SoulCure
Online Therapy Website

Mini Project Report

Submitted by

Amal Raj

Reg. No.: AJC19MCA-I012

In Partial fulfillment for the Award of the Degree of

INTEGRATED MASTER OF COMPUTER APPLICATIONS

(INMCA)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC with 'A' grade. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2023-2024

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

<u>CERTIFICATE</u>

This is to certify that the Project report, “**SOULCURE”** is the bona fide work of **AMAL RAJ (Regno: AJC19MCA-I012)** in partial fulfillment of the requirements for the award of the Degree of Integrated Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2023-24.

Ms. Lisha Varghese	**Ms. Meera Rose Mathew**
Internal Guide	**Coordinator**

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department

DECLARATION

I hereby declare that the project report **“SoulCure”** is a bona fide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Master of Computer Applications (MCA) from APJ Abdul Kalam Technological University, during the academic year 2023-2024.

Date: 23/10/2023	**AMAL RAJ**
KANJIRAPPALLY	**Reg: AJC19MCA-I012**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Manager **Rev. Fr. Dr. Mathew Paikatt** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev.Fr.Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Ms. Meera Rose Mathew** for his valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms. Lisha Varghese** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

AMAL RAJ

ABSTRACT

.

The primary goal of SoulCure is "to support users with the best care to heal and regain their physical and emotional health". It is a platform with a wide range of therapeutic techniques to promote holistic healing and well-being. The website employs various diverse therapeutic modalities like individual therapy, music therapy, family therapy, and medication management, to address various aspects of users' health and improve general wellness. Different therapies come with a variety of benefits which includes, physical healing, overcoming communication problems, recovery from drug addiction. Also offer a regularly updated blog or articles section where users can find informative and educational content related to health, well-being, and therapy. The users receive all the necessary tools and training materials for maintaining their personal wellbeing. It aims to provide a wholistic approach to healing and personal growth for people who require therapeutic treatment.

The purpose of the proposed system is to create a convenient, holistic and therapeutic platform that can comprehensively address various physical, mental and emotional health aspects. It strives to provide users with the tools and support needed to maintain personal wellbeing through therapeutic healing. During the coding and design of the system using front-end technologies such as HTML, CSS, and JavaScript with Visual Studio IDE, features are provided to elevate and enhance every stage of software development. Additionally, back-end technologies such as Python Django is utilized to enhance the system's functionality and interactivity. Furthermore, Sqlite3, a powerful and open-source relational database management system, is used as the back-end database as per the requirements of the system.

CONTENT

SL. NO	TOPIC	PAGE NO
1	INTRODUCTION	1
1.1	PROJECT OVERVIEW	2
1.2	PROJECT SPECIFICATION	2
2	SYSTEM STUDY	4
2.1	INTRODUCTION	5
2.2	EXISTING SYSTEM	5
2.3	DRAWBACKS OF EXISTING SYSTEM	6
2.4	PROPOSED SYSTEM	7
2.5	ADVANTAGES OF PROPOSED SYSTEM	7
3	REQUIREMENT ANALYSIS	8
3.1	FEASIBILITY STUDY	9
3.1.1	ECONOMICAL FEASIBILITY	9
3.1.2	TECHNICAL FEASIBILITY	10
3.1.3	BEHAVIORAL FEASIBILITY	10
3.1.4	FEASIBILITY STUDY QUESTIONNAIRE	10
3.2	SYSTEM SPECIFICATION	12
3.2.1	HARDWARE SPECIFICATION	12
3.2.2	SOFTWARE SPECIFICATION	12
3.3	SOFTWARE DESCRIPTION	12
3.3.1	DJANGO	12
3.3.2	SQLITE	13
4	SYSTEM DESIGN	14
4.1	INTRODUCTION	15
4.2	UML DIAGRAM	15
4.2.1	USE CASE DIAGRAM	16
4.2.2	SEQUENCE DIAGRAM	18
4.2.3	STATE CHART DIAGRAM	20
4.2.4	ACTIVITY DIAGRAM	22
4.2.5	CLASS DIAGRAM	24
4.2.6	OBJECT DIAGRAM	26
4.2.7	COMPONENT DIAGRAM	28

4.2.8	DEPLOYMENT DIAGRAM	29
4.3	USER INTERFACE DESIGN USING FIGMA	30
4.4	DATABASE DESIGN	34
5	SYSTEM TESTING	44
5.1	INTRODUCTION	45
5.2	TEST PLAN	45
5.2.1	UNIT TESTING	45
5.2.2	INTEGRATION TESTING	46
5.2.3	VALIDATION TESTING	46
5.2.4	USER ACCEPTANCE TESTING	47
5.2.5	AUTOMATION TESTING	47
5.2.6	SELENIUM TESTING	47
6	IMPLEMENTATION	58
6.1	INTRODUCTION	59
6.2	IMPLEMENTATION PROCEDURE	59
6.2.1	USER TRAINING	60
6.2.2	TRAINING ON APPLICATION SOFTWARE	60
6.2.3	SYSTEM MAINTENANCE	60
7	CONCLUSION & FUTURE SCOPE	61
7.1	CONCLUSION	62
7.2	FUTURE SCOPE	62
8	BIBLIOGRAPHY	63
9	APPENDIX	65
9.1	SAMPLE CODE	66
9.2	SCREEN SHOTS	73

List of Abbreviation

IDE	-	Integrated Development Environment.
HTML	-	Hyper Text Markup Language.
CSS	-	Cascading Style Sheet
UML	-	Unified Modeling Language
RDBMS	-	Relational Database Management System

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The proposed system titled "**SoulCure**" is a website that aims to provide holistic healing and wellbeing to users through a diverse range of therapy options. The main goal is to enable people to conveniently connect with qualified therapists, book appointments, access resources and get personalized recommendations to improve their physical, mental and emotional health. The system allows users to register online and create profiles. It offers services like individual therapy, music therapy, family therapy, meditation classes, health questionnaires, yoga plans, and informative articles/blogs. It aims to provide a wholistic approach to healing and personal growth for people who require therapeutic treatment.

The platform has four primary types of users - administrator, therapists, patients, and editors. The administrator manages user accounts, therapies, appointments, add therapists, content and security on the website. Therapists can create their profile, view booked appointments, conduct online sessions, and apply for leave. Patients can register, search for therapists, book and manage sessions, take assessments and access resources. Editors develop health and wellness articles, review content and publish articles after approval. This system also has chatbot technology so customers will get instant response. SoulCure is developed using HTML, CSS and JS as frontend, Python Django as a powerful backend and RDBMS Sqlite3 is used.

1.2 PROJECT SPECIFICATION

The proposed system is an online platform that aims to provide holistic healing and wellbeing through a diverse range of therapy options. The primary objective is to enable convenient access to therapeutic services by allowing users to search, book appointments, and attend online sessions with qualified therapists. The system is built to serve three primary user categories: Administrators, Therapists, and Clients. Additionally, it incorporates a set of supplementary features, including an AI yoga assistant, a basic chatbot, and notifications through WhatsApp and email. This platform is geared towards creating a seamless and efficient therapeutic experience.

1.2.1 ADMIN MODULE

The Administration module provides overarching control and management of the system. Admin users have secure login access to the platform, empowering them to oversee therapists and clients. They can add, update, or remove therapist and client profiles. Moreover, they possess the authority to manage therapists' profiles, including specifying their therapy specialization, whether it's in meditation or family therapy. Additionally, admin users can add and update various therapies offered by therapists, contributing to a well-rounded system. The admin module also encompasses appointment management. Admins can view and regulate therapist appointments while maintaining a detailed log of appointment records.

1.2.2 THERAPIST MODULE:

Therapists within the system have the ability to manage their profiles. They can create and update their profiles, including adding details about their specialization and personal information. This feature enables therapists to present a professional and informative image to potential clients. Appointment management is a crucial part of the therapist module. Therapists can view appointments relevant to their specialization, such as Family Therapy, and schedule online therapy sessions with ease. They also have access to a comprehensive appointment log that provides a history of past and upcoming sessions. Additionally, therapists can utilize the AI yoga assistant for yoga-related guidance and engage with the basic chatbot for general information and support.

1.2.3 CLIENT MODULE:

The client module offers a user-friendly experience. Clients can register for an account by providing their personal information. Once registered, they gain the capability to create and maintain their profiles. The profile management feature allows clients to keep their information up to date for a seamless experience. Clients can search for therapists based on specialization and location, ensuring that they can find the most suitable therapist for their needs. They can also book and cancel appointments, specifically within their chosen therapy area, and keep track of their appointment history. As with therapists, clients have access to the AI yoga assistant and basic chatbot for assistance and information. They can also receive important appointment reminders and other updates through WhatsApp and email notifications.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

A critical step in system development is system analysis, which involves gathering and analyzing data to identify problems and provide solutions. During this phase, effective communication between system users and developers is essential. Any system development process should always begin with a system analysis. The system analyst performs the role of an investigator by carefully examining the current system's performance. Identification of the system's inputs and outputs as well as a connection between its processes and organizational results are all included. A range of techniques, including surveys and interviews, are used to acquire information. Understanding the system's operation, locating problem areas, and suggesting fixes to the issues the business is having are the objectives. The designer assumes the role of issue solver, and the suggested fixes are rigorously compared with the current system. The user is given the chance to accept or reject the advice after the best choice has been chosen. The procedure continues until the user is pleased after the idea has been evaluated in light of their comments. The process of acquiring and analyzing data for future system studies is called a preliminary study. Conducting a thorough preliminary study is essential to ensure the success of a system development project.

2.2 EXISTING SYSTEM

The existing system for online therapy primarily connects users with qualified therapists through scheduled appointments. Patients have the option to meet with therapists either in person or through video and chat sessions. This approach provides individuals with easy access to professional support from the comfort of their own homes, ensuring convenience and accessibility. The system emphasizes the provision of quality therapeutic services, promoting mental well-being, and expanding access to exceptional therapy. It facilitates connections with qualified therapists, ultimately enhancing overall mental health support.

2.2.1 NATURAL SYSTEM STUDIED

The natural system, as it currently exists, primarily involves manual communication channels like phone calls, emails, or face-to-face interactions to schedule therapy appointments. Clients must take the initiative to contact therapy providers or individual therapists, inquiring about their availability and arranging appointments. This manual approach often results in a protracted exchange of messages to secure a mutually suitable appointment time for both clients and therapists. The limitations of this manual approach have prompted the need for a more streamlined and technologically advanced system to optimize therapy appointment scheduling, which is the primary focus of this natural system studied.

2.2.2 DESIGNED SYSTEM STUDIED

In response to the inefficiencies inherent in the existing manual processes for therapy appointment scheduling, a modern Therapy Appointment Management System is proposed. This system aims to revolutionize the way clients and therapists connect by providing a streamlined, user-friendly, and efficient solution. Key features include a digital platform for clients to access therapist profiles, book appointments, and receive real-time notifications, greatly reducing the need for time-consuming phone calls and emails. The system's design focuses on enhancing convenience, transparency, and the overall experience of accessing therapy services, while simultaneously addressing the limitations of the existing system and embracing the advantages of digital technology.

2.3 DRAWBACKS OF EXISTING SYSTEM

- **Inefficiency:** The manual process can be highly inefficient, often involving numerous rounds of communication between clients and therapists to secure a suitable appointment time.
- **Limited Access:** This system may limit access to therapy services to those who are proficient in navigating the process, potentially leaving less tech-savvy individuals at a disadvantage.
- **Lack of Transparency:** Clients may not have access to comprehensive information about therapists, making it challenging to make informed choices.
- **Missed Appointments:** Without automated reminders and notifications, there is a higher likelihood of missed appointments, leading to frustration for both clients and therapists.
- **Security Concerns:** Sharing personal information over email or phone can raise security concerns, particularly when discussing sensitive therapy topics.
- **Limited Availability:** The system's reliance on manual processes can create scheduling challenges for clients with busy schedules or those requiring therapy outside of standard working hours.
- **Inconvenience:** In-person visits can be inconvenient, especially when clients and therapists are geographically distant, necessitating multiple trips for meetings.

2.4 PROPOSED SYSTEM

The proposed technique offers people a wide range of therapy options, letting them choose the kind of assistance that best meets their needs. The platform ensures that help is always available for users so that they may receive assistance whenever they need it. The website allows users to make appointments with the nearby qualified therapists, ensuring accessibility and convenience. The system uses a wide range of therapeutic modalities, such as physiotherapy, music therapy, family therapy, and meditation classes, to provide holistic care. It aims for holistic recovery and enhance health in general. Additional modules include feedback systems, yoga assistance, and a basic chatbot, with the aim of streamlining the therapy experience, empowering clients, and making therapy services more accessible.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- **Efficiency:** Streamlined processes reduce the time and effort required for both clients and therapists.
- **Convenience:** Clients can access therapy services from the comfort of their homes, eliminating the need for physical visits.
- **Transparency:** Clients have access to detailed therapist profiles, reviews, and information about expertise to make informed decisions.
- **Secure Payments:** Payment integration ensures secure financial transactions for therapy sessions.
- **Empowering Users:** The system bridges the gap between therapy education and practice, providing opportunities for interns and law students.
- **Enhanced Scheduling:** Integration of therapist calendars allows clients to align their schedules with therapist availability efficiently.
- **Personalized Yoga Guidance:** The AI yoga assistant provides personalized yoga guidance, tailoring yoga practices to the individual's needs and goals.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software. Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study. The results of the feasibility study should be a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process.

The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives of feasibility study are listed below.

- To analyze whether the software will meet organizational requirements
- To determine whether the software can be implemented using the current technology and within the specified budget and schedule.
- To determine whether the software can be integrated with other existing software.

Feasibility study is a crucial step in determining the viability of the SoulCure platform concerning resources, time, and organizational goals. By conducting this study, the developers can assess the potential long-term advantages and opportunities offered by SoulCure. The primary objective is to ascertain whether the proposed system is practical and worthy of further investigation. The feasibility study examines SoulCure's potential effects on the holistic health industry, its capacity to satisfy client demand, and its effectiveness in making use of available resources.

3.1.1 Economical Feasibility

The web-based business has the limited measure of cash to put resources into the framework's innovative work. The expense should be upheld by proof. Because the majority of the technologies utilized are freely available, the creating system should fit within the budget. By digitizing the procedures, the proposed system eliminates the use of paper. The creating framework should be legitimate by cost and advantage. Criteria to ensure that efforts are directed toward the project that will yield the best results the quickest. The cost of developing a new system is one factor that influences its development. The proposed system is developed as part of the project's work; there are no direct costs associated with it. Additionally, every one of the assets are now accessible, it gives a sign of the framework is financially feasible for improvement.

3.1.2 Technical Feasibility

The proposed system can be implemented technically. There will be no barriers to production affecting the services or functionalities' performance. The proposed framework should have a humble prerequisite, as just insignificant or invalid changes for the executing this framework. Since the system's frontend is developed using HTML, CSS, and JavaScript. These technologies are widely used and have a large developer community, making it easier to find solutions and resolve issues. And back-end have been developed with Django and Sqlite3, the project can be developed technically.

3.1.3 Behavioral Feasibility

The proposed system includes the following questions:

- Are there adequate user support?
- Will anyone be harmed by the proposed system?

Because it would accomplish the objectives after being developed and put into action, the project would be advantageous. After carefully assessing all behavioral considerations, it is determined that the project is behaviorally feasible. Users can simply use SoulCure GUI without any training because it is user-friendly.

3.1.4 Questionnaire

1. What are the common therapy techniques you use in your current offline practice?
Answer: I frequently use behavioral therapy and mindfulness techniques in my practice.

2.How do you handle emergency situations or urgent therapy needs in your offline setting?
Answer: In case of emergencies, we have a crisis support helpline and an on-call therapist available

3.Do you primarily work with a specific group of patients in your offline practice?
Answer: Yes, I mainly work with adults aged 25-50, but I also have experience with adolescents.

4.How do you keep track of patients' therapy progress and treatment plans in your current offline practice?
Answer: I maintain patient records and session notes in physical files, which I update regularly.

5.What difficulties do you encounter in keeping patients engaged and motivated during face-to-face therapy sessions?

Answer: Some patients may struggle with consistent attendance, and others may find it challenging to stay motivated to work on their therapy goals outside of sessions.

6.How can an online therapy platform help address the geographical barriers that may limit access to therapy in offline settings?

Answer: An online platform can offer therapy to patients in remote areas or those with limited transportation options.

7.What features or functionalities do you find most valuable in your current offline hospital or clinic setting for providing therapy services?

Answer: As a therapist, I find the personalized and face-to-face interactions with patients to be the most valuable aspect of the current offline hospital setting. This allows for better understanding and empathy during therapy sessions.

8.In what ways do you believe an online platform can enhance your therapeutic services compared to traditional offline settings?

Answer: An online platform can enhance therapeutic services by expanding our reach to a wider audience, offering flexibility in scheduling appointments, and enabling convenient remote therapy options for patients who may have difficulty attending in-person sessions.

9.What additional features or tools would you like to see on the website to support your therapy practice effectively?

Answer: Additional features like integration of therapy progress tracking, session notes, and virtual whiteboards for therapeutic exercises would greatly support my therapy practice.

10. Are there any specific challenges you foresee in transitioning from offline therapy to online therapy via the website?

Answer: One challenge in transitioning to online therapy may be adapting therapeutic approaches to the online medium effectively. Training and support may be required to ensure a smooth transition.

3.1 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor	-	Intel CORE i5
RAM	-	8 GB
Hard Disk	-	512 GB

3.2.2 Software Specification

Front End	-	HTML, CSS, JS, Bootstrap
Back End	-	Django
Database	-	SQLite
Client on PC	-	Windows 7 and above.
Technologies used	-	JS, HTML5, AJAX, J Query, PHP, CSS

3.3 SOFTWARE DESCRIPTION

3.3.1 Django

Django Framework is a popular and robust web framework for Python developers. It is revered for its simplicity, clean code, and rapid development capabilities. Django is built on the Model-Template-Views (MTV) architectural pattern, which shares similarities with the Model-View-Controller (MVC) pattern used in other frameworks. One of its standout features is the Object-Relational Mapping (ORM) system, simplifying database interactions by representing database tables as Python objects. This abstraction eliminates the need for writing raw SQL queries, making database operations more straightforward.

Django also offers a built-in administrative interface, making content management a breeze. Its URL routing system allows developers to define clean and user-friendly URLs for their web applications. Furthermore, Django provides comprehensive support for form handling, data validation, and user authentication, reducing the complexity of common web development tasks. Security is a top priority in Django, with built-in protections against common web vulnerabilities like Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). The framework's modular architecture encourages extensibility, enabling developers to incorporate third-party packages or develop custom components. With a thriving community and an array of reusable packages, Django is a versatile choice for web development projects of varying sizes and complexities.

3.3.2 SQLite

SQLite is a lightweight, self-contained, and serverless relational database management system. Unlike traditional databases, it doesn't require a separate server but is embedded directly within the application. One of its standout features is the self-contained nature of SQLite databases; the entire database is stored in a single file, simplifying management, backups, and transfers. Despite its lightweight design, SQLite is ACID-compliant, meaning it ensures data integrity by supporting transactions, making it suitable for multi-user environments.

SQLite is cross-platform and compatible with various operating systems, making it a versatile choice for applications targeting different platforms. Its small code size and minimal resource usage make it suitable for resource-constrained environments, such as mobile devices. SQLite is known for its speed and efficiency, especially in read-heavy workloads. It is commonly used in mobile applications, desktop applications, and embedded systems, where a full-fledged database server might be excessive, and portability is essential.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

System design is a critical phase in the software development process that serves as the bridge between conceptualizing a software solution and bringing it to life. It plays a pivotal role in translating the requirements gathered during the earlier stages into a well-structured and efficient system. At its core, system design involves making key decisions about the system's architecture, components, modules, interfaces, and data flows, creating a blueprint for the entire system. This blueprint not only addresses the technical aspects but also factors like scalability, security, and user experience. System design is instrumental in ensuring that the resulting software system is reliable, maintainable, and performs optimally. It provides a reference point for the development team, guiding them towards the successful creation of the software system.

In this phase, designers and architects collaborate to create a comprehensive system design that offers a clear roadmap for developers to implement. The design document serves as a guiding light throughout the development process, outlining how different parts of the system will interact and function together to achieve the desired objectives. The resulting system design is a crucial foundation for the development team to build upon, ensuring that the software solution meets the needs of its users and stakeholders while adhering to best practices in software engineering. In summary, system design is a structured and methodical approach to defining how a system will work and how it will meet the requirements of the project, making it an essential step in the journey from concept to a fully functional, real-world solution.

4.2 UML DIAGRAM

Unified Modeling Language (UML) stands as a foundational tool in software engineering, renowned for its role in visually representing complex systems and processes. It provides a standardized set of graphical notations that facilitate the clear depiction of various aspects of a system's structure and behavior. Originating from the collaboration of industry experts, UML has gained widespread acceptance and adoption in both academia and industry. It serves as a powerful communication tool, enabling stakeholders, including developers, designers, and clients, to attain a shared understanding of system architecture, design, and functionality. UML diagrams act as a lingua franca, transcending language barriers and ensuring a consistent means of conveying intricate software concepts, ultimately enhancing the efficiency and effectiveness of the software development process.

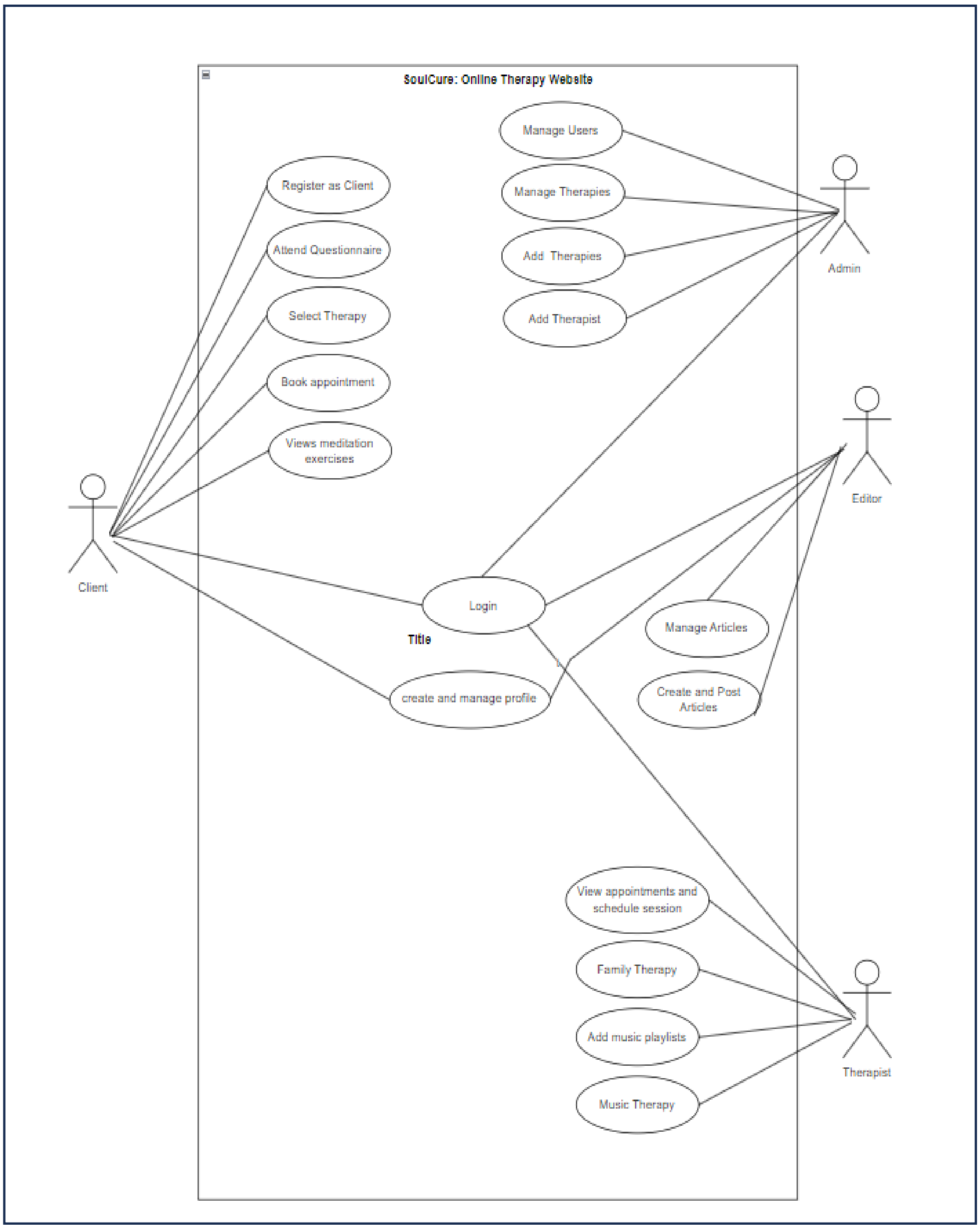
Types of UML diagrams

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

4.2.1 Use Case Diagram

Use Case Diagrams, a cornerstone in software engineering, serve as a visual representation of the interactions between a system and its external entities. At their core, they provide a structured means of identifying and defining the various functionalities a system offers and how these functionalities are accessed by different actors or entities. Actors, representing users, systems, or external entities, are depicted along with the specific use cases they engage with. Associations between actors and use cases elucidate the nature of these interactions, clarifying the roles and responsibilities of each entity within the system. This detailed visual representation not only enhances communication among stakeholders but also provides a clear blueprint for system functionality, laying the foundation for the subsequent stages of the software development process. Overall, Use Case Diagrams play a pivotal role in aligning development efforts with user expectations, ensuring that the resulting software system fulfills its intended purpose effectively and efficiently.

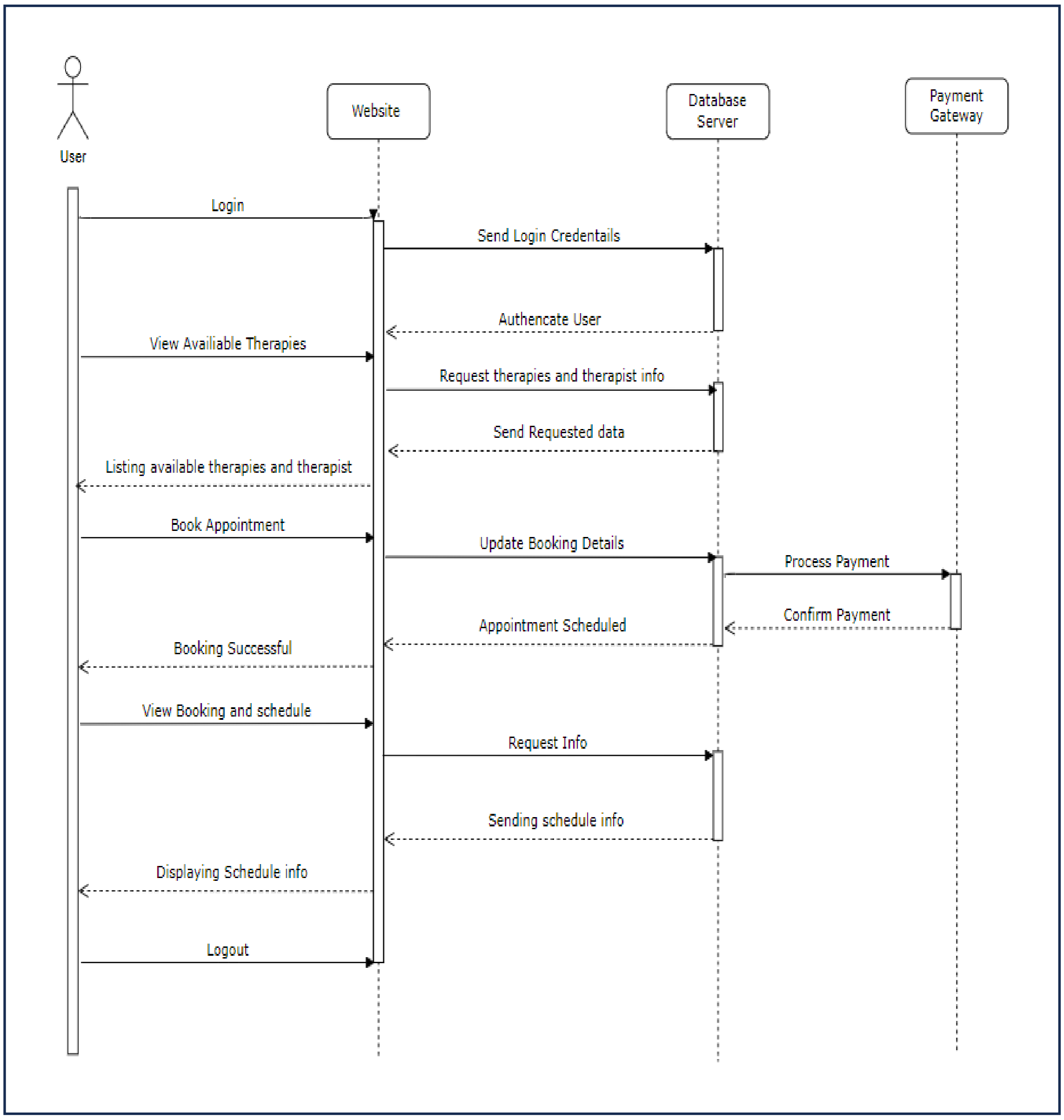
- **Actor Definition:** Clearly define and label all actors involved in the system. Actors represent external entities interacting with the system.
- **Use Case Naming:** Use descriptive names for use cases to accurately convey the functionality they represent.
- **Association Lines:** Use solid lines to represent associations between actors and use cases. This signifies the interaction between entities.
- **System Boundary:** Draw a box around the system to indicate its scope and boundaries. This defines what is inside the system and what is outside.
- **Include and Extend Relationships:** Use "include" relationships to represent common functionalities shared among multiple use cases. Use "extend" relationships to show optional or extended functionalities.


SoulCure: Online Therapy Website
Manage Users
Register as Client
Manage Therapies
Attend Questionnaire
Add Therapies
Select Therapy
Add Therapist
Admin
Book appointment
Views meditation exercises
Editor
Client
Login
Title
Manage Articles
create and manage profile
Create and Post Articles
View appointments and schedule session
Family Therapy
Add music playlists
Therapist
Music Therapy

4.2.2 Sequence Diagram

Sequence Diagrams stand as dynamic models in software engineering, portraying the chronological flow of interactions between various objects or components within a system. They spotlight the order in which messages are exchanged, revealing the behavior of the system over time. Actors and objects are represented along a vertical axis, with arrows indicating the sequence of messages and their direction. Lifelines, extending vertically from actors or objects, illustrate their existence over the duration of the interaction. These diagrams serve as a vital tool for visualizing system behavior and understanding the temporal aspects of a software process. Through Sequence Diagrams, stakeholders gain valuable insights into how different elements collaborate to achieve specific functionalities, facilitating more effective communication among development teams and stakeholders alike. This detailed representation not only aids in detecting potential bottlenecks or inefficiencies but also provides a foundation for refining system performance in the later stages of software development.

- **Vertical Ordering:** Represent actors and objects along a vertical axis, indicating the order of interactions from top to bottom.
- **Lifelines:** Extend vertical lines from actors or objects to denote their existence and participation in the interaction.
- **Activation Bars:** Use horizontal bars along lifelines to show the period during which an object is active and processing a message.
- **Messages and Arrows:** Use arrows to indicate the flow of messages between objects, specifying the direction of communication.
- **Self-Invocation:** Use a looped arrow to represent self-invocation, when an object sends a message to itself.
- **Return Messages:** Indicate return messages with a dashed arrow, showing the response from the recipient.
- **Focus on Interaction:** Sequence Diagrams focus on the chronological order of interactions, avoiding implementation details.
- **Concise Notation:** Use clear and concise notation to represent messages and interactions, avoiding unnecessary complexity.
- **Consider System Boundaries:** Clearly define the boundaries of the system to indicate what is included in the interaction.
- **Feedback and Validation**: Seek feedback from stakeholders and team members to ensure the diagram accurately represents the system behaviour.

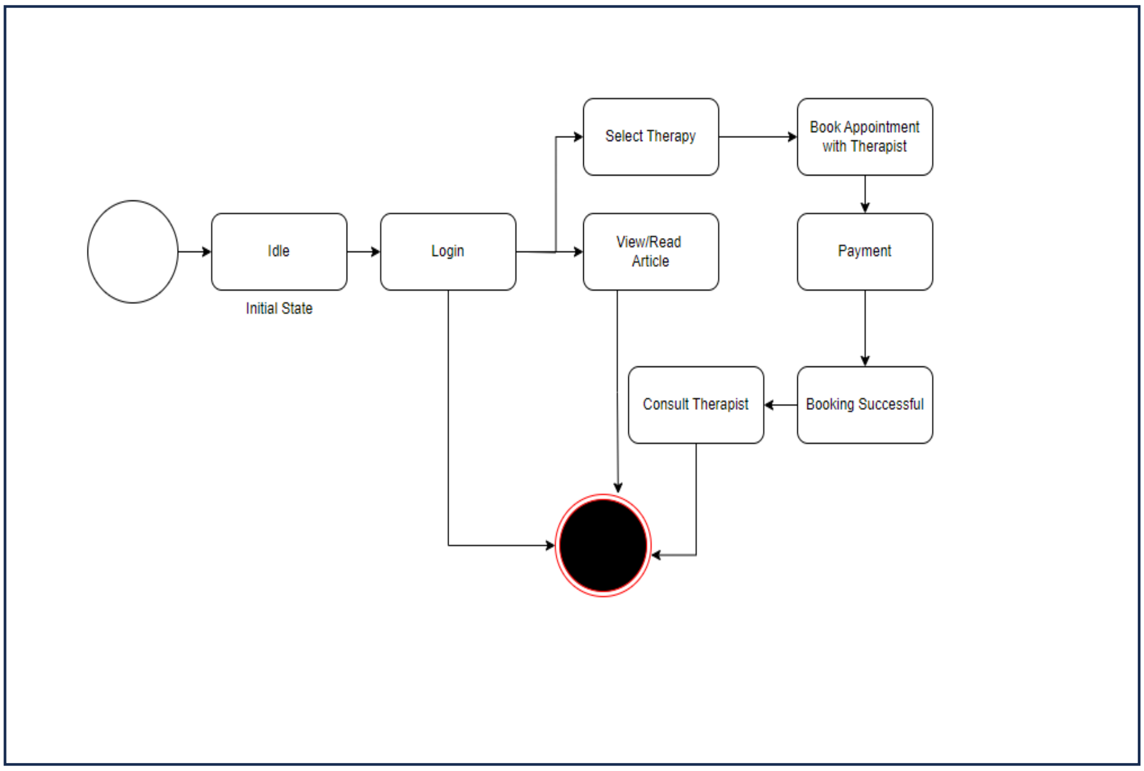

User
Website
Database Server
Payment Gateway
Login
Send Login Credentails
Authencate User
View Availiable Therapies
Request therapies and therapist info
Send Requested data
Listing available therapies and therapist
Book Appointment
Update Booking Details
Process Payment
Confirm Payment
Appointment Scheduled
Booking Successful
View Booking and schedule
Request Info
Sending schedule info
Displaying Schedule info
Logout

4.2.3 State Chart Diagram

A State Chart Diagram, a fundamental component of UML, provides a visual representation of an object's lifecycle states and the transitions between them. It depicts the dynamic behavior of an entity in response to events, showcasing how it transitions from one state to another. Each state represents a distinct phase in the object's existence, while transitions illustrate the conditions triggering state changes. Initial and final states mark the commencement and termination of the object's lifecycle. Orthogonal regions allow for concurrent states, capturing multiple aspects of the object's behavior simultaneously. Hierarchical states enable the representation of complex behaviors in a structured manner. Entry and exit actions depict activities occurring upon entering or leaving a state. Moreover, guard conditions ensure that transitions occur only under specified circumstances. State Chart Diagrams play a crucial role in understanding and designing the dynamic behavior of systems, aiding in the development of robust and responsive software applications.

Key notations for State Chart Diagrams:

- **Initial State:** Represented by a filled circle, it signifies the starting point of the object's lifecycle.
- **State:** Depicted by rounded rectangles, states represent distinct phases in an object's existence.
- **Transition Arrow:** Arrows denote transitions between states, indicating the conditions triggering a change.
- **Event:** Events, triggers for state changes, are labelled on transition arrows.
- **Guard Condition:** Shown in square brackets, guard conditions specify criteria for a transition to occur.
- **Final State:** Represented by a circle within a larger circle, it indicates the end of the object's lifecycle.
- **Concurrent State:** Represented by parallel lines within a state, it signifies concurrent behaviours.
- **Hierarchy:** States can be nested within other states to represent complex behavior.
- **Entry and Exit Actions**: Actions occurring upon entering or leaving a state are labeled within the state.
- **Transition Labels:** Labels on transition arrows may indicate actions or operations that accompany the transition.

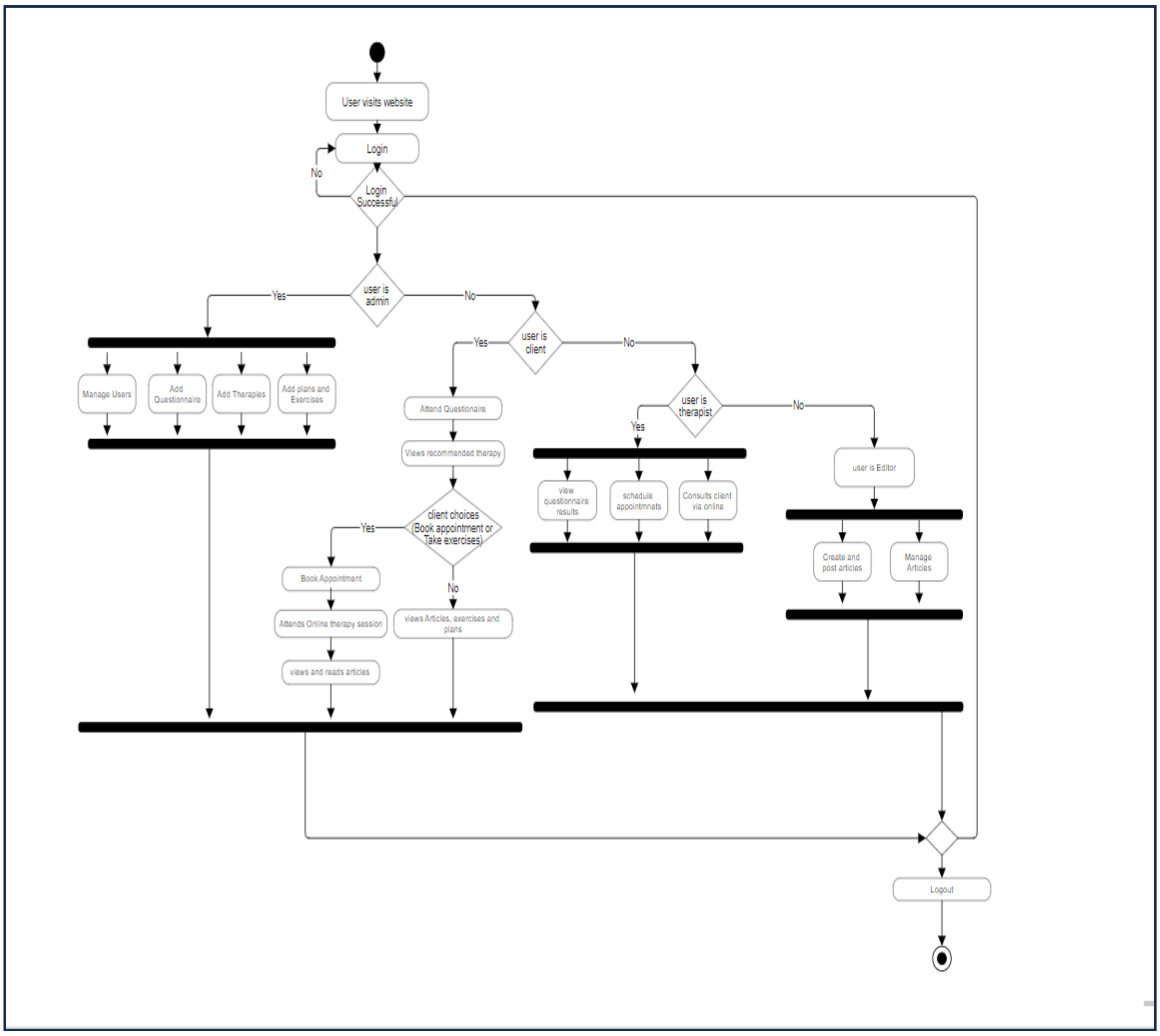

Select Therapy
Book Appointment with Therapist
Idle
Initial State
Login
View/Read Article
Payment
Consult Therapist
Booking Successful

4.2.4 Activity Diagram

An Activity Diagram is a visual representation within UML that illustrates the flow of activities and actions in a system or process. It employs various symbols to depict tasks, decision points, concurrency, and control flows. Rectangles signify activities or tasks, while diamonds represent decision points, allowing for conditional branching. Arrows indicate the flow of control from one activity to another. Forks and joins denote concurrency, where multiple activities can occur simultaneously or in parallel. Swimlane segregate activities based on the responsible entity, facilitating clarity in complex processes. Initial and final nodes mark the commencement and completion points of the activity. Decision nodes use guards to determine the path taken based on conditions. Synchronization bars enable the coordination of parallel activities. Control flows direct the sequence of actions, while object flows depict the flow of objects between activities. Activity Diagrams serve as invaluable tools for understanding, modeling, and analyzing complex workflows in systems and processes. They offer a structured visual representation that aids in effective communication and system development.

Key notations for Activity Diagrams:

- **Initial Node:** Represented by a solid circle, it signifies the starting point of the activity.
- **Activity:** Shown as a rounded rectangle, it represents a task or action within the process.
- **Decision Node:** Depicted as a diamond shape, it indicates a point where the process flow can diverge based on a condition.
- **Merge Node:** Represented by a hollow diamond, it signifies a point where multiple flows converge.
- **Fork Node:** Shown as a horizontal bar, it denotes the start of concurrent activities.
- **Join Node:** Depicted as a vertical bar, it marks the point where parallel flows rejoin.
- **Final Node**: Represented by a solid circle with a border, it indicates the end of the activity.
- **Control Flow:** Arrows connecting activities, showing the sequence of actions.
- **Object Flow:** Lines with arrows representing the flow of objects between activities.
- **Swimlane:** A visual container that groups activities based on the responsible entity or system component.

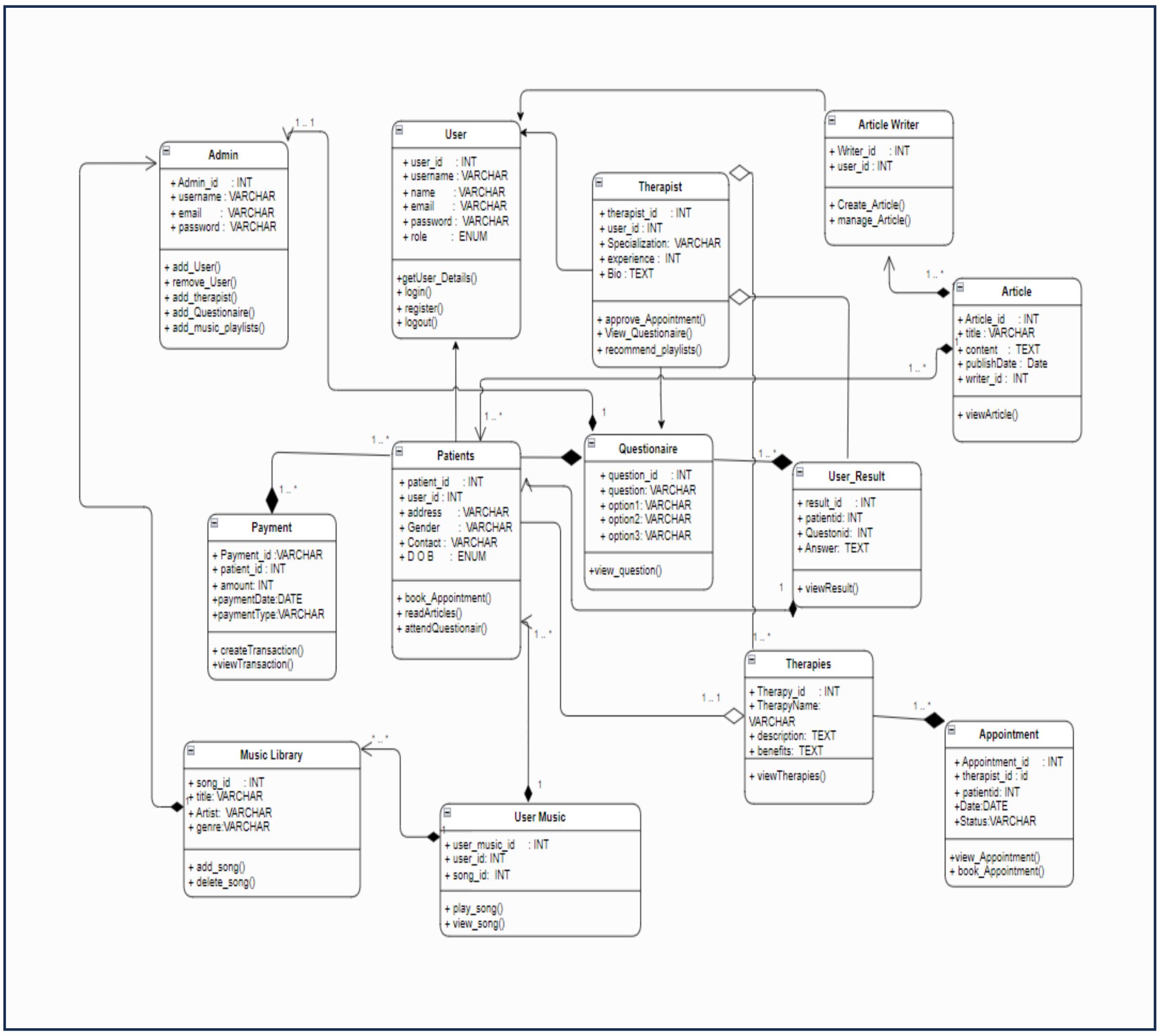

User visits website
Login
No
Login Successful
user is admin
Yes
No
Manage Users
Add Questionnaire
Add Therapies
Add plans and Exercises
user is client
Yes
No
Attend Questionaire
Views recommended therapy
client choices (Book appointment or Take exercises)
Yes
No
Book Appointment
Attends Online therapy session
views and reads articles
views Articles, exercises and plans
user is therapist
Yes
No
view questionnaire results
schedule appointmnets
Consults client via online
user is Editor
Create and post articles
Manage Articles
Logout

4.2.5 Class Diagram

A Class Diagram, a fundamental tool in UML, visually represents the structure of a system by illustrating classes, their attributes, methods, and relationships. Classes, depicted as rectangles, encapsulate data and behavior within a system. Associations between classes indicate relationships, showcasing how they interact. Multiplicity notations specify the cardinality of associations. Inheritance is denoted by an arrow indicating the subclass inheriting from a super-class. Aggregation and composition illustrate whole-part relationships between classes. Interfaces, depicted as a circle, outline the contract of behavior a class must implement. Stereotypes provide additional information about a class's role or purpose. Dependencies highlight the reliance of one class on another. Association classes facilitate additional information about associations. Packages group related classes together, aiding in system organization. Class Diagrams play a pivotal role in system design, aiding in conceptualizing and planning software architectures. They serve as a blueprint for the development process, ensuring a clear and structured approach to building robust software systems.

Key notations for Class Diagrams:

- **Class:** Represented as a rectangle, it contains the class name, attributes, and methods.
- **Attributes:** Displayed as a list within the class, they describe the properties or characteristics of the class.
- **Methods:** Also listed within the class, they define the behaviors or operations of the class
- **Associations:** Lines connecting classes, indicating relationships and connections between them.
- **Multiplicity Notation:** Indicates the number of instances one class relates to another.
- **Inheritance:** Shown as an arrow, it signifies that one class inherits properties and behaviours from another.
- **Interfaces:** Represented by a dashed circle, they define a contract of behaviour that implementing classes must follow.
- **Stereotypes:** Additional labels or annotations applied to classes to provide more information about their role or purpose.
- **Dependencies:** Shown as a dashed line with an arrow, they indicate that one class relies on another in some way.
- **Association Classes:** Represented as a class connected to an association, they provide additional information about the relationship.

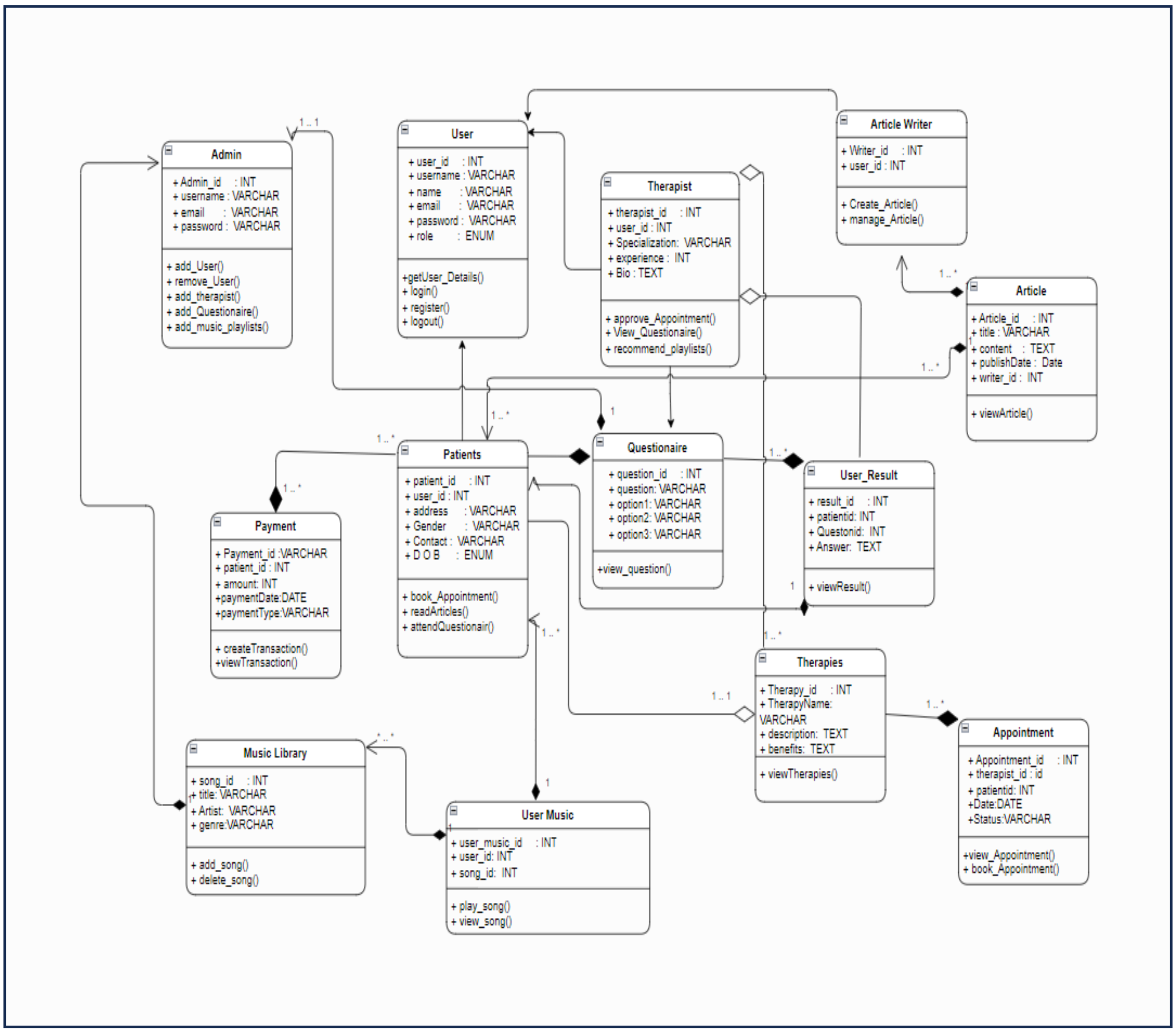

Admin
+ Admin_id : INT
+ username : VARCHAR
+ email : VARCHAR
+ password : VARCHAR
+ add_User()
+ remove_User()
+ add_therapist()
+ add_Questionaire()
+ add_music_playlists()
User
+ user_id : INT
+ username : VARCHAR
+ name : VARCHAR
+ email : VARCHAR
+ password : VARCHAR
+ role : ENUM
+getUser_Details()
+ login()
+ register()
+ logout()
Therapist
+ therapist_id : INT
+ user_id : INT
+ Specialization: VARCHAR
+ experience : INT
+ Bio : TEXT
+ approve_Appointment()
+ View_Questionaire()
+ recommend_playlists()
Article Writer
+ Writer_id : INT
+ user_id : INT
+ Create_Article()
+ manage_Article()
Article
+ Article_id : INT
+ title : VARCHAR
+ content : TEXT
+ publishDate : Date
+ writer_id : INT
+ viewArticle()
Patients
+ patient_id : INT
+ user_id : INT
+ address : VARCHAR
+ Gender : VARCHAR
+ Contact : VARCHAR
+ D O B : ENUM
+ book_Appointment()
+ readArticles()
+ attendQuestionair()
Questionaire
+ question_id : INT
+ question: VARCHAR
+ option1: VARCHAR
+ option2: VARCHAR
+ option3: VARCHAR
+view_question()
User_Result
+ result_id : INT
+ patientid: INT
+ Questonid: INT
+ Answer: TEXT
+ viewResult()
Payment
+ Payment_id :VARCHAR
+ patient_id : INT
+ amount: INT
+paymentDate:DATE
+paymentType:VARCHAR
+ createTransaction()
+viewTransaction()
Therapies
+ Therapy_id : INT
+ TherapyName: VARCHAR
+ description: TEXT
+ benefits: TEXT
+ viewTherapies()
Appointment
+ Appointment_id : INT
+ therapist_id : id
+ patientid: INT
+Date:DATE
+Status:VARCHAR
+view_Appointment()
+ book_Appointment()
Music Library
+ song_id : INT
+ title: VARCHAR
+ Artist: VARCHAR
+ genre:VARCHAR
+ add_song()
+ delete_song()
User Music
+ user_music_id : INT
+ user_id: INT
+ song_id: INT
+ play_song()
+ view_song()
1 .. 1
1 .. *
1
1 .. *
1 .. *
1 .. *
1 .. *
1
1 .. *
1 .. 1
1 .. *
* .. *
1

4.2.6 Object Diagram

An Object Diagram in UML provides a snapshot of a system at a specific point in time, displaying the instances of classes and their relationships. Objects, represented as rectangles, showcase the state and behavior of specific instances. Links between objects depict associations, highlighting how they interact. Multiplicity notations indicate the number of instances involved in associations. The object's state is displayed through attributes and their corresponding values. Object Diagrams offer a detailed view of runtime interactions, aiding in system understanding and testing. They focus on real-world instances, providing a tangible representation of class relationships. While similar to Class Diagrams, Object Diagrams emphasize concrete instances rather than class definitions. They serve as valuable tools for validating system design and verifying that classes and associations work as intended in practice. Object Diagrams play a crucial role in system validation, ensuring that the system's components and their interactions align with the intended design and requirements.

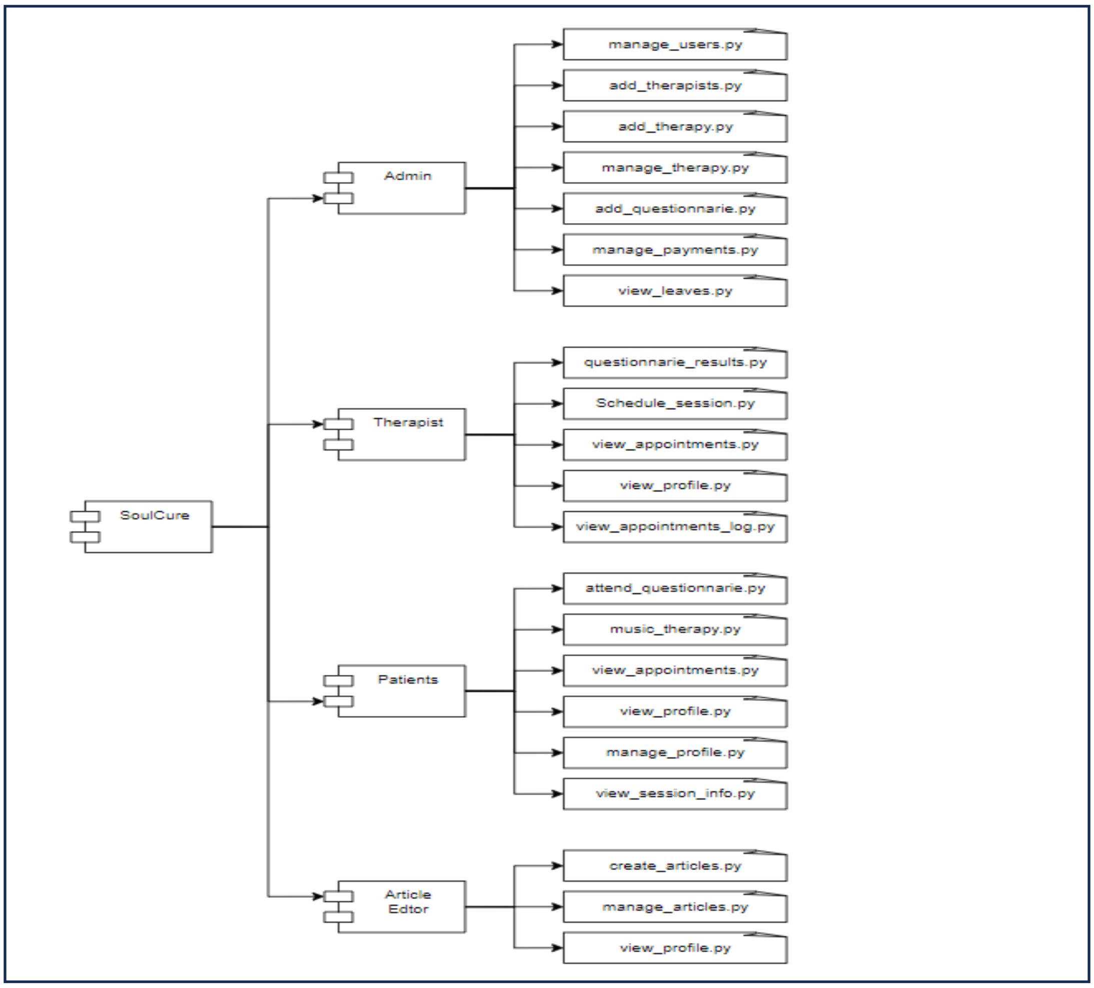
Key notations for Object Diagrams:

- **Object**: Represented as a rectangle, it contains the object's name and attributes with their values.
- **Links:** Lines connecting objects, indicating associations or relationships between them.
- **Multiplicity Notation:** Indicates the number of instances involved in associations.
- **Attributes with Values:** Displayed within the object, they represent the state of the object at a specific point in time.
- **Role Names:** Labels applied to associations, providing additional information about the nature of the relationship.
- **Object Name:** Represents the name of the specific instance.
- **Association End:** Indicates the end of an association, often with a role name and multiplicity.
- **Dependency Arrow:** Indicates a dependency relationship, where one object relies on another.
- **Composition Diamond:** Represents a stronger form of ownership, where one object encapsulates another.
- **Aggregation Diamond:** Signifies a whole-part relationship between objects.


Admin
+ Admin_id : INT
+ username : VARCHAR
+ email : VARCHAR
+ password : VARCHAR
+ add_User()
+ remove_User()
+ add_therapist()
+ add_Questionaire()
+ add_music_playlists()
User
+ user_id : INT
+ username : VARCHAR
+ name : VARCHAR
+ email : VARCHAR
+ password : VARCHAR
+ role : ENUM
+getUser_Details()
+ login()
+ register()
+ logout()
Therapist
+ therapist_id : INT
+ user_id : INT
+ Specialization: VARCHAR
+ experience : INT
+ Bio : TEXT
+ approve_Appointment()
+ View_Questionaire()
+ recommend_playlists()
Article Writer
+ Writer_id : INT
+ user_id : INT
+ Create_Article()
+ manage_Article()
Article
+ Article_id : INT
+ title : VARCHAR
+ content : TEXT
+ publishDate : Date
+ writer_id : INT
+ viewArticle()
Patients
+ patient_id : INT
+ user_id : INT
+ address : VARCHAR
+ Gender : VARCHAR
+ Contact : VARCHAR
+ D O B : ENUM
+ book_Appointment()
+ readArticles()
+ attendQuestionair()
Questionaire
+ question_id : INT
+ question: VARCHAR
+ option1: VARCHAR
+ option2: VARCHAR
+ option3: VARCHAR
+view_question()
User_Result
+ result_id : INT
+ patientid: INT
+ Questonid: INT
+ Answer: TEXT
+ viewResult()
Payment
+ Payment_id :VARCHAR
+ patient_id : INT
+ amount: INT
+paymentDate:DATE
+paymentType:VARCHAR
+ createTransaction()
+viewTransaction()
Therapies
+ Therapy_id : INT
+ TherapyName: VARCHAR
+ description: TEXT
+ benefits: TEXT
+ viewTherapies()
Appointment
+ Appointment_id : INT
+ therapist_id : id
+ patientid: INT
+Date:DATE
+Status:VARCHAR
+view_Appointment()
+ book_Appointment()
Music Library
+ song_id : INT
+ title: VARCHAR
+ Artist: VARCHAR
+ genre:VARCHAR
+ add_song()
+ delete_song()
User Music
+ user_music_id : INT
+ user_id: INT
+ song_id: INT
+ play_song()
+ view_song()
1 .. 1
1 .. *
1
* .. *

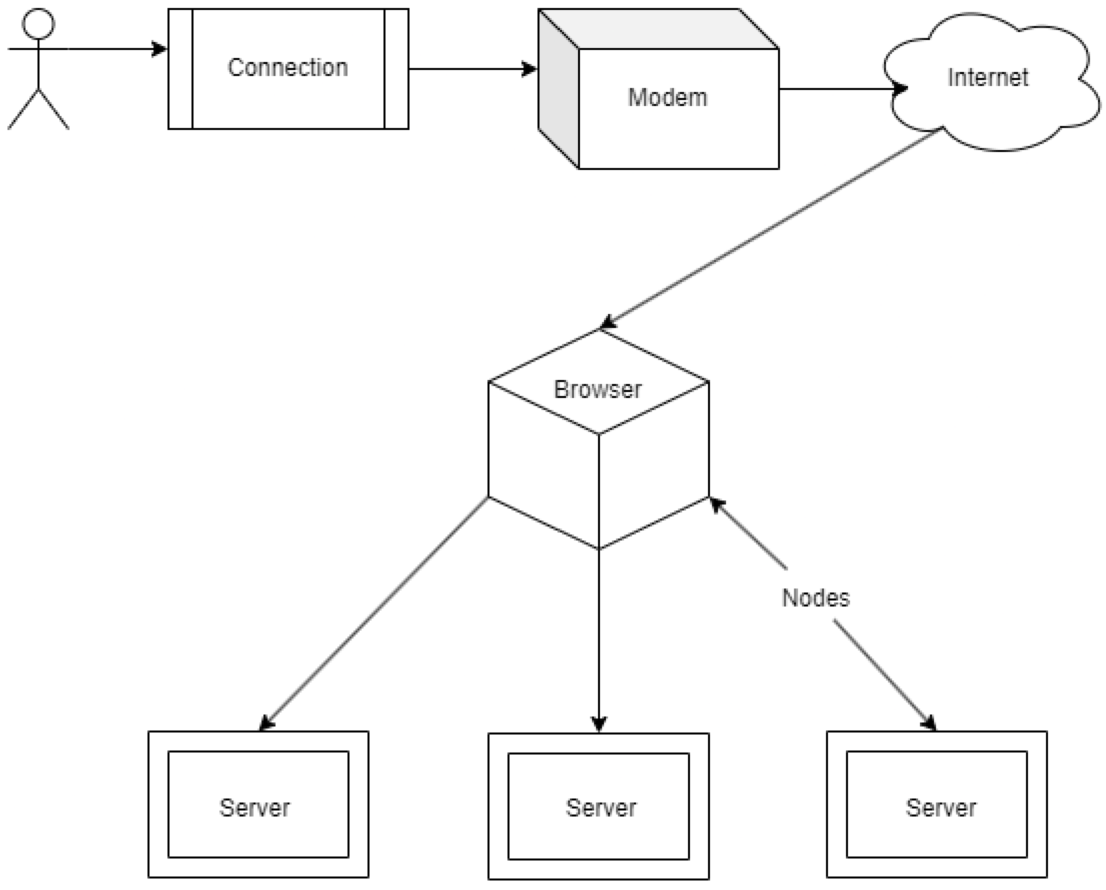
4.2.7 Component Diagram

A Component Diagram, a vital aspect of UML, offers a visual representation of a system's architecture by showcasing the high-level components and their connections. Components, depicted as rectangles, encapsulate modules, classes, or even entire systems. Dependencies between components are displayed through arrows, signifying the reliance of one component on another. Interfaces, represented by a small circle, outline the services a component offers or requires. Connectors link interfaces to denote the required or provided services. Ports, depicted as small squares, serve as connection points between a component and its interfaces. Stereotypes provide additional information about the role or purpose of a component. Deployment nodes indicate the physical location or environment in which components are deployed. Component Diagrams are instrumental in system design, aiding in the organization and visualization of system architecture. They emphasize the modular structure, facilitating ease of development, maintenance, and scalability of complex software systems. Overall, Component Diagrams play a pivotal role in planning and orchestrating the architecture of sophisticated software applications.



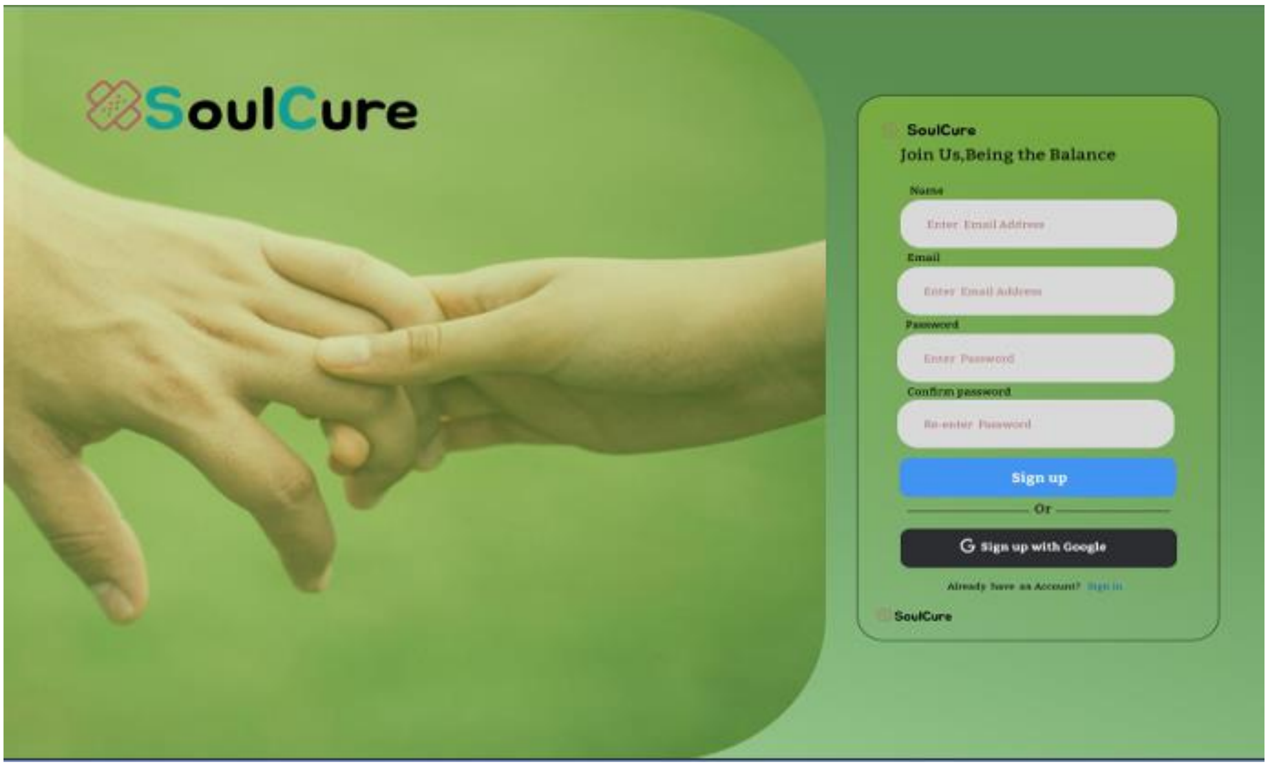
4.2.8 Deployment Diagram

A Deployment Diagram, a crucial facet of UML, provides a visual representation of the physical architecture of a system, showcasing the hardware nodes and software components. Nodes, representing hardware entities like servers or devices, are depicted as rectangles. Artifacts, denoted by rectangles with a folded corner, represent software components or files deployed on nodes. Associations between nodes and artifacts indicate the deployment of software on specific hardware. Dependencies illustrate the reliance of one node on another. Communication paths, shown as dashed lines, represent network connections between nodes. Stereotypes provide additional information about the role or purpose of nodes and artifacts. Deployment Diagrams are instrumental in system planning, aiding in the visualization and organization of hardware and software components. They emphasize the allocation of software modules to specific hardware nodes, ensuring efficient utilization of resources. Overall, Deployment Diagrams play a pivotal role in orchestrating the physical infrastructure of complex software applications.

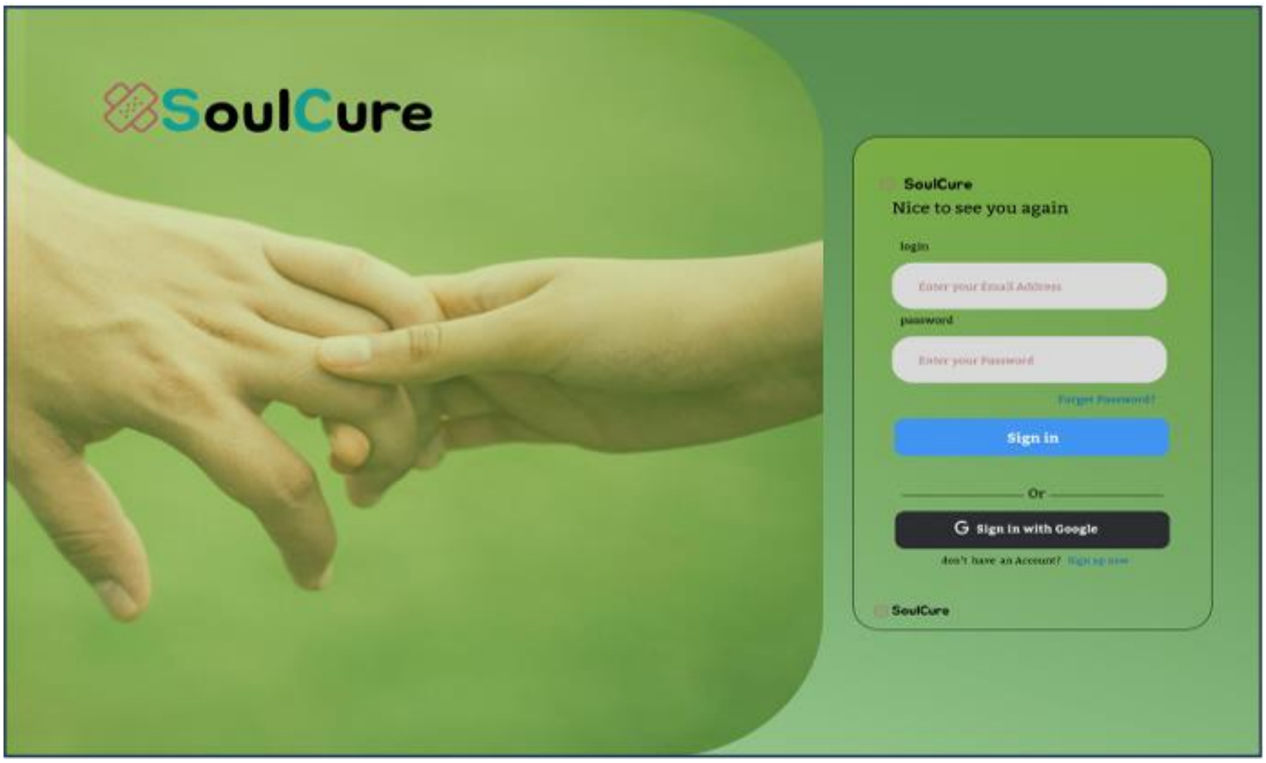


4.3 USER INTERFACE DESIGN USING FIGMA

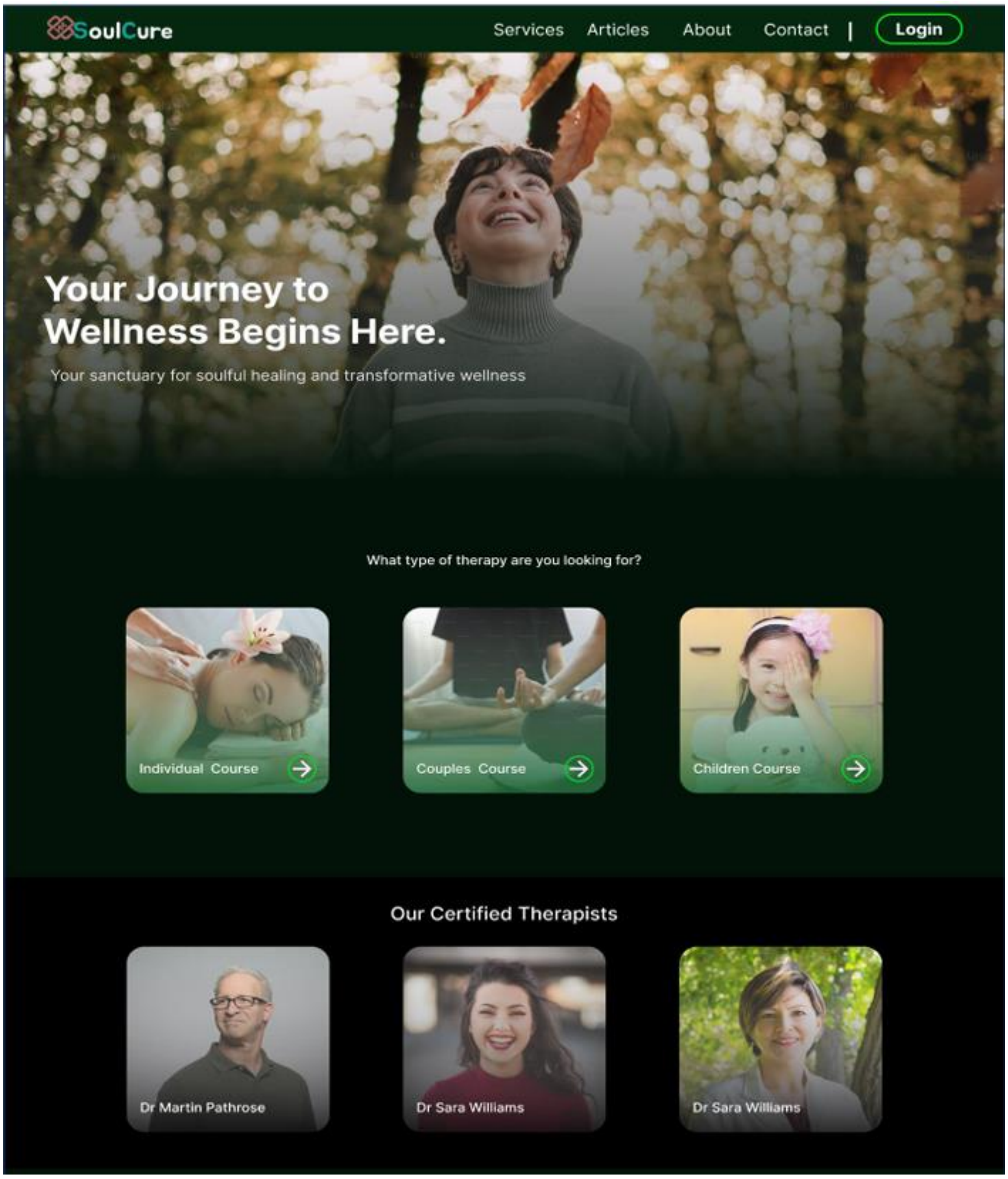
Form Name: Registration



Form Name: Login



Form Name: Home page



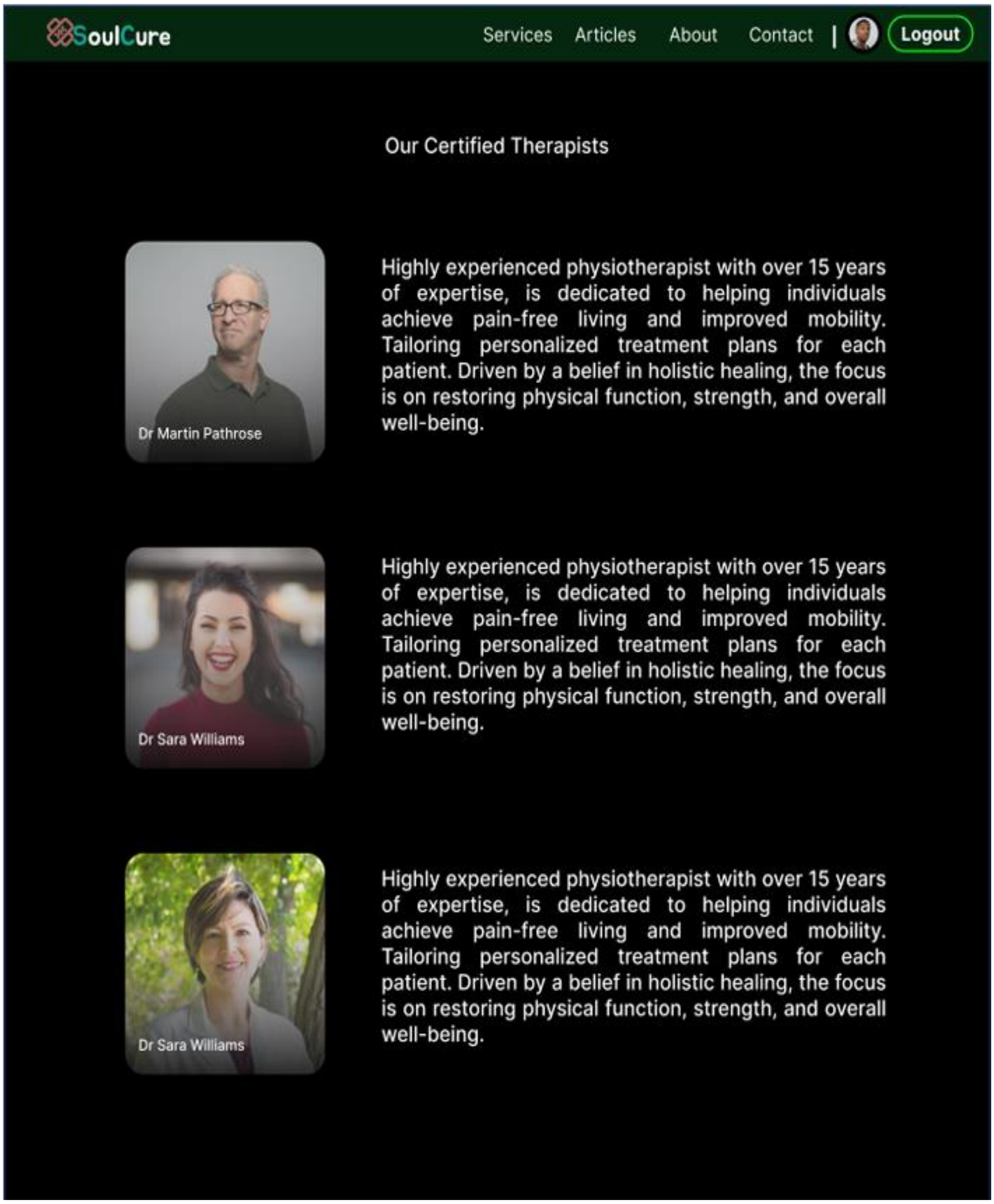
Your Journey to Wellness Begins Here.

Your sanctuary for soulful healing and transformative wellness

What type of therapy are you looking for?

Our Certified Therapists

Form Name: View Therapist page



Our Certified Therapists

Dr Martin Pathrose

Highly experienced physiotherapist with over 15 years of expertise, is dedicated to helping individuals achieve pain-free living and improved mobility. Tailoring personalized treatment plans for each patient. Driven by a belief in holistic healing, the focus is on restoring physical function, strength, and overall well-being.

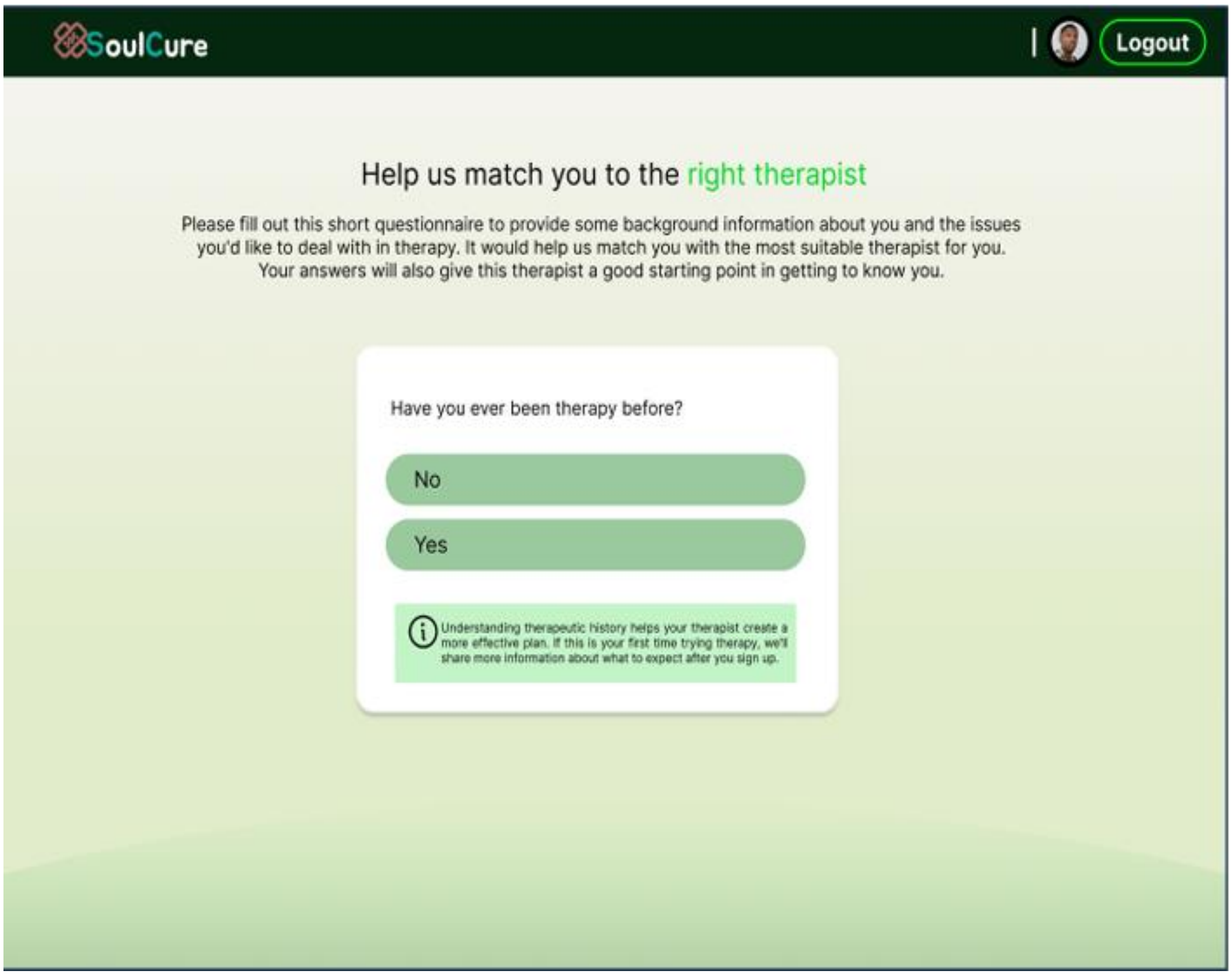
Dr Sara Williams

Highly experienced physiotherapist with over 15 years of expertise, is dedicated to helping individuals achieve pain-free living and improved mobility. Tailoring personalized treatment plans for each patient. Driven by a belief in holistic healing, the focus is on restoring physical function, strength, and overall well-being.

Dr Sara Williams

Highly experienced physiotherapist with over 15 years of expertise, is dedicated to helping individuals achieve pain-free living and improved mobility. Tailoring personalized treatment plans for each patient. Driven by a belief in holistic healing, the focus is on restoring physical function, strength, and overall well-being.

Form Name: Questionnaire page



Help us match you to the right therapist

Please fill out this short questionnaire to provide some background information about you and the issues you'd like to deal with in therapy. It would help us match you with the most suitable therapist for you. Your answers will also give this therapist a good starting point in getting to know you.

Have you ever been therapy before?

No

Yes

Understanding therapeutic history helps your therapist create a more effective plan. If this is your first time trying therapy, we'll share more information about what to expect after you sign up.

4.4 DATABASE DESIGN

A database is an organized collection of information that's organized to enable easy accessibility, administration, and overhauls. The security of information could be a essential objective of any database. The database design process comprises of two stages. In the first stage, user requirements are gathered to create a database that meets those requirements as clearly as possible. This is known as information-level design and is carried out independently of any DBMS. In the second stage, the design is converted from an information-level design to a specific DBMS design that will be used to construct the system. This stage is known as physical-level design, where the characteristics of the specific DBMS are considered. Alongside system design, there is also database design, which aims to achieve two main goals: data integrity and data independence.

4.4.1 Relational Database Management System (RDBMS)

A relational database management system (RDBMS) is a popular type of database that organizes data into tables to facilitate relationships with other stored data sets. Tables can contain vast amounts of data, ranging from hundreds to millions of rows, each of which are referred to as records. In formal relational model language, a row is called a tuple, a column heading is an attribute, and the table is a relation. A relational database consists of multiple tables, each with its own name. Each row in a table represents a set of related values.

In a relational database, relationships are already established between tables to ensure the integrity of both referential and entity relationships. A domain D is a group of atomic values, and a common way to define a domain is by choosing a data type from which the domain's data values are derived. It is helpful to give the domain a name to make it easier to understand the values it contains. Each value in a relation is atomic and cannot be further divided.

In a relational database, table relationships are established using keys, with primary key and foreign key being the two most important ones. Entity integrity and referential integrity relationships can be established with these keys. Entity integrity ensures that no primary key can have null values, while referential integrity ensures that each distinct foreign key value must have a matching primary key value in the same domain. Additionally, there are other types of keys such as super keys and candidate keys.

4.4.2 Normalization

Normalization is a database design procedure that diminishes information overt repetitiveness and takes out bothersome attributes like Insertion, Update and Deletion Anomalies. Normalization rules isolates bigger tables into more modest tables and connections them utilizing connections. The reason for Standardization in SQL is to dispense with excess (redundant) information and guarantee information is put away legitimately. The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

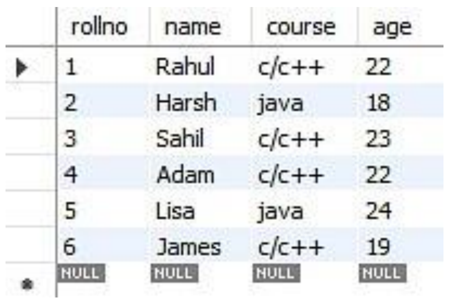
First Normal Form(1NF):

A table is considered to be in the First Normal Form if it satisfies the requirement of atomicity, which means that the table's values cannot be divided into smaller, more granular parts, and each attribute or column contains only a single value. In other words, each column should hold only one piece of information, and there should be no repeating groups or arrays of values within a single row.

The concept of atomicity in this context denotes that a cell within a database cannot contain more than one value, and must exclusively store a single-valued attribute.

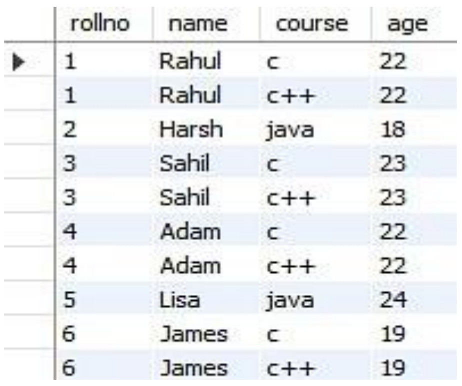
The first normal form in database normalization places limitations on attributes that have multiple values, are composed of multiple sub-attributes, or contain a combination of both. Therefore, in order to satisfy the conditions of the first normal form., these attributes need to be modified or removed, a relation must not have multi-valued or composite attributes, and any such attributes must be split into individual attributes to form atomic values.

E.g. The table contains information pertaining to students, including their roll number, name, course of study, and age.



rollno	name	course	age
1	Rahul	c/c++	22
2	Harsh	java	18
3	Sahil	c/c++	23
4	Adam	c/c++	22
5	Lisa	java	24
6	James	c/c++	19
NULL	NULL	NULL	NULL

The table containing the records of students displays a violation of the First Normal Form due to the presence of two distinct values in the course column. To ensure compliance with the First Normal Form, the table has been modified resulting in the formation of a new table.



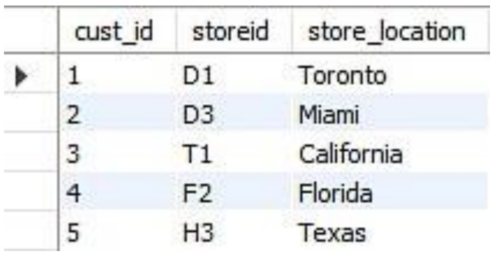
rollno	name	course	age
1	Rahul	c	22
1	Rahul	c++	22
2	Harsh	java	18
3	Sahil	c	23
3	Sahil	c++	23
4	Adam	c	22
4	Adam	c++	22
5	Lisa	java	24
6	James	c	19
6	James	c++	19

The First Normal Form is applied to achieve atomicity in the system, which ensures that each column has unique values. By following this normalization process, the data is organized into individual atomic values, eliminating any redundancies or repeating groups. As a result, data integrity is maintained, and the system can efficiently handle complex data manipulations and updates.

Second Normal Form(2NF):

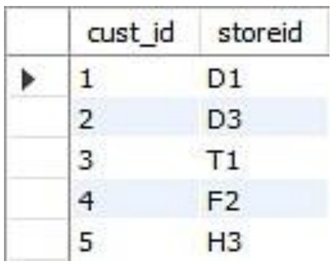
To meet requirements of Second Normal Form, a table must satisfy the criteria of First Normal Form as a prerequisite. Furthermore, the table must not exhibit partial dependency, which refers to a scenario where a non-prime attribute is dependent on a proper subset of the candidate key. In other words, the table should have no attributes that are determined by only a portion of the primary key. Now understand the Second Normal Form with the help of an example.

Consider the table Location:

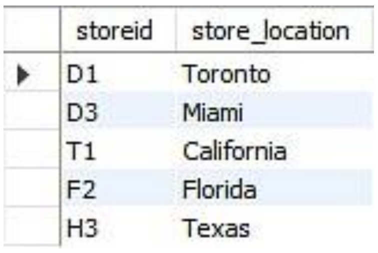


cust_id	storeid	store_location
1	D1	Toronto
2	D3	Miami
3	T1	California
4	F2	Florida
5	H3	Texas

The Location table currently has a composite primary key consisting of cust id and storied, and its non-key attribute is store location. However, the store location attribute is directly determined by the storied attribute, which is part of the primary key. As a result, this table does not meet the requirements of second normal form. To address this issue and ensure second normal form is met, it is necessary to separate the Location table into two separate tables. This will result in the creation of two distinct tables that accurately represent the relevant data and relationships: one for customer IDs and store IDs, and another for store IDs and their respective locations.:



cust_id	storeid
1	D1
2	D3
3	T1
4	F2
5	H3

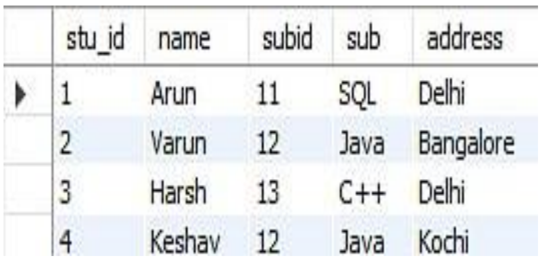


storeid	store_location
D1	Toronto
D3	Miami
T1	California
F2	Florida
H3	Texas

Third Normal Form(3NF):

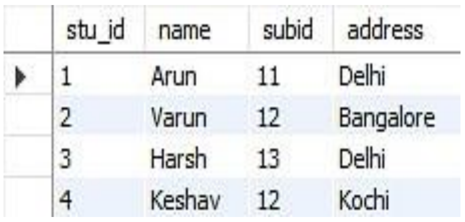
A table must meet the requirements of Second Normal Form in order to be considered in Third Normal Form, and also fulfill two additional conditions. The second condition states that non- prime attributes should not rely on non-prime characteristics that are not a part of the candidate key within the same table, thus avoiding transitive dependencies. A transitive dependency arises when A → C indirectly, due to A → B and B → C, where B is not functionally dependent on A. The main objective of achieving Third Normal Form is to reduce data redundancy and guarantee data integrity.

For instance, let's consider a student table that includes columns like student ID, student name, subject ID, subject name, and address of the student. To comply with the requirements for Third Normal Form, this table must first meet the standards of Second Normal Form and then ensure that there are no transitive dependencies between non-prime attributes.

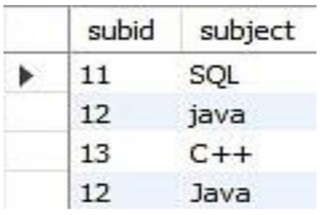


stu_id	name	subid	sub	address
1	Arun	11	SQL	Delhi
2	Varun	12	Java	Bangalore
3	Harsh	13	C++	Delhi
4	Keshav	12	Java	Kochi

Now to change the table to the third normal form, you need to divide the table as shown below: Based on the given student table, it can be observed that the stu_id attribute determines the sub_id attribute, and the sub_id attribute determines the subject (sub). This implies that there is a transitive functional dependency between stu_id and sub. As a result, the table does not satisfy the criteria for the third normal form. To adhere to the third normal form, the table must be divided into separate tables where each table represents a unique entity or relationship. In this case, the table can be divided into two tables: one for the student-subject relationship (stu_id and sub_id), and another for the subject information (sub_id and sub):



	stu_id	name	subid	address
▶	1	Arun	11	Delhi
	2	Varun	12	Bangalore
	3	Harsh	13	Delhi
	4	Keshav	12	Kochi



	subid	subject
▶	11	SQL
	12	java
	13	C++
	12	Java

The two tables illustrate that the non-key attributes are completely determined by and reliant on the primary key. In the first table, the columns for name, sub_id, and addresses are all exclusively linked to the stu_id. Likewise, the second table demonstrates that the sub column is entirely dependent on the sub_id.

4.4.3 Sanitization

Data Sanitization involves the secure and permanent erasure of sensitive data from datasets and media to guarantee that no residual data can be recovered even through extensive forensic analysis. Data sanitization has a wide range of applications but is mainly used for clearing out end-of-life electronic devices or for the sharing and use of large datasets that contain sensitive information. The main strategies for erasing personal data from devices are physical destruction, cryptographic erasure, and data erasure. While the term data sanitization may lead some to believe that it only includes data on electronic media, the term also broadly covers physical media, such as paper copies. These data types are termed soft for electronic files and hard for physical media paper copies. Data sanitization methods are also applied for the cleaning of sensitive data, such as through heuristic-based methods, machine-learning based methods, and k-source anonymity.

4.4.4 Indexing

Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. The index is a type of data structure. It is used to locate and access the data in a database table quickly.

- Primary Index − Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.
- Secondary Index − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

- Clustering Index − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

4.5 TABLE DESIGN

1.Tbl_users_login

Primary key: **loginid**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	user_id	INT	Primary key	Unique identifier for the user
2	Email	VARCHAR(50)	Unique	User's email address
3	Name	VARCHAR(50)		User's name
4	Phone	VARCHAR(12)		User's phone number
5	Password	VARCHAR(50)		User's hashed password
6	Role	SMALLINT		User's role (Client, Therapist, Editor, Admin)
7	is_admin	BOOLEAN		Whether the user is an admin
8	is_staff	BOOLEAN		Whether the user is staff
9	is_active	BOOLEAN		Whether the user is active
10	is_superadmin	BOOLEAN		Whether the user is a superadmin

2.Tbl_users_registration

Primary key: **reg_id**

Foreign key: **user_id** references table **Tbl_users_login**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	reg_id	INT	PRIMARY KEY	Unique identifier for the user's profile
2	user_id	INT	FOREIGN KEY	References the user associated with this profile
3	profile_picture	VARCHAR(255)		Path to the user's profile picture
4	address	VARCHAR(50)		User's address
5	addressline1	VARCHAR(50)		User's address line 1
6	addressline2	VARCHAR(50)		User's address line 2
7	country	VARCHAR(50)		User's country
8	state	VARCHAR(40)		User's state
9	city	VARCHAR(15)		User's city
10	pin_code	VARCHAR(8)		User's PIN code
11	gender	VARCHAR(4)		User's gender
12	dob	DATE		User's date of birth
13	profile_created_at	DATETIME		Date and time when the profile was created
14	profile_modified_at	DATETIME		Date and time when the profile was last modified

3.Tbl_appointments

Primary key: **appointment_id**

Foreign key: **client_id** references table **Tbl_users_login**

Foreign key: **therapist_id** references table **Tbl_therapist**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	id	INT	PRIMARY KEY	Unique identifier for the appointment
2	date	DATE		Date of the appointment
3	client_id	INT	FOREIGN KEY	References the client user
4	therapist_id	INT	FOREIGN KEY	References the therapist user
5	time_slot	TIME		Time slot for the appointment
6	created_date	DATETIME		Date and time when the appointment was created
7	modified_date	DATETIME		Date and time when the appointment was last modified
8	cancelled_date	DATETIME		Date and time when the appointment was cancelled
9	status	VARCHAR(20)		Status of the appointment (e.g., Not Paid, Pending, Scheduled, Completed, Cancelled)
10	payment_status	BOOLEAN		Payment status of the appointment (True or False)
11	cancel_status	BOOLEAN		Cancellation status of the appointment (True or False)

4.Tbl_Therapy

Primary key: **therapy_id**

Foreign key: **client_id** references table **Tbl_users_login**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Therapy_id	INT	PRIMARY KEY	Unique identifier for the therapy
2	therapy_name	VARCHAR(100)	UNIQUE	Name of the therapy
3	description	TEXT		Description of the therapy
4	duration	VARCHAR(30)		Duration of the therapy
5	benefits	TEXT		Benefits of the therapy
6	status	BOOLEAN		Status of the therapy (True or False)
7	fees	DECIMAL(10,2)		Fees for the therapy

5.Tbl_Therapist

Primary key: **therapist_id**

Foreign key: **user_id** references table **Tbl_users_login**

Foreign key: **therapy_id** references table **Tbl_therapy**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Therapist_id	INT	PRIMARY KEY	Unique identifier for the therapist
2	bio	TEXT		Bio of the therapist
3	certification_name	VARCHAR(50)		Name of the certification
4	certificate_id	VARCHAR(100)	UNIQUE	Unique identifier for the certification
5	experience	INT		Experience of the therapist
6	user_id	INT	FOREIGN KEY	References the user associated with this therapist
7	therapy_id	INT	FOREIGN KEY	References the therapy associated with this therapist

6.Tbl_payment

Primary key: **payment_id**

Foreign key: **user_id** references table **Tbl_users_login**

Foreign key: **appointment_id** references table **Tbl_appointment**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Payment_id	INT	PRIMARY KEY	Unique identifier for the payment
2	user_id	INT	FOREIGN KEY	References the user associated with this payment
3	razorpay_order_id	VARCHAR(255)		Razorpay order ID
4	payment_id	VARCHAR(255)		Razorpay payment ID
5	amount	DECIMAL(8,2)		Amount paid
6	currency	VARCHAR(5)		Currency code (e.g., "INR")
7	timestamp	DATETIME		Timestamp of the payment
8	payment_status	VARCHAR(20)		Payment status (e.g., Pending, Successful, Failed)
9	appointment_id	INT	FOREIGN KEY	References the appointment associated with this payment

7.Tbl_therapy_session_schedule

Primary key: **session_id**

Foreign key: **appointment_id** references table **Tbl_appointment**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Session_id	INT	PRIMARY KEY	Unique identifier for the therapy session schedule
2	appointment_id	INT	FOREIGN KEY	References the appointment associated with this therapy session
3	platform	VARCHAR(20)		Platform for the therapy session (e.g., Zoom, Microsoft Teams)
4	meeting_url	VARCHAR		URL for the therapy session meeting
5	scheduled_time	DATETIME		Scheduled time for the therapy session
6	duration_minutes	INT		Duration of the therapy session in minutes
7	status	VARCHAR(20)		Status of the therapy session (e.g., Pending, Scheduled, Completed)
8	created_date	DATETIME		Date and time when the therapy session was created
9	modified_date	DATETIME		Date and time when the therapy session was last modified

8.Tbl_leave_request

Primary key: **request_id**

Foreign key: **therapist_id** references table **Tbl_therapist**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Request_id	INT	PRIMARY KEY	Unique identifier for the leave request
2	therapist_id	INT	FOREIGN KEY	References the therapist associated with this leave request
3	date	DATE		Date of the leave request
4	status	VARCHAR(10)		Status of the leave request (e.g., Pending, Accepted, Rejected)

8.Tbl_therapist_day_off

Primary key: **day_off_id**

Foreign key: **therapist_id** references table **Tbl_therapist**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Day_off_id	INT	PRIMARY KEY	Unique identifier for the leave request
2	therapist_id	INT	FOREIGN KEY	References the therapist associated with this leave request
3	date	DATE		Date of the therapist's day off

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

Software testing is a thorough method of analysing a piece of software's performance to see if it performs as planned. This procedure, which is also known as verification and validation, entails evaluating a product to make sure it complies with relevant standards and meets user needs. Software testing is accomplished using a variety of techniques, including reviews, analyses, inspections, and walkthroughs, as well as associated processes like static analysis. Static analysis, which evaluates source code without running it, and dynamic analysis, which keeps track of program behavior while it is in use and produces information like execution traces, timing profiles, and test coverage statistics. The various tasks that make up testing can be pre-organized and completed in a methodical manner. The testing process for computer-based systems begins with separate modules and moves forward through system integration. There are many laws that can be used as testing objectives, and testing is necessary for the system testing objectives to be successful.

5.2 TEST PLAN

A test plan defines the steps necessary to carry out different testing approaches, as well as the tasks that must be accomplished. The task of generating computer programmes, documentation, and data structures falls under the purview of software developers. Individual testing is required to make sure that every part of the programme operates as planned. An independent test group (ITG) is ready to give developers feedback and spot any potential problems. The test strategy must include a quantification of the testing objectives. These goals may include measurements like the mean time to failure, the cost of fixing errors, the density of remaining errors, the frequency of recurrence, and the number of hours needed for regression testing. The testing levels could include:

- Unit testing
- Integration Testing
- Validation Testing or System Testing
- Output Testing or User Acceptance Testing
- Automation Testing
- Selenium Testing

5.2.1 Unit Testing

Unit testing is a crucial stage of software verification that concentrates on testing individual components or modules, which are the fundamental building blocks of software design. This testing process entails reviewing the component-level design specifications to detect any errors within the module's boundaries by analyzing key control paths. The complexity of the test and the untested areas are determined during the process of unit testing, which is designed to be white-box focused, and multiple components can be tested simultaneously. To ensure proper functionality of the program unit being tested, the modular interface undergoes checks to ensure correct data flow in and out. Additionally, the integrity of temporarily stored data in the local data structure is examined throughout the algorithm's execution. The evaluation of boundary conditions confirms that every statement within the module has been executed at least once. Finally, inspection of each error management path is conducted to ensure proper error handling.

Performing data flow tests across module interfaces prior to any other testing is crucial for ensuring the effectiveness of the testing process. This is because if data cannot flow in and out of the system correctly, all other tests will be useless. Error handling channels must be established, and fault scenarios must be anticipated during the design stage to ensure that the system can redirect or stop working when an error occurs. The final stage of unit testing is boundary testing, which involves testing the software at its boundaries. This is because software often fails at its boundaries.

5.2.2 Integration Testing

Integration testing is a rigorous procedure that entails creating the structure of a program and executing tests to identify any problems related to the connections between different components. The objective is to establish a program structure based on design, utilizing components that have undergone unit testing. The overall program is then tested as a whole. However, correcting any issues can be challenging since the program's size makes it difficult to isolate the root causes. Once these errors are corrected, new ones may emerge, and the process may seem to repeat itself endlessly. After completing unit testing on all modules, they are integrated into the system to verify that there are no interface inconsistencies. This integration process also leads to the development of a distinctive program structure, as any discrepancies in the program structures are eliminated.

5.2.3 Validation Testing or System Testing

After conducting the testing phase, the entire system, including its code, modules, and class modules, was thoroughly examined using a process known as system tests or black box testing. Black box testing specifically aims to ensure that the software's functional requirements are met by creating input conditions that replicate all possible scenarios. The objective of this testing procedure is to detect and address a range of issues spanning from inadequate or inaccurate functionalities, interface discrepancies, errors in data arrangement or external data retrieval, performance drawbacks, initialization malfunctions, to termination glitches.

5.2.4 Output Testing or User Acceptance Testing

The system under consideration is tested for user acceptance; in this case, it must satisfy the business' requirements. The programme should consult the user and the perspective system while it is being developed in order to make any necessary adjustments. This was accomplished in regards to the following areas:

- Input Screen Designs
- Output Screen Designs

A variety of test data are used to conduct the aforementioned tests. The process of system testing requires the preparation of test data. After the preparation of sample data, the system being analyzed is put to the test using that data. Test data errors are discovered once more and resolved using the testing techniques mentioned above when the system is tested. Additionally, the fixes are noted for future use.

5.2.5 Automation Testing

Automation testing is a software testing process that verifies whether a software product meets specific requirements by automatically executing tests. The main purpose of automation testing is to detect bugs, defects, and other issues in software products and ensure that they perform as designed.

Benefits of using automation testing for software development:

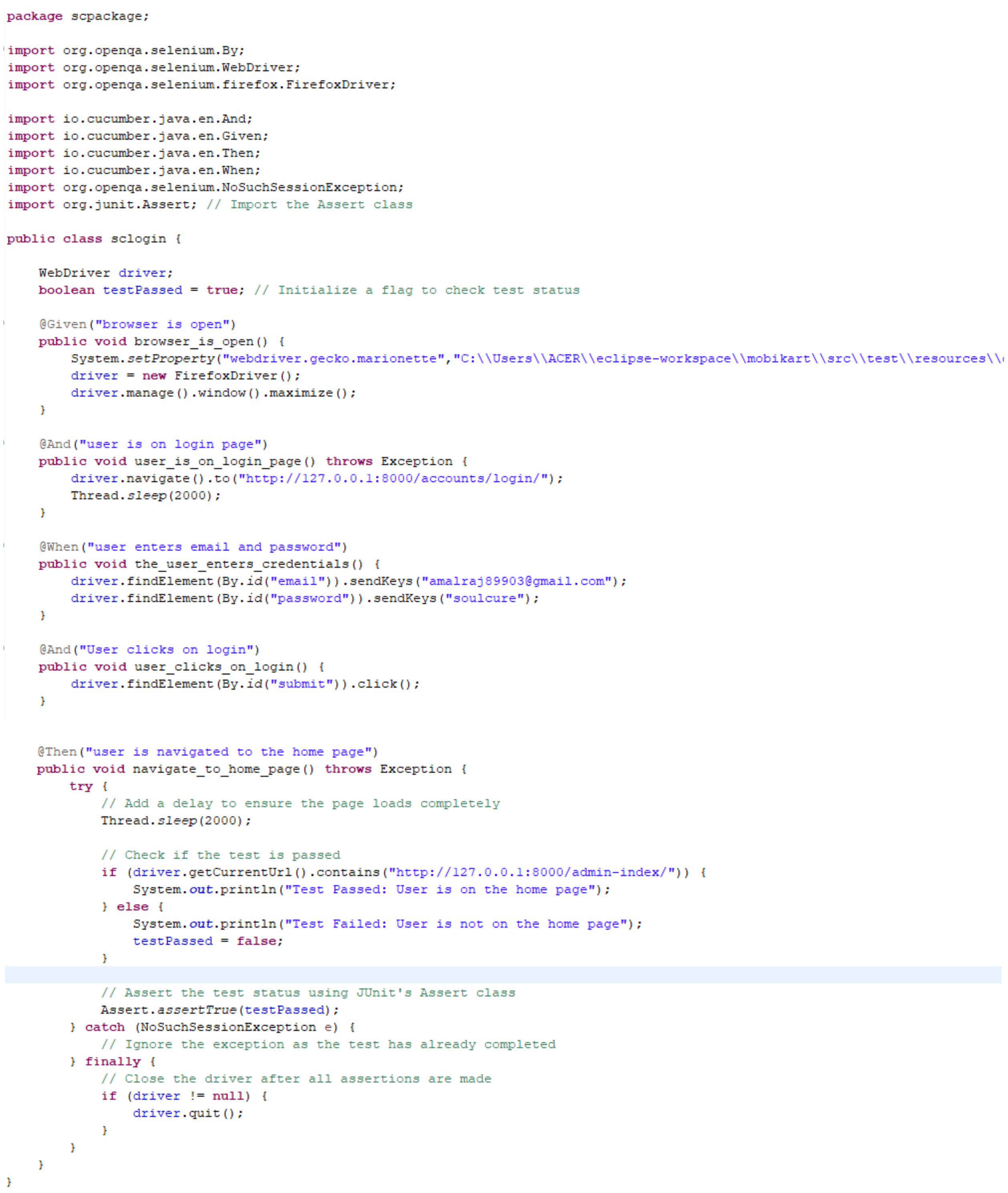
- Detailed reporting capabilities
- Improved bug detection
- Simplifies testing
- Speeds up the testing process
- Reduces human intervention

5.2.6 Selenium Testing

Selenium is a widely used open-source automation testing suite for web UI. Originally developed by Jason Huggins in 2004, Selenium supports automation across different browsers, platforms, and programming languages. It can be deployed on Windows, Linux, Solaris, Macintosh, and even supports mobile OS such as iOS, Windows Mobile, and Android. Selenium supports various programming languages through drivers specific to each language including C#, Java, Perl, PHP, Python, and Ruby. Selenium Web Driver is the most popular with Java and C#. Test scripts can be coded in any supported language and run directly in modern web browsers such as Internet Explorer, Mozilla Firefox, Google Chrome, and Safari. Selenium is not only used for functional tests but can also be integrated with automation test tools like Maven, Jenkins, and Docker for continuous testing. Furthermore, it can be integrated with TestNG and JUnit for managing test cases and generating reports.

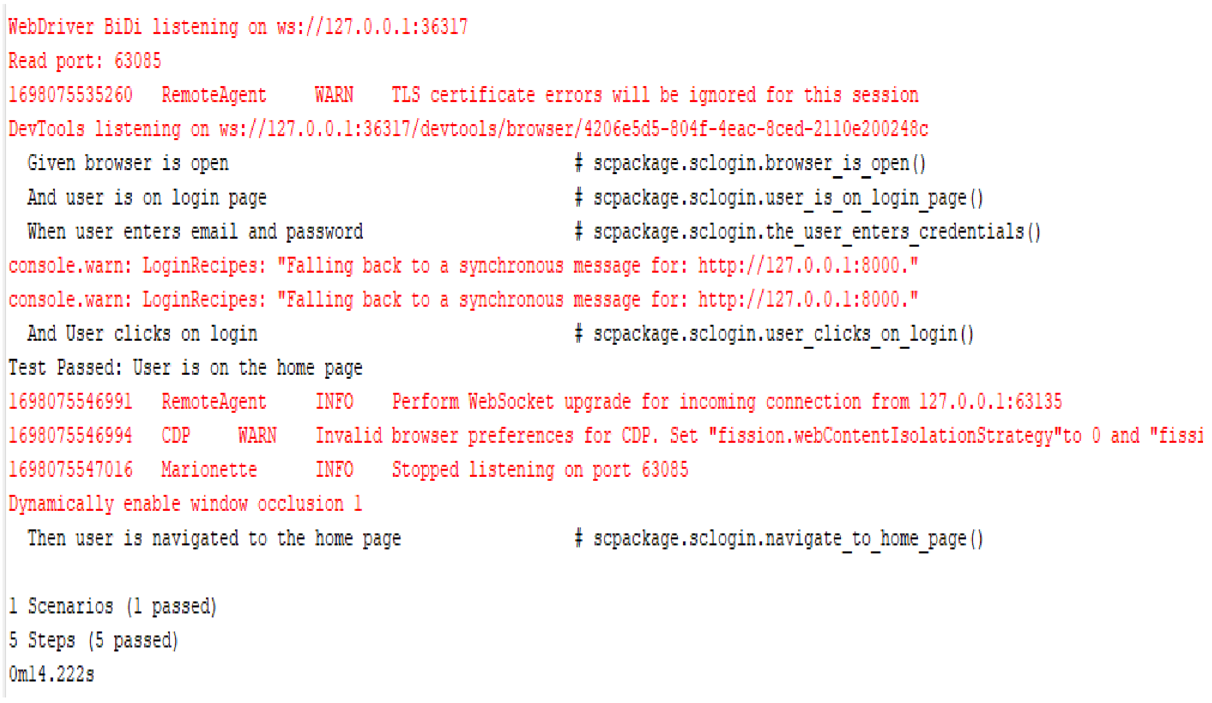
Test Case 1

Code



```
package scpackage;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.openqa.selenium.NoSuchSessionException;
import org.junit.Assert; // Import the Assert class

public class sclogin {

    WebDriver driver;
    boolean testPassed = true; // Initialize a flag to check test status

    @Given("browser is open")
    public void browser_is_open() {
        System.setProperty("webdriver.gecko.marionette","C:\\Users\\ACER\\eclipse-workspace\\mobikart\\src\\test\\resources\\
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
    }

    @And("user is on login page")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/accounts/login/");
        Thread.sleep(2000);
    }

    @When("user enters email and password")
    public void the_user_enters_credentials() {
        driver.findElement(By.id("email")).sendKeys("amalraj89903@gmail.com");
        driver.findElement(By.id("password")).sendKeys("soulcure");
    }

    @And("User clicks on login")
    public void user_clicks_on_login() {
        driver.findElement(By.id("submit")).click();
    }


    @Then("user is navigated to the home page")
    public void navigate_to_home_page() throws Exception {
        try {
            // Add a delay to ensure the page loads completely
            Thread.sleep(2000);

            // Check if the test is passed
            if (driver.getCurrentUrl().contains("http://127.0.0.1:8000/admin-index/")) {
                System.out.println("Test Passed: User is on the home page");
            } else {
                System.out.println("Test Failed: User is not on the home page");
                testPassed = false;
            }

            // Assert the test status using JUnit's Assert class
            Assert.assertTrue(testPassed);
        } catch (NoSuchSessionException e) {
            // Ignore the exception as the test has already completed
        } finally {
            // Close the driver after all assertions are made
            if (driver != null) {
                driver.quit();
            }
        }
    }
}
```

Output:



```
WebDriver BiDi listening on ws://127.0.0.1:36317
Read port: 63085
1698075535260   RemoteAgent     WARN    TLS certificate errors will be ignored for this session
DevTools listening on ws://127.0.0.1:36317/devtools/browser/4206e5d5-804f-4eac-8ced-2110e200248c
  Given browser is open                                    # scpackage.sclogin.browser_is_open()
  And user is on login page                                # scpackage.sclogin.user_is_on_login_page()
  When user enters email and password                      # scpackage.sclogin.the_user_enters_credentials()
console.warn: LoginRecipes: "Falling back to a synchronous message for: http://127.0.0.1:8000."
console.warn: LoginRecipes: "Falling back to a synchronous message for: http://127.0.0.1:8000."
  And User clicks on login                                 # scpackage.sclogin.user_clicks_on_login()
Test Passed: User is on the home page
1698075546991   RemoteAgent     INFO    Perform WebSocket upgrade for incoming connection from 127.0.0.1:63135
1698075546994   CDP     WARN    Invalid browser preferences for CDP. Set "fission.webContentIsolationStrategy"to 0 and "fissi
1698075547016   Marionette      INFO    Stopped listening on port 63085
Dynamically enable window occlusion 1
  Then user is navigated to the home page                  # scpackage.sclogin.navigate_to_home_page()

1 Scenarios (1 passed)
5 Steps (5 passed)
0m14.222s
```

Test Report

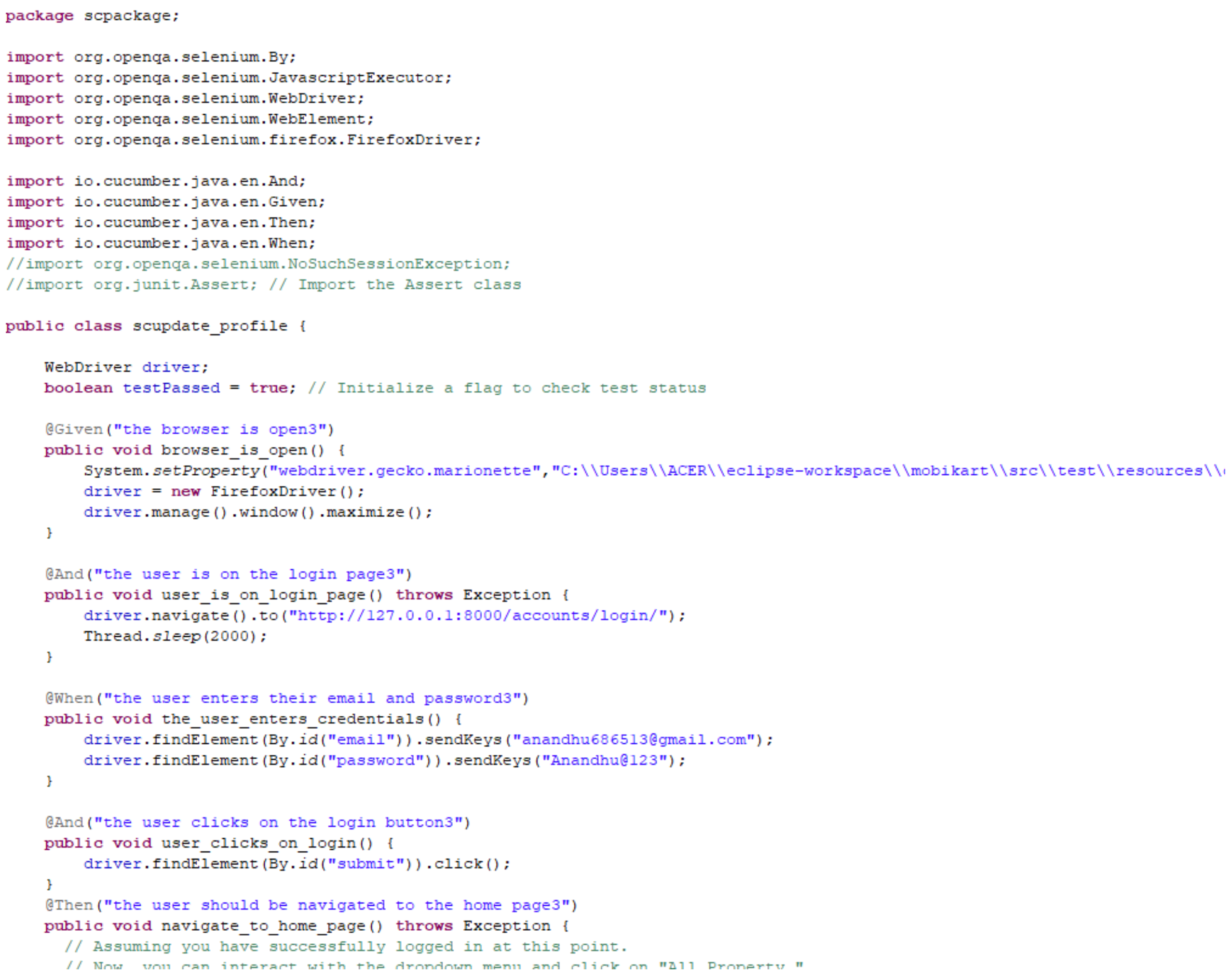
Project Name : SoulCure	
Login Test Case	
Test Case ID: 1	**Test Designed By: Amal Raj**
Test Priority (Low/Medium/High): High	**Test Designed Date: 23-**10-2023
Module Name: Login Page	**Test Executed By: Ms. Lisha Varghese**
Test Title: Verify login with valid email and password	**Test Execution Date: 23-**10-2023
Description: Test the Login Page	
Pre-Condition: User has valid email id and password	

Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Navigation to Login Page		Login Page should be displayed	Login page displayed	Pass
2	Provide Valid email	User Name: amalraj89903@gmail.com	User should be able to Login	User Logged in and navigated to the dashboard with records	Pass
3	Provide Valid Password	Password: soulcure			
4	Click on Sign In button				

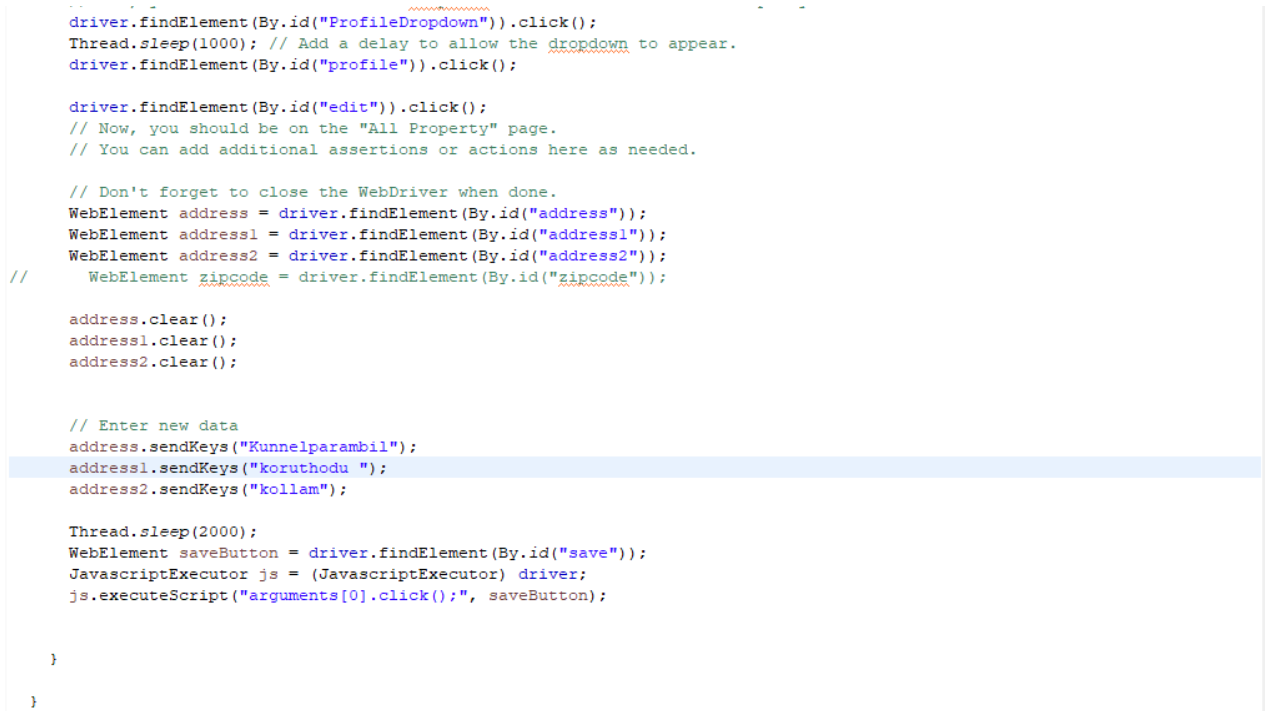
Post-Condition: User is validated with database and successfully login into account. The Account session details are logged in database

Test Case 2:

Code

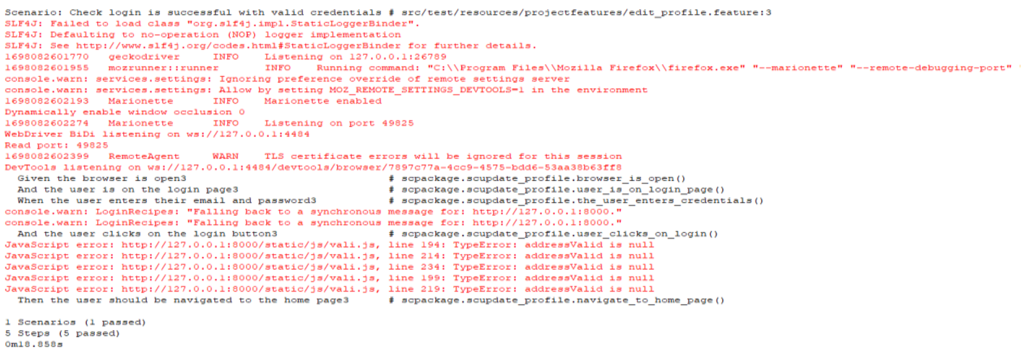


```
package scpackage;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
//import org.openqa.selenium.NoSuchSessionException;
//import org.junit.Assert; // Import the Assert class

public class scupdate_profile {

    WebDriver driver;
    boolean testPassed = true; // Initialize a flag to check test status

    @Given("the browser is open3")
    public void browser_is_open() {
        System.setProperty("webdriver.gecko.marionette","C:\\Users\\ACER\\eclipse-workspace\\mobikart\\src\\test\\resources\\
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
    }

    @And("the user is on the login page3")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/accounts/login/");
        Thread.sleep(2000);
    }

    @When("the user enters their email and password3")
    public void the_user_enters_credentials() {
        driver.findElement(By.id("email")).sendKeys("anandhu686513@gmail.com");
        driver.findElement(By.id("password")).sendKeys("Anandhu@123");
    }

    @And("the user clicks on the login button3")
    public void user_clicks_on_login() {
        driver.findElement(By.id("submit")).click();
    }
    @Then("the user should be navigated to the home page3")
    public void navigate_to_home_page() throws Exception {
      // Assuming you have successfully logged in at this point.
      // Now, you can interact with the dropdown menu and click on "All Property."
```



```
        driver.findElement(By.id("ProfileDropdown")).click();
        Thread.sleep(1000); // Add a delay to allow the dropdown to appear.
        driver.findElement(By.id("profile")).click();

        driver.findElement(By.id("edit")).click();
        // Now, you should be on the "All Property" page.
        // You can add additional assertions or actions here as needed.

        // Don't forget to close the WebDriver when done.
        WebElement address = driver.findElement(By.id("address"));
        WebElement address1 = driver.findElement(By.id("address1"));
        WebElement address2 = driver.findElement(By.id("address2"));
//      WebElement zipcode = driver.findElement(By.id("zipcode"));

        address.clear();
        address1.clear();
        address2.clear();


        // Enter new data
        address.sendKeys("Kunnelparambil");
        address1.sendKeys("koruthodu ");
        address2.sendKeys("kollam");

        Thread.sleep(2000);
        WebElement saveButton = driver.findElement(By.id("save"));
        JavascriptExecutor js = (JavascriptExecutor) driver;
        js.executeScript("arguments[0].click();", saveButton);


      }

  }
```

Output



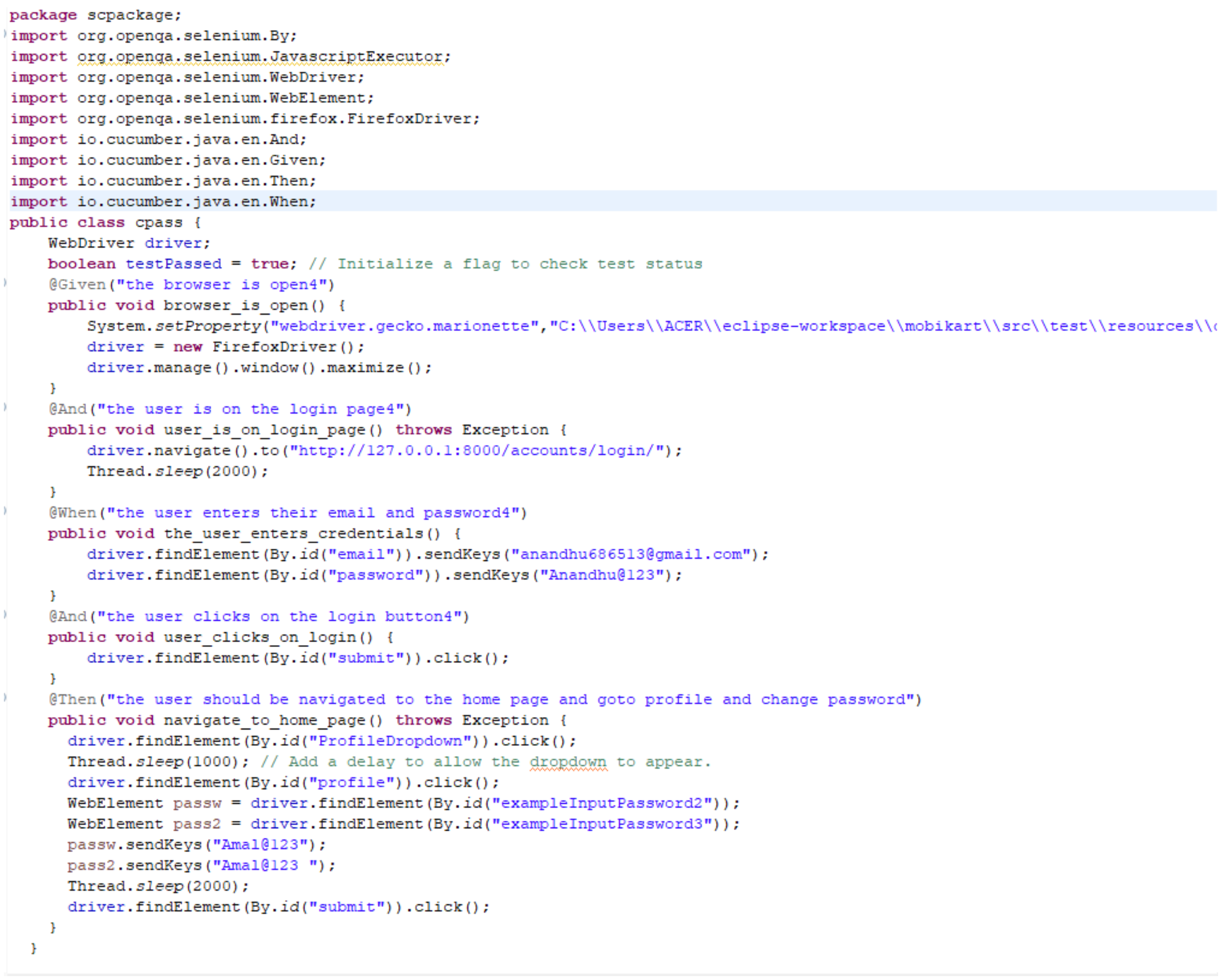
```
Scenario: Check login is successful with valid credentials # src/test/resources/projectfeatures/edit_profile.feature:3
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
1698082601770   geckodriver     INFO    Listening on 127.0.0.1:26789
1698082601955   mozrunner::runner       INFO    Running command: "C:\\Program Files\\Mozilla Firefox\\firefox.exe" "--marionette" "--remote-debugging-port"
console.warn: services.settings: Ignoring preference override of remote settings server
console.warn: services.settings: Allow by setting MOZ_REMOTE_SETTINGS_DEVTOOLS=1 in the environment
1698082602193   Marionette      INFO    Marionette enabled
Dynamically enable window occlusion 0
1698082602274   Marionette      INFO    Listening on port 49825
WebDriver BiDi listening on ws://127.0.0.1:4484
Read port: 49825
1698082602399   RemoteAgent     WARN    TLS certificate errors will be ignored for this session
DevTools listening on ws://127.0.0.1:4484/devtools/browser/7897c77a-4cc9-4575-bdd6-53aa38b63ff8
  Given the browser is open3                          # scpackage.scupdate_profile.browser_is_open()
  And the user is on the login page3                  # scpackage.scupdate_profile.user_is_on_login_page()
  When the user enters their email and password3      # scpackage.scupdate_profile.the_user_enters_credentials()
console.warn: LoginRecipes: "Falling back to a synchronous message for: http://127.0.0.1:8000."
console.warn: LoginRecipes: "Falling back to a synchronous message for: http://127.0.0.1:8000."
  And the user clicks on the login button3            # scpackage.scupdate_profile.user_clicks_on_login()
JavaScript error: http://127.0.0.1:8000/static/js/vali.js, line 194: TypeError: addressValid is null
JavaScript error: http://127.0.0.1:8000/static/js/vali.js, line 214: TypeError: addressValid is null
JavaScript error: http://127.0.0.1:8000/static/js/vali.js, line 234: TypeError: addressValid is null
JavaScript error: http://127.0.0.1:8000/static/js/vali.js, line 199: TypeError: addressValid is null
JavaScript error: http://127.0.0.1:8000/static/js/vali.js, line 219: TypeError: addressValid is null
  Then the user should be navigated to the home page3 # scpackage.scupdate_profile.navigate_to_home_page()

1 Scenarios (1 passed)
5 Steps (5 passed)
0m18.858s
```

Test report

Project Name : SoulCure					
Profile Update Test Case					
Test Case ID: 4			**Test Designed By: Amal Raj**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 23-**10-2023		
Module Name: Update profile			**Test Executed By: Ms. Lisha Varghese**		
Test Title: Verify update profile			**Test Execution Date: 23-**10-2023		
Description: Testing the update profile page					
Pre-Condition: Require valid profile fields and update button					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Navigation to profile Page		Profile page should be displayed	Profile page displayed	Pass
2	Provide Address line1	kunnelparambil	User should be able to update profile	Profile updated and message should be displayed.	Pass
3	Provide Address line2	Korthudu po			
4	Provide Address line3	kollam			
4	Click on save button				
Post-Condition: User is inserted into the database and successfully registered. Profile details updated and inserted successfully into database.					

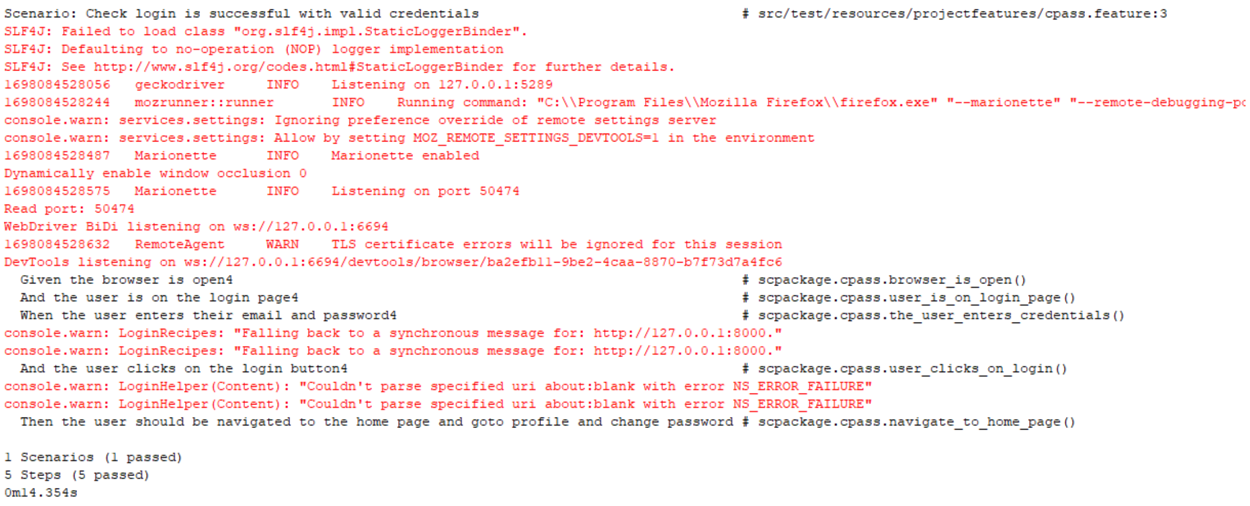
Test Case 3:

Code



```
package scpackage;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class cpass {
    WebDriver driver;
    boolean testPassed = true; // Initialize a flag to check test status
    @Given("the browser is open4")
    public void browser_is_open() {
        System.setProperty("webdriver.gecko.marionette","C:\\Users\\ACER\\eclipse-workspace\\mobikart\\src\\test\\resources\\
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
    }
    @And("the user is on the login page4")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/accounts/login/");
        Thread.sleep(2000);
    }
    @When("the user enters their email and password4")
    public void the_user_enters_credentials() {
        driver.findElement(By.id("email")).sendKeys("anandhu686513@gmail.com");
        driver.findElement(By.id("password")).sendKeys("Anandhu@123");
    }
    @And("the user clicks on the login button4")
    public void user_clicks_on_login() {
        driver.findElement(By.id("submit")).click();
    }
    @Then("the user should be navigated to the home page and goto profile and change password")
    public void navigate_to_home_page() throws Exception {
      driver.findElement(By.id("ProfileDropdown")).click();
      Thread.sleep(1000); // Add a delay to allow the dropdown to appear.
      driver.findElement(By.id("profile")).click();
      WebElement passw = driver.findElement(By.id("exampleInputPassword2"));
      WebElement pass2 = driver.findElement(By.id("exampleInputPassword3"));
      passw.sendKeys("Amal@123");
      pass2.sendKeys("Amal@123 ");
      Thread.sleep(2000);
      driver.findElement(By.id("submit")).click();
    }
  }
```

Output:



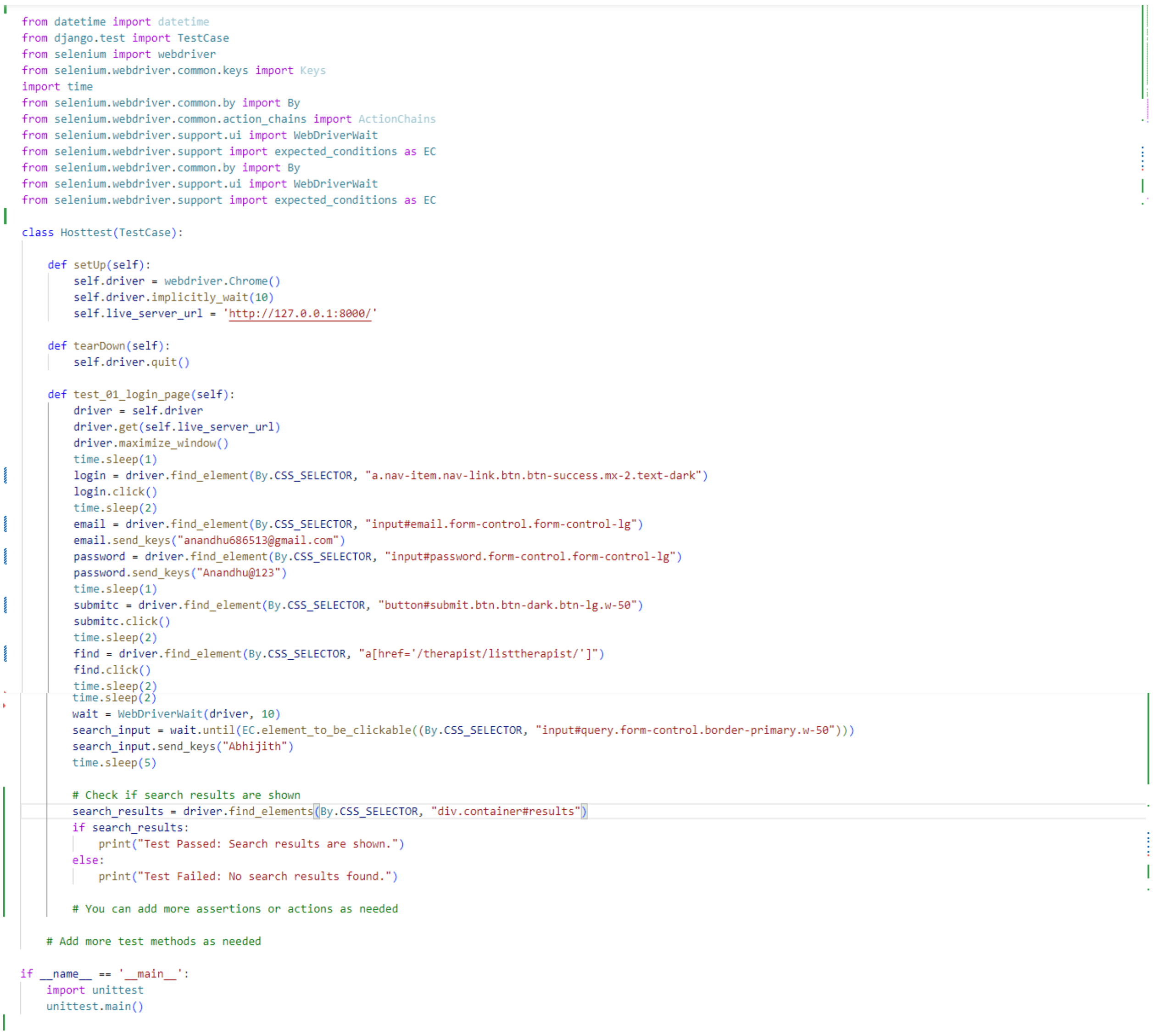
```
Scenario: Check login is successful with valid credentials                              # src/test/resources/projectfeatures/cpass.feature:3
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
1698084528056   geckodriver     INFO    Listening on 127.0.0.1:5289
1698084528244   mozrunner::runner       INFO    Running command: "C:\\Program Files\\Mozilla Firefox\\firefox.exe" "--marionette" "--remote-debugging-pc
console.warn: services.settings: Ignoring preference override of remote settings server
console.warn: services.settings: Allow by setting MOZ_REMOTE_SETTINGS_DEVTOOLS=1 in the environment
1698084528487   Marionette      INFO    Marionette enabled
Dynamically enable window occlusion 0
1698084528575   Marionette      INFO    Listening on port 50474
Read port: 50474
WebDriver BiDi listening on ws://127.0.0.1:6694
1698084528632   RemoteAgent     WARN    TLS certificate errors will be ignored for this session
DevTools listening on ws://127.0.0.1:6694/devtools/browser/ba2efb11-9be2-4caa-8870-b7f73d7a4fc6
  Given the browser is open4                                                             # scpackage.cpass.browser_is_open()
  And the user is on the login page4                                                     # scpackage.cpass.user_is_on_login_page()
  When the user enters their email and password4                                         # scpackage.cpass.the_user_enters_credentials()
console.warn: LoginRecipes: "Falling back to a synchronous message for: http://127.0.0.1:8000."
console.warn: LoginRecipes: "Falling back to a synchronous message for: http://127.0.0.1:8000."
  And the user clicks on the login button4                                               # scpackage.cpass.user_clicks_on_login()
console.warn: LoginHelper(Content): "Couldn't parse specified uri about:blank with error NS_ERROR_FAILURE"
console.warn: LoginHelper(Content): "Couldn't parse specified uri about:blank with error NS_ERROR_FAILURE"
  Then the user should be navigated to the home page and goto profile and change password # scpackage.cpass.navigate_to_home_page()

1 Scenarios (1 passed)
5 Steps (5 passed)
0m14.354s
```

Test Report

Project Name : SoulCure					
Change Password Test Case					
Test Case ID: 5			**Test Designed By: Amal Raj**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 23-**10-2023		
Module Name: Change password			**Test Executed By: Ms. Lisha Varghese**		
Test Title: Verify change passwsord			**Test Execution Date: 23-**10-2023		
Description: Testing change password					
Pre-Condition: Require valid password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Navigation to change password page		Profile page should be displayed	Profile page displayed	Pass
2	Provide old password	Nandhu@123	User should be able click add on change password button	Password should be changed.	Pass
4	Provide new password	Anandhu@123			
5	Provide confirm password	Anandhu@123	User should not be able click button	Password should not be changed.	Pass
7	Click on Change password button				
Post-Condition: Password Change and inserted successfully into database					

Test Case 4:

Code



```
from datetime import datetime
from django.test import TestCase
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

class Hosttest(TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.live_server_url = 'http://127.0.0.1:8000/'

    def tearDown(self):
        self.driver.quit()

    def test_01_login_page(self):
        driver = self.driver
        driver.get(self.live_server_url)
        driver.maximize_window()
        time.sleep(1)
        login = driver.find_element(By.CSS_SELECTOR, "a.nav-item.nav-link.btn.btn-success.mx-2.text-dark")
        login.click()
        time.sleep(2)
        email = driver.find_element(By.CSS_SELECTOR, "input#email.form-control.form-control-lg")
        email.send_keys("anandhu686513@gmail.com")
        password = driver.find_element(By.CSS_SELECTOR, "input#password.form-control.form-control-lg")
        password.send_keys("Anandhu@123")
        time.sleep(1)
        submitc = driver.find_element(By.CSS_SELECTOR, "button#submit.btn.btn-dark.btn-lg.w-50")
        submitc.click()
        time.sleep(2)
        find = driver.find_element(By.CSS_SELECTOR, "a[href='/therapist/listtherapist/']")
        find.click()
        time.sleep(2)
        time.sleep(2)
        wait = WebDriverWait(driver, 10)
        search_input = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, "input#query.form-control.border-primary.w-50")))
        search_input.send_keys("Abhijith")
        time.sleep(5)

        # Check if search results are shown
        search_results = driver.find_elements(By.CSS_SELECTOR, "div.container#results")
        if search_results:
            print("Test Passed: Search results are shown.")
        else:
            print("Test Failed: No search results found.")

        # You can add more assertions or actions as needed

    # Add more test methods as needed

if __name__ == '__main__':
    import unittest
    unittest.main()
```

Output:

```
(scenv) D:\SoulCure\Main\soulcure>py manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:51763/devtools/browser/9083bc50-87d7-4b0d-abc9-5007ce943305
Test Passed: Search results are shown.
.
----------------------------------------------------------------------
Ran 1 test in 38.960s

OK
Destroying test database for alias 'default'...
```

Project Name : SoulCure					
Change Password Test Case					
Test Case ID: 5			**Test Designed By: Amal Raj**		
Test Priority (Low/Medium/High): High			**Test Designed Date:** 23-10-2023		
Module Name: Search			**Test Executed By: Ms. Lisha Varghese**		
Test Title: Verify Search Function			**Test Execution Date:** 23-10-2023		
Description: Testing Search Function					
Pre-Condition: Require valid Search data					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Navigation to home page		Nav bar should shown with search bar	Nav bar shown with search bar	Pass
2	Provide query to be searched	Abhijith	User should be able input in search bar	User was able input in search bar and search results are shown	
3	Results are shown				Pass
Post-Condition: Results for query are shown in the page					

CHAPTER 6

IMPLEMENTATION

6.1INTRODUCTION

During the implementation stage of a project, the theoretical design is transformed into a functional system, which is essential in achieving a successful outcome and user confidence in the system's effectiveness and accuracy. User training and documentation are the primary focus during this stage, and conversion typically takes place around the same time as user training or afterwards. Implementation involves converting the new system design into an operational one and can create chaos and confusion if not carefully planned or controlled. The implementation process encompasses all activities necessary to convert from the existing system to the new system, whether it be a totally new or modified system. It is crucial to provide a reliable system that meets organizational requirements. The system can only be implemented after thorough testing is done, and it is found to be working according to specifications. Feasibility checks are performed by system personnel, and the more complex the system being implemented, the more involved the system analysis and design effort required for education and training, system testing, and changeover.
The implementation state involves the following tasks:

- Careful planning.
- Investigation of system and constraints.
- Design of methods to achieve the changeover.

6.2 IMPLEMENTATION PROCEDURES

Implementation of software refers to the final installation of the package in its real environment, to the satisfaction of the intended uses and the operation of the system. In many organizations someone who will not be operating it, will commission the software development project. In the initial stage people doubt about the software but we have to ensure that the resistance does not build up, as one has to make sure that:

- The active user must be aware of the benefits of using the new system.
- Their confidence in the software is built up.
- Proper guidance is imparted to the user so that he is comfortable in using the application.

Before going ahead and viewing the system, the user must know that for viewing the result, the server program should be running in the server. If the server object is not up running on the server, the actual process won’t take place.

6.2.1 User Training

The goal of user training is to give users the confidence to test and alter computer-based systems in order to finally accomplish the intended goals. Training becomes more important as systems get more complicated. As part of the user training process, participants are exposed to a variety of crucial tasks like data entering, handling error alerts, querying databases, calling up routines to generate reports, among others.

6.2.2 Training on the Application Software

The new application software requires users to first complete a basic computer literacy training course before utilizing it. They should be shown how to access help resources, traverse the software panels, correct entry errors, and update data that has already been entered. Specific topics pertaining to the user group and their function in using the system or its components should also be covered in the training. The training for the programme should be adapted to the requirements of various user groups and hierarchical levels.

6.2.3 System Maintenance

The software maintenance phase is an essential part of the software development life cycle and occurs after a software product has been successfully implemented and is performing useful work. Maintenance is necessary to ensure that the system remains adaptable to changes in the system environment. While finding mistakes is a part of software maintenance, it is not the only task involved. There are many other activities involved in maintenance, such as updating documentation, fixing bugs, enhancing existing features, and adding new features. Effective maintenance can help ensure that the software remains functional, reliable, and efficient over time.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

In an age of rapid technological advancement, the SoulCure project is a pioneering innovation in therapy services. It ushers in a new era, departing from traditional methods and embracing efficiency and user-friendliness. SoulCure's primary objective is to provide a cutting-edge web platform that simplifies the search for therapists and therapy modalities, enhancing the user experience. Administrators can efficiently manage therapy offerings, ensuring the platform remains up to date and secure. Users benefit from an intuitive and informative system, where they can explore therapist profiles, access various therapy modalities, and book appointments with ease. The system's rigorous testing ensures a seamless and dependable user experience, fostering mental well-being and enhancing quality of life. Overall, SoulCure strives to be a comprehensive holistic healing platform that leverages automation and personalization to enable progress towards mental, emotional and physical wellbeing goals.

7.2 FUTURE SCOPE

Going forward, SoulCure has immense potential for growth and innovation. The platform can expand its therapy offerings to include more modalities like art, dance, occupational therapies etc. Integrating wearables data and health tracking capabilities can provide richer insights during assessments. The portal can be developed into a wellness marketplace by bringing on board nutritionists, fitness experts, life coaches etc. Enhanced personalization can be enabled by adding features like sentiment analysis during therapy sessions. Partnerships with hospitals can drive clinical use cases by facilitating doctor consultations and remote patient monitoring. From a business perspective, a subscription model can be introduced to provide bundled wellness packages. Investing in emerging technologies like VR can make therapies more experiential and engaging. Through a combination of an expanded holistic network, deeper personalization and immersive formats, SoulCure can fulfil its vision of making whole-person wellbeing care accessible to all.

CHAPTER 8

BIBLIOGRAPHY

REFERENCES:

- Gary B. Shelly, Harry J. Rosenblatt, “*System Analysis and Design*”, 2009.
- Roger S Pressman, “*Software Engineering*”, 1994.
- Pankaj Jalote, “So*ftware engineering*: a precise approach”, 2006.
- IEEE Std 1016 Recommended Practice for Software Design Descriptions.

WEBSITES:

- www.w3schools.com
- https:/stackoverflow.com/
- https://www.youtube.com/brototype/
- https://app.diagrams.net/

CHAPTER 9

APPENDIX

9.1 Sample Code

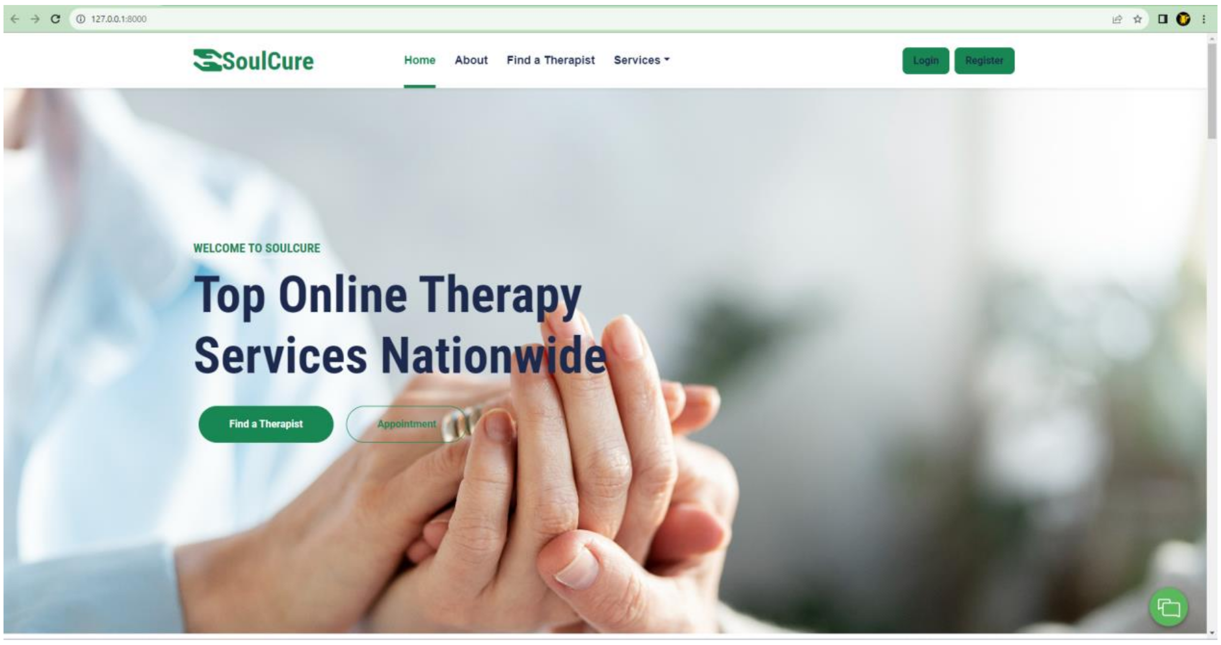
Views.py

```
def userlogin(request):
    if request.user.is_authenticated:
        if request.user.role == CustomUser.CLIENT:
            return redirect('http://127.0.0.1:8000/')
        elif request.user.role == CustomUser.THERAPIST:
            return redirect(reverse('therapist'))
        elif request.user.role == CustomUser.ADMIN:
            return redirect(reverse('adminindex'))
        else:
            return redirect('http://127.0.0.1:8000/')
    elif request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')
        print(email)
        print(password)

        if email and password:
            user = authenticate(request, email=email, password=password)
            print("Authenticated user:", user)
            if user is not None:
                auth_login(request, user)
                print("User authenticated:", user.email, user.role)
                if request.user.role == CustomUser.CLIENT:
                    return redirect('http://127.0.0.1:8000/')
                elif request.user.role == CustomUser.THERAPIST:
                    return redirect(reverse('therapist'))
                elif request.user.role == CustomUser.ADMIN:
                    return redirect(reverse('adminindex'))
                else:
                    return redirect('http://127.0.0.1:8000/')

            else:
                error_message = "Invalid login credentials."
                return render(request, 'login.html', {'error_message': error_message})
        else:
            error_message = "Please fill out all fields."
            return render(request, 'login.html', {'error_message': error_message})

    return render(request, 'login.html')


@user_not_authenticated
def register(request):
    if request.user.is_authenticated:
        if request.user.role == CustomUser.CLIENT:
            return redirect('http://127.0.0.1:8000/')
        elif request.user.role == CustomUser.THERAPIST:
            return redirect(reverse('therapist'))
        elif request.user.role == CustomUser.ADMIN:
            return redirect(reverse('adminindex'))
        else:
```

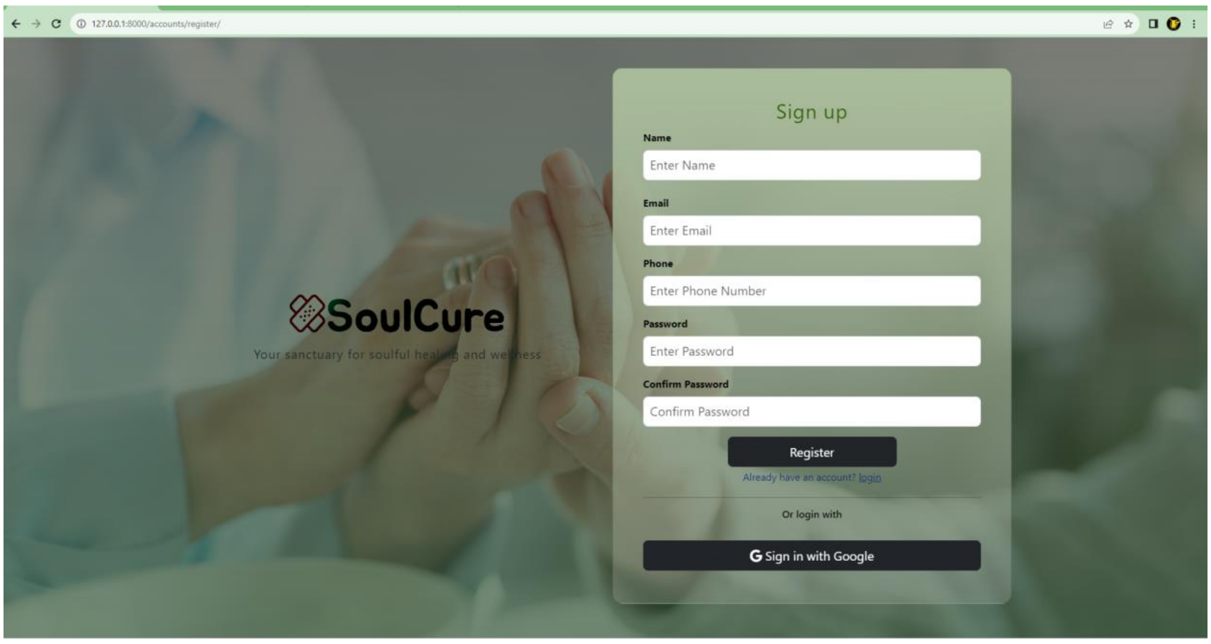
```
        return redirect('http://127.0.0.1:8000/')
    elif request.method == 'POST':
        name1 = request.POST.get('name', None)
        email = request.POST.get('email', None)
        phone = request.POST.get('phone', None)
        password = request.POST.get('pass', None)
        confirm_password = request.POST.get('cpass', None)
        role = User.CLIENT

        if name1 and email and phone and password and role:
            if User.objects.filter(email=email).exists():
                error_message = "Email is already registered."
                return render(request, 'register2.html', {'error_message': error_message})

            elif password!=confirm_password:
                error_message = "Password's Don't Match, Enter correct Password"
                return render(request, 'register2.html', {'error_message': error_message})


            else:
                user = User(name=name1, email=email, phone=phone,role=role)
                user.set_password(password)  # Set the password securely
                user.is_active=False
                user.save()
                user_profile = UserProfile(user=user)
                user_profile.save()
                activateEmail(request, user, email)
                return redirect('login')

    return render(request, 'register2.html')


def userLogout(request):
    logout(request)
    return redirect('http://127.0.0.1:8000/')


from django.shortcuts import render
from django.http import JsonResponse
from django.db.models import Q
from therapist.models import Therapist



@login_required
def search_therapists(request):
    return render(request, 'demosearch.html')



@login_required
def search_therapists2(request):
    query = request.GET.get('query')

    # Use Q objects to filter therapists based on therapy name or therapist name
    therapists = Therapist.objects.filter(
        Q(user__name__icontains=query) | Q(therapy__therapy_name__icontains=query)
    )

    therapists_data = [
        {
            'id': therapist.user.id,
            'name': therapist.user.name,
            'therapy': therapist.therapy.therapy_name,
            'profile_picture': therapist.user.userprofile.profile_picture.url,
            'experience': therapist.experience,
            'certification_name': therapist.certification_name,
        }
        for therapist in therapists
    ]

    return JsonResponse({'therapists': therapists_data})


@login_required
def book_appointment(request, t_id):
    therapist = get_object_or_404(CustomUser, id=t_id)
    print(therapist)
    current_therapists=Therapist.objects.filter(user=therapist)
    print(current_therapists)
    if current_therapists.exists():
    # Access the first therapist in the queryset (you may need to loop through if there are multiple
therapists)
        current_therapist = current_therapists.first()

        # Access the associated therapy
        associated_therapy = current_therapist.therapy

        if associated_therapy:
            # Access the fee for the associated therapy
            therapy_fee = associated_therapy.fees
            print("Therapy Fee:", therapy_fee)
        else:
            print("Therapist is not associated with any therapy.")
    else:
        print("Therapist not found.")



    context = None

    if request.method == 'POST':
        date = request.POST.get('date')
        time_slot = request.POST.get('time_slot')

        # Check if the current user (client) has already booked an appointment for the same date and
time slot
        existing_appointment = Appointment.objects.filter(client=request.user, date=date,
time_slot=time_slot).exclude(time_slot__isnull=True).first()

        # Check if the current user (client) has already booked an appointment for the same date
        existing_appointment_same_date = Appointment.objects.filter(client=request.user,
date=date).exclude(time_slot__isnull=True).first()

        # Check if the selected date is in the therapist's day-offs
        therapist_day_off = TherapistDayOff.objects.filter(therapist=therapist, date=date).first()

        if therapist_day_off:
            context = {
                'error': f'{therapist.name} is on leave on {date}. Please select a different date.',
                'therapist': therapist,
            }
        elif existing_appointment:
            apps = Appointment.objects.filter(date=date,
time_slot=time_slot).exclude(time_slot__isnull=True)
            time_slots = {time(9, 0): 1, time(11, 0): 1, time(13, 0): 1, time(15, 0): 1, time(17, 0): 1}
            for app in apps:
                time_slots[app.time_slot] = 0

            available_slots = [time_slot.strftime('%I:%M %p') for time_slot, available in
time_slots.items() if available]
            available_slots = ", ".join(available_slots)

            user = request.user
            initial_data = {
                'client': user,
                'client_name': user.name,
                'client_phone': user.phone,
                'therapist': therapist.id,
                'therapist_name': therapist.name,
            }
            appointment_form = AppointmentForm(initial=initial_data, therapist_leave_dates=[])
            user_form = CurrentUserForm(instance=user)
            context = {
                'error': 'You have already scheduled an appointment for the selected Date and Time Slot',
                'therapist': therapist,
                'appointment_form': appointment_form,
                'user_form': user_form,
                'available_slots': available_slots
            }
        elif existing_appointment_same_date:
            user = request.user
            initial_data = {
                'client': user,
                'client_name': user.name,
                'client_phone': user.phone,
                'therapist': therapist.id,
                'therapist_name': therapist.name,
            }
            appointment_form = AppointmentForm(initial=initial_data, therapist_leave_dates=[])
            user_form = CurrentUserForm(instance=user)
            context = {
                'error': 'You have already scheduled an appointment for the selected Date',
                'therapist': therapist,
                'appointment_form': appointment_form,
                'user_form': user_form,
            }
        else:
            form = AppointmentForm(request.POST, therapist_leave_dates=[])
            form.instance.client = request.user
            form.instance.therapist = therapist

            if form.is_valid():

                # appointment1=form.save()
                appointment1 = form.save(commit=False)  # Save the form data to the appointment
instance but don't commit to the database yet

                # appointment1_id = appointment1.id

                # client = razorpay.Client(auth=(settings.RAZORPAY_KEY_ID,
settings.RAZORPAY_KEY_SECRET))


                # Save the appointment instance to the database

                appointment1.save()
                t_fee=int(therapy_fee)
                return redirect('payment',appointment_id=appointment1.id,t_fees=therapy_fee)
    else:
        user = request.user
        initial_data = {
            'client': user,
            'client_name': user.name,
            'client_phone': user.phone,
            'therapist': therapist.id,
            'therapist_name': therapist.name,
        }
        current_date = c_date.today()
        leave_dates = [str(date) for date in TherapistDayOff.objects.filter(therapist=therapist,
date__gte=current_date).values_list('date', flat=True)]
        appointment_form = AppointmentForm(initial=initial_data, therapist_leave_dates=leave_dates)
        user_form = CurrentUserForm(instance=user)

        context = {'appointment_form': appointment_form, 'user_form': user_form, 'therapist': therapist,
'leave_dates': leave_dates}

    return render(request, 'appointment.html', context)



def payment(request, appointment_id,t_fees):
    # Use get_object_or_404 to get the Subscription object based on sub_id
        # Retrieve subscription features from a specific Subscription instance
    # You may want to retrieve a specific subscription
    print(t_fees)
    t_fees = float(request.resolver_match.kwargs['t_fees'])
    print(t_fees)

    appointments = Appointment.objects.all()
    current_appointment = Appointment.objects.get(pk=appointment_id)
    # For Razorpay integration
    currency = 'INR'
    amount = t_fees  # Get the subscription price
    amount_in_paise = int(amount * 100)  # Convert to paise
    print(amount_in_paise)

    # Create a Razorpay Order
    razorpay_order = razorpay_client.order.create(dict(
        amount=amount_in_paise,
        currency=currency,
        payment_capture='0'
    ))

    # Order ID of the newly created order
    razorpay_order_id = razorpay_order['id']
    callback_url = reverse('paymenthandler', args=[appointment_id])  # Define your callback URL
here

    phone=current_appointment.client.phone
    print(phone)
    payment = Payment.objects.create(
        user=request.user,
        razorpay_order_id=razorpay_order_id,
        payment_id="",
        amount=amount,
        currency=currency,
        payment_status=Payment.PaymentStatusChoices.PENDING,
        appointment=current_appointment
    )
    appointment=current_appointment
    # Prepare the context data
    context = {
        'user': request.user,
        'appointment':appointment,
        # 'therapy_fee':t_fees,
        'razorpay_order_id': razorpay_order_id,
        'razorpay_merchant_key': settings.RAZOR_KEY_ID,
        'razorpay_amount': amount_in_paise,
        'currency': currency,
        'amount': amount_in_paise / 100,
        'callback_url': callback_url,
        'phone':phone,

    }

    return render(request, 'client/razorpay_payment.html', context)


# @csrf_exempt
# def payment_confirmation(request, order_id):
#     try:
#         # Retrieve the appointment based on the order_id
@csrf_exempt
def paymenthandler(request, appointment_id):
    # Only accept POST requests.
    if request.method == "POST":
        # Get the required parameters from the POST request.
        payment_id = request.POST.get('razorpay_payment_id', '')
        razorpay_order_id = request.POST.get('razorpay_order_id', '')
        signature = request.POST.get('razorpay_signature', '')
        params_dict = {
            'razorpay_order_id': razorpay_order_id,
            'razorpay_payment_id': payment_id,
            'razorpay_signature': signature
        }
    # Verify the payment signature.
        result = razorpay_client.utility.verify_payment_signature(params_dict)

        payment = Payment.objects.get(razorpay_order_id=razorpay_order_id)
        amount = int(payment.amount * 100)  # Convert Decimal to paise

        # Capture the payment.
        razorpay_client.payment.capture(payment_id, amount)
        payment = Payment.objects.get(razorpay_order_id=razorpay_order_id)

        # Update the order with payment ID and change status to "Successful."
        payment.payment_id = payment_id
        payment.payment_status = Payment.PaymentStatusChoices.SUCCESSFUL
        payment.save()

        try:
            update_appointment = Appointment.objects.get(id=appointment_id)
            print(update_appointment)
            update_appointment.status = 'pending'
            update_appointment.save()
            pay_amt= payment.amount
            payee = update_appointment.client.name
            email = update_appointment.client.email
            ap_date=update_appointment.date
            ap_time=update_appointment.time_slot
            therapist = update_appointment.therapist.name
            appointment_email(email,  payee, ap_date, ap_time, pay_amt,therapist,payment)
        except Appointment.DoesNotExist:
            # Handle the case where the appointment with the given ID does not exist
            return HttpResponseBadRequest("Invalid appointment ID")

        # Render the success page on successful capture of payment.
        return render(request, 'client/payment_confirmation.html',{'appointment':update_appointment})

    else:
        update_appointment = Appointment.objects.get(id=appointment_id)
        update_appointment.payment_status = False
        update_appointment.save()

        # If other than POST request is made.
        return HttpResponseBadRequest()
```

9.1 Screen Shots

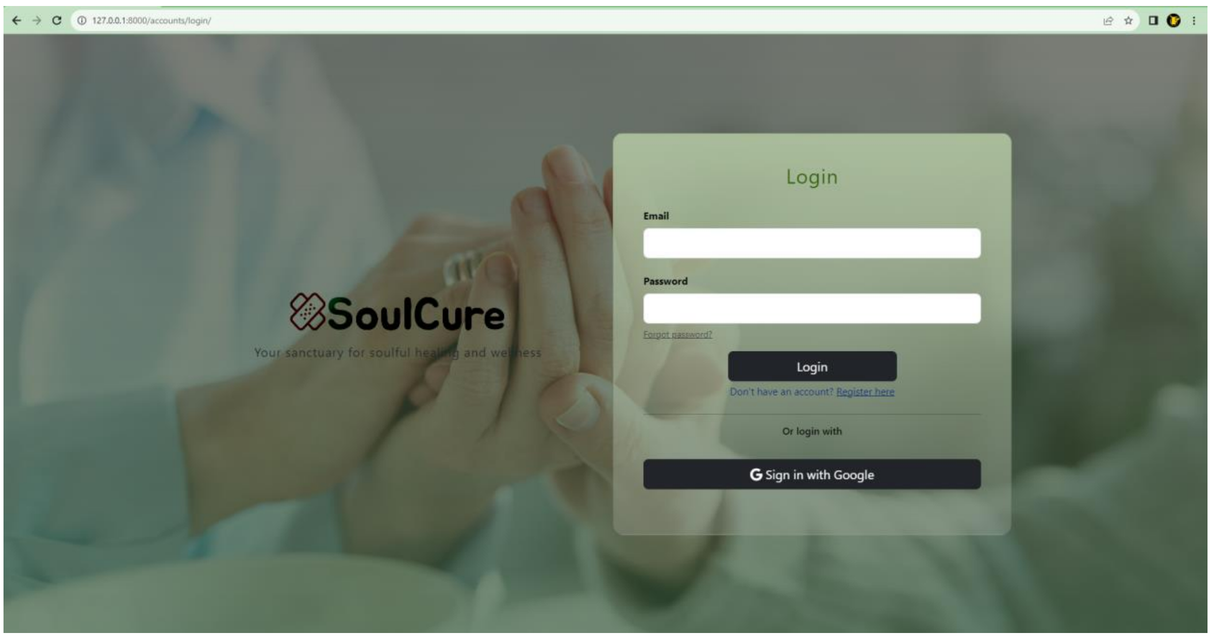
Landing Page



Registration Page



Login Page


127.0.0.1:8000/accounts/login/

SoulCure

Your sanctuary for soulful healing and wellness

Login

Email

Password

Forgot password?

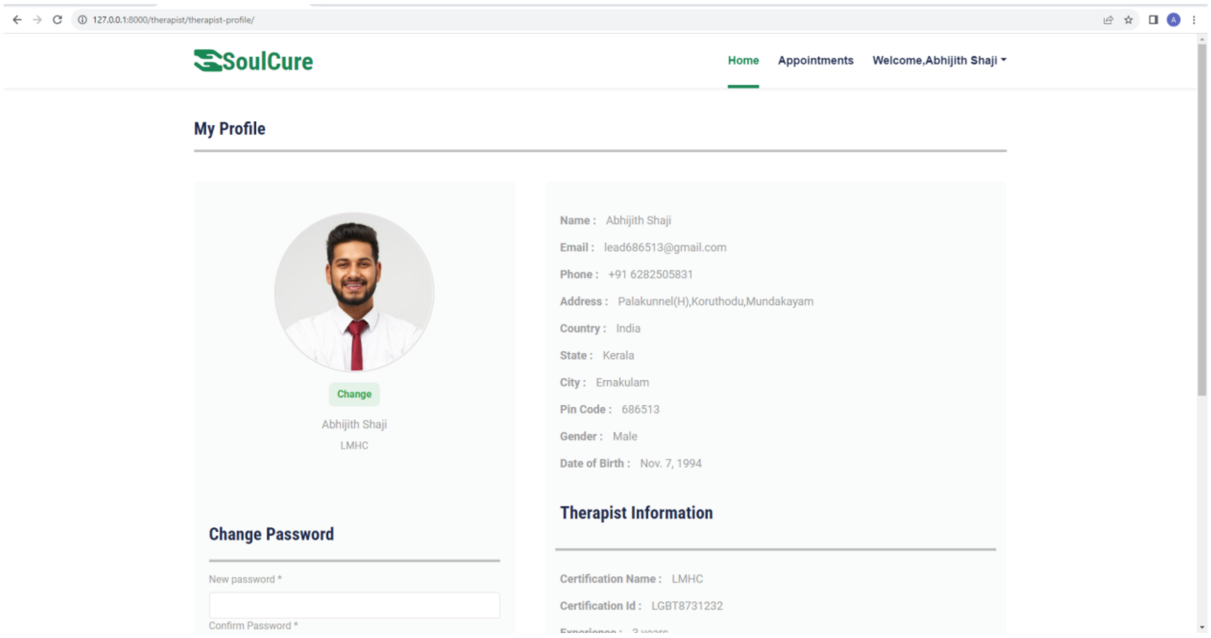
Login

Don't have an account? Register here

Or login with

Sign in with Google

Profile Page


127.0.0.1:8000/therapist/therapist-profile/

SoulCure

Home Appointments Welcome,Abhijith Shaji

My Profile

Change

Abhijith Shaji

LMHC

Name : Abhijith Shaji

Email : lead686513@gmail.com

Phone : +91 6282505831

Address : Palakunnel(H),Koruthodu,Mundakayam

Country : India

State : Kerala

City : Ernakulam

Pin Code : 686513

Gender : Male

Date of Birth : Nov. 7, 1994

Change Password

New password *

Confirm Password *

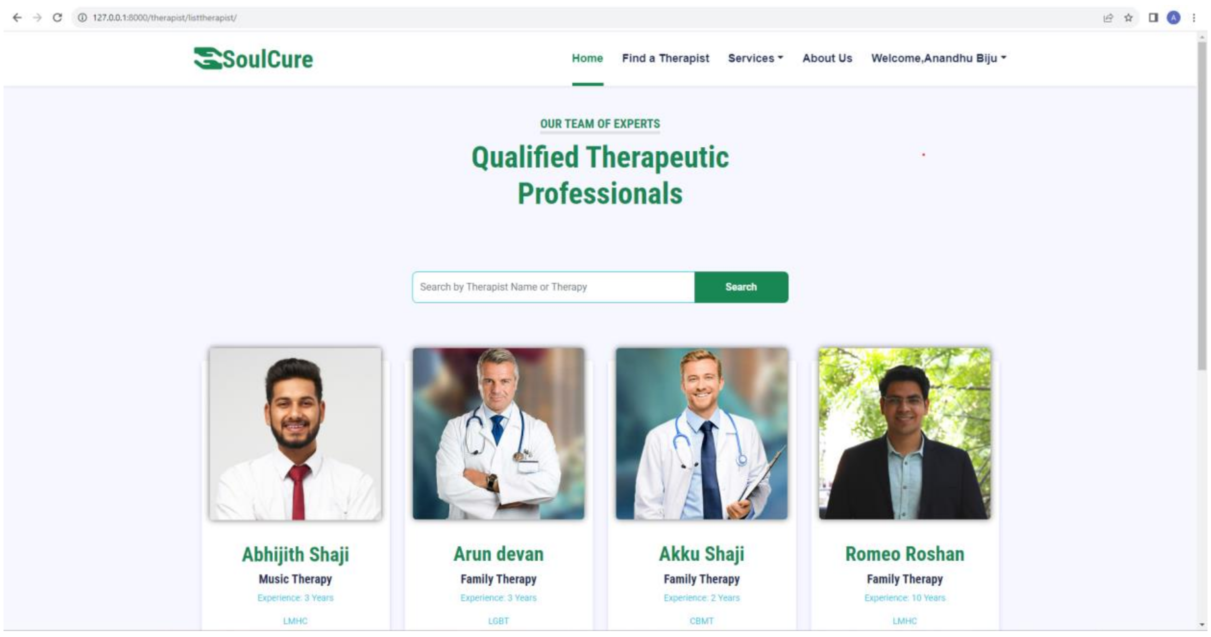
Therapist Information

Certification Name : LMHC

Certification Id : LGBT8731232

Experience : 3 years

View Therapists Page


127.0.0.1:8000/therapist/listtherapist/
SoulCure
Home
Find a Therapist
Services
About Us
Welcome,Anandhu Biju
Search by Therapist Name or Therapy
Search

OUR TEAM OF EXPERTS

Qualified Therapeutic Professionals

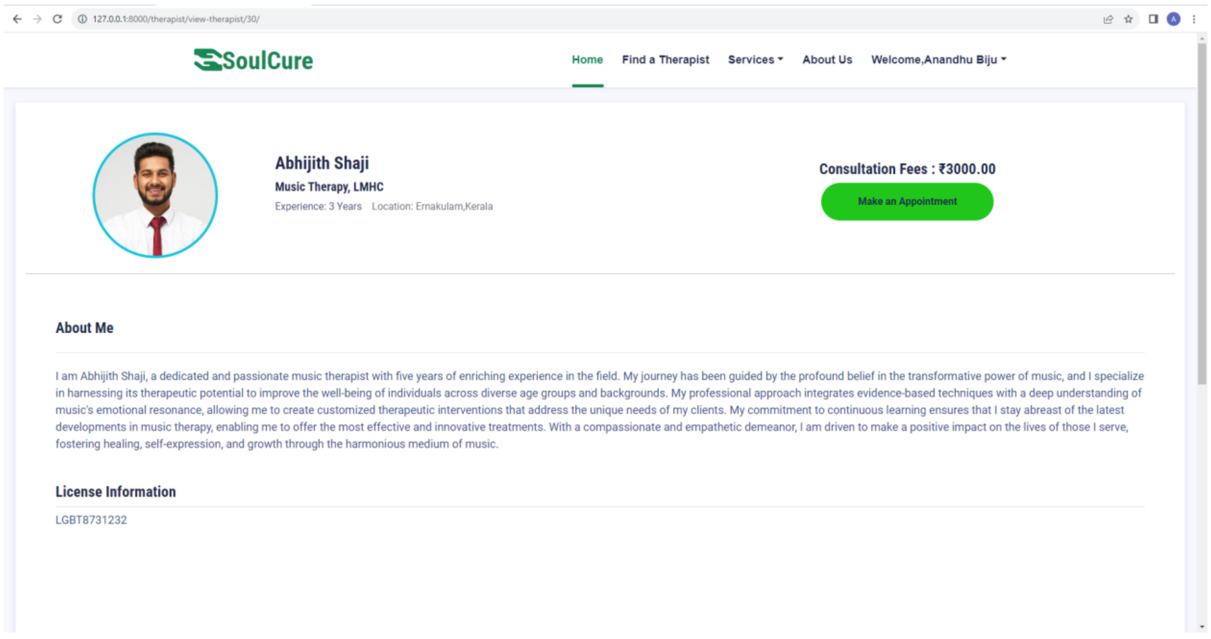
Abhijith Shaji
Music Therapy
Experience: 3 Years
LMHC

Arun devan
Family Therapy
Experience: 3 Years
LGBT

Akku Shaji
Family Therapy
Experience: 2 Years
CBMT

Romeo Roshan
Family Therapy
Experience: 10 Years
LMHC

Single Therapist Page



Abhijith Shaji
Music Therapy, LMHC
Experience: 3 Years Location: Ernakulam,Kerala

Consultation Fees : ₹3000.00

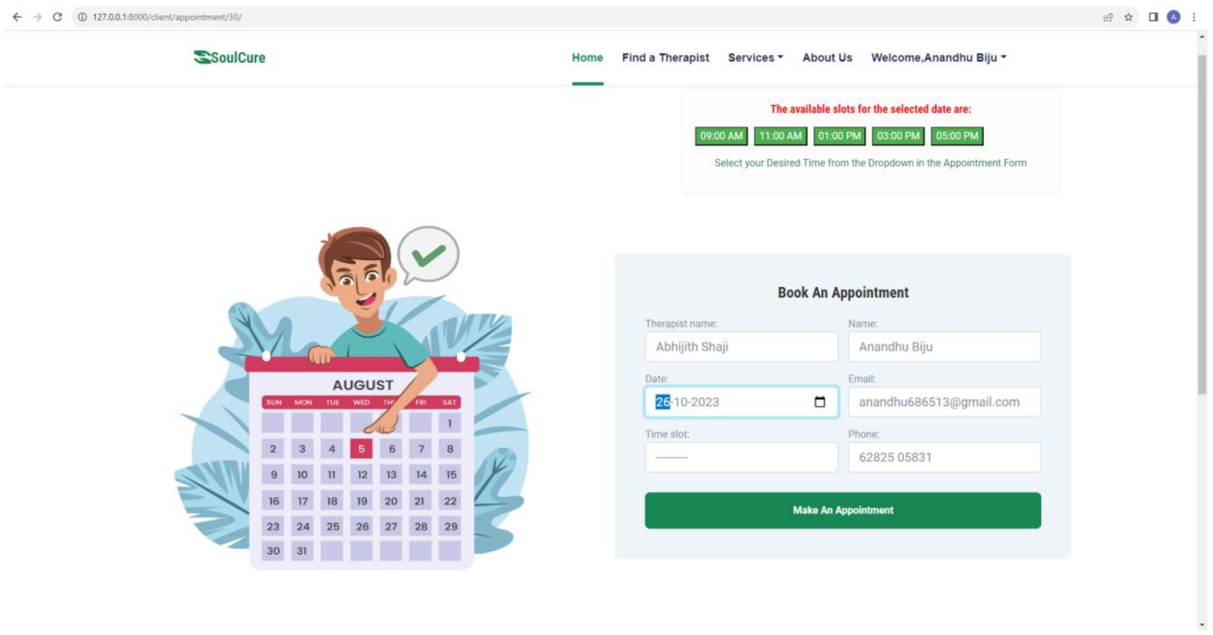
About Me

I am Abhijith Shaji, a dedicated and passionate music therapist with five years of enriching experience in the field. My journey has been guided by the profound belief in the transformative power of music, and I specialize in harnessing its therapeutic potential to improve the well-being of individuals across diverse age groups and backgrounds. My professional approach integrates evidence-based techniques with a deep understanding of music's emotional resonance, allowing me to create customized therapeutic interventions that address the unique needs of my clients. My commitment to continuous learning ensures that I stay abreast of the latest developments in music therapy, enabling me to offer the most effective and innovative treatments. With a compassionate and empathetic demeanor, I am driven to make a positive impact on the lives of those I serve, fostering healing, self-expression, and growth through the harmonious medium of music.

License Information

LGBT8731232

Book Appointment Page



The available slots for the selected date are:

09:00 AM 11:00 AM 01:00 PM 03:00 PM 05:00 PM

Select your Desired Time from the Dropdown in the Appointment Form

Book An Appointment

Therapist name: Abhijith Shaji	Name: Anandhu Biju
Date: 26-10-2023	Email: anandhu686513@gmail.com
Time slot: ---------	Phone: 62825 05831

Make An Appointment