



# IT2080 IT PROJECT

Progress Report

ITP\_2025\_Y2\_S2\_WD-155

## FeelsFIX Online Therapy Booking System

### Group Details

**Campus:** SLIIT Campus Malabe.

**Group:** Weekday Batch 07 (Group Index – 155)

	Student ID	Student Name	Email	Contact Number
1	IT23166110	D.D. Haputhanthri	it23166110@my.sliit.lk	0775907458
2	IT23275560	A.D. Athauda	it23275560@my.sliit.lk	0774341445
3	IT23423992	K.H. Dissanayake	it23423992@my.sliit.lk	0774960272
4	IT23268258	B.P.L. Fernando	it23268258@my.sliit.lk	0719930179
5	IT23257436	E.M.W.S. Ekanayake	it23257436@my.sliit.lk	0703546465

## Table of Contents

1 Introduction .....	3
2 Team Members' Progress .....	4
2 Repository Link .....	6
3 ER diagram .....	7
4 Normalized Schema.....	8
5 Network design .....	9
6 High-level system design diagram.....	10
7 Test Case Design .....	11
7.1 User Management - IT23166110 .....	11
7.2 Appointment Management- IT23268258 .....	13
7.3 Payment Management- IT23275560 .....	15
7.4 Workshop Management- IT23423992 .....	17
7.5 Wellness Content & Feedback Management- IT23257436 .....	19
8 Innovative Parts of the Project.....	21
9 User Management - Level of Progress .....	22
10 Appointment Management - Level of Progress.....	29
11 Payment Management - Level of Progress .....	41
12 Workshop Management - Level of Progress.....	48
13 Wellness Content & Feedback Management - Level of Progress.....	54

# 1 Introduction

This progress report details the development of FeelsFix, an online therapy booking system intended to bridge professionals' therapists to clients for timely access, economical and quick mental healthcare. Overview of the system Optimized for therapy session scheduling, therapist availability, collecting payments and engaging in wellness.

- **User Management:** Enables clients, therapists, and admins to create and manage accounts, verify credentials, and oversee user interactions within the platform.
- **Appointment Management:** Facilitates real-time scheduling, rescheduling, and cancellation of therapy sessions with automated reminders.
- **Payment Management:** Supports secure online payments through bank transfers, allowing users to upload payment slips for verification.
- **Workshop Management:** Allows therapists to create and manage wellness workshops while enabling clients to browse, register, and participate.
- **Wellness Content & Feedback Management:** Provides a space for therapists and users to post wellness blogs, engage in discussions, and leave feedback on sessions and therapists.

This report documents each module progress — what is already implemented, in-progress development, individual contributions and aspects of the system like test cases, algorithms, pseudocode, system architecture diagrams etc. through which work has been performed. The aim is to make sure everything runs towards objectives and milestones of the project.

## 2 Team Members' Progress

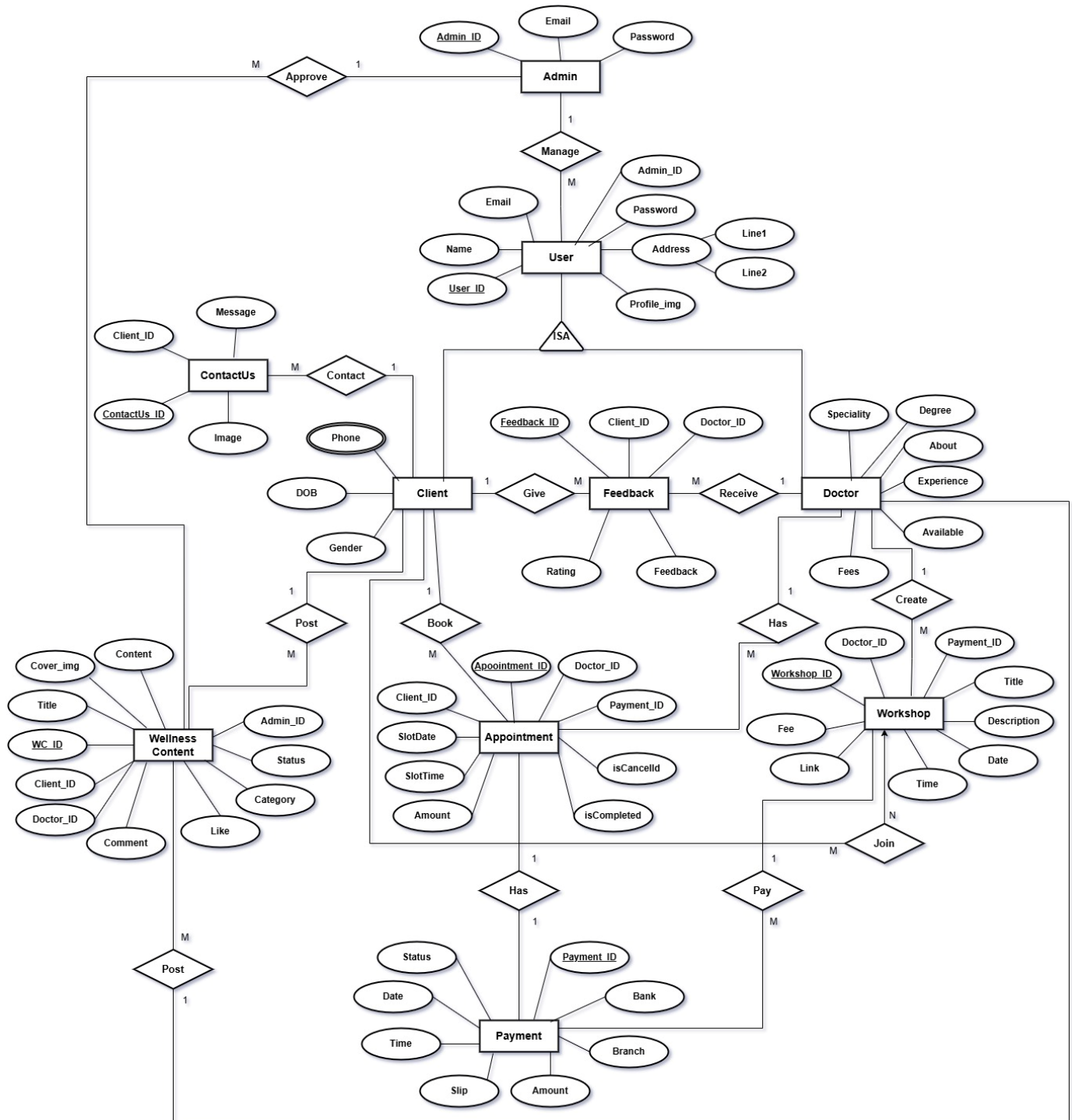
Name of the Member	Functionalities Developed	Functionalities to be Completed	Repository
<b>D.D. Haputhanthri</b> <b>(User Management)</b>	<ul style="list-style-type: none"> <li>• User Registration &amp; Authentication</li> <li>• Profile Management</li> <li>• Role-Based Access Control (Users, Therapists, Admins)</li> <li>• Database Integration &amp; Data Validation</li> <li>• Session &amp; Security Management</li> </ul>	<ul style="list-style-type: none"> <li>• Email Verification &amp; Account Activation</li> <li>• Password Reset &amp; Recovery System</li> </ul>	
<b>B.P.L. Fernando</b> <b>(Appointment Management)</b>	<ul style="list-style-type: none"> <li>• Therapist Profile &amp; Availability Display</li> <li>• Appointment Booking System</li> <li>• Booking Confirmation &amp; Notifications</li> <li>• Upcoming Appointments Tracking</li> <li>• Session Completion Status</li> </ul>	<ul style="list-style-type: none"> <li>• Advanced Therapist Search &amp; Filtering</li> <li>• Automated Notifications &amp; Logs</li> <li>• Enhanced Appointment Sorting &amp; Management</li> <li>• Post-Session Feedback System</li> <li>• Admin Alerts &amp; Monitoring</li> </ul>	
<b>A.D. Athauda</b> <b>(Payment Management)</b>	<ul style="list-style-type: none"> <li>• Secure Bank Slip Upload &amp; Storage</li> <li>• Session Fee Display &amp; Payment</li> </ul>	<ul style="list-style-type: none"> <li>• Transaction History &amp; Payment Tracking</li> <li>• Clear &amp; Transparent Payment Display</li> </ul>	

	<p>Confirmation</p> <ul style="list-style-type: none"> <li>• Admin Verification &amp; Approval System</li> <li>• Automated Booking Status Updates</li> <li>• Admin Dashboard for Payment Management</li> </ul>	<ul style="list-style-type: none"> <li>• User Payment Notifications</li> </ul>	
<p><b>K.H. Dissanayake</b></p> <p><b>(Workshop Management)</b></p>	<ul style="list-style-type: none"> <li>• Workshop Creation &amp; Management</li> <li>• Workshop Listing &amp; Display</li> <li>• Workshop Availability Tracking</li> </ul>	<ul style="list-style-type: none"> <li>• Advanced Search &amp; Filtering</li> <li>• Workshop Registration System</li> <li>• Confirmation &amp; Notifications</li> <li>• Automated Reminder System</li> </ul>	
<p><b>E.M.W.S. Ekanayake</b></p> <p><b>(Wellness Content &amp; Feedback Management)</b></p>	<ul style="list-style-type: none"> <li>• Blog Post Submission System</li> <li>• Approval &amp; Status Tracking</li> <li>• Feedback Management</li> </ul>	<ul style="list-style-type: none"> <li>• Draft Saving for Blog Posts</li> <li>• Admin Notification System for New Submissions</li> <li>• Feedback Notification System for Therapists/Admins</li> <li>• Therapist Rating Calculation Algorithm</li> <li>• Featured Section for Top-Rated</li> </ul>	

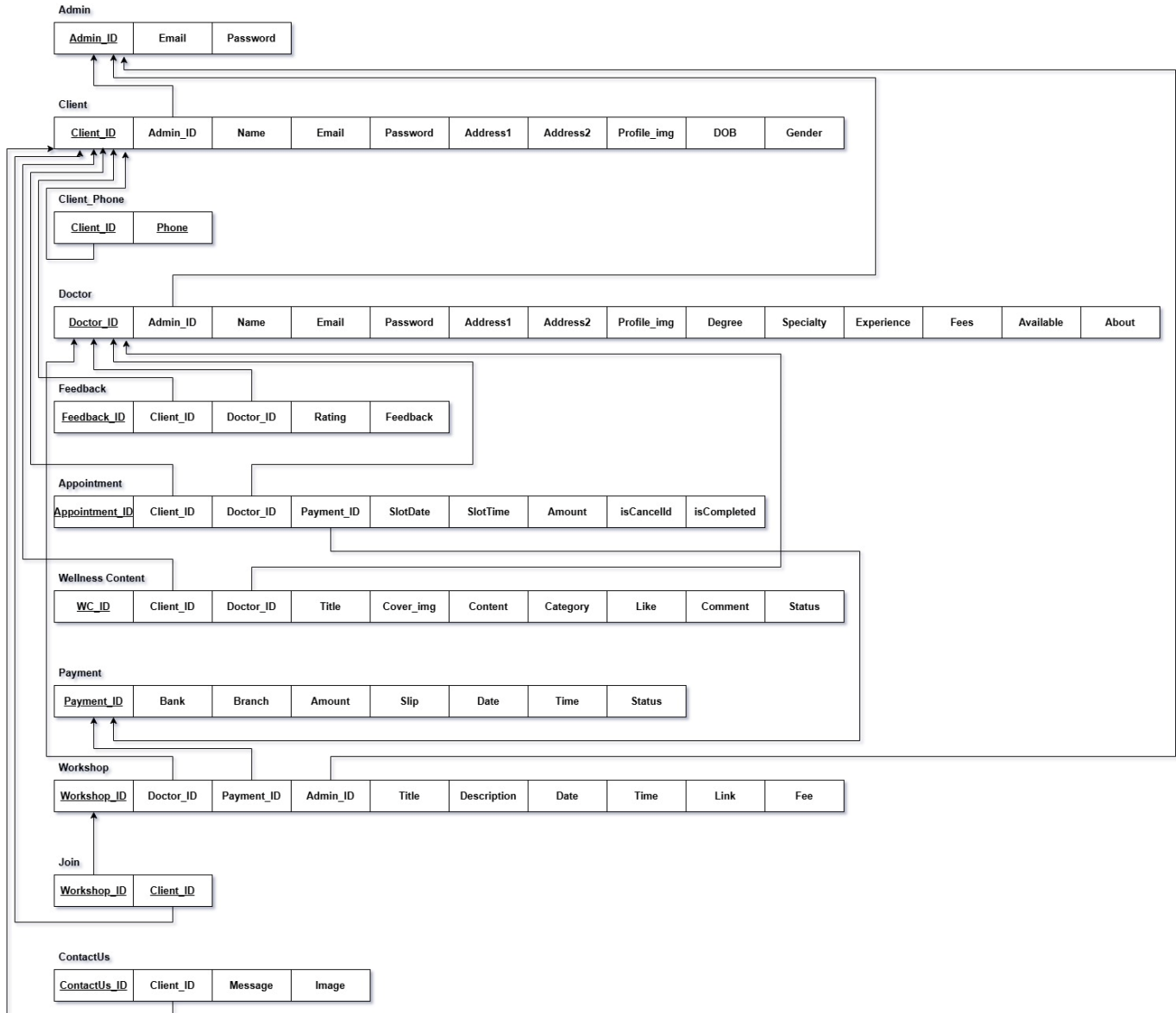
		Therapists	
		<ul style="list-style-type: none"><li>• Automated Post Status Notifications</li></ul>	

**2 Repository Link**

### 3 ER diagram

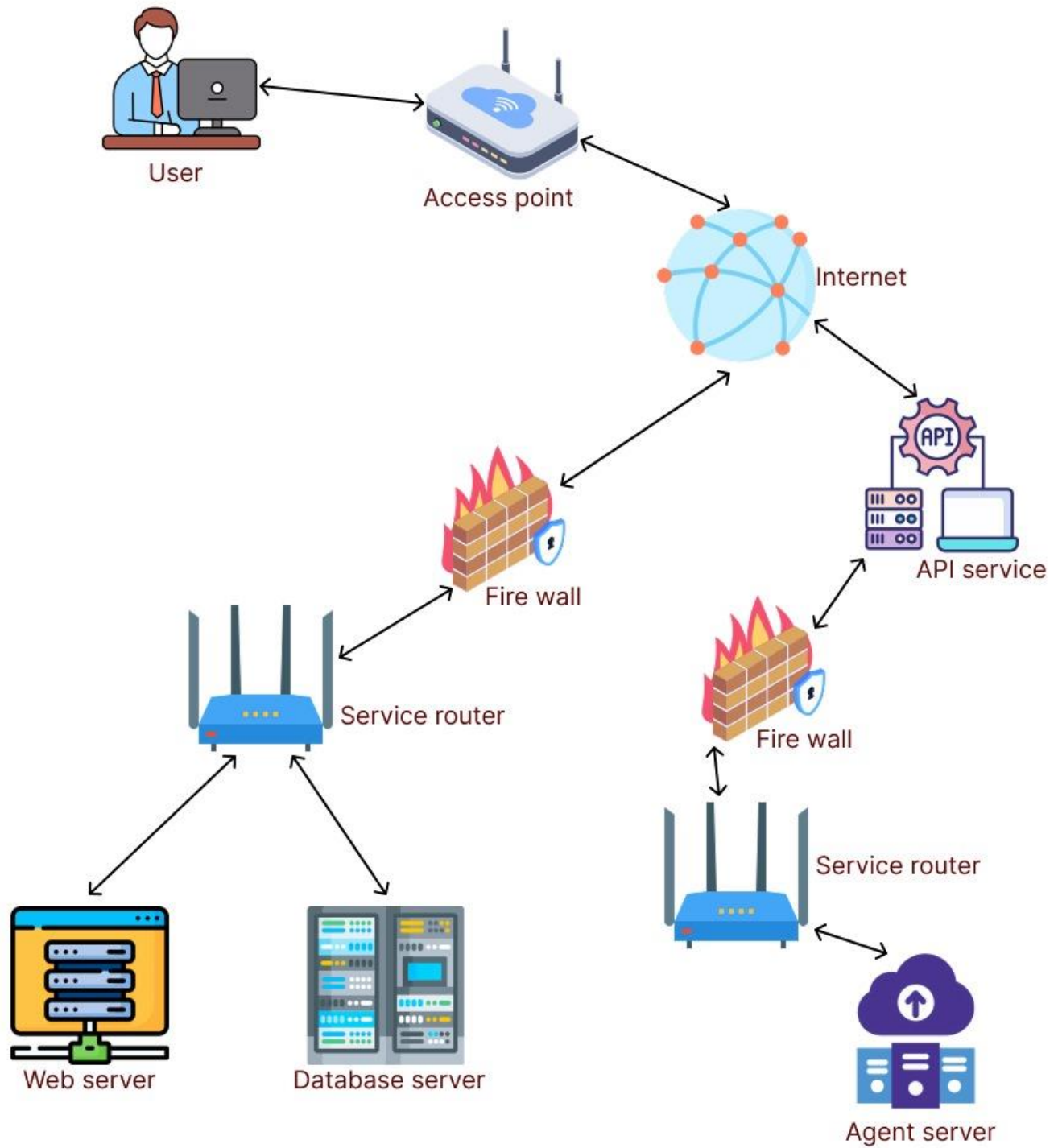


## 4 Normalized Schema

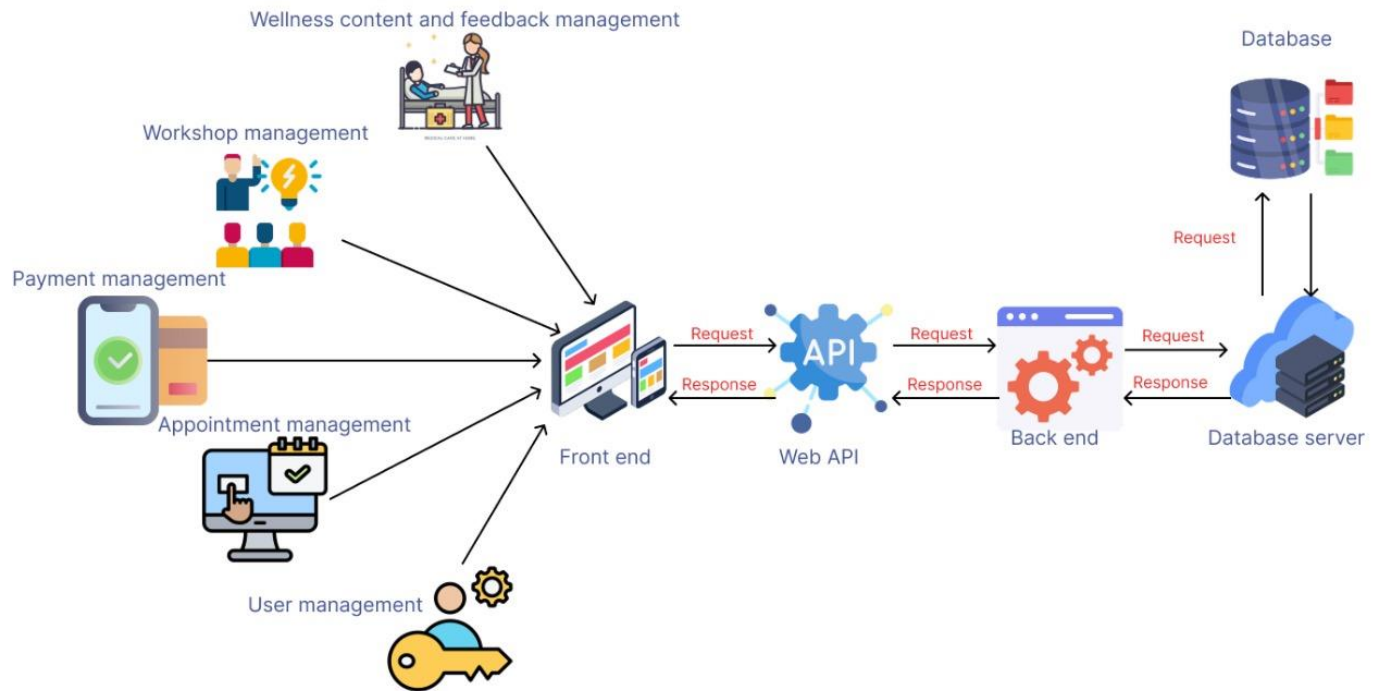




## 5 Network design



## 6 High-level system design diagram



## 7 Test Case Design

### 7.1 User Management – IT23166110

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Completed (Yes/No)
UM_TC_01	User registration process	1. Navigate to the sign-up page. 2. Fill in valid user details. 3. Submit registration.	The user account should be created successfully, and an email verification link should be sent.	The user account should be created successfully, and an email verification link should be sent.	
UM_TC_02	Admin approval of therapist profile	1. Admin logs into the admin panel. 2. Views therapist profile. 3. Clicks "approve" for the therapist's profile.	Therapist profile should be approved, and the therapist is notified of the approval.	Therapist profile should be approved, and the therapist is notified of the approval.	
UM_TC_03	User password reset	1. Navigate to the password reset page. 2. Enter registered email. 3. Follow the instructions in the email to reset password.	Passwords should be reset successfully, and the user should be able to log in with the new password.	Passwords should be reset successfully, and the user should be able to log in with the new password.	

<b>UM_TC_04</b>	Account deletion request	1. User submits an account deletion request. 2. Admin receives and reviews the request. 3. Admin approves the deletion.	The user account should be deleted successfully after admin approval.	The user account should be deleted successfully after admin approval.	
<b>UM_TC_05</b>	User suspension for policy violation	1. Admin reviews reported misconduct or violation. 2. Admin suspends the user account. 3. Confirm action.	The user should be suspended and should no longer have access to the platform.	The user should be suspended and should no longer have access to the platform.	

## 7.2 Appointment Management– IT23268258

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Completed (Yes/No)
AM_TC_01	Book an appointment	1. Client logs in. 2. Selects a therapist and time slot. 3. Chooses a communication mode. 4. Confirms booking.	Appointment is successfully booked and confirmation is sent via email/SMS.	Appointment is successfully booked and confirmation is sent via email/SMS.	
AM_TC_02	Reschedule an appointment	1. Client navigates to upcoming appointments. 2. Select an appointment. 3. Picks a new available time slot. 4. Saves changes.	Appointment is rescheduled, and both therapist and client receive updated details.	Appointment is rescheduled, and both therapist and client receive updated details.	
AM_TC_03	Therapist approves or declines appointment	1. Therapist logs in. 2. Views pending appointment requests. 3. Approves or declines a request.	Client receives notification of the therapist's decision.	Client receives notification of the therapist's decision.	
AM_TC_04	Cancel an appointment	1. Client selects an appointment.	Appointment is canceled; cancellation	Appointment is canceled; cancellation	

		2. Click "Cancel" and provides reason (if applicable). 3. Confirms action.	reason is saved; therapist is notified.	reason is saved; therapist is notified.	
<b>AM_TC_05</b>	No-show detection & admin alert	1. Session time passes without therapist joining. 2. System detects absence. 3. Send alert to admin.	Admin is notified of the therapist no-show for follow-up action.	Admin is notified of the therapist no-show for follow-up action.	

### 7.3 Payment Management– IT23275560

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Completed (Yes/No)
PM_TC_01	Upload bank transfer slip	1. User navigates to payment section. 2. Uploads valid bank slip. 3. Submits for verification.	Slip is uploaded successfully and marked as pending approval.	Slip is uploaded successfully and marked as pending approval.	
PM_TC_02	Admin approves/rejects slip	1. Admin logs in. 2. Views pending slips. 3. Reviews and approves/rejects the slip.	User is notified of approval or rejection; booking status updates accordingly.	User is notified of approval or rejection; booking status updates accordingly.	
PM_TC_03	View and track payment history	1. User logs in. 2. Navigates payment history. 3. Reviews of past payments and statuses.	Users can view all completed, pending, and refunded payments.	Users can view all completed, pending, and refunded payments.	
PM_TC_04	Therapist receives automated payout	1. System verifies confirmed session. 2. Checks payment is completed. 3. Initiates payout process.	Therapist receives notification of payout; funds processed if automated.	Therapist receives notification of payout; funds processed if automated.	

<b>PM_TC_05</b>	Apply discount or promo code	1. User books a session or subscribes. 2. Enters valid promo code. 3. Applies code.	The discount is calculated, and an updated amount is shown before payment.	The discount is calculated, and an updated amount is shown before payment.	
-----------------	------------------------------	---	--	--	--



## 7.4 Workshop Management– IT23423992

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Completed (Yes/No)
WM_TC_01	Browse and view available workshops	1. User logs in. 2. Navigates to workshops section. 3. Views list of available workshops.	Workshop listings are displayed with details like title, date, and therapist.	Workshop listings are displayed with details like title, date, and therapist.	
WM_TC_02	Register for a workshop	1. User selects a workshop. 2. Clicks "Register". 3. Confirms registration.	User receives a successful message and confirmation email.	User receives a successful message and confirmation email.	
WM_TC_03	Admin approves workshop created by therapist	1. Therapist creates workshop. 2. Admin logs in and reviews pending workshops. 3. Approves or rejects listing.	Approved workshop becomes visible to users; rejected ones remain hidden.	Approved workshop becomes visible to users; rejected ones remain hidden.	
WM_TC_04	Receive workshop reminder	1. User registers for a workshop. 2. Time approaches. 3. The system sends automated reminders via email/notification.	User receives a timely reminder before the session.	User receives a timely reminder before the session.	
WM_TC_05	Cancel workshop registration	1. User goes to registered workshops. 2. Clicks "Cancel	Users are unregistered, and slot is reopened for	Users are unregistered, and slot is reopened for	

		Registration”. 3. Confirms cancellation.	others.	others.	
--	--	--	---------	---------	--

## 7.5 Wellness Content & Feedback Management– IT23257436

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Completed (Yes/No)
WC_TC_01	Submit blog post for approval	1. User logs in. 2. Navigates to blog section. 3. Creates and submits a post.	Post is marked as "Pending Approval" and not publicly visible until approved.	Post is marked as "Pending Approval" and not publicly visible until approved.	
WC_TC_02	Admin approves or rejects blog post	1. Admin logs in. 2. Opens pending blog posts. 3. Approves or rejects a post.	Post is either published or rejected with notification sent to the user.	Post is either published or rejected with notification sent to the user.	
WC_TC_03	Submit feedback on therapist/session/workshop	1. User logs in. 2. Goes to feedback page. 3. Selects therapist/session/workshop. 4. Submits feedback.	Feedback is saved and visible to an admin and therapist if needed.	Feedback is saved and visible to an admin and therapist if needed.	
WC_TC_04	Report inappropriate blog content or comment	1. User logs in. 2. Navigates to a blog post. 3. Clicks "Report". 4. Provides reason and submits report.	Report is sent to admin for review.	Report is sent to admin for review.	

<b>WC_TC_05</b>	Like, comment, and share a blog post	1. User browses published blog posts. 2. Likes, comments, or shares any blog post.	Interaction is recorded and visible publicly (except shares, if external).	Interaction is recorded and visible publicly (except shares, if external).	
-----------------	--------------------------------------	---	--	--	--

## **8 Innovative Parts of the Project**

### **1. Personalized Therapist Recommendation**

- Suggests therapists based on user preferences, past sessions, and feedback.
- Helps clients find the most suitable therapist quickly.

### **2. Dynamic Pricing for Therapy Sessions**

- Implements pricing variations based on therapist expertise, session type, and peak hours.
- Allows for cost-effective options like group sessions or off-peak discounts.

### **3. Wellness Content & Interactive Blog**

- Users and therapists can share mental health articles, tips, and experiences.
- Admin-approved posts ensure quality content and credibility.
- Allows users to like, comment, and share posts for engagement.

### **4. Subscription-Based Therapy Plans**

- Offers monthly or annual plans for users who require regular therapy.
- Provides cost-effective packages for ongoing mental health support.

### **5. Therapist Availability & Instant Booking System**

- Displays real-time therapist availability with time slot selection.
- Users can instantly book sessions without waiting for manual approvals.

### **6. Bank Transfer-Based Payment with Slip Verification**

- Secure payment system where users upload proof of bank transfer.
- Admin manually verifies transactions, reducing fraud risks.

### **7. Automated Session Reminders & Follow-Ups**

- Sends SMS & email reminders before sessions.
- Triggers post-session feedback requests to improve therapist performance.

### **8. Comprehensive User & Therapist Dashboard**






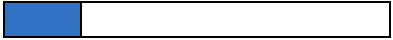

- Users can track past and upcoming sessions, payments, and wellness content.
- Therapists manage their schedule, earnings, and client interactions in one place.

### **9. Workshop Management with Attendance Tracking**

- Enables therapists to organize mental health workshops.
- Tracks participant attendance and engagement for effective learning.

## 9 User Management – Level of Progress

### 9.1 Completion Level of the Component

Component	Description	Progress
User Registration UI	Designing and implementing a user-friendly registration interface.	 100%
Login & Authentication	Implementing secure login with session management.	 90%
Profile Management	Allowing users to update profile details, including profile pictures.	 90%
Password Reset & OTP Verification	Implementing password recovery via email with OTP verification.	 20%
Role-Based Access Control	Managing different user roles (Admin, Therapist, Client).	 90%
Account Verification	Send verification emails and updating user status after verification.	 20%
User Activity Logs	Tracking user activity for security and management purposes.	 20%

## 9.2 Functionalities need to be Implemented

### ❖ Account Verification

**Description:** Ensuring that only genuine users can access the platform by verifying their accounts through email confirmation.

**Features:**

- Generate and send a verification email with a unique token.
- Provide a verification link for users to activate their accounts.
- Update user status after successful verification.
- Display success or failure messages based on verification status.

### ❖ User Activity Logs

**Description:** Tracking user actions on the platform to enhance security and provide insights into platform usage.

**Features:**

- Log user login and logout activities.
- Track important user interactions (e.g., booking sessions, posting feedback).
- The store logs in a secure database for reference.
- Allow admin access to view logs for monitoring suspicious activities.

### ❖ Password Reset & OTP Verification

**Description:** Allowing users to securely reset their passwords in case of forgotten credentials.

**Features:**

- Generate and send a **password reset token** via email.
- Create a **password reset UI** for users to enter a new password.
- Validate **OTP (One-Time Password)** before allowing password reset.
- Update the **new password** in the database after successful verification.

## 9.3 Queries Used

### User Inquiry Management in Node.js and MongoDB

```
4  const createContact = async (request, response) => {
5    try {
6      if (
7        !request.body.name ||
8        !request.body.email ||
9        !request.body.phone ||
10       !request.body.message
11      ) {
12        return response.status(400).send({
13          message: "Send all required fields: name, email, phone, message",
14        });
15      }
16
17      const newContact = {
18        name: request.body.name,
19        email: request.body.email,
20        phone: request.body.phone,
21        message: request.body.message,
22        photo: request.file ? request.file.filename : null,
23      };
24
25      const contact = await Contact.create(newContact);
26
27      return response.status(201).send(contact);
28    } catch (error) {
29      console.log(error.message);
30      response.status(500).send({ message: error.message });
31    }
32  };
33
```



## Fetching and Displaying User Inquiries in a Node.js User Management System

```
35 const getAllContacts = async (request, response) => {
36   try {
37     const contacts = await Contact.find({}).sort({ createdAt: -1 }); // Sort by newest first
38
39     return response.status(200).json({
40       count: contacts.length,
41       data: contacts,
42     });
43   } catch (error) {
44     console.log(error.message);
45     response.status(500).send({ message: error.message });
46   }
47 };
48
```

## Retrieving a Specific User Inquiry by ID from MongoDB

```
50 const getContactById = async (request, response) => {
51   try {
52     const { id } = request.params;
53
54     const contact = await Contact.findById(id);
55
56     if (!contact) {
57       return response
58         .status(404)
59         .json({ message: "Contact message not found" });
60     }
61
62     return response.status(200).json(contact);
63   } catch (error) {
64     console.log(error.message);
65     response.status(500).send({ message: error.message });
66   }
67 };
68
```

## Updating User Inquiries in a MongoDB-Based Inquiry Management System

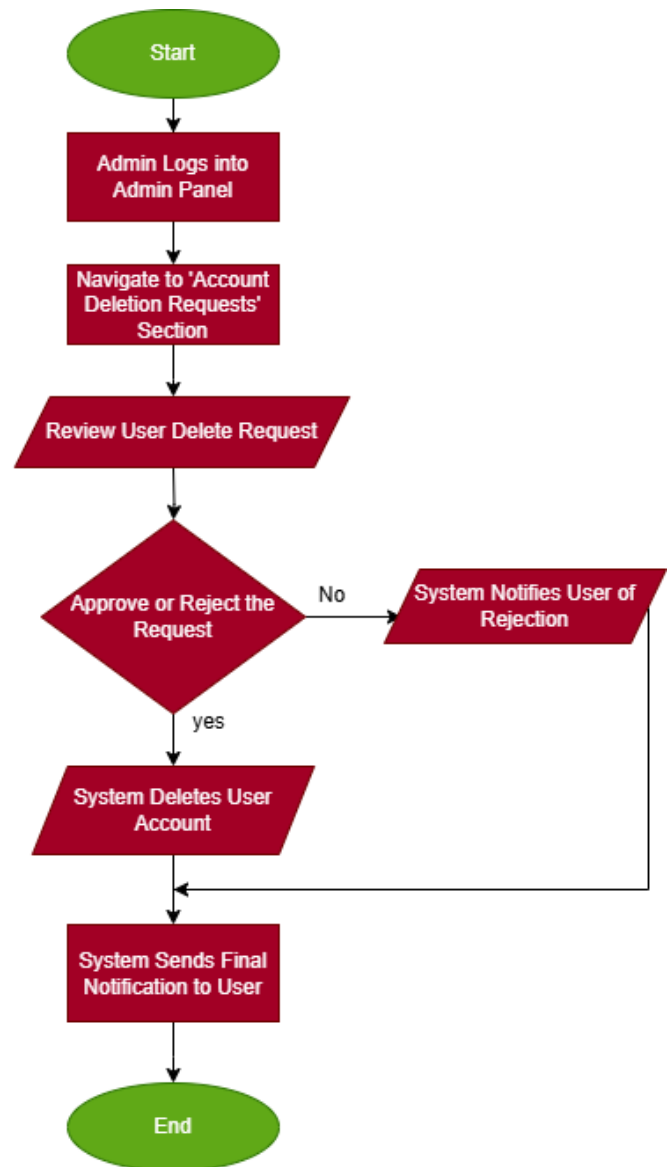
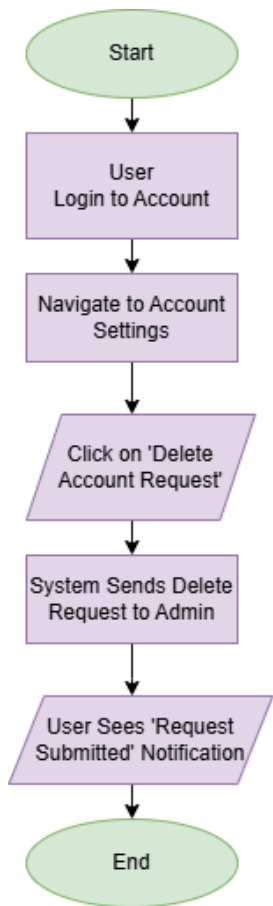
```
70 ✓ const updateContact = async (request, response) => {
71 ✓   try {
72     if (
73       !request.body.name ||
74       !request.body.email ||
75       !request.body.phone ||
76       !request.body.message
77 ✓   ) {
78     return response.status(400).send({
79       message: "Send all required fields: name, email, phone, message",
80     });
81   }
82
83   const { id } = request.params;
84   const { name, email, phone, message } = request.body;
85   const photo = request.file ? request.file.filename : null;
86
87 ✓   const result = await Contact.findByIdAndUpdate(id, {
88     name,
89     email,
90     phone,
91     message,
92     ...(photo && { photo }), // Only update photo if a new one is uploaded
93   });
94
95 ✓   if (!result) {
96     return response
97       .status(404)
98       .json({ message: "Contact message not found" });
99   }
100
101   return response
```

```
100
101     return response
102       .status(200)
103       .send({ message: " message updated successfully" });
104 ✓   } catch (error) {
105     console.log(error.message);
106     response.status(500).send({ message: error.message });
107   }
108 };
109
```

## Deleting User Inquiries from a MongoDB-Based Inquiry Management System










```
111  ✓ const deleteContact = async (request, response) => {  
112  ✓    try {  
113      const { id } = request.params;  
114  
115      const result = await Contact.findByIdAndDelete(id);  
116  
117  ✓    if (!result) {  
118      return response  
119      .status(404)  
120      .json({ message: "Contact message not found" });  
121    }  
122  
123    return response  
124    .status(200)  
125    .send({ message: "Contact message deleted successfully" });  
126  ✓  } catch (error) {  
127    console.log(error.message);  
128    response.status(500).send({ message: error.message });  
129  }  
130  };  
131
```

## 9.4 Flow charts regarding User Management



## 10 Appointment Management – Level of Progress

### 10.1 Completion Level of the Component

Component	Description	Progress
<b>Appointment Booking UI</b>	Designing and implementing a user-friendly interface for booking appointments	 100%
<b>Appointment Management</b>	Managing appointments <ul style="list-style-type: none"> <li>• Create an appointment</li> <li>• View appointment</li> <li>• Cancel appointment</li> <li>• Mark as completed appointment</li> </ul>	 100%
<b>Form Validation</b>	Implementing validation and restrictions for forms.	 90%
<b>Therapist Form UI</b>	Designing a user-friendly form for therapist registration/editing.	 100%
<b>Therapist Management</b>	Managing therapist profiles and data.	 90%
<b>Therapist Availability</b>	Managing therapist schedules and availability.	 100%
<b>Therapist Dashboard</b>	A dashboard for therapists to view appointments, income and manage profiles.	 70%
<b>Therapist Search &amp; Filter</b>	Enabling users to search and filter therapists. <ul style="list-style-type: none"> <li>• Search by name</li> <li>• Filters by specialty</li> </ul>	 50%
<b>Admin Report Generation</b>	Generating reports for admin insights.	 20%

## **10.2 Functionalities need to be Implemented**

### **1. Therapist Dashboard**

**Description:** A dashboard for therapists to view appointments, income and manage profiles.

**Features:**

- Upcoming appointments list
- Earnings
- Availability management

### **2. Therapist Search & Filter**

**Description:** Enabling users to search and filter therapists.

**Features:**

- Search by name
- Filter by specialty

### **3. Admin Report Generation**

**Description:** Generating reports for admin insights.

**Features:**

- Appointment statistics (monthly/weekly)
- Revenue reports
- Therapist Details Sheet

## 10.3 Queries Used

### MongoDB Data Fetching Functions in Appointment management

```
//API to book appointment
const bookAppointment = async (req, res) => {
  try {
    const { userId, docId, slotDate, slotTime } = req.body;
    const docData = await doctorModel.findById(docId).select("-password");
    if (!docData.available) {
      return res.json({ success: false, message: "Doctor not Available" });
    }
    let slots_booked = docData.slots_booked;

    //checking for slot availability
    if (slots_booked[slotDate]) {
      if (slots_booked[slotDate].includes(slotTime)) {
        return res.json({ success: false, message: "Slot not Available" });
      } else {
        slots_booked[slotDate].push(slotTime);
      }
    } else {
      slots_booked[slotDate] = [];
      slots_booked[slotDate].push(slotTime);
    }

    const userData = await userModel.findById(userId).select("-password");
    delete docData.slots_booked;
    const appointmentData = {
      userId,
      docId,
      userData,
      docData,
      amount: docData.fees,
      slotTime,
      slotDate,
      date: Date.now(),
    };

    const newAppointment = new appointmentModel(appointmentData);
    await newAppointment.save();

    //save new slots data in docData
    await doctorModel.findByIdAndUpdate(docId, { slots_booked });
    res.json({ success: true, message: "Appointment Booked" });
  } catch (error) {
```

```

    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

```

```

//API to get user appointment
const listAppointment = async (req, res) => {
  try {
    const { userId } = req.body;
    const appointments = await appointmentModel.find({ userId });
    res.json({ success: true, appointments });
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

//API to cancel appointment
const cancelAppointment = async (req, res) => {
  try {
    const { userId, appointmentId } = req.body;
    const appointmentData = await appointmentModel.findById(appointmentId);
    //verify appointment user
    if (appointmentData.userId !== userId) {
      return res.json({ success: false, message: "Unauthorized action" });
    }

    await appointmentModel.findByIdAndUpdate(appointmentId, {
      cancelled: true,
    });

    //releasing doctor slot
    const { docId, slotDate, slotTime } = appointmentData;
    const doctorData = await doctorModel.findById(docId);
    let slots_booked = doctorData.slots_booked;
    slots_booked[slotDate] = slots_booked[slotDate].filter(
      (e) => e !== slotTime
    );
    await doctorModel.findByIdAndUpdate(docId, { slots_booked });
    res.json({ success: true, message: "Appointment Cancelled" });
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

```



## MongoDB Data Fetching Functions in Therapist management

```
//API for doctor login
const loginDoctor = async (req, res) => {
  try {
    const { email, password } = req.body;
    const doctor = await doctorModel.findOne({ email });
    if (!doctor) {
      return res.json({ success: false, message: "Invalid credentials" });
    }
    const isMatch = await bcrypt.compare(password, doctor.password);
    if (isMatch) {
      const token = jwt.sign({ id: doctor._id }, process.env.JWT_SECRET);
      res.json({ success: true, token });
    } else {
      return res.json({ success: false, message: "Invalid credentials" });
    }
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

//API to get doctor appointments for doctor panel
const appointmentsDoctor = async (req, res) => {
  try {
    const { docId } = req.body;
    const appointments = await appointmentModel.find({ docId });
    res.json({ success: true, appointments });
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

//API to mark appointment completed for doctor panel
const appointmentComplte = async (req, res) => {
  try {
    const { docId, appointmentId } = req.body;
    const appointmentData = await appointmentModel.findById(appointmentId);
    if (appointmentData && appointmentData.docId === docId) {
      await appointmentModel.findByIdAndUpdate(appointmentId, {
        isCompleted: true,
      });
      return res.json({ success: true, message: "Appointment Completed" });
    } else {
      return res.json({ success: false, message: "Mark Failed" });
    }
  }
};
```

```

    }
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

//API to cancel appointment for doctor panel
const appointmentCancel = async (req, res) => {
  try {
    const { docId, appointmentId } = req.body;
    const appointmentData = await appointmentModel.findById(appointmentId);
    if (appointmentData && appointmentData.docId === docId) {
      await appointmentModel.findByIdAndUpdate(appointmentId, {
        cancelled: true,
      });
      return res.json({ success: true, message: "Appointment Cancelled" });
    } else {
      return res.json({ success: false, message: "Cancellation Failed" });
    }
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

//API to get dashboard data for doctor panel
const doctorDashboard = async (req, res) => {
  try {
    const { docId } = req.body;
    const appointments = await appointmentModel.find({ docId });
    let earnings = 0;
    appointments.map((item) => {
      if (item.isCompleted || item.payment) {
        earnings += item.amount;
      }
    });

    let patients = [];
    appointments.map((item) => {
      if (!patients.includes(item.userId)) {
        patients.push(item.userId);
      }
    });

    const dashData = {
      earnings,
      appointments: appointments.length,
    };
  }
};

```

```

    patients: patients.length,
    latestAppointments: appointments.reverse().slice(0, 5),
  };

  res.json({ success: true, dashData });
} catch (error) {
  console.log(error);
  res.json({ success: false, message: error.message });
}
};

//API to get doctor profile for doctor panel
const doctorProfile = async (req, res) => {
  try {
    const { docId } = req.body;
    const profileData = await doctorModel.findById(docId).select("-password");
    res.json({ success: true, profileData });
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

//API to update doctor profile data from doctol panel
const updateDoctorProfile = async (req, res) => {
  try {
    const { docId, fees, address, available } = req.body;
    await doctorModel.findByIdAndUpdate(docId, { fees, address, available });
    res.json({ success: true, message: "Profile Updated" });
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

```

## MongoDB Data Fetching Functions in Admin Dashboard

```
//API for adding doctor
const addDoctor = async (req, res) => {
  try {
    const {
      name,
      email,
      password,
      speciality,
      degree,
      experience,
      about,
      fees,
      address,
    } = req.body;
    const imageFile = req.file;

    //checkin for all data to add doctor
    if (
      !name ||
      !email ||
      !password ||
      !speciality ||
      !degree ||
      !experience ||
      !about ||
      !fees ||
      !address
    ) {
      return res.json({ success: false, message: "Missing Details" });
    }

    //validating email format
    if (!validator.isEmail(email)) {
      return res.json({
        success: false,
        message: "Please enter a valid email",
      });
    }

    //validating strong password
    if (password.length < 8) {
      return res.json({
        success: false,
        message: "Please enter a strong password",
      });
    }
  }
}
```

```

//hashing doctor password
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);

//upload image to cloudinary
const imageUpload = await cloudinary.uploader.upload(imageFile.path, {
  resource_type: "image",
});
const imageUrl = imageUpload.secure_url;

const doctorData = {
  name,
  email,
  image: imageUrl,
  password: hashedPassword,
  speciality,
  degree,
  experience,
  about,
  fees,
  address: JSON.parse(address),
  date: Date.now(),
};

const newDoctor = new doctorModel(doctorData);
await newDoctor.save();

res.json({ success: true, message: "Doctor Added" });
} catch (error) {
  console.log(error);
  res.json({ success: false, message: error.message });
}
};

//API for admin login
const loginAdmin = async (req, res) => {
  try {
    const { email, password } = req.body;
    if (
      email === process.env.ADMIN_EMAIL &&
      password === process.env.ADMIN_PASSWORD
    ) {
      const token = jwt.sign(email + password, process.env.JWT_SECRET);
      res.json({ success: true, token });
    } else {
      res.json({ success: false, message: "Invalid Credentials" });
    }
  }
};

```

```

} catch (error) {
  console.log(error);
  res.json({ success: false, message: error.message });
}
};

//API to get all doctor
const allDoctors = async (req, res) => {
  try {
    const doctors = await doctorModel.find({}).select("-password");
    res.json({ success: true, doctors });
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

//API to get all appointment list
const appointmentsAdmin = async (req, res) => {
  try {
    const appointments = await appointmentModel.find({});
    res.json({ success: true, appointments });
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

//API for appointment cancellation
const appointmentCancel = async (req, res) => {
  try {
    const { appointmentId } = req.body;
    const appointmentData = await appointmentModel.findById(appointmentId);

    await appointmentModel.findByIdAndUpdate(appointmentId, {
      cancelled: true,
    });

    //releasing doctor slot
    const { docId, slotDate, slotTime } = appointmentData;
    const doctorData = await doctorModel.findById(docId);
    let slots_booked = doctorData.slots_booked;
    slots_booked[slotDate] = slots_booked[slotDate].filter(
      (e) => e !== slotTime
    );
    await doctorModel.findByIdAndUpdate(docId, { slots_booked });
    res.json({ success: true, message: "Appointment Cancelled" });
  } catch (error) {

```

```

    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

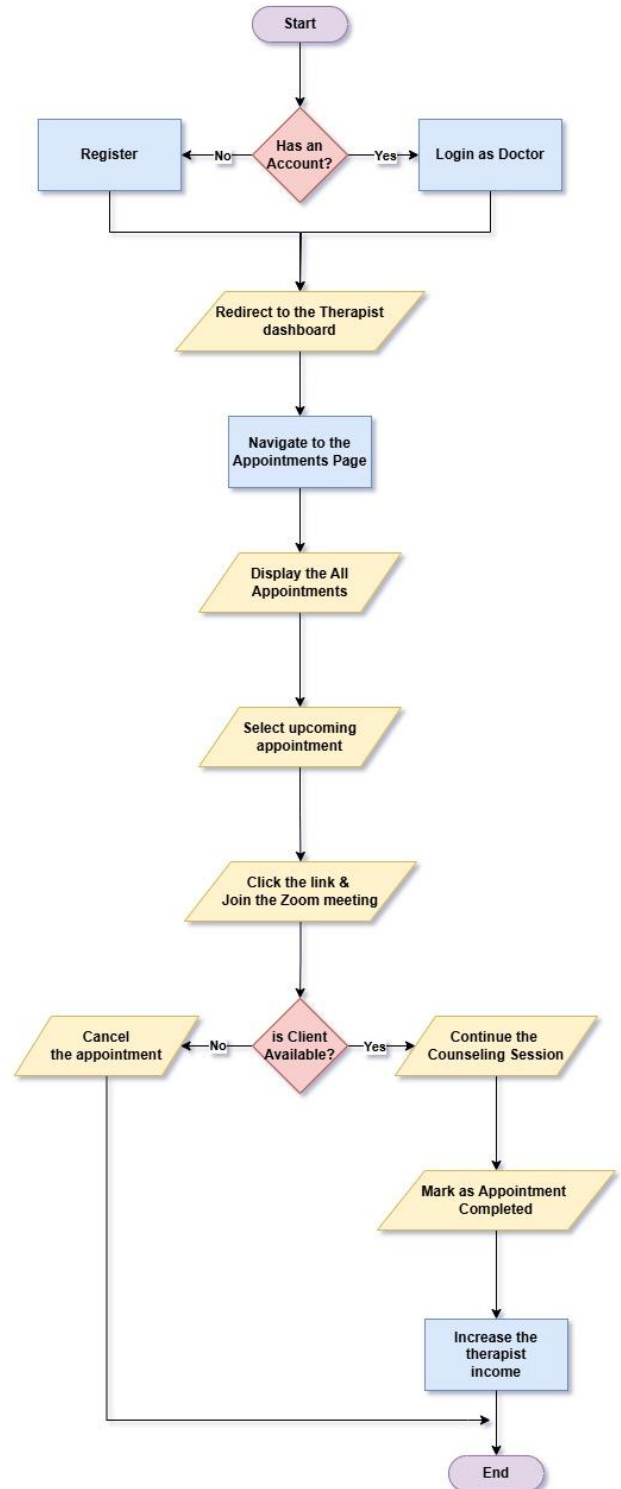
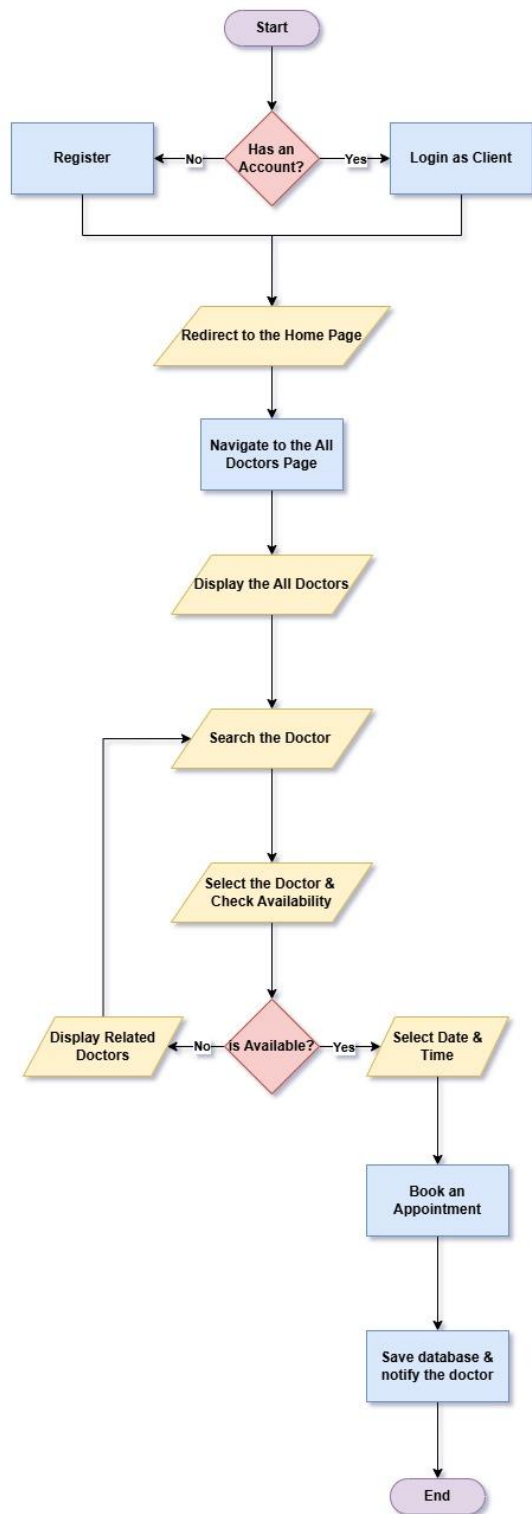
//API to get dashboard data for admin panel
const adminDashboard = async (req, res) => {
  try {
    const doctors = await doctorModel.find({});
    const users = await userModel.find({});
    const appointments = await appointmentModel.find({});

    const dashData = {
      doctors: doctors.length,
      appointments: appointments.length,
      patients: users.length,
      latestAppointments: appointments.reverse().slice(0, 5),
    };

    res.json({ success: true, dashData });
  } catch (error) {
    console.log(error);
    res.json({ success: false, message: error.message });
  }
};

```




## 10.4 Flow charts regarding Appointment Management





## 11 Payment Management – Level of Progress

### 11.1 Completion Level of the Component

Component	Description	Progress
Bank Transfer Slip Upload System	upload payment proof after making a bank transfer	 (100%)
Notification System	Keeps users updated on payment status.	 (50%)
Payment Verification & Approval (Admin)	reviews and approves payments.	 (100%)

### 11.2 Functionalities need to be Implemented

- Refund & Cancellation Handling:
  - Admin manages "*refunds*" based on cancellation policies.
- Financial Reporting (Admin Panel):
  - Help admin track *revenue and payments*.

## 11.3 Queries Used

### MongoDB Data Fetching Functions in Finance Management System

- Fetch All Payment Records (Payment.find())

Fetch all payments for display:

```
// Get All Payments
const getPayments = async (req, res) => {
  try {
    const payments = await Payment.find().sort({ createdAt: -1 });
    res.status(200).json(payments);
  } catch (error) {
    console.error("Fetch error:", error);
    res.status(500).json({ error: "Failed to fetch payments." });
  }
};
```

Display payments in the admin panel:

```
const fetchPayments = async () => {
  try {
    setIsLoading(true);
    const response = await axios.get('http://localhost:5000/api/payments');
    setPayments(response.data);
  } catch (error) {
    console.error('Error fetching payments:', error);
    setMessage('Failed to load payments. Please try again.');
```

```
  } finally {
    setIsLoading(false);
  }
};

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

## Payment.find()

- Retrieves **all** payment records from the MongoDB payments collection, sorted by creation date (newest first).
  - Used in getPayments() to list payments for approval/rejection.
  - It works like:  
SELECT \* FROM payments ORDER BY createdAt DESC;
  - Frontend calls GET /api/payments.
  - Backend executes Payment.find() and returns the data.
  - Returns 500 Internal Server Error if the database query fails.
- Create New Payment Record(new Payment().save())

```
const newPayment = new Payment({
  paymentId,
  name,
  bank,
  amount: parseFloat(amount),
  status: 'Pending',
  fileUrl: `/uploadPayment/${req.file.filename}`,
});

await newPayment.save();
```

## new Payment().save()

- Saves a new payment slip upload to the database.
- Sets status: Pending (pending admin approval).
- It works like:  
INSERT INTO payments (name, bank, amount, status, fileUrl)  
VALUES ('John Doe', 'Bank of Ceylon', 100.50, false,  
'/uploadPayment/12345-proof.jpg');
- Checks for req.file (returns 400 Bad Request if missing).

- Approve/Decline Payment (Admin)

Approve:

```
// Approve Payment
const approvePayment = async (req, res) => {
  try {
    const { id } = req.params;
    const payment = await Payment.findByIdAndUpdate(
      id,
      { status: 'Approved' },
      { new: true }
    );

    if (!payment) {
      return res.status(404).json({ error: "Payment not found" });
    }

    res.status(200).json({ message: "Payment approved", payment });
  } catch (error) {
    console.error("Approval error:", error);
    res.status(500).json({ error: "Failed to approve payment" });
  }
};
```

Decline:

```
//// decline Payment
const declinePayment = async (req, res) => {
  try {
    const { id } = req.params;
    const payment = await Payment.findByIdAndUpdate(
      id,
      { status: 'Declined' },
      { new: true }
    );

    if (!payment) {
      return res.status(404).json({ error: "Payment not found" });
    }

    res.status(200).json({ message: "Payment Declined", payment });
  } catch (error) {
    console.error("Decline error:", error);
    res.status(500).json({ error: "Failed to decline payment" });
  }
};
```

**Payment.findByIdAndUpdate(id, { status: 'Approved' }, { new: true }):**

- Updates a payment's status to Approved or Declined based on admin action.
- Uses findByIdAndUpdate to target the payment by its MongoDB \_id.
- It works like:

UPDATE payments SET status = 'Approved' WHERE \_id = '123abc';

- **File Upload Handling**

```
// Upload Payment Slip
const uploadPayment = async (req, res) => {
  try {
    const { paymentId, name, bank, amount } = req.body;

    if (!req.file) {
      return res.status(400).json({ error: "File is required" });
    }

    const newPayment = new Payment({
      paymentId,
      name,
      bank,
      amount: parseFloat(amount),
      status: 'Pending',
      fileUrl: `/uploadPayment/${req.file.filename}`,
    });

    await newPayment.save();
    res.status(201).json({
      message: "Payment uploaded successfully!",
      payment: newPayment
    });
  } catch (error) {
    console.error("Upload error:", error);
    res.status(500).json({
      error: error.message || "Server error during upload."
    });
  }
}
```

- Saves the payment slip file path (fileUrl) to the database after Multer processes the upload.

- **Notification system**

- Update Payment Status (Triggers Notifications):

```
const payment = await Payment.findByIdAndUpdate(
  id,
  { status: 'Approved' },
  { new: true }
);
```

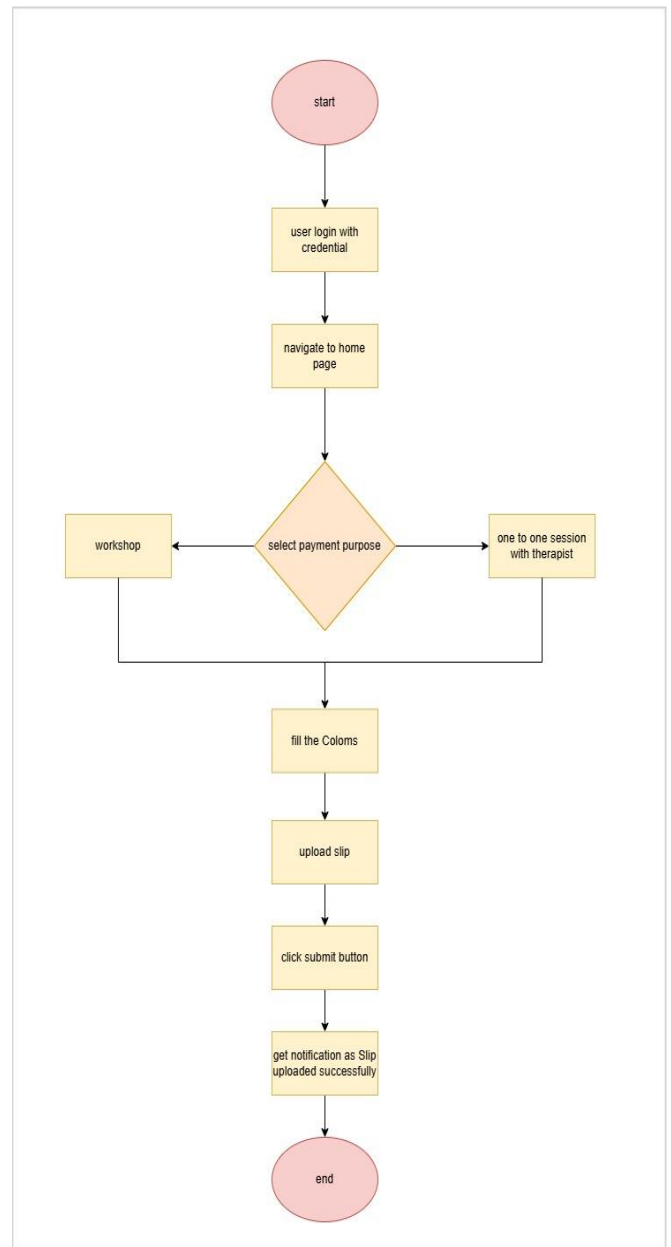
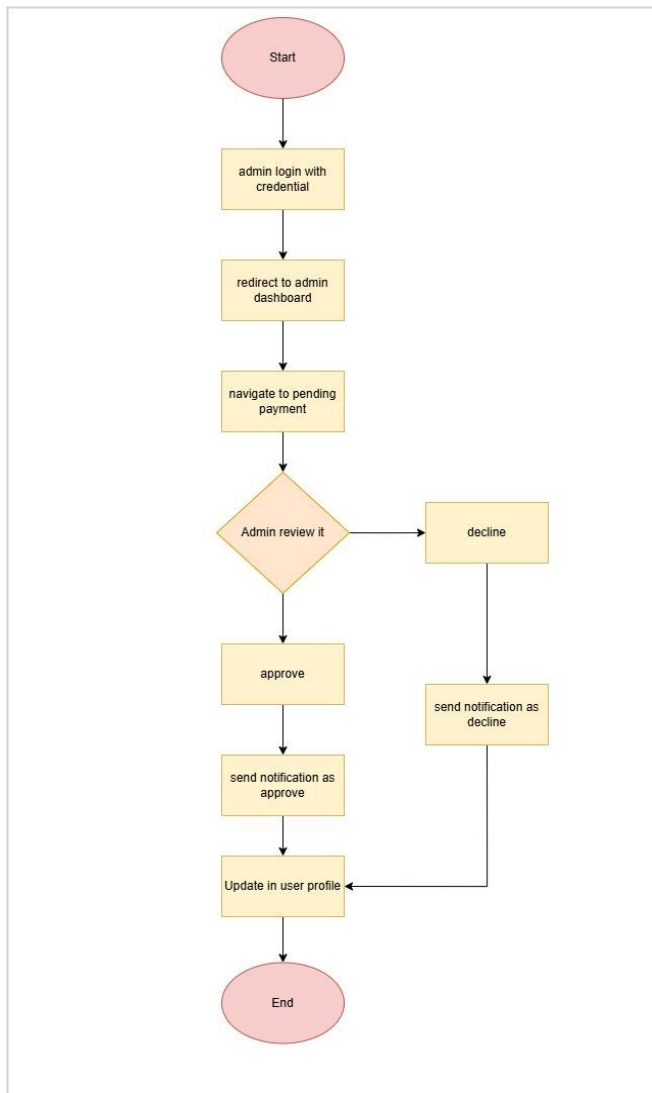
```
const payment = await Payment.findByIdAndUpdate(
  id,
  { status: 'Approved' },
  { new: true }
);
```

- Updates payment status (admin action) which would trigger notifications to users.

- Upload Payment slip Status (Triggers Notifications):

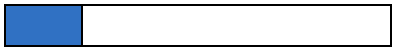






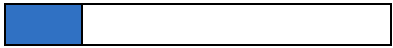
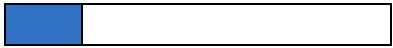
```
res.status(201).json({
  message: "Payment uploaded successfully!",
  payment: newPayment
});
```

## 11.4 Flow charts regarding Appointment Management



## 12 Workshop Management – Level of Progress

### 12.1 Completion Level of the Component

Component	Description	Progress
<b>Workshop Details UI</b>	Designing and implementing a user-friendly workshop interface that displays an overview of workshop details.	 20%
<b>Workshop List UI</b>	Allow therapists and admins to create and manage workshops.	 100%
<b>Image upload for workshop Images</b>	Allowing therapists and admins to upload images for their Workshop.	 100%
<b>Workshop Registration &amp; Enrollment</b>	Allow users to register for workshops.	 50%
<b>Attendance Tracking</b>	Ensures users who signed up actually attend.	 20%
<b>Workshop Cancellation &amp; Rescheduling</b>	Allow organizers to handle cancellations & reschedules.	 100%
<b>Workshop Feedback &amp; Ratings</b>	Collect feedback to improve future workshops.	 20%
<b>Notifications &amp; Reminders</b>	Keeps users updated about their workshops.	 20%
<b>Reporting &amp; Analytics</b>	Tracks workshop performance and engagement.	 20%



## **12.2 Functionalities need to be Implemented**

### **1. Workshop Details UI**

- Redesign workshop listings with clear descriptions, instructor bios, schedules, and multimedia (images).

### **2. Workshop Registration & Enrollment**

- Develop a seamless registration flow with secure payment integration (if applicable) and instant enrollment confirmations.

### **3. Notifications & Reminders**

- Automate email/SMS/push notifications for registration confirmations, upcoming sessions, and last-minute changes.

### **4. Reporting & Analytics**

- Create dashboards for admins to track attendance rates, participant feedback, revenue (if paid), and workshop popularity.

### **5. Attendance Tracking**

- Introduce a real-time attendance system with check-in/check-out functionality.

## 12.3 Queries Used

### MongoDB Data Fetching Functions in Workshop List Page

- Fetch all workshops for the workshop List Page.

```
7  const getAllWorkshops = async (req, res) => {
8    try {
9      const workshops = await Workshop.find().sort({ createdAt: -1 });
10     res.status(200).json({
11       success: true,
12       count: workshops.length,
13       data: workshops
14     });
15   } catch (err) {
16     console.error("Error fetching workshops:", err);
17     res.status(500).json({
18       success: false,
19       error: "Server error while fetching workshops"
20     });
21   }
22 };
23
```

- Add workshops to the workshop List.

```
42  const addWorkshop = async (req, res) => {
43    try {
44      const { title, description, date, duration, price } = req.body;
45      const imagePath = req.file ? req.file.path.replace(/\\/g, '/') : null;
46
47      if (!title || !description || !date || !duration || !price || !imagePath) {
48        return res.status(400).json({
49          success: false,
50          error: "Please provide all required fields including an image"
51        });
52      }
53
54      const workshop = new Workshop({
55        title,
56        description,
57        date,
58        duration,
59        price,
60        image: imagePath
61      });
62
63      const savedWorkshop = await workshop.save();
64      res.status(201).json({
65        success: true,
66        data: savedWorkshop
67      });
68    } catch (err) {
69      console.error("Error adding workshop:", err);
70      res.status(500).json({
71        success: false,
72        error: "Server error while adding workshop"
73      });
74    }
75  };
76
```

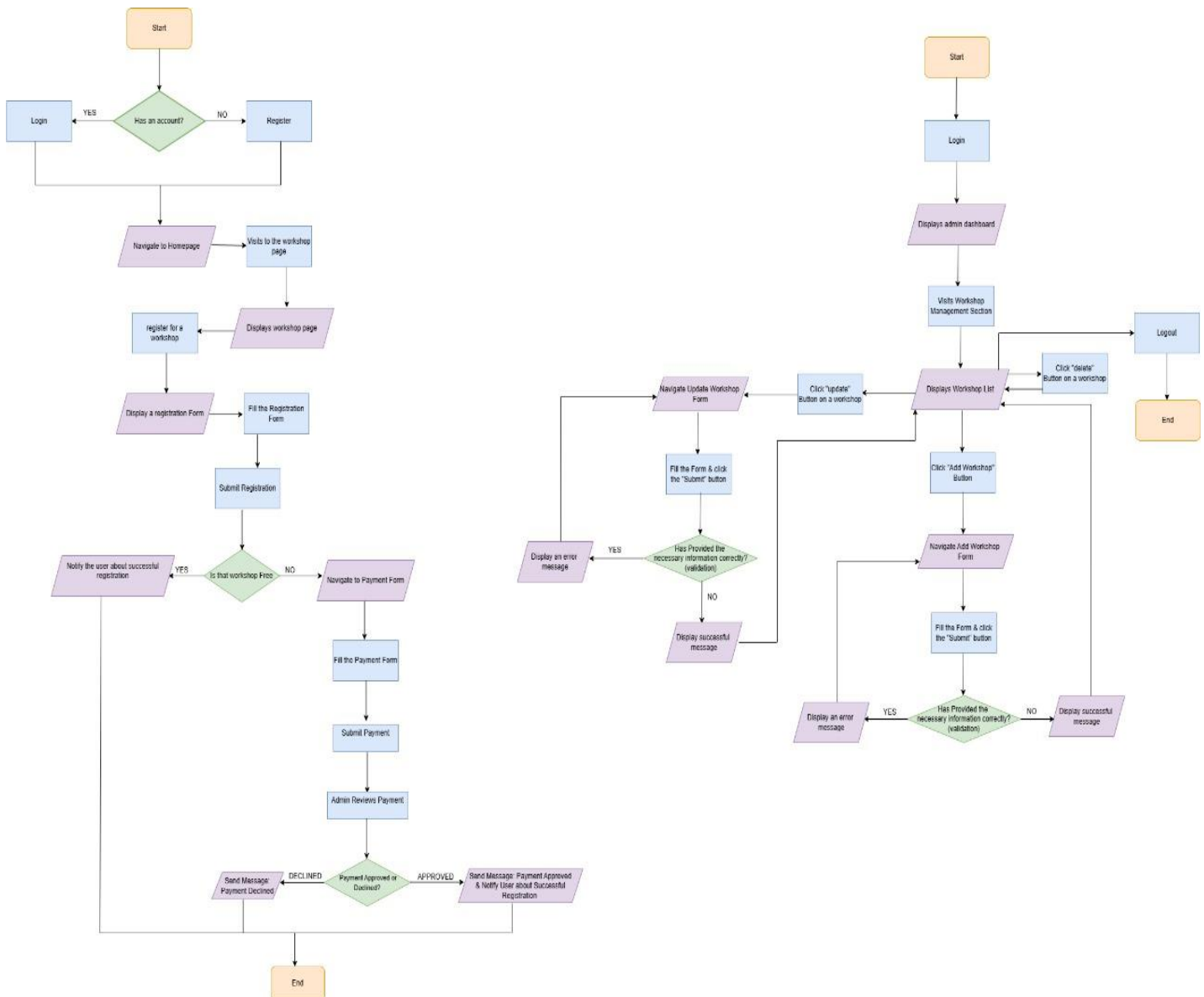
- Updating the workshop details of a specific workshop.

```
101 const updateWorkshop = async (req, res) => {
102   try {
103     const { title, description, date, duration, price } = req.body;
104     const updateData = { title, description, date, duration, price };
105
106     if (req.file) {
107       updateData.image = req.file.path.replace(/\\/g, '/');
108     }
109
110     const workshop = await Workshop.findByIdAndUpdate(
111       req.params.id,
112       updateData,
113       { new: true, runValidators: true }
114     );
115
116     if (!workshop) {
117       return res.status(404).json({
118         success: false,
119         error: "Workshop not found"
120       });
121     }
122
123     res.status(200).json({
124       success: true,
125       data: workshop
126     });
127   } catch (err) {
128     console.error("Error updating workshop:", err);
129     res.status(500).json({
130       success: false,
131       error: "Server error while updating workshop"
132     });
133   }
134 };
135
```

- Delete a specific workshop from the workshop List.






```
137 const deleteworkshop = async (req, res) => {
138   try {
139     const workshop = await Workshop.findByIdAndDelete(req.params.id);
140     if (!workshop) {
141       return res.status(404).json({
142         success: false,
143         error: "Workshop not found"
144       });
145     }
146     res.status(200).json({
147       success: true,
148       data: { id: req.params.id }
149     });
150   } catch (err) {
151     console.error("Error deleting workshop:", err);
152     res.status(500).json({
153       success: false,
154       error: "Server error while deleting workshop"
155     });
156   }
157 };
158
```

## 12.4 Flow charts regarding Workshop Management



## 13 Wellness Content & Feedback Management – Level of Progress

### 13.1 Completion Level of the Component

Components	Description	Progress
Vlog Creation & Upload	A platform for wellness experts to share video content.	 100%
Post & Blog Management	Creating and publishing wellness-related articles.	 100%
User Reactions & Comments	Allowing users to engage with vlogs and posts.	 50%
Content Moderation	Ensuring posted content follows community guidelines.	 50%
Push Notifications	Sending reminders about new content, sessions, or updates.	 20%

## **13.2 Functionalities need to be Implemented**

### **1. Content Upload Interface**

- Design an easy-to-use UI for therapists and wellness experts to upload vlogs and write blog posts, with support for video embedding, images, and rich text formatting.

### **2. Engagement Features**

- Implement like, comment, and share functionalities for blogs and vlogs to boost community interaction and engagement.

### **3. Content Moderation System**

- Develop admin tools for reviewing submitted content, managing reported posts/comments, and blocking inappropriate users based on policy violations.

### **4. Feedback Submission & Analysis**

- Create forms for users to give feedback on therapists, sessions, and the platform; integrate data visualization tools for admins to analyze trends and satisfaction levels.

### **5. Notification & Alert System**

- Set up automated email/push notifications to inform users about new content, feedback responses, comment replies, and content approval/rejection status.

## 13.3 Queries Used

### Create, Get , Update , Delete Feedbacks

```
import { UserOutlined, EditOutlined, DeleteOutlined } from "@ant-design/icons";
import { useAuth } from "../../auth/authContext";
import { useNavigate } from "react-router-dom";
import {
  createFeedback,
  getFeedbacks,
  getUserFeedbacks,
  updateFeedback,
  deleteFeedback,
} from "../../api/Api";

const { TextArea } = Input;
const { Option } = Select;

const FeedbackForm = () => {
  const [form] = Form.useForm();
  const { username, user } = useAuth();
  const navigate = useNavigate();
  const [feedbacks, setFeedbacks] = useState([]);
  const [userFeedbacks, setUserFeedbacks] = useState([]);
  const [editingFeedback, setEditingFeedback] = useState(null);
  const [isModalVisible, setIsModalVisible] = useState(false);
  const [loading, setLoading] = useState(false);
  const [submitting, setSubmitting] = useState(false);

  const therapists = [
    {
      id: "1",
      name: "Dr. Sarah Johnson",
      specialization: "Cognitive Behavioral Therapy",
    },
    { id: "2", name: "Dr. Michael Chen", specialization: "Family Counseling" },
  ];

  useEffect(() => {
    const loadData = async () => {
      setLoading(true);
      try {
        const [allFeedbacks, userFb] = await Promise.all([
```



```

        getFeedbacks(),
        username ? getUserFeedbacks() : Promise.resolve([]),
    ]);
    setFeedbacks(allFeedbacks.data || []);
    setUserFeedbacks(userFb.data || []);
} catch (error) {
    message.error(
        error.response?.data?.message || "Failed to load feedbacks"
    );
} finally {
    setLoading(false);
}
};

loadData();
}, [username]);

const handleSubmit = async (values) => {
    if (!username) {
        message.warning("Please login to submit feedback");
        navigate("/login");
        return;
    }

    try {
        setSubmitting(true);
        const feedbackData = {
            rating: values.rating,
            feedback: values.feedback,
            therapist: values.therapist,
            anonymous: values.anonymous,
        };

        if (editingFeedback?._id) {
            const response = await updateFeedback(
                editingFeedback._id,
                feedbackData
            );
            const updated = response.data;
            setFeedbacks(
                feedbacks.map((f) => (f._id === updated._id ? updated : f))
            );
            setUserFeedbacks(
                userFeedbacks.map((f) => (f._id === updated._id ? updated : f))
            );
        }
    }
};

```

```

        message.success("Feedback updated successfully");
    } else {
        const response = await createFeedback(feedbackData);
        const newFeedback = response.data;
        setFeedbacks([newFeedback, ...feedbacks]);
        setUserFeedbacks([newFeedback, ...userFeedbacks]);
        message.success("Feedback submitted successfully");
    }

    form.resetFields();
    setEditingFeedback(null);
    setIsModalVisible(false);
} catch (error) {
    console.error("Feedback error:", error);
    message.error(
        error.response?.data?.message || "Failed to submit feedback"
    );
} finally {
    setSubmitting(false);
}
};

const handleEdit = (feedback) => {
    if (!feedback?._id) {
        message.error("Invalid feedback data");
        return;
    }

    setEditingFeedback(feedback);
    form.setFieldsValue({
        therapist: feedback.therapist,
        rating: feedback.rating,
        feedback: feedback.feedback,
        anonymous: feedback.name === "Anonymous",
    });
    setIsModalVisible(true);
};

const handleDelete = async (id) => {
    Modal.confirm({
        title: "Delete Feedback",
        content: "Are you sure you want to delete this feedback?",
        okText: "Yes",
        okType: "danger",
        cancelText: "No",
    });
};

```

```

onOk: async () => {
  try {
    // Optimistically remove from UI
    setFeedbacks(prev => prev.filter(f => f._id !== id));
    setUserFeedbacks(prev => prev.filter(f => f._id !== id));

    const response = await deleteFeedback(id);

    if (!response.success) {
      // If deletion failed, reload the data
      const [allFeedbacks, userFb] = await Promise.all([
        getFeedbacks(),
        getUserFeedbacks()
      ]);
      setFeedbacks(allFeedbacks.data);
      setUserFeedbacks(userFb.data);
      throw new Error(response.message);
    }

    message.success(response.message);
  } catch (error) {
    message.error(error.message || "Failed to delete feedback");
  }
},
});
});

const getTherapistName = (therapistId) => {
  const therapist = therapists.find((t) => t.id === therapistId);
  return therapist
    ? `${therapist.name} (${therapist.specialization})`
    : "Unknown Therapist";
};

if (loading && feedbacks.length === 0) {
  return (
    <div
      style={{ display: "flex", justifyContent: "center", padding: "50px" }}
    >
      <Spin size="large" />
    </div>
  );
}

return (
  <div style={{ maxWidth: 800, margin: "auto", padding: 20 }}>

```

```

<h2 style={{ textAlign: "center", marginBottom: 20 }}>
  Share Your Experience
</h2>
<p style={{ textAlign: "center", marginBottom: 30 }}>
  Your feedback helps us improve our services.
</p>

{username ? (
  <Button
    type="primary"
    onClick={() => {
      form.resetFields();
      setEditingFeedback(null);
      setIsModalVisible(true);
    }}
    style={{ marginBottom: 20 }}
  >
    Add New Feedback
  </Button>
) : (
  <Button
    type="primary"
    onClick={() => navigate("/login")}
    style={{ marginBottom: 20 }}
  >
    Login to Submit Feedback
  </Button>
)}

<Modal
  title={editingFeedback ? "Edit Feedback" : "Add Feedback"}
  open={isModalVisible}
  onCancel={() => {
    setIsModalVisible(false);
    setEditingFeedback(null);
    form.resetFields();
  }}
  footer={null}
  destroyOnClose
>
  <Form
    form={form}
    layout="vertical"
    onFinish={handleSubmit}
    initialValues={{ rating: 5, anonymous: false }}

```

```

>
<Form.Item
  name="therapist"
  label="Select Your Therapist"
  rules={[
    { required: true, message: "Please select your therapist" },
  ]}
>
  <Select placeholder="Select therapist">
    {therapists.map((therapist) => (
      <Option key={therapist.id} value={therapist.id}>
        {therapist.name} ({therapist.specialization})
      </Option>
    ))}
  </Select>
</Form.Item>

<Form.Item
  name="rating"
  label="Rate Your Experience"
  rules={[{ required: true, message: "Please rate your experience" }]}
>
  <Rate allowHalf />
</Form.Item>

<Form.Item
  name="feedback"
  label="Your Feedback"
  rules={[
    { required: true, message: "Please provide your feedback" },
    { min: 10, message: "Feedback must be at least 10 characters" },
  ]}
>
  <TextArea rows={4} placeholder="Share your experience..." />
</Form.Item>

<Form.Item name="anonymous" valuePropName="checked">
  <Checkbox>Submit anonymously</Checkbox>
</Form.Item>

<Form.Item>
  <Button
    type="primary"
    htmlType="submit"
    loading={submitting}
  >

```

```

        style={{ width: "100%" }}
      >
        {editingFeedback ? "Update Feedback" : "Submit Feedback"}
      </Button>
    </Form.Item>
  </Form>
</Modal>

{username && userFeedbacks.length > 0 && (
  <>
    <h3>Your Feedback</h3>
    <List
      itemLayout="horizontal"
      dataSource={userFeedbacks}
      renderItem={(item) => (
        <List.Item
          actions={[
            <Button
              icon={<EditOutlined />}
              onClick={() => handleEdit(item)}
              disabled={loading}
            />,
            <Button
              icon={<DeleteOutlined />}
              onClick={() => handleDelete(item._id)}
              danger
              // disabled={loading}
            />,
          ]}
        >
          <List.Item.Meta
            avatar={<Avatar icon={<UserOutlined /> />} />}
            title={item.name}
            description={
              <>
                <Rate disabled value={item.rating} />
                <p>{item.feedback}</p>
                <p>
                  <small>
                    Therapist: {getTherapistName(item.therapist)}
                  </small>
                </p>
                <p>
                  <small>
                    {new Date(item.createdAt).toLocaleDateString()}

```

```

        </small>
      </p>
    </>
  }
  />
</List.Item>
)}
/>
</>
)}

<h3 style={{ marginTop: 30 }}>Community Feedback</h3>
{feedbacks.length === 0 ? (
  <p>No feedback available yet. Be the first to share!</p>
) : (
  <List
    itemLayout="horizontal"
    dataSource={feedbacks}
    renderItem={(item) => (
      <List.Item>
        <List.Item.Meta
          avatar={<Avatar icon={<UserOutlined />} />}
          title={item.name}
          description={
            <>
              <Rate disabled value={item.rating} />
              <p>{item.feedback}</p>
              <p>
                <small>
                  Therapist: {getTherapistName(item.therapist)}
                </small>
              </p>
              <p>
                <small>
                  {new Date(item.createdAt).toLocaleDateString()}
                </small>
              </p>
            </>
          }
        </List.Item>
      )}
    />
  </List.Item>
)}
/>
)}
</div>

```

```
);  
};  
  
export default FeedbackForm;
```



## 13.4 Flow charts regarding Wellness Content & Feedback Management

