Abstract

Therapy Co: A Digital Platform for Mental Health Management

The project aims to develop *TherapyCo*, a mental-health support and therapist-booking platform integrating AI assistance and a full-stack appointment management system. The platform offers an intuitive user interface, therapist discovery, appointment booking, secure login, and a 24/7 AI chatbot for initial guidance. The system uses React.js, Tailwind CSS, Node.js, Express.js, MongoDB, and an external AI model. The platform improves accessibility and makes mental-health support easy to navigate for patients, doctors, and administrators.

Keywords: Mental Health, Teletherapy, Practice Management, Appointment Management, Full-Stack Development, Node.js, JavaScript, Healthcare Administration.

Contents

1		Introduction	8
	1.1	Project Goal and Scope	8
	1.2	Type of Problem Addressed	9
	1.3	Key System Components	9
2		Literature Review	11
	2.1	Core Concepts and Platforms	11
		Digital Mental Health Platforms (DMP)	11
		Generative AI and Large Language Models (LLMs)	11
	2.2	Technologies and Tools Used	12
		React	12
		Node.js & Express.js	12
		MongoDB (with Mongoose)	12
		Vite / Tailwind CSS / HTML / CSS / Docker	13
3		Problem Statement / Requirement Specifications	14
	3.1	Project Planning (3.1.1 Overview of the Project)	14
	3.2.1	Functional Requirements	14
		3.2.1.1 - User and Doctor Profile Management	14
		3.2.1.2 - Appointment Scheduling & Management	15
		3.2.1.3 - Mental Wellness Chatbot & Initial Screening	15
		3.2.1.4 - Urgent Support & Crisis Escalation	15
		3.2.1.5 - Doctor-Patient Interaction & Record Keeping	15
		3.2.1.6 - Payment and Billing	15
		3.2.1.7 - Platform Analytics & Reporting (Admin)	16

	3.2.2	Non-Functional Requirements	16
		3.2.2.1 Performance	16
		3.2.2.2 Security	16
		3.2.2.3 Usability	16
		3.2.2.4 Scalability	17
		3.2.2.5 Maintainability	17
		3.2.2.6 Reliability and Data Integrity	17
		3.2.2.7 Compliance and Confidentiality	17
	3.3	System Design	17
		3.3.1 - Design Constraints	17
		3.3.1.1 - Technology Constraints	17
		3.3.1.2 - Performance Constraints	18
		3.3.2 - Block Diagram/ Sequence Diagram	19
4		System Design and Testing	23
	4.1	System Architecture and Implementation	23
		4.1.1. Web Application Workflow	23
		4.1.2. Frontend Implementation	23
		4.1.3. Backend Implementation	24
		4.1.4. AI/Chatbot for Initial Screening and Support	24
		4.1.5. API Development & Integration	25
	4.2	Challenges and AI Limitations	26
		4.2.1. Implementation Challenges & Key Issues	26
		4.2.2 AI/Chatbot Model Performance and Technical Considerations	26
	4.3	Functional Testing	28
	4.4	Performance Testing	29

	4.5	Security Testing	30
	4.6	Usability Testing	31
5		Standard Adopted	33
6		Conclusion and Future Scope	39
7		References	41
8	8.1	Appendix A	42
	8.2	Appendix B	43

List of Figures

3.1	Use Case Diagram	18
3.2	Sequence Diagram	19
B.1	Landing Page	43
B.2	All Doctors Page	43
B.3	About Us Page	44
B.4	Admin / Doctor Login Page	44
B.5	Login/ Create Account page for Patients	45
B.6	Admin's Dashboard	46
B.7	Admin→ All Appointments	46
B.8	Admin → Add Doctor	47
B.9	Doctor's Dashboard	47
B.10	Doctor's Profile	48
B.11	Doctor's Patient list	48
B.12	Patient's Account Page	
B.13	My Appointments Page	
B.14	Razorpay Payment Gateway	
B.15	Chatbot	

List of Tables

4.1	Functional Testing	29
4.2	Performance Testing	30
4.3	Security Testing	31
4.4	Usability Testing	32

Chapter 1

Introduction

The rapid shift toward digital healthcare has highlighted a critical need for integrated, efficient, and secure platforms to manage mental health services. Access to qualified therapists and streamlined administrative processes are frequently hindered by disjointed legacy systems, leading to administrative burdens for providers and a challenging user experience for patients seeking care. The rapid shift to digital healthcare necessitates secure, integrated, and efficient mental health service platforms. Fragmented legacy systems often hinder access to therapists and efficient administration, causing high overhead for providers and a poor patient experience. This inefficiency compromises the consistency and quality of therapeutic services, vital for positive mental health outcomes.

Project Goal and Scope

The overarching goal of the **Therapy Co** project is to develop a comprehensive **full-stack web application** that unifies the management of mental health services for three primary user groups: **Admins**, **Doctors**, and **Patients**. This platform moves beyond simple website brochureware to provide a complete digital ecosystem for therapy practice management, from initial patient contact to session completion.

Therapy Co is an ambitious project aiming to build a complete **full-stack web application** for managing mental health services. Its core objective is to create a unified digital ecosystem that serves three main user groups: **Admins**, **Doctors**, and **Patients**. This platform is designed to be a comprehensive practice management tool, streamlining the entire therapy process from initial patient engagement through to session completion, and goes far beyond the functionality of a simple informational website.

The core objective is to deliver a functional system that allows users to:

- **Patients:** Easily search for practitioners and securely book, view, and manage their appointments.

- **Doctors:** Manage their schedule, confirm appointments, and access relevant patient information.
- **Admins:** Oversee the entire system, manage user accounts (doctors and patients), and maintain system-level configurations.

Type of Problem Addressed

This project addresses a **Service Management and Workflow Optimization** problem in the specialized domain of mental health. It is fundamentally an **end-to-end management solution** aimed at digitizing and automating the operational logistics of a therapy practice.

Key areas of functionality include:

1. **Secure Multi-Role Authentication:** Ensuring distinct, secure access for patients, doctors, and administrators.
2. **Centralized Scheduling System:** Providing real-time appointment booking and management capabilities.
3. **Data Management:** Storing and retrieving user and appointment data reliably and securely.

Key System Components

The **full-stack architecture** of the platform is designed for responsiveness and scalability, built primarily on **JavaScript** technologies. The primary components of the system include:

- **Frontend Interface:** Developed using a modern JavaScript framework (e.g., React/Vue.js) to deliver an intuitive and responsive user experience.
- **Backend Server:** Utilizes Node.js with a framework like Express.js to handle business logic, manage API requests, and ensure data integrity.
- **Database:** A robust database solution (e.g., MongoDB, PostgreSQL) is employed to persistently store all user and appointment data.
- **Deployment:** The application is containerized using **Docker** to facilitate seamless and portable deployment on cloud platforms like Render.

Chapter 2

Literature Review

In this project, full-stack web development, artificial intelligence (AI), and API integration were used to create a digital mental health web platform. Providing initial AI-driven conversational support and enabling safe communication between patients and physicians are two of the platform's primary functions. The next section will provide specifics on the tools and techniques used in this initiative.

2.1 Core Concepts and Platforms

Digital Mental Health Platforms (DMP)

Tele-counseling systems, also known as Digital Mental Health Platforms (DMPs), greatly improve health technology by increasing access to psychological services. They handle secure onboarding, appointment scheduling, and private health information in addition to facilitating virtual patient-professional connections. For sensitive medical records, their user-centric design guarantees simple navigation and stringent data privacy compliance.

Generative AI and Large Language Models (LLMs)

The project integrates cutting-edge Generative AI capabilities, utilizing packages, to power an intelligent chatbot. Large Language Models (LLMs) are used in this technology to process and produce text responses that resemble those of a human. In this regard, the LLM functions as an initial support tool, able to offer broad emotional support, respond to fundamental questions about mental health, and direct users to pertinent professional resources prior to a consultation with a human physician. By providing instantaneous, nonjudgmental conversational support, this feature improves accessibility.

2.2 Technologies and Tools Used

React

React, a powerful JavaScript library by Meta, is a leading choice for developing UIs and UI components, especially for large-scale single-page applications (SPAs) with frequent data updates. Its component-based architecture enhances code reusability and simplifies complex UIs, benefiting platforms like user, doctor, and admin portals. The innovative Virtual DOM ensures high performance by efficiently updating and rendering components only when data changes.

Node.js & Express.js

The project's robust and scalable backend is built with Node.js and Express.js. Node.js is a cross-platform JavaScript runtime used for server-side execution, while Express.js is a minimalist web framework for Node.js. Together, they efficiently create RESTful APIs to manage database transactions, user authentication, and data delivery to the React frontend.

MongoDB (with Mongoose)

MongoDB, a prevalent NoSQL document database, employs flexible, JSON-like documents for data storage. This fundamental schema-less nature is highly beneficial in agile development, enabling rapid changes and updates to data structures. To bolster application stability and streamline data handling, the project utilizes Mongoose. As an Object Data Modeling (ODM) library for Node.js, Mongoose enforces structure and validation, establishing precise schemas for crucial entities like users, doctors, and appointments within the MongoDB data.

Vite

Vite is a cutting-edge frontend toolchain, offering superior performance and developer experience. It surpasses older bundlers like Webpack by using native ES modules, leading to much faster build times and rapid development server start-up. Its fast Hot Module Replacement (HMR) instantly updates code changes without losing application state, greatly enhancing development workflow and productivity.

Tailwind CSS

Tailwind CSS is a utility-first CSS framework designed to speed up the design and styling workflow. Instead of offering ready-made components like other CSS frameworks, Tailwind provides a set of low-level utility classes (such as flex, pt-4, or text-center). These classes are applied directly within HTML or React components. This approach promotes design consistency, greatly reduces the need to write custom CSS, and simplifies responsive design through its integrated breakpoint utilities.

HTML

HTML (HyperText Markup Language) serves as the foundational markup language for web content creation and structure. It defines the essential elements of a webpage using nested elements enclosed in tags (like <h1>, <p>, <img>). HTML is responsible for providing the semantic structure, which differentiates content such as headings, paragraphs, images, and links. The current standard, HTML5, further advances web development by including features for multimedia and better semantic organization.

CSS

CSS (Cascading Style Sheets) is the styling language for HTML, dictating the visual presentation of web components. It allows developers to manage all aesthetic elements of a webpage, such as colors, typography, spacing, and overall layout, through a rule-based syntax. This syntax uses selectors to apply specific visual styles to targeted HTML elements. A key feature is Media Queries, which are essential for creating responsive designs that automatically adjust the layout to fit the screen size of the user's device.

Docker

Docker is a platform for building, packaging, and running applications in lightweight, isolated **containers**. It bundles an application and its dependencies (libraries, configs, tools) into a portable unit, ensuring consistent behavior across different machines. Docker uses an optimized, layered image system. Docker Compose helps manage multi-container setups. It greatly simplifies deployment, scalability, and environment management.

Chapter 3

Problem Statement / Requirement Specifications

Escalating global mental health challenges persist due to stigma, long waiting lists, and difficulty finding practitioners. Professionals are burdened by admin, and current basic digital tools are insufficient. This systemic failure heightens suffering and costs.

Our Solution: A Comprehensive Mental Health Teleconsultation and Support Platform is proposed. It will offer a robust practitioner management system, a discreet user booking frontend, and an integrated AI chatbot for immediate initial support, effectively closing the gap between need and professional care.

3.1 Project Planning

3.1.1 Overview of the Project : To develop a comprehensive Online Mental Wellness Platform that facilitates access to mental health professionals and provides interactive support. The application is a full-stack system that includes:

- A User Interface (Frontend) for browsing mental health specialists, scheduling appointments, and managing a personal profile.
- A Chatbot Utility for providing immediate, on-demand mental health information, initial screening, and chat support.
- A Doctor Portal for managing appointments and patient prescriptions.
- An Admin Dashboard for overseeing the platform, managing the list of doctors, and tracking all user appointments.

3.2.1 Functional Requirements

3.2.1.1 - User and Doctor Profile Management

- Allow users (patients) to register, create, and manage their personal profiles.
- Enable licensed mental health professionals (doctors) to register, submit credentials for verification, and maintain their public profiles (specialty, availability, fees).
- Store all profile data and credentials securely in the backend database.

3.2.1.2 - Appointment Scheduling & Management

- Provide a web interface for users to browse available doctors based on specialty, rating, and availability.
- Allow users to select a time slot and submit an appointment request.
- Enable doctors to view pending requests, approve or reject appointments, and manage their calendar.
- Send automated notifications/reminders to both user and doctor upon booking, approval, or cancellation.

3.2.1.3 - Mental Wellness Chatbot & Initial Screening

- Implement a **Chatbot** feature for immediate, non-crisis, supportive chat interactions.
- Allow the chatbot to conduct an **initial mental health screening** based on user input (e.g., using a short questionnaire).
- Based on the assessment, the chatbot should provide relevant, general psychoeducational information and suggest appropriate professional specialties (e.g., Clinical Psychologist, Psychiatrist).

3.2.1.4 - Urgent Support & Crisis Escalation

- Integrate a mechanism within the chatbot and platform for detecting high-risk or crisis language (e.g., suicidal ideation).
- Immediately **flag and alert** the user with clear contact information for national emergency hotlines and crisis centers.
- Provide a one-click escalation pathway to emergency resources, bypassing regular platform features.

3.2.1.5 - Doctor-Patient Interaction & Record Keeping

- Provide a secure portal for doctors to view their patient's medical history and current appointment notes.
- Allow doctors to digitally write and manage prescriptions or therapeutic notes following a session.
- Enable users to view their session notes and prescriptions securely via their profile.

3.2.1.6 - Payment and Billing

- Integrate with a third-party payment gateway (e.g., Stripe) to handle consultation fees.
- Generate digital invoices/receipts for users upon successful payment.
- Process payouts to doctors after the consultation is completed.

3.2.1.7 - Platform Analytics & Reporting (Admin)

- Generate aggregate reports for the platform administrator, including:
  - Total number of users and active doctors.
  - Appointment volume, completion rates, and cancellations.
  - Total revenue generated (earnings report).

3.2.2 Non-Functional Requirements

3.2.2.1 Performance

- Appointment scheduling and confirmation must be completed in under **3 seconds**.
- The Chatbot must respond to user input in **real-time** (under 1 second).
- The system should maintain stability and responsiveness when handling **200+ concurrent user sessions** (including chat and booking).

3.2.2.2 Security

- Implement **secure multi-factor authentication** for all user types (User, Doctor, Admin).
- Enforce **Role-based Access Control (RBAC)** to ensure users can only access their own data, doctors can only access their patients' data, and admins have platform-wide oversight.
- Encrypt all sensitive Protected Health Information (**PHI**) both in transit and at rest.

3.2.2.3 Usability

- The user interface (UI) must be **intuitive and clean** (utilizing a design system like Tailwind CSS, as indicated by the project files).
- The platform must be fully **responsive** and optimized for use on mobile, tablet, and desktop devices.
- Adhere to **Web Content Accessibility Guidelines (WCAG)** standards to ensure usability for individuals with disabilities.

3.2.2.4 Scalability

- The database and application architecture must be capable of **scaling horizontally** to support exponential growth in registered users and concurrent appointments.

3.2.2.5 Maintainability

- The system must have a **modular and component-based** architecture (e.g., React/Next.js) to simplify updates, debugging, and feature additions. Consistent, detailed documentation for both front-end and back-end services is also mandatory.

3.2.2.6 Reliability and Data Integrity

- Implement routine, automated database backups to ensure **data integrity**.
- Ensure high availability with minimal downtime for crucial services (e.g., appointment booking and the Chatbot).

3.2.2.7 Compliance and Confidentiality

- Ensure strict compliance with **data privacy regulations** (e.g., GDPR, and ideally, principles similar to HIPAA for health data).
- Implement a detailed **audit logging system** to record all critical user actions, data access, and changes for accountability and confidentiality verification.

3.3 System Design

3.3.1 - Design Constraints

3.3.1.1 - Technology Constraints

- Frontend: Use **React/Vite** and **Tailwind CSS** for a fast, component-based, and responsive user interface.
- Backend: Developed with Node.js and Express.js to efficiently manage asynchronous API requests.
- Database: Utilize **MongoDB** (NoSQL) for flexible storage of complex records like patient notes and doctor details.
- Authentication: Implement **JWT Tokens** to enforce secure, role-based access for User, Doctor, and Admin portals.

- AI/ML Integration: The chatbot feature requires connecting to a dedicated **NLP model API** for initial mental health screening.
- Media Storage: Doctor profile pictures and verification documents must be managed externally via a cloud storage service like **Cloudinary**.

3.3.1.2 - Performance Constraints

- **Chatbot Speed:** AI model inference for conversational responses must be delivered in **under 1.5 seconds**.
- **Core Transactions:** Critical operations like appointment booking and profile retrieval must complete in **under 2 seconds**.
- **Concurrency:** The system must maintain stability and low latency while handling **200+ concurrent user sessions**.
- **Resource Management:** Server CPU utilization must not exceed **70%** under peak load conditions.

3.3.2 -Sequence Diagram

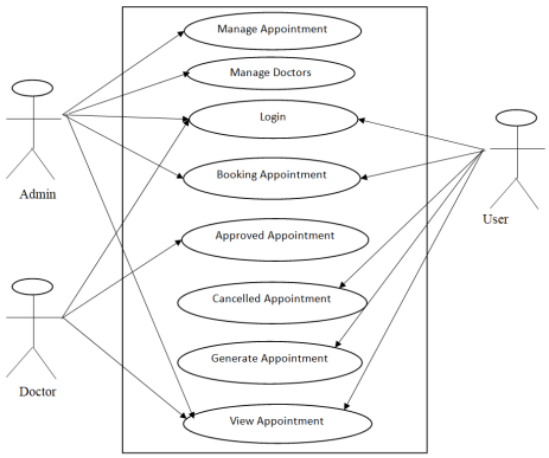


Fig 3.1 - Use Case Diagram

Sequence Diagram

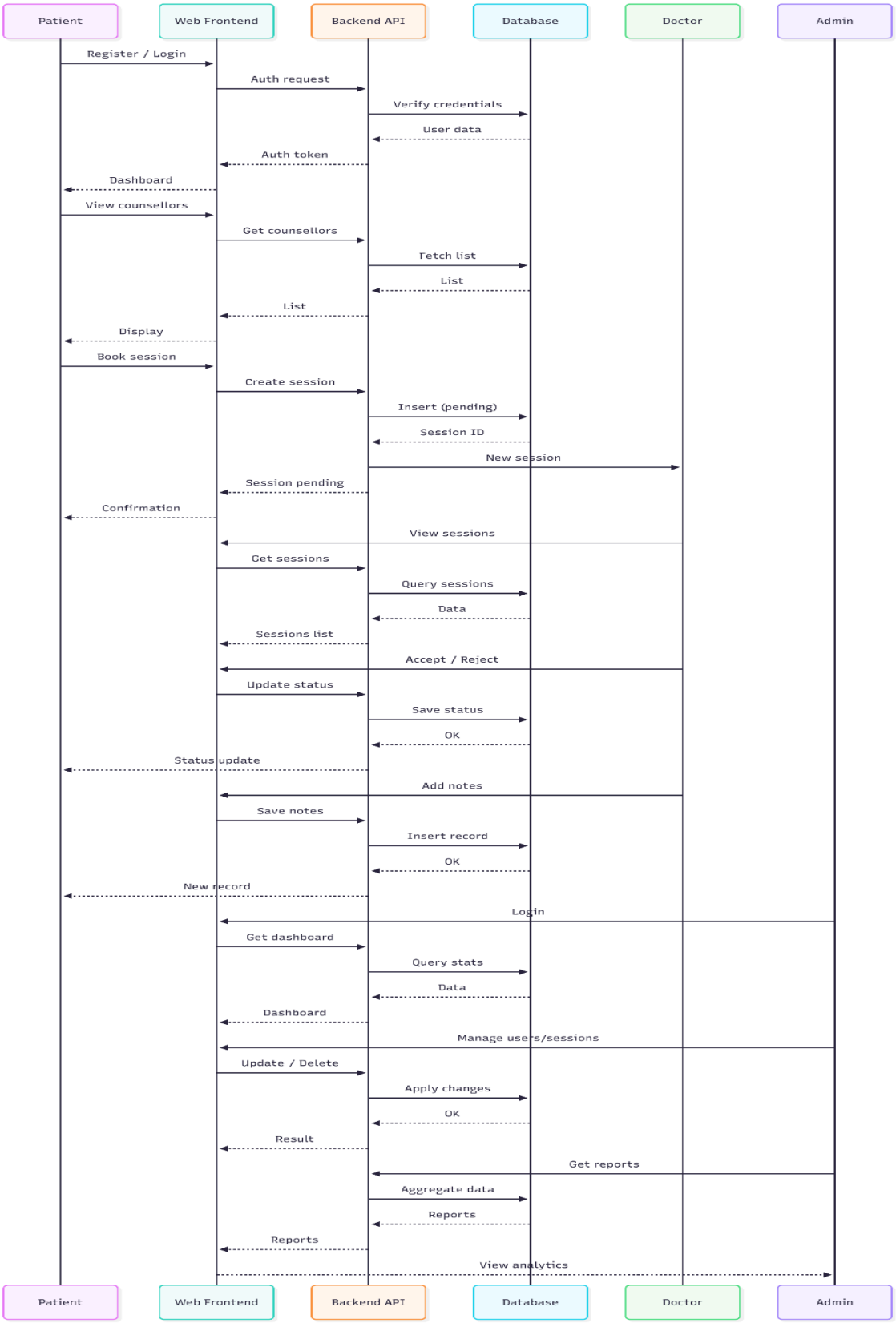


Fig 3.2 - Sequence diagram

Sequence Diagram Explanation:

- **Patient → Web Frontend**

  The patient begins the process by registering or logging into the TherapyCo platform using the web interface.

- **Web Frontend → Backend API**

  The frontend sends an authentication request to the backend API for verification.

- **Backend API → Database**

  The backend checks the stored credentials in the database.

- **Database → Backend API**

  The database returns the user data, and the backend sends an authentication token back to the frontend.

- **Web Frontend → Patient**

  The patient receives access to their dashboard after authentication.

- **Patient → Web Frontend**

  The patient requests to view available counselors.

- **Web Frontend → Backend API**

  The frontend requests the counsellor list from the backend.

- **Backend API → Database**

  The backend fetches the list of counselors.

- **Database → Backend API → Web Frontend → Patient**

  The list is returned to the frontend and displayed to the patient.

- **Patient → Web Frontend**

The patient books a session.

- **Web Frontend → Backend API**

  The frontend sends a “Create Session” request.

- **Backend API → Database**

  The backend inserts a new pending session into the database.

- **Database → Backend API → Doctor**

  The database confirms the entry and the backend notifies the concerned doctor about the new session.

- **Backend API → Web Frontend → Patient**

  The patient receives confirmation that their session request is pending.

- **Doctor → Web Frontend**

  The doctor views their pending sessions.

- **Web Frontend → Backend API → Database → Backend API → Doctor**

  The backend fetches the doctor’s session list from the database and returns it to the doctor.

- **Doctor → Web Frontend → Backend API**

  The doctor accepts or rejects the session.

- **Backend API → Database → Backend API**

  The updated session status is saved to the database.

- **Backend API → Patient**

  The patient is notified of the updated session status.

- **Doctor → Web Frontend → Backend API**

  The doctor adds session notes after completing the therapy session.

- **Backend API → Database → Backend API → Web Frontend → Patient**

  The notes are saved in the database and the patient receives a “new record added” notification.

- **Admin → Web Frontend → Backend API**

  The admin logs into their dashboard.

- **Backend API → Database → Backend API → Web Frontend → Admin**

  System statistics and analytics are fetched and displayed.

- **Admin → Web Frontend → Backend API**

  The admin manages users or sessions (update/delete actions).

- **Backend API → Database → Backend API → Web Frontend → Admin**

  The changes are applied, saved, and confirmed back to the admin.

- **Admin → Backend API**

  The admin requests system reports.

- **Backend API → Database → Backend API → Web Frontend → Admin**

  The system generates aggregated analytics and returns the final reports.

Chapter 4

System Design and Testing

4.1 System Architecture and Implementation

4.1.1. Web Application Workflow

Here is a reorganized and rephrased workflow for a patient's journey:

1. **User Onboarding:** A new patient registers and completes their personal profile.
2. **Initial Triage (Self-Service):** The patient uses the **Chatbot Utility** for an initial assessment to receive guidance on their next steps.
3. **Appointment Selection:** The patient searches, filters, and selects a verified doctor based on their needs, then books an available time slot.
4. **Doctor Confirmation:** The selected doctor receives a notification, logs into their dedicated portal, and formally **approves** the scheduled appointment.
5. **Post-Session Record Keeping:** Following the appointment, the doctor logs essential information, such as the prescription or session notes. This data is then securely stored by the system and made accessible to the patient via their user profile.

4.1.2. Frontend Implementation

Built with React and Vite for fast development, the user interface provides a smooth, dynamic, and engaging user experience.

Technology Stack and Architecture

- **Frontend Framework:** Built with **React.js**, utilizing **Vite** for optimized development and bundling.
- **Styling:** Implemented a utility-first approach using **Tailwind CSS** to achieve a modern, fully responsive, and clean design.
- **State Management:** Application state is managed using a combination of **React Context** for global data and standard local React state for component-specific management.

Key Application Features

- **Doctor Discovery and Listing:** Users can browse, filter, and view detailed profiles of verified mental health specialists across various categories (e.g., Clinical Psychologist, Psychiatrist).
- **Appointment Management:**
    - **Booking Interface:** A streamlined, multi-step process for users to select a doctor, choose an available time slot, and submit their booking request.
    - **Doctor/Admin Dashboard:** A central hub displaying critical operational metrics, including patient lists, appointment schedules, tools for managing prescription issuance, and approving new appointment requests.
- **Personal User Portal:** Features a comprehensive **User Profile & History** section, providing secure access to a user's past appointment history, session notes, and prescriptions issued by their healthcare provider.

4.1.3. Backend Implementation

The backend handles all business logic, data persistence, and API routing using a robust Node.js/Express.js architecture.

- **Framework:** Built on **Node.js** with the **Express.js** framework.
- **Database:** Used **MongoDB** for flexibility in storing user, doctor, and unstructured consultation notes.
- **Authentication:** Implemented **JSON Web Tokens (JWT)** for secure, session-less authentication and enforced **Role-Based Access Control (RBAC)** across User, Doctor, and Admin roles.

4.1.4. AI/Chatbot for Initial Screening and Support

This feature integrates a conversational AI utility designed to provide immediate supportive interaction and offer an initial, non-clinical screening based on user-described concerns.

- **Core Function & Technology**

  This feature integrates a conversational AI using **Google's Gemini API** to provide immediate, supportive interaction. The primary function is to analyze a user's described

concerns and, through prompt engineering, identify a relevant medical specialty. It then provides the user with a **direct hyperlink to the 'All Doctors' page, pre-filtered** for that specific specialty, streamlining the user's journey from triage to booking.

- **Operational Process**
  - **Input:** Real-time user text detailing feelings, concerns, or questions.
  - **Output:** Generates supportive conversational responses and offers a preliminary suggestion for a therapist specialty.
  - **Crisis Handling:** If the user expresses crisis-related or harmful intent, the prompt instructs the model to provide safety-focused guidance and recommend emergency helplines. It *does not* use an automated crisis-classification model.
- **Deployment**
  - The AI model functions as an external API service.
  - The Express.js backend forwards the user's message and system instructions to the dedicated Chatbot API endpoint.
  - The backend returns the generated response to the client interface.

4.1.5. API Development & Integration

The platform's data flow between its frontend and backend services is managed through several essential APIs:

- **User and Doctor Management:** This API is responsible for all CRUD (Create, Read, Update, Delete) operations for both user and doctor profiles, ensuring data is synchronized and verification statuses are properly updated.
- **Appointment Scheduling:** This component handles the doctor's availability calendar, processes booking requests, updates the status of appointments (e.g., Approved, Cancelled, Completed), and manages all related notifications.
- **Chatbot Interface**:This API acts as the real-time bridge to the external Large Language Model (LLM), handling the transmission of user messages and returning supportive, prompt-guided responses.
- **Media Storage:** The platform utilizes the **Cloudinary** service via this API for the secure, external storage of assets such as profile pictures and official doctor credentials.

4.2 Challenges and AI Limitations

4.2.1. Implementation Challenges & Key Issues

Real-time scheduling and sensitive data platform deployment faced significant technical and operational obstacles:

- **Complex Doctor Search and Filtering:** Accurate doctor search required reliable filtering by specialty, insurance, language, and availability. Partial matching caused irrelevant results and poor user experience.
- **Managing High-Concurrency Appointment Booking:** Safely handling simultaneous booking for the *same* slot required transaction controls (database locking) to prevent data conflicts and ensure atomic bookings.
- **Data Security and PHI Compliance:** Protecting sensitive **Patient Health Information (PHI)** and session notes increased complexity. Meeting compliance necessitated stringent security, including **encryption at rest and in transit**.

4.2.2 AI/Chatbot Model Performance and Technical Considerations

The project's conversational system was implemented using **a cloud-hosted large language model (LLM) accessed via API calls**. No custom machine-learning model, dataset preparation, or fine-tuning pipeline was developed. All behavioural control relied on **system-level instructions**, input formatting, and safety-oriented prompt engineering.

4.2.2.1 System Architecture Overview

- **Frontend → Backend → LLM API architecture**.
- The backend transmitted user messages to the LLM along with predefined system instructions.
- The model returned a structured conversational output, which the backend delivered back to the user.
- No local model inference, GPU-based training, or storage of conversation embeddings was used.

4.2.2.2 Prompt Engineering and Behaviour Control

The primary technical mechanism for shaping the chatbot's behaviour was prompt design.

The system prompt defined:

- The tone (empathetic and non-judgmental)
- Scope limitations (no diagnosis, no medical instructions)
- Safety boundaries (how to handle harmful content)
- Conversation structure (short steps, supportive suggestions)

Performance was therefore dependent on the LLM's ability to follow these instructions. While generally reliable, deviations occasionally occurred when user queries were ambiguous or complex.

4.2.2.3 Emotional Understanding Limitations

Because no classifier or rule-based NLP pipeline was implemented, emotional interpretation was fully dependent on:

- The LLM's general pre-trained capabilities
- Keywords and linguistic patterns in the user message
- Instructional hints embedded in the system prompt

This resulted in good performance on explicit emotional statements, but inconsistent performance for:

- sarcasm
- vague distress signals
- indirect cries for help

4.2.2.4 Response Quality Characteristics

The LLM typically produced helpful and context-aware messages. However, some technical limitations were noted:

- **Repetition**: Reused certain empathy phrases due to pattern-matching tendencies.
- **Token-based variability**: Output length varied depending on temperature and prompt length.
- **Statelessness**: No conversation memory; each request was processed as a standalone API call.

4.2.2.5 Ethical and Safety Technical Constraints

Because no crisis-detection model (e.g., classifier, regex engine, or scoring pipeline) was implemented, the system relied on:

- Built-in LLM safety filters
- Prompt-level instructions for crisis handling
  Manual testing for edge cases

This approach is limited, as LLM safety filters operate as black-box systems and may fail in borderline scenarios. Additionally, due to the proprietary nature of the LLM, the team could not apply bias-mitigation techniques, dataset filtering, or adversarial training, resulting in potential downstream bias risks.

4.3 Functional Testing

Functional testing ensures that each feature of the **TherapyCo** operates as expected by users.

Feature Tested	Scenario	Expected Outcome	Status
User Registration	New user signs up with valid details	Account successfully created and stored in database	**Pass**
Invalid Registration Submission	Missing fields or invalid input during signup	Displays appropriate error messages	**Pass**
Login Authentication	User enters valid login credentials	Successfully logs in and is redirected to dashboard	**Pass**
Unauthorized Login	Incorrect password or unregistered email	Displays “Invalid credentials” error	**Pass**

Profile Update	User updates personal information or preferences	Changes saved and reflected in database	**Pass**
Chatbot Interaction	User messages the chatbot	Bot responds with relevant guidance	**Pass** (with occasional delays)

Table 4.1

Functional Testing Result

- The system successfully passes all major functional tests, including user registration, login, profile updates, quiz submission, article access, and chatbot interaction.
- However, improved error handling is recommended for specific cases such as incomplete quiz submissions, slow chatbot responses, and invalid search inputs to ensure smoother user experience.

4.4 Performance Testing

Performance testing evaluates the system's response time, load handling, and overall efficiency.

Process	Expected Time	Actual Time	Performance Status
User Registration	< 2 sec	~1.4 sec	Fast
User Login	< 2 sec	~1.1 sec	Fast
Loading Mental Health Articles/Resources	< 3 sec	~2.7 sec	Acceptable

Self-Assessment Quiz Submission	< 3 sec	~2.4 sec	Good
Chatbot Response Retrieval	< 2 sec	~2.1 sec (slight delay)	Acceptable
Bulk Resource Fetch (large dataset)	< 5 sec	~4.6 sec	Acceptable

Table 4.2

Performance Testing Result

- Overall performance is acceptable across major features.
- Some delays occur when loading large datasets; optimization is recommended.
- Batch processing or improved caching can help enhance efficiency.

4.5 Security Testing

Security testing ensures that the system is protected from vulnerabilities.

Tested Scenario	Potential Risk	Security Measures
SQL Injection	Malicious queries can alter or access user data	Strong input validation & parameterized queries
Brute Force Login Attempts	Repeated login tries may compromise accounts	Account lockout after multiple failed attempts
Data Encryption	Exposure of sensitive user information	AES-256 encryption for stored data and secure HTTPS communication

API Key Exposure	Hardcoded keys may be leaked or misused	All API keys stored safely in environment variables
Session Hijacking	Unauthorized access through stolen cookies	Secure, HTTP-only cookies with expiry and token-based authentication

Table 4.3

Security Testing Result

- Adding **Two-Factor Authentication (2FA)** or **Google Sign-In** can further strengthen login security.
- Implementing **automatic logout after user inactivity or browser closure** will help prevent unauthorized access.

4.6 Usability Testing

User feedback was collected from students and general users to improve overall usability.

Aspect	User Feedback
Ease of Use	Clean interface with easy navigation across resources and dashboard.
Content Clarity	Articles are helpful, but users want more visual elements and interactive content.
Error Messages	Clear, specific error messages need slight improvement.
Mobile Compatibility	Works well on mobile devices, though some pages require better responsiveness.

Table 4.4

Improvement Suggestions

- Add interactive visuals to enhance user engagement.
- Improve responsiveness for a smoother experience across devices.

In summary, the testing phase confirmed that all core functionalities of the platform are operational. The system successfully passed critical functional and security tests. The partial-pass results in performance and edge-case testing provided a clear roadmap for future optimizations, specifically in database query caching and refining the chatbot's handling of ambiguous user input.

Chapter 5

Standards Adopted

5.1 Design Standards

In all engineering streams, predefined design standards ensure quality, safety, and consistency. Some widely used standards include IEEE, ISO, ANSI, ASME, and IEC. In software development, UML diagrams and database design standards are followed to maintain structured architecture. This project, TherapyCo, adheres to best practices across different layers:

5.1.1 Frontend Design Standards

Framework & Styling: The frontend is developed using React and bundled with Vite.The interface is styled using Tailwind CSS, ensuring a clean and consistent UI.

Component-Based Architecture: The frontend follows a modular, reusable component structure, improving maintainability.

Accessibility & Responsiveness: Tailwind CSS ensures responsiveness across mobile, tablet, and desktop devices. Basic accessibility practices are followed for readable and user-friendly interfaces

Routing & Navigation: Client-side routing is handled using react-router-dom, enabling smooth navigation across pages such as login, resources, chatbot, and dashboard.

State Management: The project uses React Hooks (useState, useEffect, useContext) for managing UI and component-level state.

5.1.2 Backend Design Standards

Database Design: The backend uses MongoDB, managed through Mongoose and MongoDB Atlas for data modeling and interaction.

API Development: Backend services are built using Express.js, following RESTful API structure. API endpoints are organized based on functionality for clear and maintainable communication between frontend and backend.

Authentication & Security: User authentication is implemented using bcryptjs for password hashing jsonwebtoken (JWT) for generating and validating authentication tokens

File Upload & Media Handling: Cloudinary is used for storing and managing media files
LLM / AI Integration: The system integrates a pre-trained Large Language Model (LLM) through an external API.

5.2 Coding Standards

Coding standards are essential for maintaining code quality, readability, and consistency. The following practices have been adopted for this project:

Concise Code: Writing as few lines as possible while maintaining clarity and functionality.

Naming Conventions: Using meaningful and consistent naming for variables, functions, and components (e.g., camelCase for JavaScript variables, PascalCase for React components).

Code Segmentation: Segmenting blocks of code into logical paragraphs to improve readability.

Indentation and Structure: Clearly marking the beginning and end of control structures using indentation, ensuring clean and structured code.

Function Simplicity: Keeping functions short and focused, ensuring that each function performs a single, well-defined task.

Reusable Components: Following a modular approach by creating reusable components in Next.js to enhance maintainability.

Error Handling: Implementing proper error handling mechanisms using try-catch blocks and logging errors for debugging.

Comments and Documentation: Providing meaningful comments and maintaining

documentation for complex logic to aid future developers

5.3 Testing Standards

Quality assurance and testing follow established ISO and IEEE standards to ensure reliability

and performance. This project uses structured testing methods for all major modules of the , validating workflow accuracy and system behavior.

<u>Test Case 1 — User Authentication and Login</u>

Test Case ID: TC-001
Objective: Verify that patients, doctors and admins can log in with valid credentials and are prevented from accessing the system with invalid credentials or without the correct role.
Preconditions:

1. User accounts (patient, doctor, admin) exist in the database.
2. Authentication service / API is running (frontend -> backend reachable).

Test Steps:

1. Open the site and navigate to the Login page.
2. Attempt login as a **patient** with valid email/username + password.
3. Repeat login using a **doctor** account and an **admin** account.
4. Attempt login with an incorrect password for a known user.
5. Attempt login with a non-existent email/username.
6. After successful login, try to access an admin-only page with a non-admin account.

Expected Result:

1. Valid credentials authenticate the user and redirect them to the appropriate landing page (patient dashboard / doctor dashboard / admin dashboard).

2. Invalid credentials show a clear error message (e.g., “Invalid email or password”) and do not log the user in.
3. Role-based access control prevents non-admins from accessing admin pages (should show 403/redirect).

Post-conditions: User is either logged in with the correct role or access is denied.

Test Case 2 — Doctor / Patient Search and Filtering

Test Case ID: TC-002
Objective: Validate search and filtering for finding doctors (or patients) — e.g., by name, specialty, availability or location.
Preconditions:

1. The system contains doctor and patient records with specialties, availability slots, and locations.
2. Search/filter UI and backend endpoints are available.

Test Steps:

1. Navigate to the “Find Doctor” / “Profiles” page.
2. Enter a doctor’s name in the search bar and submit.
3. Apply filter: specialty (e.g., “Clinical Psychologist”).
4. Apply filter: availability (e.g., “Next 7 days” or “Online consultations only”).
5. Combine search + multiple filters.
6. Click “Reset filters” or clear search.

Expected Result:

1. Search returns matching profiles by name.
2. Filters narrow results dynamically (only doctors matching specialty and availability appear).
3. Combined filters work together and return intersection of criteria.
4. Reset/clear restores full list.

Post-conditions: Users can locate suitable doctors/patients efficiently.

Test Case 3 — Appointment Booking & Management Flow

Test Case ID: TC-003
Objective: Ensure appointment booking, modification, cancellation and notifications work correctly between patients, doctors and admin.
Preconditions:

1. Appointment API and database are up and reachable.
2. At least one patient and one doctor exist; doctor has available slots.

Test Steps:

1. As a patient, navigate to a doctor's profile and select an available slot.
2. Submit booking (provide reason/notes as required).
3. Verify booking appears in patient's "My Appointments" and doctor's schedule.
4. As doctor, accept or reject the appointment (if such flow exists).
5. As patient, attempt to modify the appointment time to another available slot.
6. Cancel the appointment from patient side.
7. Check that appointment status and history are updated and stored.
8. Verify that notifications/emails (or in-app alerts) are sent to doctor and patient for booking/acceptance/cancellation.

Expected Result:

1. Booking created successfully and shows in both patient and doctor views.
2. Status transitions (pending → confirmed → cancelled) update correctly and persist.
3. Modifications are validated against availability to prevent double-booking.
4. Notifications are delivered when actions occur.

Post-conditions: Appointment records reflect the final state (created/modified/cancelled) and logs/notifications exist.

Test Case 4 — Add New User (Doctor / Patient) — Admin Flow

Test Case ID: TC-004

Objective: Verify that admin can create new doctor or patient accounts and that validations work.

Preconditions:

1. The tester has admin credentials.
2. User creation form / endpoint is available.

Test Steps:

1. Log in as admin and navigate to "User Management" or "Add New Doctor/Patient."
2. Fill required fields (full name, email, password, role, specialty for doctors, contact info).
3. Intentionally leave required fields blank to test validation.
4. Submit the form to create the user.
5. Search for the newly created user in the user list.
6. Attempt to create another user with the same email to test duplicate-check.

Expected Result:

1. Valid user details create the account and it appears in user list.
2. Form validation prevents submission if required fields are empty and displays helpful error messages.
3. Duplicate email is rejected with an appropriate error.
4. Created user can log in (if required verification completed) or receives invitation/email as per app flow.

Post-conditions: New user record exists and is accessible per role.

Chapter 6

Conclusion and Future Scope

6.1 Final Thoughts

Developing TherapyCo has been a meaningful and insightful experience. What started as an effort to create a simple platform for awareness gradually evolved into a system that offers curated mental health resources, personalized recommendations, and an interactive chatbot. Throughout the journey, we looked into how technology can support emotional well-being and make mental health information more accessible. We learned to manage real-world constraints like responsiveness, performance, and user-centered design, balancing an intuitive UI with reliable backend functionality.

Building this platform wasn’t just a technical task; it was a chance to understand the importance of mental health in daily life. The process helped us improve our skills in full-stack development, API integration, testing, documentation, and UX design. It also promoted better collaboration as a team. This project helped us see how digital tools can empower individuals to learn, self-reflect, and seek guidance while keeping privacy, simplicity, and ease of access in mind.

6.2 Looking Ahead

Although the Mental Health Support Website works well as a prototype right now, there is still a lot of space for improvement and scalability in the future. Our next course of action is to increase the system's intelligence, personalisation, and support by extending its capabilities.

Scaling the platform to accommodate more users, content, and real-time interactions is one possible improvement.

- Incorporating more sophisticated ML models to fortify the recommendation engine.
- Enhancing the chatbot with natural language processing to give more sympathetic, contextual responses.

- Incorporating sophisticated analytics, including progress monitoring, mood trends, and customised insights.
- Incorporating peer-supporting community features like group discussions or moderated forums.
- Introducing structured journaling tools or guided programs for mental health.

The platform has the potential to develop into a full-fledged digital wellness companion in the future, providing tools for emotional well-being, personalised content, and preventive support. We hope this project can have a positive impact on people who are seeking help and spur future advancements in digital mental health solutions with ongoing user feedback and incremental development.

References

1. Tailwind Labs, "Tailwind CSS Documentation," 2024.
   Available: https://tailwindcss.com/docs

2. JavaScript, "MDN Web Docs - Mozilla".
   Available: https://developer.mozilla.org/en-US/docs/Web/JavaScrip

3. ReactJS, "React Official Documentation," Meta Open Source, 2024.
   Available: https://react.dev

4. Google, "Material Design Guidelines for Accessibility," 2023.
   Available: https://material.io/design

5. World Health Organization (WHO), "Mental Health: Strengthening Our Response," 2023.
   Available: https://www.who.int

6. American Psychological Association, "Mental Health Resources," APA, 2024.
   Available: https://www.apa.org

7. W3C, "Web Content Accessibility Guidelines (WCAG 2.1)," 2018.
   Available: https://www.w3.org/WAI/standards-guidelines/wcag

8. OpenAI, "Building Conversational AI Interfaces," OpenAI Guides, 2024.
   Available: https://platform.openai.com/docs

9. GPT-based Chatbot Integration Docs, "Best Practices for Chat UI & API Usage," 2024.
10. F. C. Bartlett, Health Psychology and Digital Interventions, Springer, 2020.

11. A. Cooper et al., About Face: The Essentials of Interaction Design, Wiley, 2014.

12. J. Nielsen, Usability Engineering, Morgan Kaufmann, 1993.

13. Mental Health America (MHA), "Online Mental Health Screening Information," 2024.
    Available: https://mhanational.org

14. GitHub, "Open Source Mental Health Web Projects," 2024.
    Available: https://github.com

Appendix A: Project Dependencies and Environment Setup

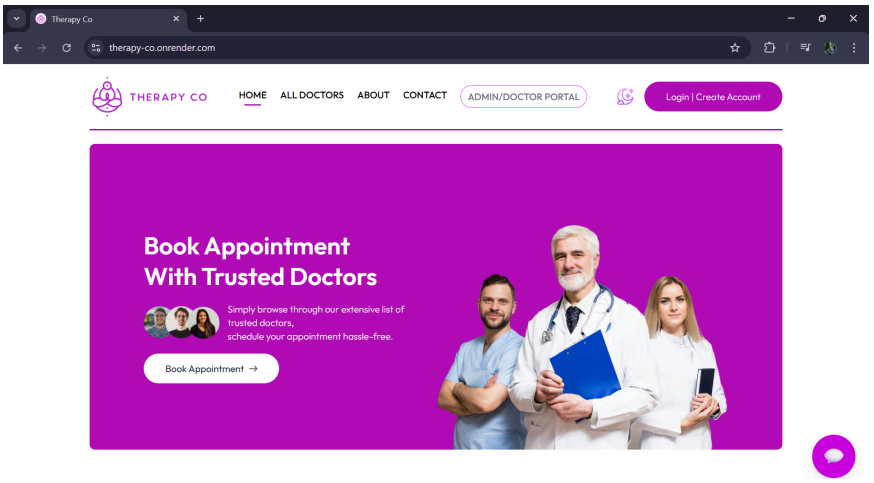
The project requires the following primary dependencies across the frontend, backend, and AI/ML components:

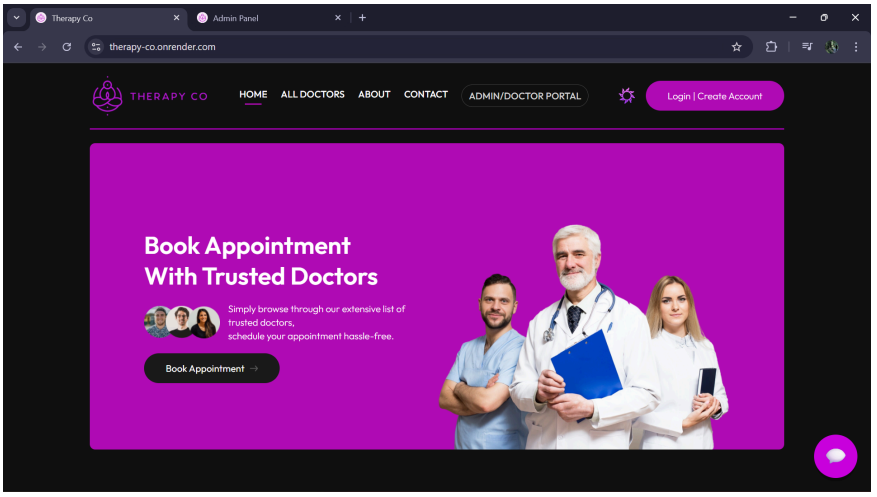
- **Frontend Technologies:**
  - react
  - react-router-dom
  - tailwindcss
  - vite
- **Backend Technologies:**
  - express
  - mongoose (for MongoDB interaction)
  - bcryptjs (for secure password hashing)
  - jsonwebtoken (for JSON Web Tokens - JWTs)
  - multer
  - cloudinary
- **AI/ML Technologies and Dependencies:**

The system leverages a **pre-trained Large Language Model (LLM)** accessed via an external API, resulting in minimal dependencies. Crucially, no local Natural Language Processing (NLP) libraries (such as Transformers, TensorFlow, or PyTorch) are needed. The only required dependencies are those necessary for the backend's API integration.

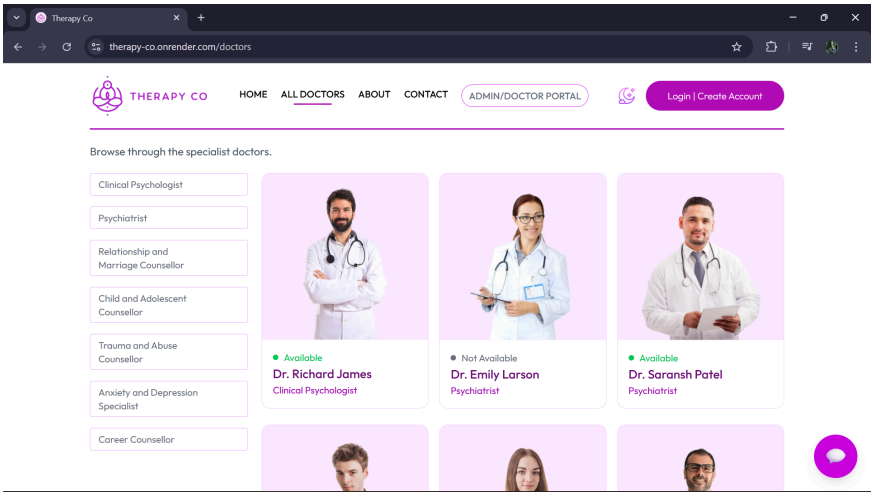
Appendix B: Application Screenshots.

B.1 Landing Page:

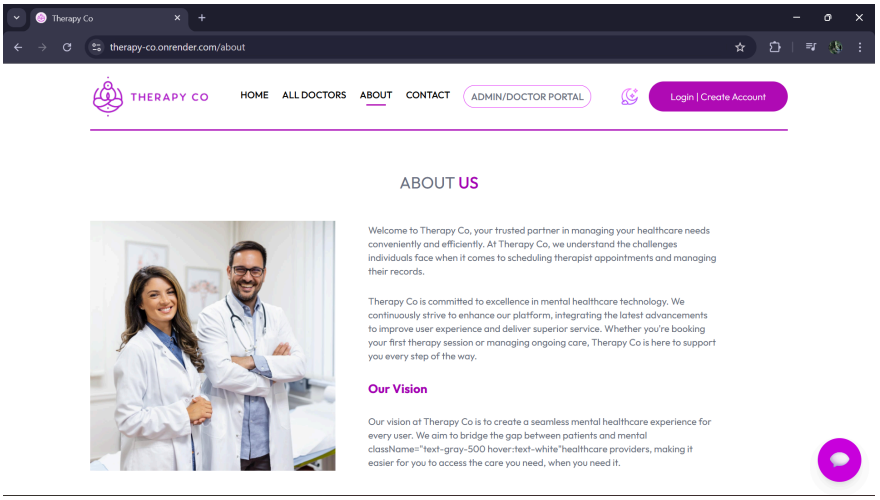




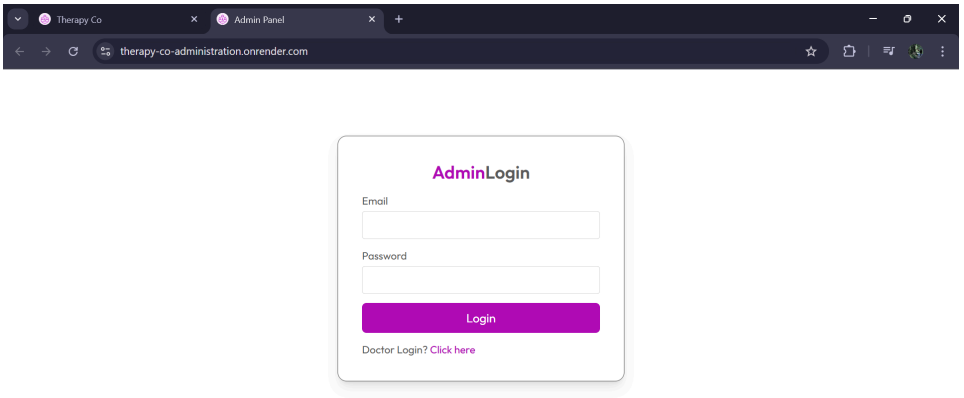
B.2 All Doctors Page:



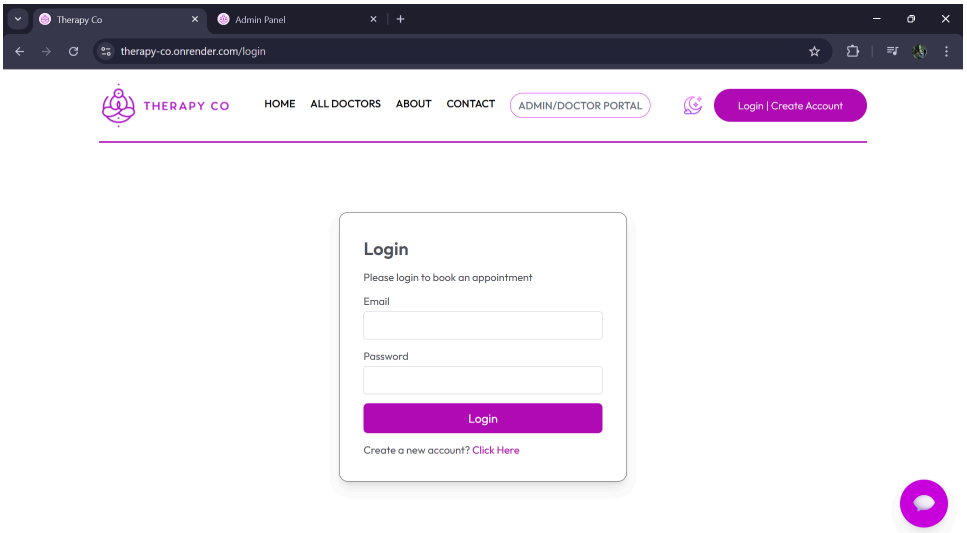
B.3 About Us Page:

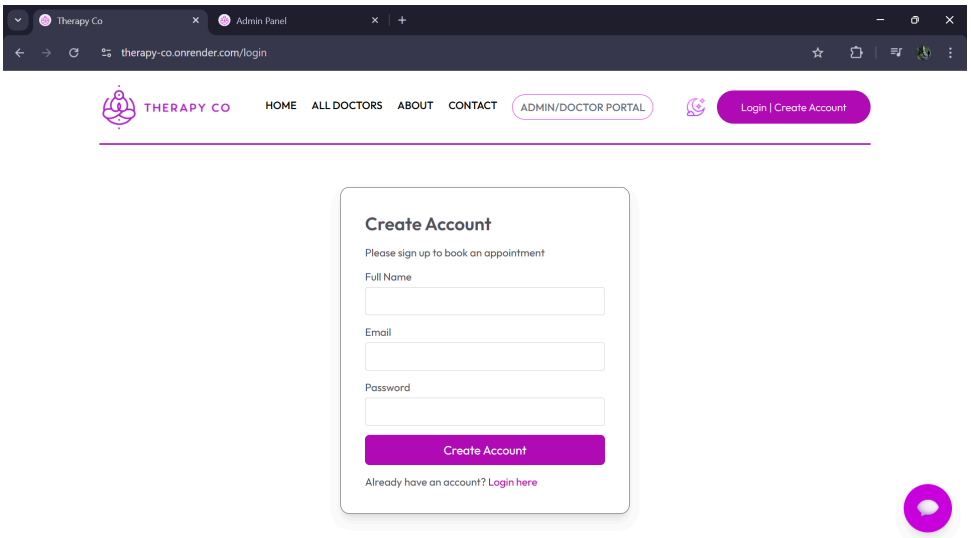


B.4 Admin/Doctor Login Page:

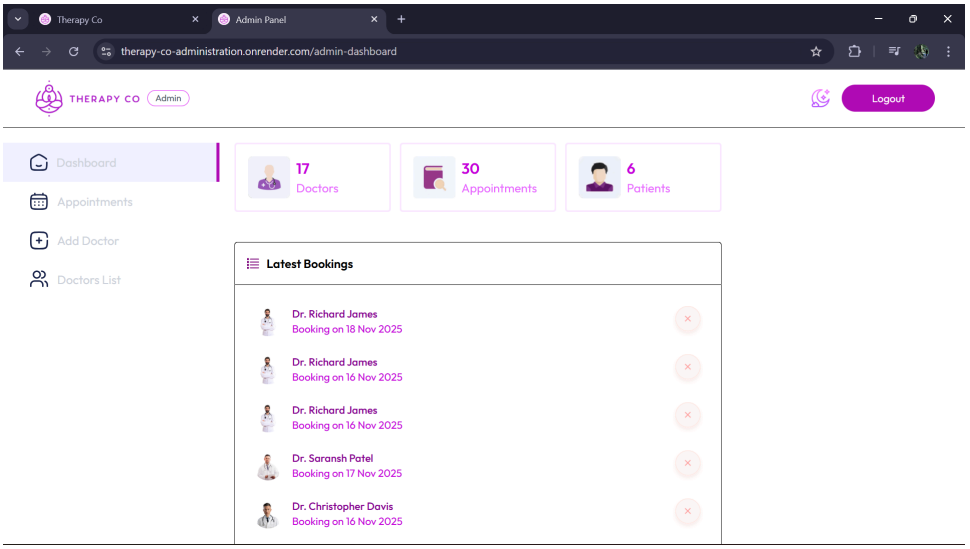


B.5 Login/ Create Account page for Patients:

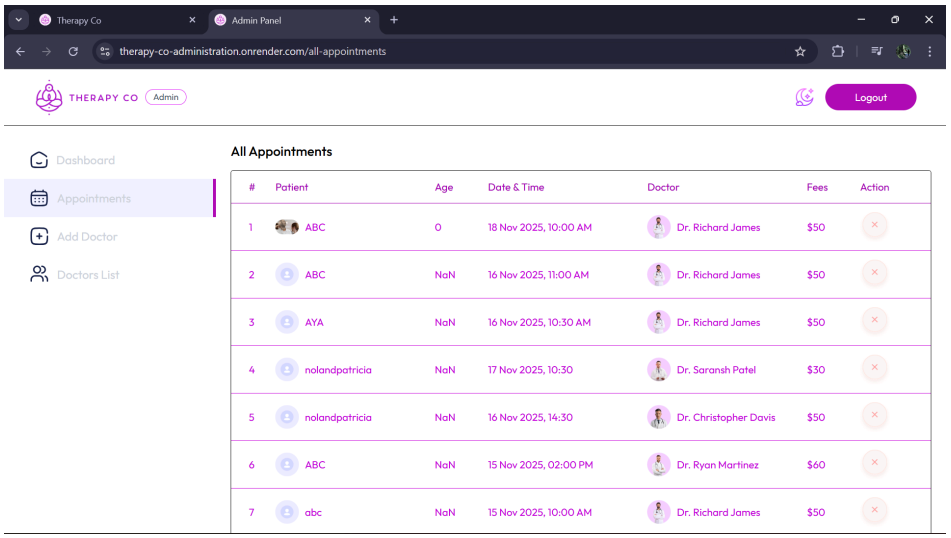




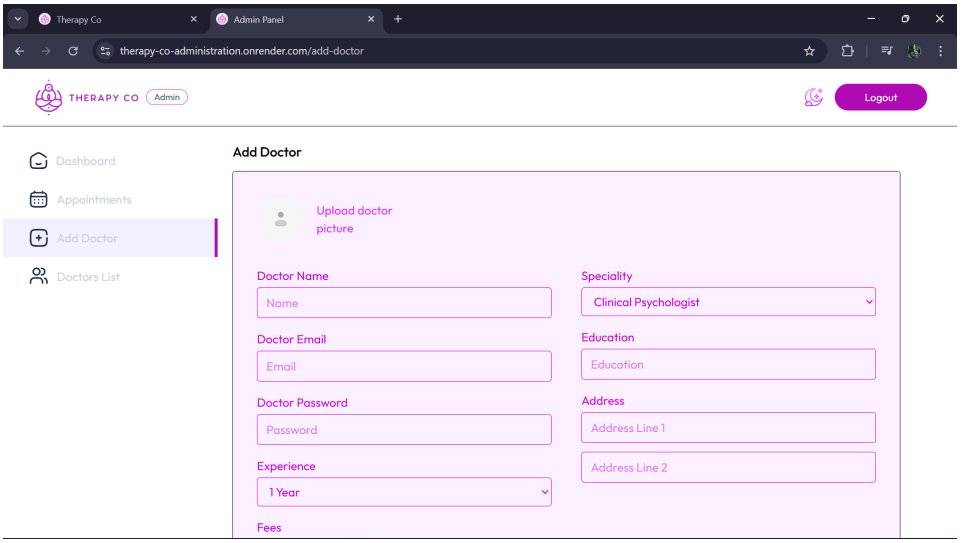
B.6 Admin's Dashboard



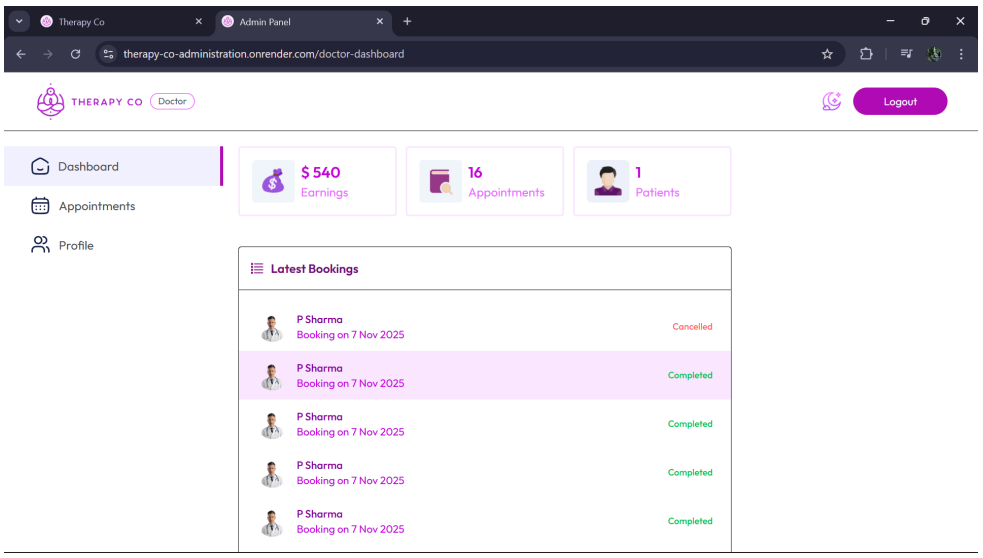
B.7 Admin→ All Appointments



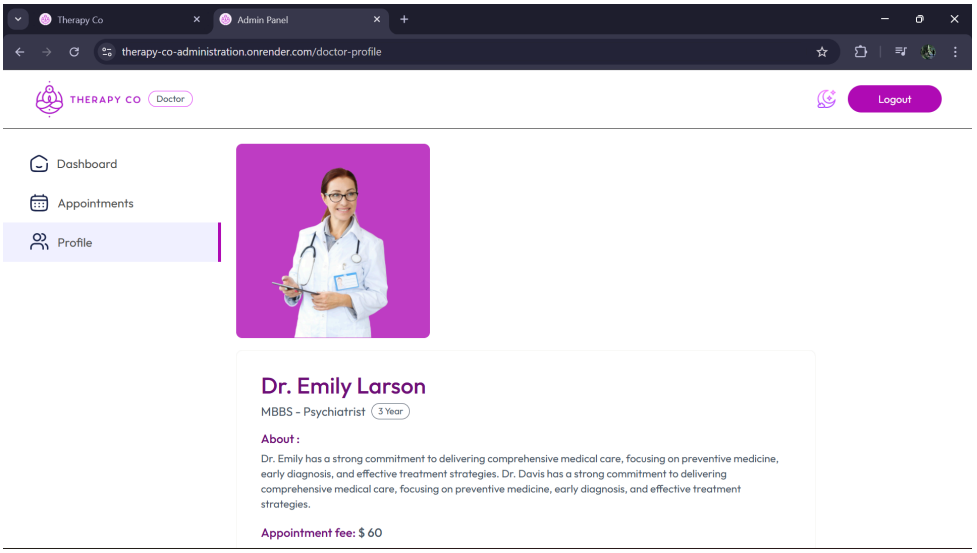
B.8 Admin → Add Doctor



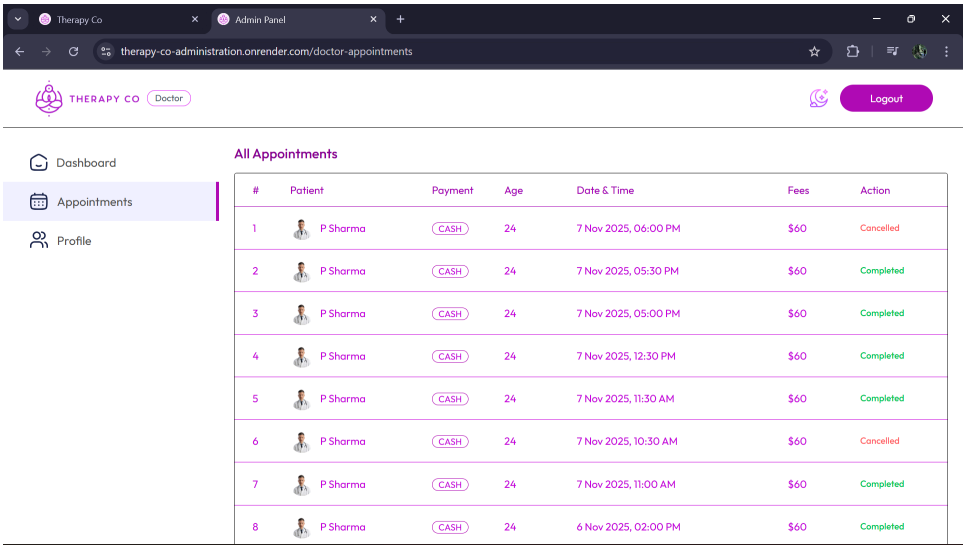
B.9 Doctor's Dashboard



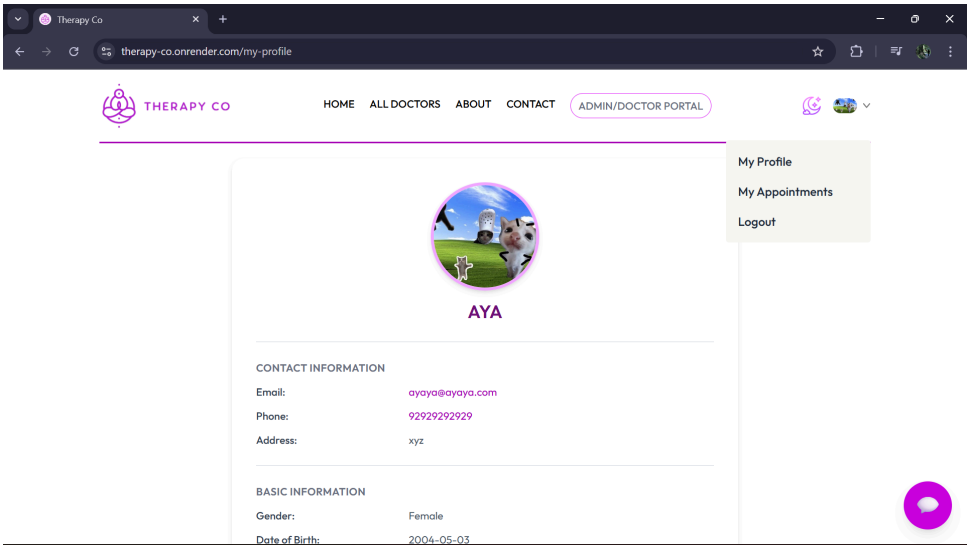
B.10 Doctor's Profile



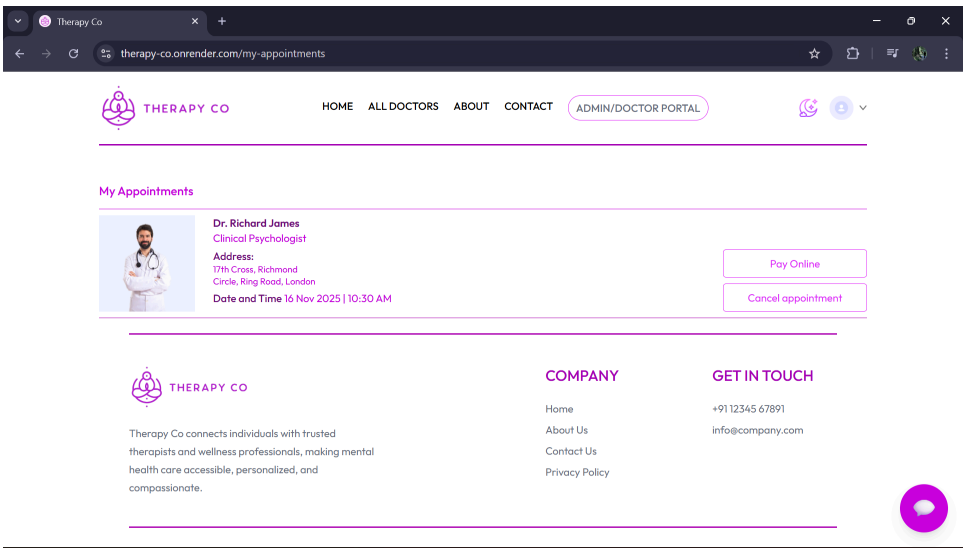
B.11 Doctor's Patient list



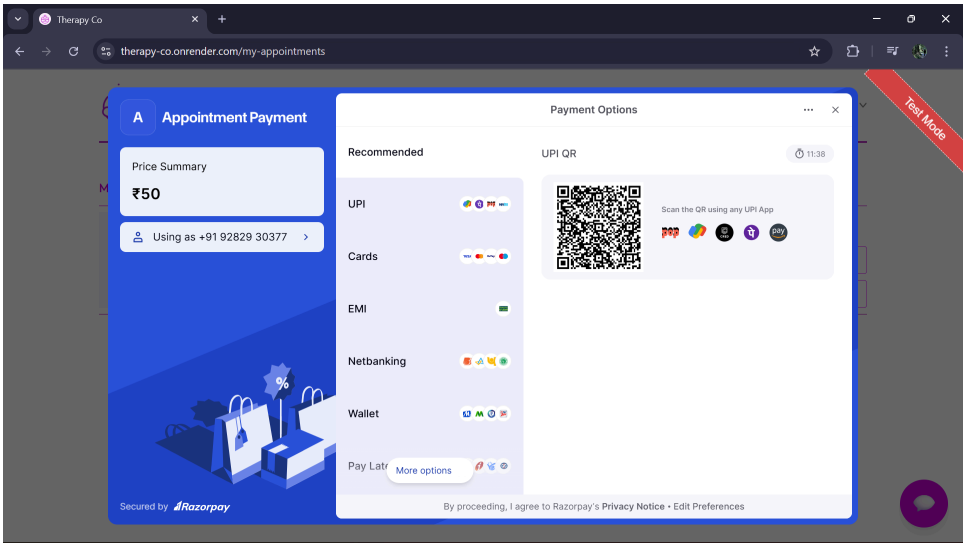
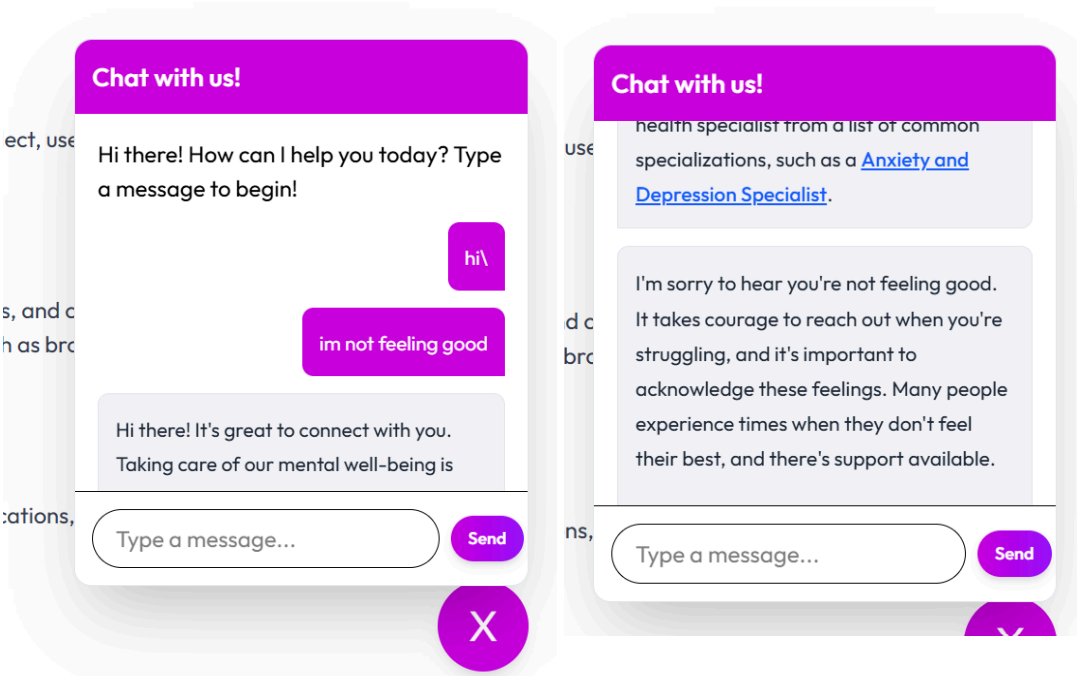
B.12 Patient's Account Page



B.13 My Appointments Page



B.14 Razorpay Payment Gateway



B.15 Chatbot