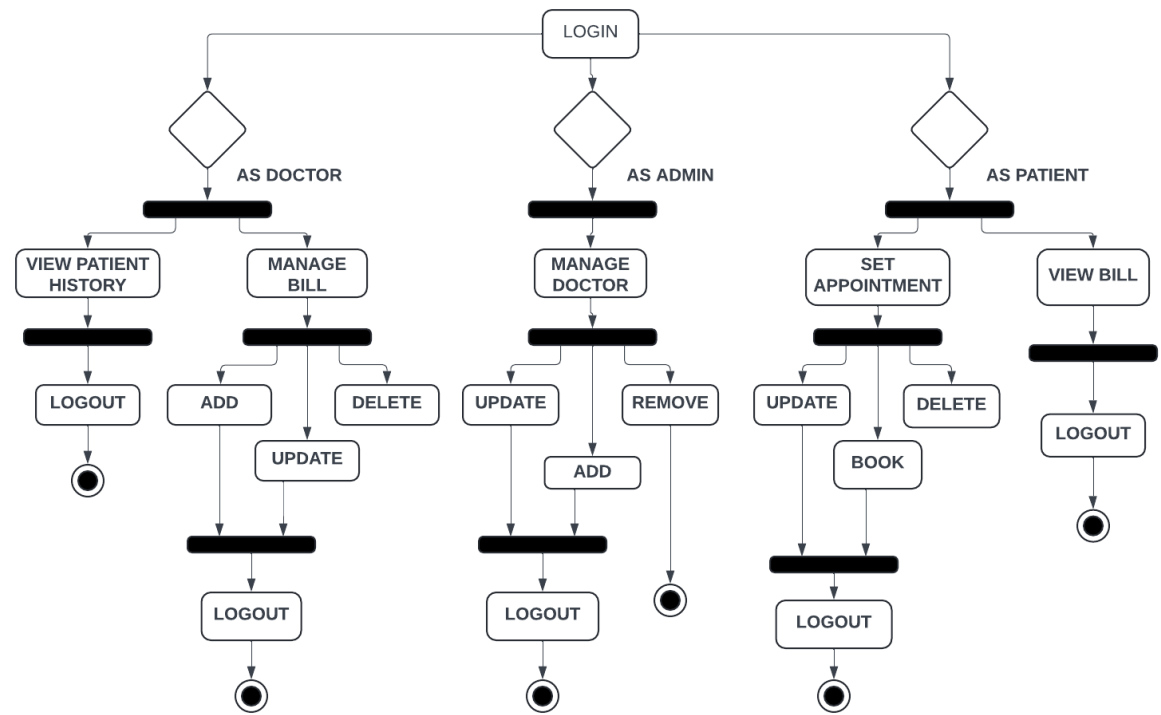


Figure 3.1 Activity Diagram

3.4 DATA FLOW DIAGRAM:

LEVEL 0:

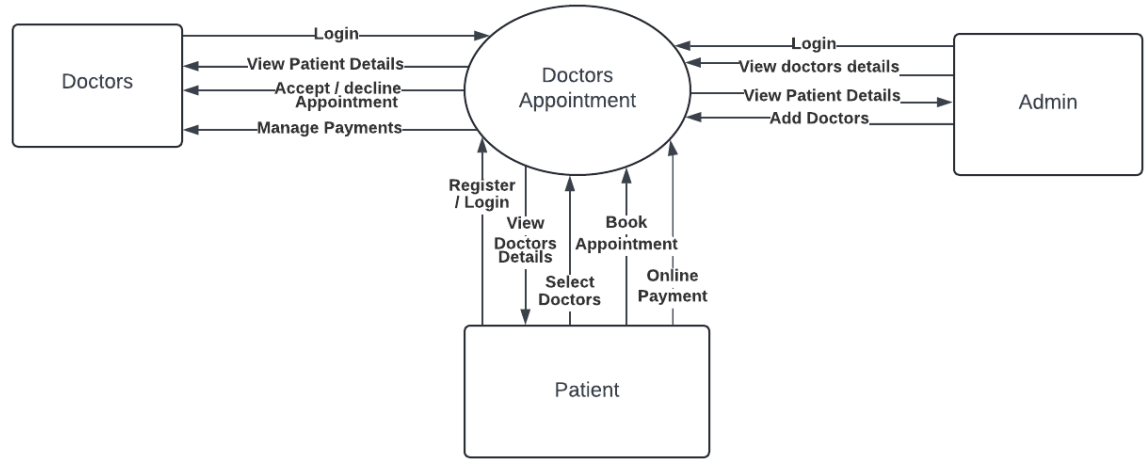


Figure 3.2 Data Flow Diagram (Level 0)

LEVEL 1:

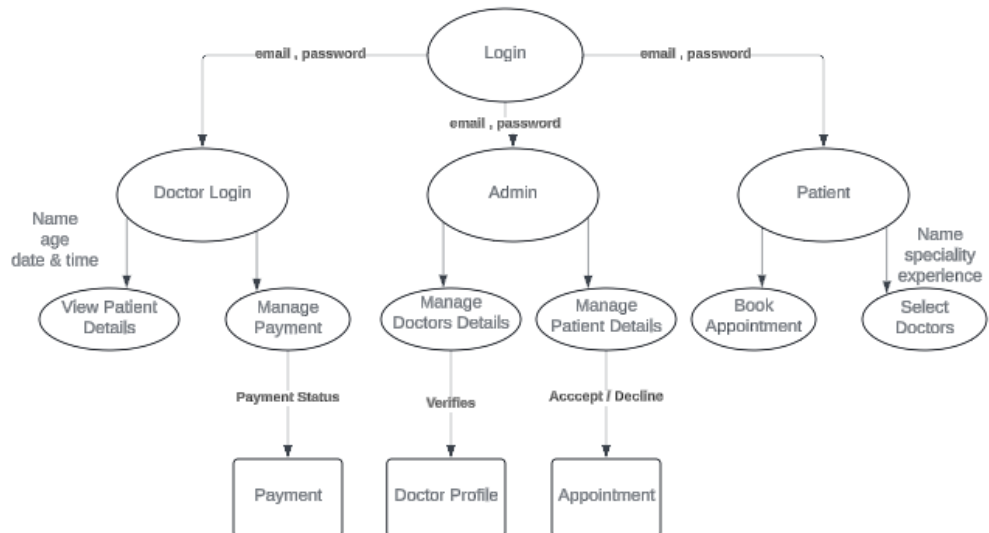


Figure 3.3 Data Flow Diagram (Level 1)

3.5 USECASE DIAGRAM:

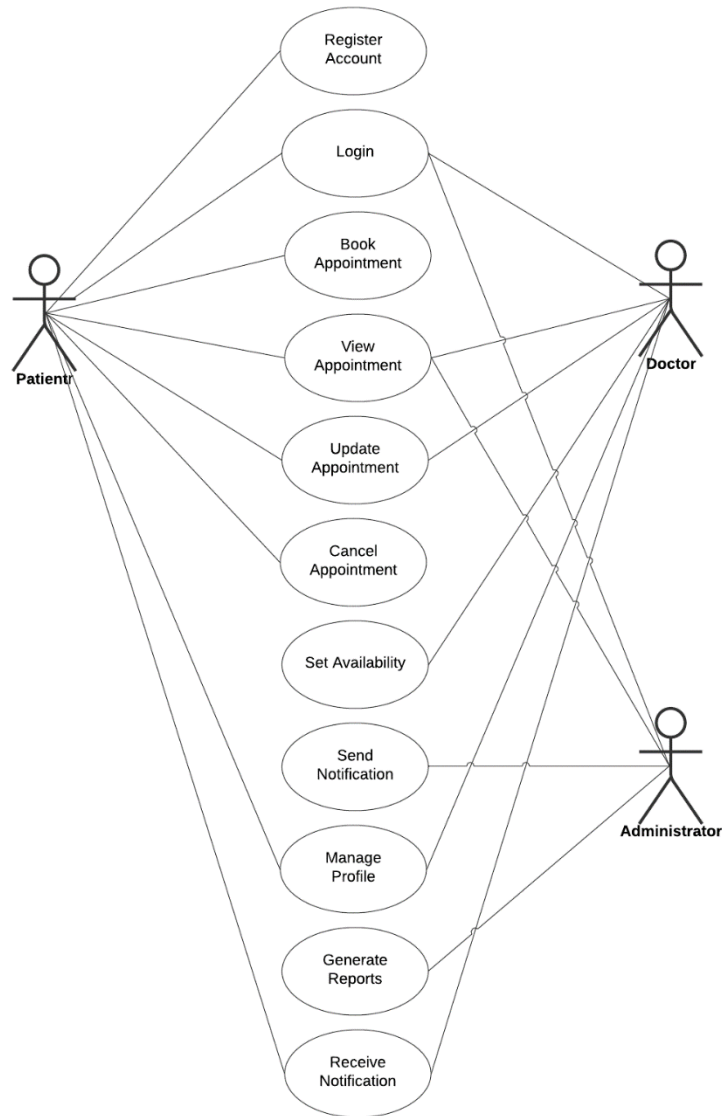


Figure 3.4 Use Case Diagram

3.6 ER-DIAGRAM:

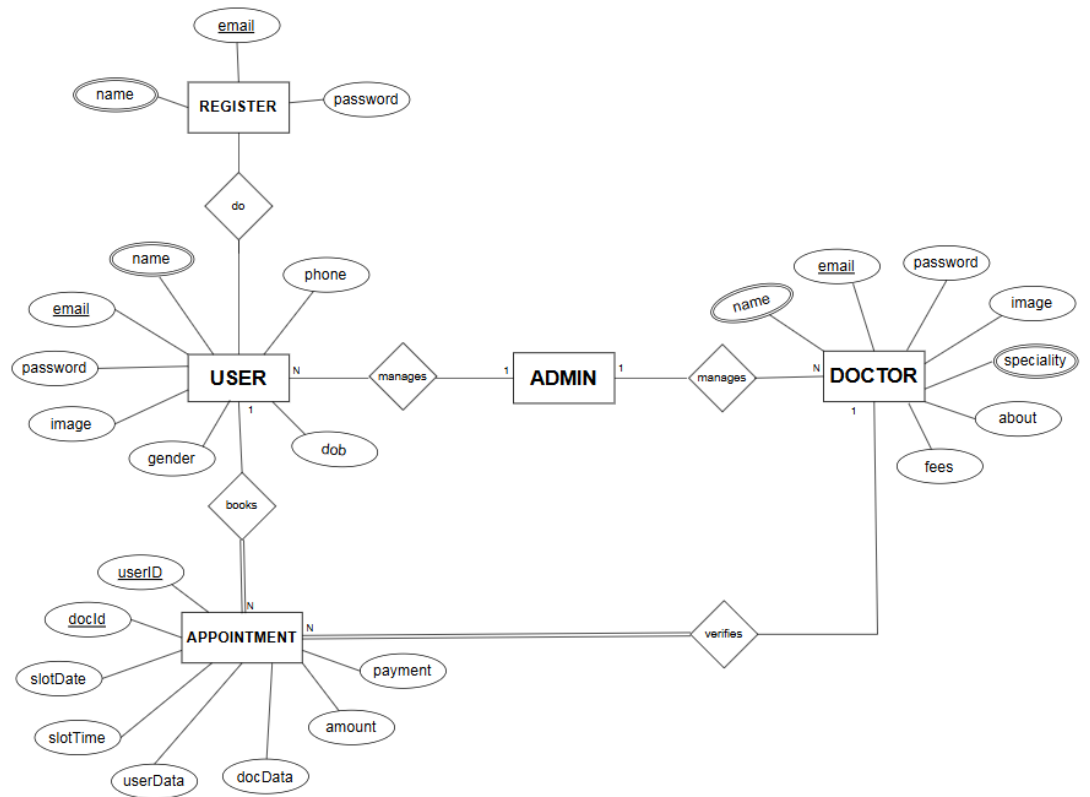


Figure 3.5 ER Diagram

3.7 DATABASE DESIGN:

Table 3.1 Appointment Details

S.No	Field Name	Type	Size	Constraints
1	id	Int	4 bytes	Primary Key
2	UserID	String	50 chars	Not Null
3	DocID	String	50 chars	Not Null
4	SlotDate	String	10 chars	Not Null
5	SlotTime	String	8 chars	Not Null
6	UserData	Object	N/A	Not Null
7	DocData	Object	N/A	Not Null
8	Amount	Int	4 bytes	Not Null
9	Date	Double	8 bytes	Not Null
10	Cancelled	Boolean	1 bit	Not Null
11	Payment	Boolean	1 bit	Not Null
12	IsCompleted	Boolean	1 bit	Not Null

Table 3.2 Doctor Details

S.No	Field Name	Type	Size	Constraints
1	Id	Int	4 bytes	Primary Key
2	Name	String	100 chars	Not Null
3	Email	String	100 chars	Not Null
4	Password	String	255 chars	Not Null
5	Image	String	255 chars	Not Null
6	Speciality	String	100 chars	Not Null
7	Degree	String	100 chars	Not Null
8	Experience	String	200 chars	Not Null
9	About	String	500 chars	Not Null
10	Available	String	10 chars	Not Null
11	Fees	Int	4 bytes	Not Null
12	Address	Object	N/A	Not Null
13	Date	Double	8 bytes	Not Null
14	Slots_booked	Object	N/A	Not Null

Table 3.3 User Details

S.No	Field Name	Type	Size	Constraints
1	Id	Int	4 bytes	Primary Key
2	Name	String	100 chars	Not Null
3	Email	String	100 chars	Not Null
4	Image	String	255 chars	Not Null
5	Phone	String	15 chars	Not Null
6	Address	Object	N/A	Not Null
7	Gender	String	10 chars	Not Null
8	DOB	String	10 chars	Not Null
9	Password	String	255 chars	Not Null

INPUT DESIGN:

The input design allows for accurate data collecting using user-friendly forms. During registration, patients and doctors enter crucial information such as their names, email addresses, and secure passwords, which are validated to verify accuracy. Patients provide appointment preferences such as date, time, and doctor specialisation, while doctors manage availability via scheduling inputs. Users can send complaints or feedback using structured fields, dropdown categories, and file uploads for supporting documents. Payment inputs are secure, ensuring confidentiality and dependability.

OUTPUT DESIGN:

The output design is to provide clear and actionable information. Users receive confirmation messages following successful registrations and appointment bookings. Role-specific dashboards provide live updates on appointment statuses, scheduling, and complaint progress. Notifications via email and SMS keep users aware of important events such as appointment confirmations, cancellations, and complaint resolution. Doctors and patients can see important profile changes, while administrators can obtain complete reports on system usage, booking trends, and complaint handling. This input-output design creates a holistic experience that balances efficiency and clarity, resulting in higher user satisfaction.

CHAPTER 4

IMPLEMENTATION

4.1 CODE DESCRIPTION

Code description can be used to summarize code or to explain the programmer's intent. Good comments don't repeat the code or explain it. They clarify its intent. Comments are sometimes processed in various ways to generate documentation external to the source code itself by document generator or used for integration with systems and other kinds of external programming tools. Developer have chosen React.js as front end, Node.js and Express.js as backend and MongoDB as Database.

4.2 STANDARDIZATION OF THE CODING

Coding standards define a programming style. A coding standard does not usually concern itself with wrong or right in a more abstract sense. It is simply a set of rules and guidelines for the formatting of source code. The other common type of coding standard is the one used in or between development teams. Professional code performs a job in such a way that is easy to maintain and debug. All the coding standards are followed while creating this project. Coding standards become easier, the earlier you start. It is better to do a neat job than cleaning up after all is done. Every coder will have a unique pattern than others to. Such a style might include the conventions he uses to name variables and functions and how he comments his work. When the said pattern and style is standardized, it pays off the effort well in the long.

4.3 ERROR HANDLING

Exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code. Mainly if-else block is used to handle errors using condition checking, if-else catch errors as a conditional statement. In many cases there are many corner cases

which must be checking during an execution but if-else can only handle the defined conditions. In if-else, conditions are manually generated based on the task.

An error is a serious problem than an application doesn't usually get pass without incident. Errors cause an application to crash, and ideally send an error message offering some suggestion to resolve the problem and return to a normal operating state, there no way to deal with errors live or in production the only solution is to detect them via error monitoring and bug tracking and dispatch a developer or two to sort out the code.

Exceptions, on the other hand are exceptional conditions an application should reasonably be accepted to handle. Programming language allow developers to include try. Catch statements to handle exceptions and apply a sequence of logic to deal with the situation instead of crashing. And when an application encounters an exception that there is no work around for, that is called an unhandled exception, which is the same thing as an error.

CHAPTER 5

TESTING AND RESULTS

5.1 TESTING

It is the last check of the specification, design, and code, software testing is an important aspect of software quality assurance. In fact, testing is the only step in the software development process that might be regarded as detrimental rather than beneficial.

A software testing strategy combines software action design approaches with a well-thought-out set of steps that result in efficient software development.

Testing is a set of tasks that may be scheduled ahead of time and completed in a systematic way. The basic purpose of programme testing is to check software quality utilizing cost-effective and effective methods for both large and small systems.

5.2 Unit Testing

The construction of test cases to confirm that the core programme logic is operating properly and that programme inputs result in genuine outputs is known as unit testing. The appliance's individual software modules are evaluated. It is completed after a private unit has been completed but before integration. Unit tests are component-level tests that look at the specified inputs and expected effects of a given business process, application, or system configuration.

Test Case

Module : Admin Login

Input : Email, Password

Expected output : Authentication done, Redirected to next Admin page

Sample Test

Input Email: jeevaa.21bsr@kongu.edu

Input password : 12345678

Output : Redirect to page

Analysis : Expected Page will open.

5.3 Integration Testing

Integration tests check if software components that are expected to function together really do. Integration tests show that, while the individual components were satisfactory, the combination of components is correct and consistent, as seen by successful unit testing.

Integration testing is a sort of testing that focuses on identifying problems caused by a number of different components. Integration testing ensures that key design aspects fulfil functional, performance, and reliability requirements.

All projects are carried out in line with the event's schedule. The fundamental idea is to employ a "building block" technique in which confirmed assemblages are added to a verified base and then used to help with blending testing.

Test case

Module : Admin

Input : Add Doctors

Expected Output : Doctors will be added.

Sample Test

Input : Add Doctors

Output : Redirect to page

Analysis : Expected page will open

5.4 Validation Testing

Integration testing concludes with the application being fully packed as a package, the discovery and correction of interface issues, and a final set of software tests. Validation testing might begin right now. Validation may be defined in a variety of ways, but one basic definition is when software behaves as the customer would reasonably expect.

Data validation testing includes validation testing. The information type and value are assessed during the validation testing phase. In the proposed approach, the password is the only data used for testing. There is no need to analyse the information type of any data in the proposed system.

Test Case

Module : Appointments

Input : Input to all fields

Expected Output : Appointment should be Booked.

Sample Test

Input : Email – keerthi@gmail.com

Name : keerthi

Address : 12A, Erode

State : TamilNadu

Output : Redirect to payment module

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

6.1 CONCLUSION:

The new Doctor's Appointment Booking System efficiently addresses the issues that plagued the prior system. It is intended to be extremely user-friendly and efficient, ensuring seamless interaction for all users. The major purpose is to create an interactive and streamlined platform for scheduling and monitoring healthcare visits. Patients can register online to get information about doctors, services, and availability. They can simply arrange, reschedule, and cancel appointments as necessary. The system also includes safe payment methods for any mandatory fees, making it more convenient for users.

By automating the process, the technology eliminates human mistake, shortens wait times, and increases operational efficiency. The system generates accurate and well-structured reports, which help administrators and doctors manage schedules and data more effectively.

6.2 FUTURE ENHANCEMENTS:

The Doctor's Appointment Booking System can be improved by adding timely reminders, complete appointment histories, and AI-powered doctor recommendations. Telemedicine integration for video consultations, as well as extensive reporting and analytics, could boost functionality even further. Security methods such as two-factor authentication, data encryption, and cloud-based backups would improve data reliability and protection. Built with React JS, MongoDB, and Node.js, the system is well-positioned for future improvements that will make it more user-friendly and efficient.

APPENDICES

A. SAMPLE CODING

ADD DOCTOR:

```
import React, { useContext, useState } from 'react'
import { assets } from '../assets/assets'

import { toast } from 'react-toastify'

import axios from 'axios'

import { AdminContext } from '../context/AdminContext'
import { AppContext } from '../context/AppContext'

const AddDoctor = () => {

  const [docImg, setDocImg] = useState(false)

  const [name, setName] = useState("")
  const [email, setEmail] = useState("")
  const [password, setPassword] = useState("")
  const [experience, setExperience] = useState('1 Year')
  const [fees, setFees] = useState("")
  const [about, setAbout] = useState("")
  const [speciality, setSpeciality] = useState('General physician')
  const [degree, setDegree] = useState("")
  const [address1, setAddress1] = useState("")
  const [address2, setAddress2] = useState("")

  const { backendUrl } = useContext(AppContext)
  const { aToken } = useContext(AdminContext)

  const onSubmitHandler = async (event) => {
```

```
event.preventDefault()

try {
  if (!docImg) {
    return toast.error('Image Not Selected')
  }

  const formData = new FormData();

  formData.append('image', docImg)
  formData.append('name', name)
  formData.append('email', email)
  formData.append('password', password)
  formData.append('experience', experience)
  formData.append('fees', Number(fees))
  formData.append('about', about)
  formData.append('speciality', speciality)
  formData.append('degree', degree)
  formData.append('address', JSON.stringify({ line1: address1, line2: address2 }))

  // console log formdata

  formData.forEach((value, key) => {
    console.log(`${key}: ${value}`);
  });

  const { data } = await axios.post(backendUrl + '/api/admin/add-doctor', formData, {
    headers: { aToken } })

  if (data.success) {
    toast.success(data.message)

    setDocImg(false)
```

```

setName("")
setPassword("")
setEmail("")
setAddress1("")
setAddress2("")
setDegree("")
setAbout("")
setFees("")
} else {
toast.error(data.message)
}
} catch (error) {
toast.error(error.message)
console.log(error)
}
}
return (
<form onSubmit={onSubmitHandler} className='m-5 w-full'>
<p className='mb-3 text-lg font-medium'>Add Doctor</p>
<div className='bg-white px-8 py-8 border rounded w-full max-w-4xl max-h-[80vh] overflow-y-scroll'>
<div className='flex items-center gap-4 mb-8 text-gray-500'>
<label htmlFor="doc-img">
<img className='w-16 bg-gray-100 rounded-full cursor-pointer' src={docImg ?
URL.createObjectURL(docImg) : assets.upload_area} alt="" />

```



```

</label>

<input onChange={(e) => setDocImg(e.target.files[0])} type="file" name=""
id="doc-img" hidden />

<p>Upload doctor <br /> picture</p>

</div>

<div className='flex flex-col lg:flex-row items-start gap-10 text-gray-600'>

<div className='w-full lg:flex-1 flex flex-col gap-4'>

<div className='flex-1 flex flex-col gap-1'>

<p>Your name</p>

<input onChange={e => setName(e.target.value)} value={name}
className='border rounded px-3 py-2' type="text" placeholder='Name' required />

</div>

<div className='flex-1 flex flex-col gap-1'>

<p>Doctor Email</p>

<input onChange={e => setEmail(e.target.value)} value={email}
className='border rounded px-3 py-2' type="email" placeholder='Email' required
/>

</div>

<div className='flex-1 flex flex-col gap-1'>

<p>Set Password</p>

<input onChange={e => setPassword(e.target.value)} value={password}
className='border rounded px-3 py-2' type="password" placeholder='Password'
required />

</div>

<div className='flex-1 flex flex-col gap-1'>

<p>Experience</p>

```

```

<select onChange={e => setExperience(e.target.value)} value={experience}
className='border rounded px-2 py-2' >

<option value="1 Year">1 Year</option>

<option value="2 Year">2 Years</option>

<option value="3 Year">3 Years</option>

<option value="4 Year">4 Years</option>

<option value="5 Year">5 Years</option>

<option value="6 Year">6 Years</option>

<option value="8 Year">8 Years</option>

<option value="9 Year">9 Years</option>

<option value="10 Year">10 Years</option>

</select>

</div>

<div className='flex-1 flex flex-col gap-1'>

<p>Fees</p>

<input onChange={e => setFees(e.target.value)} value={fees} className='border
rounded px-3 py-2' type="number" placeholder='Doctor fees' required />

</div>

</div>

<div className='w-full lg:flex-1 flex flex-col gap-4'>

<div className='flex-1 flex flex-col gap-1'>

<p>Speciality</p>

<select onChange={e => setSpeciality(e.target.value)} value={speciality}
className='border rounded px-2 py-2'>

<option value="General physician">General physician</option>

<option value="Gynecologist">Gynecologist</option>

```

```

<option value="Dermatologist">Dermatologist</option>
<option value="Pediatricians">Pediatricians</option>
<option value="Neurologist">Neurologist</option>
<option value="Gastroenterologist">Gastroenterologist</option>
</select>

</div>

<div className='flex-1 flex flex-col gap-1'>

<p>Degree</p>

<input  onChange={e  =>    setDegree(e.target.value)}    value={degree}
className='border rounded px-3 py-2' type="text" placeholder='Degree' required />

</div>

<div className='flex-1 flex flex-col gap-1'>

<p>Address</p>

<input  onChange={e  =>    setAddress1(e.target.value)}    value={address1}
className='border rounded px-3 py-2' type="text" placeholder='Address 1'
required />

<input  onChange={e  =>    setAddress2(e.target.value)}    value={address2}
className='border rounded px-3 py-2' type="text" placeholder='Address 2'
required />

</div>

</div>

</div>

</div>

<div>

<p className='mt-4 mb-2'>About Doctor</p>

<textarea  onChange={e  =>    setAbout(e.target.value)}    value={about}
className='w-full px-4 pt-2 border rounded' rows={5} placeholder='write about
doctor'></textarea>

```

```

</div>

<button type='submit' className='bg-primary px-10 py-3 mt-4 text-white rounded-
full'>Add doctor</button>

</div>

</form> )}export default AddDoctor

```

ALL APPOINTMENTS

```

import React, { useEffect } from 'react'

import { assets } from '../assets/assets'

import { useContext } from 'react'

import { AdminContext } from '../context/AdminContext'

import { AppContext } from '../context/AppContext'

const AllAppointments = () => {

  const { aToken, appointments, cancelAppointment, getAllAppointments } =
    useContext(AdminContext)

  const { slotDateFormat, calculateAge, currency } = useContext(AppContext)

  useEffect(() => {

    if (aToken) {

      getAllAppointments()

    }

  }, [aToken])

  return (

    <div className='w-full max-w-6xl m-5 '>

      <p className='mb-3 text-lg font-medium'>All Appointments</p>

      <div className='bg-white border rounded text-sm max-h-[80vh] overflow-y-
scroll'>

```

```
<div className='hidden sm:grid grid-cols-[0.5fr_3fr_1fr_3fr_3fr_1fr_1fr] grid-
flow-col py-3 px-6 border-b'>
```

```
<p>#</p>
```

```
<p>Patient</p>
```

```
<p>Age</p>
```

```
<p>Date & Time</p>
```

```
<p>Doctor</p>
```

```
    <p>Fees</p>
```

```
<p>Action</p>
```

```
</div>
```

```
{appointments.map((item, index) => (
```

```
<div className='flex flex-wrap justify-between max-sm:gap-2 sm:grid sm:grid-
cols-[0.5fr_3fr_1fr_3fr_3fr_1fr_1fr] items-center text-gray-500 py-3 px-6 border-b
hover:bg-gray-50' key={index}>
```

```
<p className='max-sm:hidden'>{index+1}</p>
```

```
<div className='flex items-center gap-2'>
```

```
<img src={item.userData.image} className='w-8 rounded-full' alt="" />
```

```
<p>{item.userData.name}</p>
```

```
</div>
```

```
<p className='max-sm:hidden'>{calculateAge(item.userData.dob)}</p>
```

```
<p>{slotDateFormat(item.slotDate)}, {item.slotTime}</p>
```

```
<div className='flex items-center gap-2'>
```

```
<img src={item.docData.image} className='w-8 rounded-full bg-gray-200' alt=""
```

```
/> <p>{item.docData.name}</p>
```

```
</div>
```

```
<p>{currency}{item.amount}</p>
```

```

{ item.cancelled ? <p className='text-red-400 text-xs font-medium'>Cancelled</p>
: item.isCompleted ? <p className='text-green-500 text-xs
font-medium'>Completed</p> : <img onClick={() =>
cancelAppointment(item._id)} className='w-10 cursor-pointer'
src={assets.cancel_icon} alt="" />}

</div>

)}}

</div>

</div>

)

}

export default AllAppointments

```

DOCTOR PROFILE:

```

import React, { useContext, useEffect, useState } from 'react'
import { DoctorContext } from '../context/DoctorContext'

import { AppContext } from '../context/AppContext'

import { toast } from 'react-toastify'

import axios from 'axios'

const DoctorProfile = () => {

const { dToken, profileData, setProfileData, getProfileData } =
useContext(DoctorContext)

const { currency, backendUrl } = useContext(AppContext)

const [isEdit, setIsEdit] = useState(false)

const updateProfile = async () => {

try {

```

```

const updateData = {
  address: profileData.address,
  fees: profileData.fees,
  about: profileData.about,
  available: profileData.available
}

const { data } = await axios.post(backendUrl + '/api/doctor/update-profile',
updateData, { headers: { dToken } })

if (data.success) {
  toast.success(data.message)

  setIsEdit(false)

  getProfileData()
} else {
  toast.error(data.message)
}

setIsEdit(false)
} catch (error) {
  toast.error(error.message)

  console.log(error)
}
}

useEffect(() => {
  if (dToken) {
    getProfileData()
  }
}

```

```

    }, [dToken])

    return profileData && (

      <div>

        <div className='flex flex-col gap-4 m-5'>

          <div>

            <img      className='bg-primary/80      w-full      sm:max-w-64      rounded-lg'
            src={profileData.image} alt="" />

          </div>

          <div className='flex-1 border border-stone-100 rounded-lg p-8 py-7 bg-white'>

            <p      className='flex items-center gap-2 text-3xl font-medium text-gray-
            700'>{profileData.name}</p>

            <div className='flex items-center gap-2 mt-1 text-gray-600'>

              <p>{profileData.degree} - {profileData.speciality}</p>

              <button      className='py-0.5      px-2      border      text-xs      rounded-
              full'>{profileData.experience}</button>

            </div>

            <div>

              <p className='flex items-center gap-1 text-sm font-medium text-[#262626] mt-
              3'>About :</p>

              <p className='text-sm text-gray-600 max-w-[700px] mt-1'>

                {

                  isEdit

                  ? <textarea onChange={(e) => setProfileData(prev => ({ ...prev, about:
                  e.target.value }))) type='text' className='w-full outline-primary p-2' rows={8}
                  value={profileData.about} />

                  : profileData.about

                }

              </p>

```


</p>

</div>

<p className='text-gray-600 font-medium mt-4'>

Appointment fee: {currency} {isEdit ? <input type='number' onChange={(e) => setProfileData(prev => ({ ...prev, fees: e.target.value })))} value={profileData.fees} /> : profileData.fees}

</p>

<div className='flex gap-2 py-2'>

<p>Address:</p>

<p className='text-sm'>

{isEdit ? <input type='text' onChange={(e) => setProfileData(prev => ({ ...prev, address: { ...prev.address, line1: e.target.value } })))} value={profileData.address.line1} /> : profileData.address.line1 }

{isEdit ? <input type='text' onChange={(e) => setProfileData(prev => ({ ...prev, address: { ...prev.address, line2: e.target.value } })))} value={profileData.address.line2} /> : profileData.address.line2 }

</p>

</div>

<div className='flex gap-1 pt-2'>

<input type="checkbox" onChange={() => isEdit && setProfileData(prev => ({ ...prev, available: !prev.available })))} checked={profileData.available} />

<label htmlFor="">Available</label>

</div>

{

isEdit

```
? <button onClick={updateProfile} className='px-4 py-1 border border-primary
text-sm rounded-full mt-5 hover:bg-primary hover:text-white transition-
all'>Save</button>
```

```
: <button onClick={() => setIsEdit(prev => !prev)} className='px-4 py-1 border
border-primary text-sm rounded-full mt-5 hover:bg-primary hover:text-white
transition-all'>Edit</button>
```

```
}
```

```
</div>
```

```
</div>
```

```
</div>
```

```
)
```

```
}
```

```
export default DoctorProfile
```

B. SCREENSHOTS

HOME PAGE

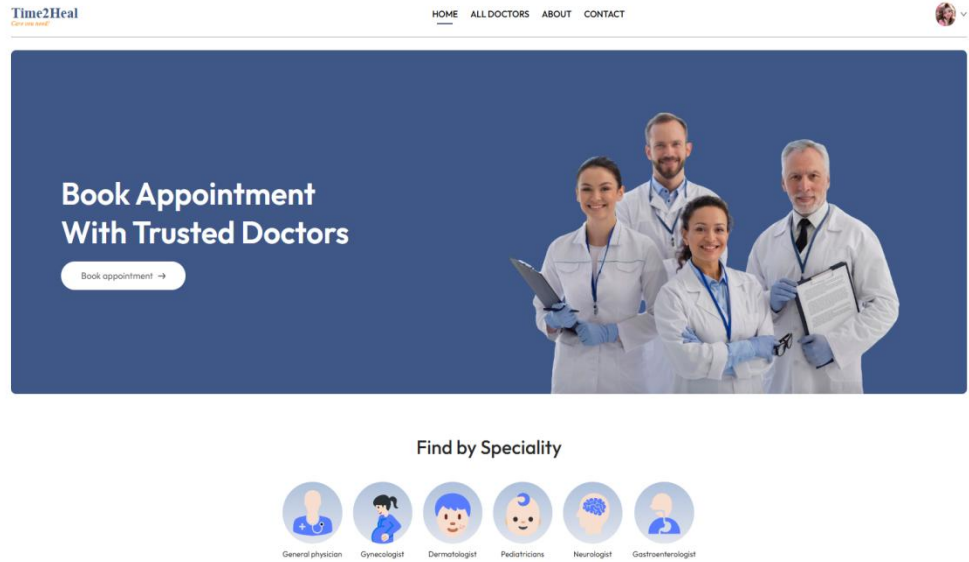


Figure B.1 Home Page

ADD DOCTOR

Figure B.2 Add Doctors

DOCTORS LIST

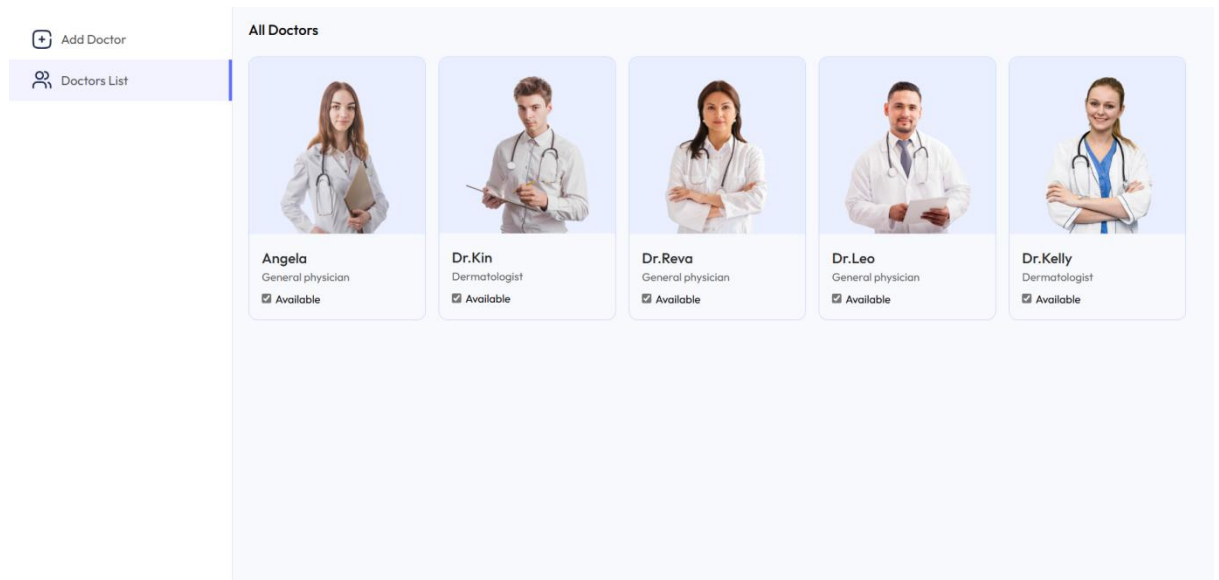


Figure B.3 All Doctors List

DOCTORS PROFILE

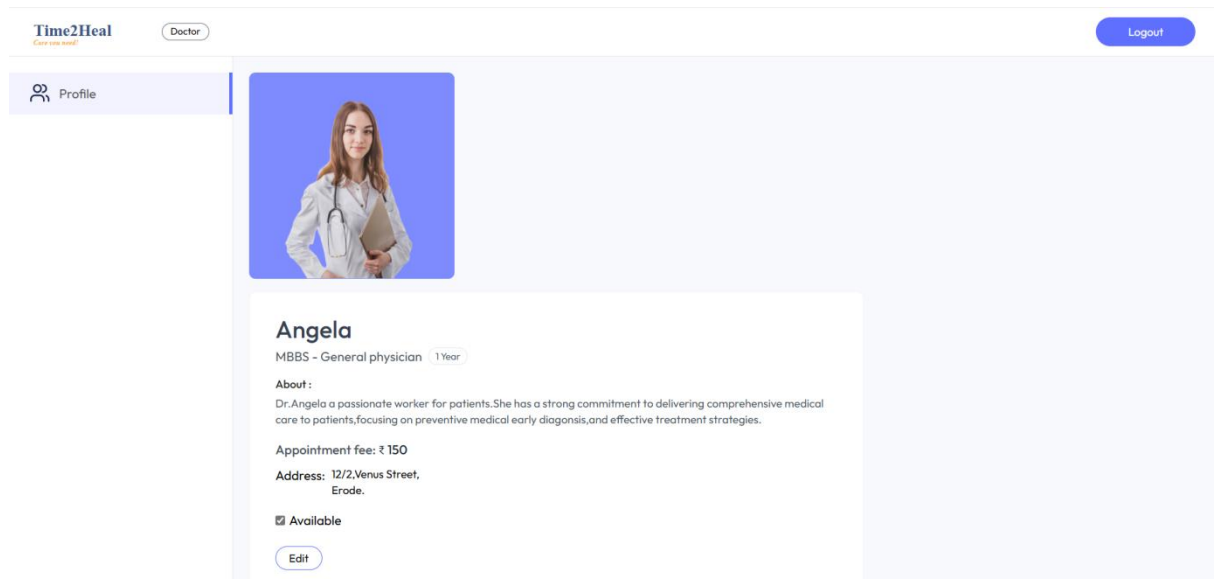


Figure B.4 Profile

ALL DOCTORS

Time2Heal

Let's get you healed!

HOME

ALL DOCTORS

ABOUT

CONTACT

General physician

Gynecologist

Dermatologist

Pediatricians

Neurologist

Gastroenterologist

Angela

General physician

Available

Dr.Kin

Dermatologist

Available

Dr.Reva

General physician

Available

Dr.Leo

General physician

Available

Dr.Kelly

Dermatologist

Available

Figure B.5 All Doctors

ALL APPOINTMENTS

Time2Heal

Let's get you healed!

Admin

Logout

Appointments

+

Add Doctor


Doctors List

All Appointments


#	Patient	Age	Date & Time	Doctor	Fees	Action
1	<div><div></div>jeeva</div>	21	18 Dec 2024, 02:00 pm	<div><div></div>Dr.Kelly</div>	₹250	<div><div></div></div>
2	<div><div></div>jeeva</div>	21	13 Dec 2024, 11:00 am	<div><div></div>Dr.Kelly</div>	₹250	<div><div></div></div>
3	<div><div></div>jeeva</div>	21	16 Dec 2024, 12:30 pm	<div><div></div>Dr.Leo</div>	₹250	Completed
4	<div><div></div>jeeva</div>	21	19 Nov 2024, 11:00 am	<div><div></div>Angela</div>	₹150	Cancelled


Figure B.6 All Appointments

DOCTORS PROFILE



[HOME](#)
[ALL DOCTORS](#)
[ABOUT](#)
[CONTACT](#)





Dr. Leo

MBBS - General physician 5 Year

About

Dr. Leo a passionate worker for patients. He has a strong commitment to delivering comprehensive medical care to patients, focusing on preventive medical early diagnosis, and effective treatment strategies.

Appointment fee: ₹250

Booking slots

FRI
13

SAT
14

SUN
15

MON
16

TUE
17

WED
18

10:00 am

10:30 am

11:00 am

11:30 am

12:00 pm

12:30 pm

01:00 pm

01:30 pm


02:00 pm

02:30 pm


Book an appointment

Figure B.7 Doctors Profile


ALL DOCTORS PROFILE



[HOME](#)
[ALL DOCTORS](#)
[ABOUT](#)
[CONTACT](#)



My appointments



Dr. Kelly


Dermatologist

Address:
12/2, Venus Street,
Erode.

Date & Time: 18 Dec 2024 | 02:00 pm

Pay Online

Cancel appointment



Dr. Kelly


Dermatologist

Address:
12/2, Venus Street,
Erode.

Date & Time: 13 Dec 2024 | 11:00 am

Pay Online

Cancel appointment



Dr. Leo

General physician

Address:
12/2, Venus Street,
Erode.

Date & Time: 16 Dec 2024 | 12:30 pm

Completed

Figure B.8 All Doctors Profile

REFERENCES

BOOKS

- [1] Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js, 2nd Edition by Shama Hoque
- [2] Pro MERN Stack: Full Stack Web App Development With Mongo, Express, React, And Node (Kindle Edition) By Vasam Subramanian
- [3] MongoDB: The Definitive Guide by Kristina Chodorow
- [4] Learning React: A Hands-On Guide to Building Web Applications Using React and Redux by Kirupa Chinnathambi
- [5] Node.js Design Patterns by Mario Casciaro

WEBSITES

- [1] <https://www.kmchhospitals.com/>
- [2] <https://www.apollohospitals.com/book-appointment/>
- [3] <https://gemhospitals.com/appointment/book-appointment>