Project Report on

SMART HEALTHCARE MANAGEMENT WEBSITE

Submitted in partial fulfilment of the requirement for the award of the

Degree of

BACHELOR OF COMPUTER APPLICATION

in

COMPUTER APPLICATION

by

Aastha Singh(221117106002)

Abhismita Nayak(221117106013)

Aditi Kumari (221117106014)

Under the Guidance of

Papita Chaudhary

(Assistant Professor, GLBIM.)

GL BAJAJ
Institute of Management
FIND YOUR SPARK
Affiliated to CCS University, Meerut.

G.L. BAJAJ INSTITUTE OF MANAGEMENT

(**CHAUDHARY CHARAN SINGH UNIVERSITY, MEERUT)**

DEC, 2024

DECLARATION

I declare that the work presented in this report titled "**Smart healthcare management website**" submitted to the Department of **COMPUTER APPLICATION**, **G. L. BAJAJ INSTITUTE OF MANAGEMENT,** Greater Noida, for the Bachelor of Computer Application in **Computer APPLICATION** is our original work. I have not plagiarized unless cited or the same report has not submitted anywhere for the award of any other degree. I understand that any violation of the above will be cause for disciplinary action by the university against me as per the University rule

Place:NOIDA **Signature of the Student**

Date: **ADITI KUMARI (221117106014)**

AASTHA SINGH (221117106002)

ABHISMITA NAYAK (221117106013)

GL BAJAJ
Institute of Management
FIND YOUR SPARK
Affiliated to CCS University, Meerut.

G. L. BAJAJ INSTITUTE OF MANAGEMENT

CERTIFICATE

This is to certify that the project titled “**Smart healthcare management website** ’’ is under my supervision. As per best of my knowledge this project work is not submitted or published anywhere else carried out **by Aditi kumari , Aastha Singh , Abhismita Nayak** during the academic year 2023-24. We approve this project for submission in partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Application in, **CHAUDHARY CHARAN SINGH UNIVERSITY, MEERUT.** I wish him/her all the best for his/her bright future ahead.

The Project is Satisfactory / Unsatisfactory.

Papita Chaudhary	Approved by
(Assistant professor)	Dr. Mukul Gupta
	Principal, GLBIM

ACKNOWLEDGEMENT

I am grateful to The Department of Computer Application, for giving me the opportunity to carry out this project, which is an integral fragment of the curriculum in Bachelor of Computer Application program at the **CHAUDHARY CHARAN SINGH UNIVERSITY, MEERUT**. I would like to express our heartfelt gratitude and regards to my project guide, Papita Chaudhary, Assistant Professor, **G.L.BAJAJ INSTITUTE OF MANAGEMENT,** for his/her unflagging support and continuous encouragement throughout the project. Special thanks to our **Principal Dr. Mukul Gupta, G. L. BAJAJ INSTITUTE OF MANAGEMENT.** I am also obliged to the staff of **G.L.BAJAJ INSTITUTE OF MANAGEMENT** for aiding me during the course of our project. I offer my heartiest thanks to my friends for their help in collection of data samples whenever necessary. Last but not the least; I want to acknowledge the contributions of my parents and family members, for their constant and never-ending motivation.

Signature of the Student

ADITI KUMARI (221117106014)

AASTHA SINGH (221117106002)

ABHISMITA NAYAK (221117106013)

TABLE OF CONTENTS

Abstract

Wellnessify is a cutting-edge health management platform designed to empower individuals in managing their health effectively and holistically. In an era where access to affordable and efficient healthcare solutions remains a global challenge, Wellnessify bridges the gap by offering an innovative, usercentric approach to personal healthmanagement. The platform allows users to input their symptoms, access a curated database of homeremedies derived from scientific research and naturopathic traditions, and connect with local medical professionals when necessary.

The unique value proposition of Wellnessify lies in its focus on non-pharmaceutical interventions. By prioritizing natural, affordable, and accessible solutions, the platform addresses the growing demand for holistic health alternatives. Beyond remedy recommendations, Wellnessify utilizes advanced algorithms to ensure that users receive highly personalized suggestions tailored to their specific conditions. Moreover, its integration with local healthcare services provides a seamless transition from selfcare to professional care, ensuring comprehensive health support.

Wellnessify's mission extends beyond symptom management; it strives to foster a healthier, more informed society by combining the best of modern technology and natural medicine. With its intuitive design, robust database, and innovative features, Wellnessify is redefining how individuals approach everyday health challenges, empowering them to take proactive control of their well-being.

Beyond symptom management, Wellnessify aims to cultivate a healthier, more informed society by merging cutting-edge technology with the principles of natural medicine. Its intuitive design and user-friendly interface make it accessible to all, while its robust database ensures reliability and depth. By redefining how individuals approach everyday health challenges, Wellnessify empowers users to proactively manage their health, paving the way for a future where well-being is accessible to all.

The cornerstone of Wellnessify’s value lies in its emphasis on non-pharmaceutical interventions. In an age where many seek natural and affordable alternatives to synthetic medications, Wellnessify delivers personalized, evidence-based recommendations tailored to each user’s unique conditions and preferences. This personalization is powered by advanced algorithms that analyze user data to ensure precise and effective solutions. The platform’s commitment to holistic health doesn’t end with remedy suggestions; it also integrates with local healthcare services, bridging the gap between self-care and professional medical attention. The platform's broader vision is to promote a global culture of wellness, where individuals are equipped with the knowledge and tools to address everyday health concerns without unnecessary reliance on pharmaceuticals. Wellnessify stands as a testament to the power of combining tradition and technology, offering a modern solution to age-old health challenges. It’s not just about managing symptoms; it’s about transforming lives and creating healthier, more resilient communities. Through its innovative features and user-focused philosophy.

INTRODUCTION

Health and wellness are essential factors of leading a fulfilling life, yet millions face barriers to accessing adequate healthcare for minor ailments. The over-reliance on pharmaceuticals and limited awareness of natural, non-invasive alternatives has created a void in the healthcare landscape. Wellnessify was conceived to fill this gap, offering a revolutionary platform that combines the accessibility of technology with the efficiency of natural medicine.

At its core, Wellnessify is designed to simplify the process of managing minor health issues. Users can easily input their symptoms into the platform and receive personalized remedies derived from an extensive, scientifically validated database. These remedies focus on promoting natural healing, avoiding the side effects and costs often associated with pharmaceutical treatments. Additionally, recognizing that some conditions require professional intervention, Wellnessify integrates local healthcare service providers into its ecosystem, ensuring users have access to the care they need when necessary.

The inspiration behind Wellnessify came from the increasing global awareness of the benefits of holistic health approaches. By leveraging advanced symptom matching algorithms, the platform ensures that users receive accurate and effective recommendations tailored to their unique needs. Its intuitive interface is designed for users of all ages and technical proficiencies, making healthcare more accessible than ever before.

One of Wellnessify's standout features is its commitment to education. The platform doesn't merely provide remedies; it actively seeks to enhance users' understanding of their health. Each recommendation comes with detailed explanations, offering insights into how and why a remedy works, as well as lifestyle tips for long-term well-being. This focus on education bridges the gap between self-care and preventive health, fostering a culture where individuals feel confident in managing their own health.

Moreover, Wellnessify's integration of technology goes beyond basic symptom analysis. Its algorithms are designed to continuously learn and improve, ensuring that users receive increasingly accurate and relevant suggestions. This adaptability allows the platform to cater to diverse demographics, addressing varying health concerns based on geography, age, lifestyle, and cultural practices.Users can transition seamlessly from at-home remedies to professional interventions, ensuring comprehensive support for all health concerns.

Wellnessify's vision extends into the realm of global health equity. By prioritizing natural, cost-effective solutions, the platform addresses the needs of underserved populations, making healthcare accessible to those who might otherwise struggle to afford it. Its mission to promote sustainable practices aligns with the growing global emphasis on environmentally friendly healthcare, reducing reliance on over-processed pharmaceuticals and their associated ecological impact.

Ultimately, Wellnessify is more than a platform; it is a catalyst for change. By blending science, tradition, and innovation, it empowers individuals to take ownership of their health, paving the way for a future where wellness is not a privilege but a fundamental right for all.

Literature Review

- User-Centric Design and Accessibility The design and accessibility of smart healthcare websites are critical for their success. Lim et al. (2019) discussed the integration of wearable technologies in healthcare, suggesting that user interaction with health technologies must be optimized for better health outcomes. This is particularly important for websites that provide health information and remedies, as users may have varying levels of health literacy.
- Nutritional Benefits of Recommended Remedies Natural remedies often possess nutritional benefits that can contribute to overall health. For instance, turmeric, a common ingredient in natural remedies, has been widely studied for its bioactive properties. Xu et al. (2018) reviewed the health benefits of curcumin, a principal compound in turmeric, highlighting its anti-inflammatory and antioxidant effects. This underscores the importance of providing users with detailed information about the ingredients in recommended remedies, including their health benefits, preparation instructions, and any contraindications.
- Symptom Assessment and Natural Remedies Research indicates that symptom assessment is crucial for providing accurate health advice. Wu et al. (2018) developed SymMap, an integrative database that enhances traditional Chinese medicine through symptom mapping, demonstrating the potential of aligning symptoms with natural remedies. The findings suggest that a structured approach to symptom input can facilitate the identification of suitable natural treatments, such as turmeric milk for common ailments like colds (Wu et al., 2018).
- E-healtheare: an analysis of key themes in research

Avinandan Mukherjee, John McGinnis -International Journal of Pharmaceutical and Healtheare Marketing 2007 Purpose - Healthcare is among the fastest-growing sector in both developed and emerging economies. E-healtheare is contributing to the explosive growth within this industry by utilizing the internet and all its capabilities to support its stakeholders with information searches and communication processes. The purpose of this paper is to present the state-of-the-art and to identify key themes in research on e-healthcare. Design/methodology/approach - A review of the literature in the marketing and management of e-healtheare was conducted to determine the major themes pertinent to e-healtheare research as well as the commonalities and differences within these themes. Findings - Based on the literature review, the five major themes of e-healtheare research identified are: cost savings; virtual networking; electronic medical records; source credibility and privacy concerns; and physician-patient relationships. Originality/value Based on these major themes, managerial implications for e-healtheare are formulated. Suggestions are offered to facilitate healthcare service organizations' attempts to further implement and properly utilize e-healtheare in their facilities. These propositions will also help these stakeholders develop and streamline their e-healtheare processes already in use. E-healtheare systems enable firms to improve efficiency, to reduce costs, and to facilitate the coordination of care across multiple facilities.

Objective

In an era marked by the over-reliance on pharmaceuticals and fragmented healthcare systems, Wellnessify offers a transformative vision for personal health management. Its mission is grounded in the principles of holistic healing, accessibility, and technological innovation, making it a platform that redefines how individuals approach health and wellness. Central to this mission are six key goals that drive its development and impact:

1. Harness Natural Forces

Wellnessify prioritizes the use of natural, scientifically validated remedies to address minor health concerns. This approach aims to reduce dependency on pharmaceutical drugs, which often carry side effects and contribute to the growing issue of drug overuse. By leveraging natural forces and traditional healing methods, the platform provides safe, effective, and accessible alternatives for users. Remedies are chosen based on rigorous research, ensuring that users benefit from solutions that are both gentle and impactful. This focus on natural medicine empowers users to address their health concerns without resorting to "hard-hitting" drugs unless absolutely necessary.

2. Improve Accidental Inclination Towards Healthcare

Health management should not require advanced technical knowledge or specialized training. Wellnessify’s advanced symptom-matching algorithms and user-friendly interface ensure that anyone, regardless of their technical proficiency, can easily navigate the platform and access personalized remedies. By making the process seamless, the platform bridges the gap between individuals and effective health solutions, encouraging even those who might not typically seek healthcare to take charge of their well-being.

3. Encourage Both Curative and Preventive Strategies

Wellnessify emphasizes a dual approach to health management: curative and preventive care. Users are guided to address health concerns early using mild, natural remedies, thereby preventing escalation to severe conditions. Simultaneously, the platform educates users on preventive strategies, such as lifestyle changes, dietary adjustments, and stress management techniques. This dual strategy not only resolves immediate issues but also fosters long-term health resilience, encouraging users to be proactive in maintaining their well-being.

4. Encourage Informed Self-Care

Education is a cornerstone of Wellnessify’s philosophy. The platform provides users with detailed explanations for each remedy and its benefits, fostering a deeper understanding of health and wellness. This approach empowers users to make informed decisions about their health, cultivating personal responsibility and awareness. By building knowledge and encouraging positive attitudes toward health, Wellnessify promotes a culture where self-care becomes a natural and sustainable habit.

Exploring Data

Data set created manually.

Remedies

_id: ObjectId('6746366c6fb8d283141e88dc')

name: "Ginger Tea" ingredients:

Array (4)

instructions: "Boil water, add grated ginger, simmer for 10 minutes, strain, and add"

contraindications: "Avoid if you have gallstones or are allergic to ginger."

symptoms:

Array (3)

posts

> Open MongoDB shell

Explain

Reset

Find

<>

Options ▸

25 1-13 of 13

>

≡ {}

remedies

symptoms

users

_id: ObjectId('674636fa9586755d782e7116') name: "Turmeric Milk"

ingredients: Array (3)

instructions: "Warm milk, add turmeric powder and honey, stir well and serve."

contraindications: "Avoid if you have gallbladder issues or are taking blood thinning medi.."

symptoms: Array (3)

59

Q Search

_id: ObjectId('6746373995807550782671l8')

name: "Peppermint Tea"

ingredients: Array (3)

instructions: "Boil water, add peppermint leaves, steep for 5-7 minutes, strain, and.." contraindications: "Avoid if you have acid reflux or peppermint allergies."

_id: ObjectId('6746366c6fb8d283141e88dc')

name: "Ginger Tea" ingredients:

Array (4)

instructions: "Boil water, add grated ginger, simmer for 10 minutes, strain, and add"

contraindications: "Avoid if you have gallstones or are allergic to ginger."

symptoms:

Array (3)

posts

> Open MongoDB shell

Explain

Reset

Find

<>

Options ▸

25 1-13 of 13

>

≡ {}

remedies

symptoms

users

_id: ObjectId('674636fa9586755d782e7116')

name: "Turmeric Milk"

ingredients: Array (3)

instructions: "Warm milk, add turmeric powder and honey, stir well and serve."

contraindications: "Avoid if you have gallbladder issues or are taking blood thinning medi.."

symptoms: Array (3)

59

Q Search

_id: ObjectId('674637399580755078267118')

name: "Peppermint Tea"

ingredients: Array (3)

instructions: "Boil water, add peppermint leaves, steep for 5-7 minutes, strain, and.." contraindications: "Avoid if you have acid reflux or peppermint allergies."

id: ObjectId('674637399580755078267116')

name: "Peppermint Tea"

ingredients: Array (3)

> Open MongoDB shell

Explain

Reset

Find

<>

Options

25

1-13 of 13

instructions: "Boil water, add peppermint leaves, steep for 5-7 minutes, strain, and "

contraindications: "Avoid if you have acid reflux or peppermint allergies."

symptoms:

Array (3)

_id: ObjectId('67463776958075507826711a')

name: "Lemon Honey Water"

ingredients: Array (3)

instructions: "Mix fresh lemon juice and honey into warm water, stir well and drink."

contraindications: "Avoid if you have citrus allergies or diabetes (monitor honey consumpt"

symptoms

: Array (3)

=

{}

田

59°

Q Search

_id: ObjectId('6746379c95807550782e711c')

name: "Garlic Water"

ingredients: Array (2)

instructions: "Crush garlic cloves, boil in water for 10 minutes, strain and serve wa..."

contraindications: "Avoid if you have low blood pressure or are allergic to garlic."

ingredients:

Array (2)

instructions: "Crush garlic cloves, boil in water for 16 minutes, strain and serve wa..."

contraindications: "Avoid if you have low blood pressure or are allergic to garlic."

symptoms:

Array (3)

_id: ObjectId('674637c49580755d78207110")

name: "Honey and Cinnamon"

ingredients: Array (2)

instructions: "Mix 1 tablespoon of honey with 1/2 teaspoon of cinnamon powder and con..."

contraindications: "Avoid if allergic to honey or cinnamon."

symptoms Array (3)

_id: name: "

ObjectId('674637ef95880755d78267129) Saltwater Gargle"

ingredients: Array (2)

instructions: "Dissolve 1 teaspoon of salt in warm water and gargle for 30 seconds, t.."

contraindications: "Avoid swallowing the saltwater solution."

symptoms:

Array (2)

name: "Ginger Compress"

ingredients: Array (3)

instructions: "Grate ginger, steep in warm water for 5 minutes, soak a cloth in the s.."

contraindications: "Avoid if skin irritation occurs."

symptoms:

Array (2)

_id

: ObjectId('6746390695867550762e712a')

name: "Aloe Vera Juice"

ingredients:

Array (3)

instructions: "Extract fresh aloe vera gel, blend with water and honey, and consume o."

contraindications: "Avoid if pregnant, breastfeeding, or allergic to aloe vera."

symptoms:

Array (3)

_id: ObjectId(+674639379566755d782e712c')

name: "Clove 011"

ingredients:

Array (2)

instructions: "Dip a cotton ball in clove oil and apply gently to the affected area.

contraindications: "Avoid ingesting clove oil directly."

‣ symptoms: Array (2)

Symptoms

name: "Ginger Compress"

ingredients: Array (3)

instructions: "Grate ginger, steep in warm water for 5 minutes, soak a cloth in the s.."

contraindications: "Avoid if skin irritation occurs."

symptoms:

Array (2)

_id

: ObjectId('6746390695867550762e712a')

name: "Aloe Vera Juice"

ingredients:

Array (3)

instructions: "Extract fresh aloe vera gel, blend with water and honey, and consume o."

contraindications: "Avoid if pregnant, breastfeeding, or allergic to aloe vera."

symptoms:

Array (3)

_id: ObjectId(+674639379566755d782e712c')

name: "Clove 011"

ingredients:

Array (2)

instructions: "Dip a cotton ball in clove oil and apply gently to the affected area.

contraindications: "Avoid ingesting clove oil directly."

‣ symptoms: Array (2)

name: "cough"

dateEntered: 2024-11-26T19:51:49.956+00:00 V

practice

comments

posts

remedies

symptoms

users

_id: ObjectId('674626d66fb8d283141e88891)

name: "sore throat!

dateEntered: 2024-11-26T19:51:50.891+00:00

_id: ObjectId('674626d76fb8d283141e888c') name: "stomach ache"

dateEntered: 2024-11-26T19:51:51.537+00:00 v:e

Search

id: ObjectId('674626d76fb8d283141e888f')

name: "indigestion"

dateEntered: 2024-11-26T19:51:51.752+00:00 v:

users

_id: ObjectId('674626665fb8d283141e887a')

name: "isha"

email: "isha@gmail.com"

password: "$2a$10$XSc7PPq20ec3QdsX81aV8.87YP8EulfA1VyYF6LSgqF/bmPBHszyq"

createdAt: 2024-11-26T19:49:58.103+00:00

updatedAt: 2824-11-26T19:57:28.169+00:00

_id: ObjectId('675e82b03fdbb2fd6923adde')

name: "admin"

email: "admin@gmail.com"

password: "$2a$1@SPxusEXy2E1HpF7fzItPuduANCdpDUxeTRUFUNYJVhQ0F1F72.j.Gu"

role: "admin"

createdAt: 2024-12-15T07:18:08.527+00:00

updatedAt: 2024-12-15T07:18:08.527+00:00

V:

posts

id: ObjectId(1675ea04c3fdbb2fd0923ade8")

title: "Natural Remedies for Common Cold Relief" content: "The common cold, a viral infection affecting millions annually, often"

author: ObjectId('675e82b83fdbb2fd0923adde')

createdAt: 2024-12-15T09:24:27.980+00:00

updatedAt: 2025-01-10T13:11:27.228+00:00

remedies

symptoms

users

id: ObjectId('67811e2ef3982a64404935a7')

title: "Harnessing the Power of Exercise and Natural Remedies for Holistic Wel"

content:"

In the fast-paced world we live in, maintaining a balance between me.."

author: ObjectId('67811baaf3982a644d493599') createdAt: 2625-01-16T13:18:38.694+00:00

updatedAt: 2025-01-16T13:18:38.894+00:00

id: ObjectId('67812223f3982a644d4935af)

title: "Understanding the Importance of Sleep for Health"

id: ObjectId('6781224ff3982a6446493563")

title: "Superfoods to Include in Your Diet for Better Health"

content: "Superfoods are nutrient-dense ingredients that offer extraordinary hea.."

author: ObjectId('67811baaf39822644d0493599')

createdAt: 2025-01-16T13:36:15.650+00:00

updatedAt: 2025-01-16T13:36:15.650+00:00 V

id: ObjectId('67812275f3982a6446493567*)

title: "DIY Skin Care with Natural Ingredients: Face Masks and Scrubs" content: "Natural skin care is gaining popularity as a safe, cost-effective way" author: ObjectId('67811baaf3982a6440493599')

createdAt: 2025-01-10T13:36:53.264+00:00

updatedAt: 2025-01-10T13:36:53.264+00:00

_id: ObjectId('678122b7f3982a644d4935bb')

title: "How to Stay Active While Working a Desk Job"

content: "Sitting for prolonged hours at a desk can negatively impact your healt.."

1 _id: ObjectId('675ea04c3fdbb2fd6923adeß')

local

2 title: "Natural Remedies for Common Cold Relief

practice

3 content: "The common cold, a viral infection affecting millions annually, ofter

comments

posts

remedies

symptoms

1. Garlic: Known for its antimicrobial and antiviral properties, g

2. Honey: A time-honored remedy, honey is particularly effective ir

3. Herbal Ingredients: Ayurveda and traditional systems like Jamu r

4. Echinacea: This herbal supplement is widely used for cold prever

users

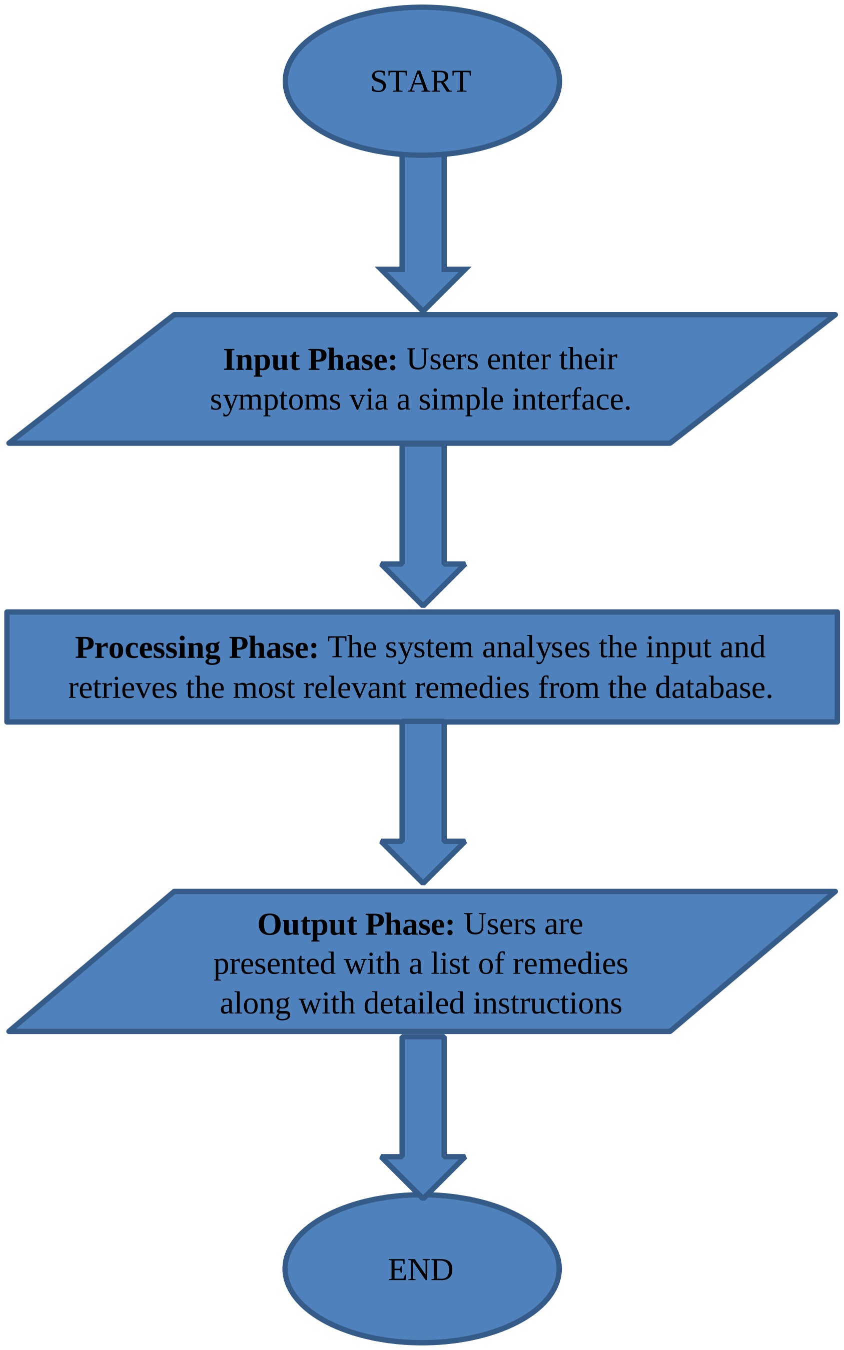
4 author: 675e82b03fdbb2fd0923adde

createdAt: 2024-12-15T09:24:27.980+00:00

6 updatedAt: 2025-01-10T13:11:27.228+00:00

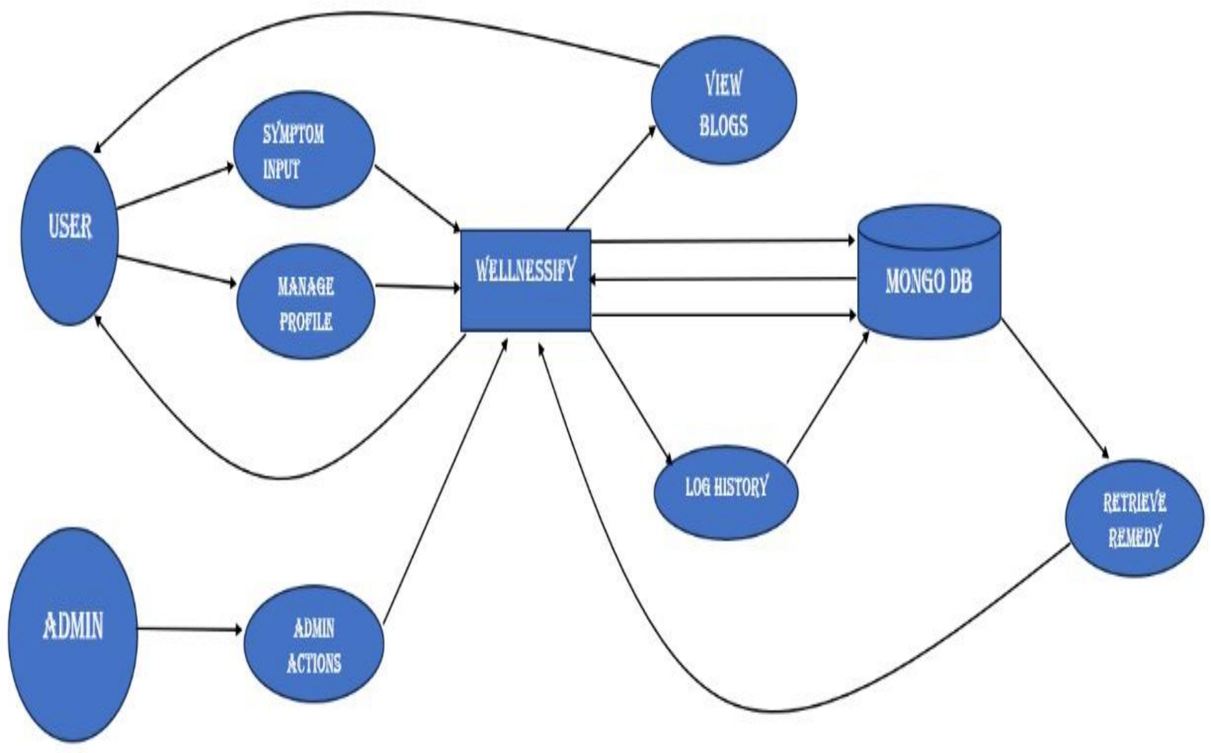
Flowchart

The process flow of Wellnessify is designed to optimize user interaction:

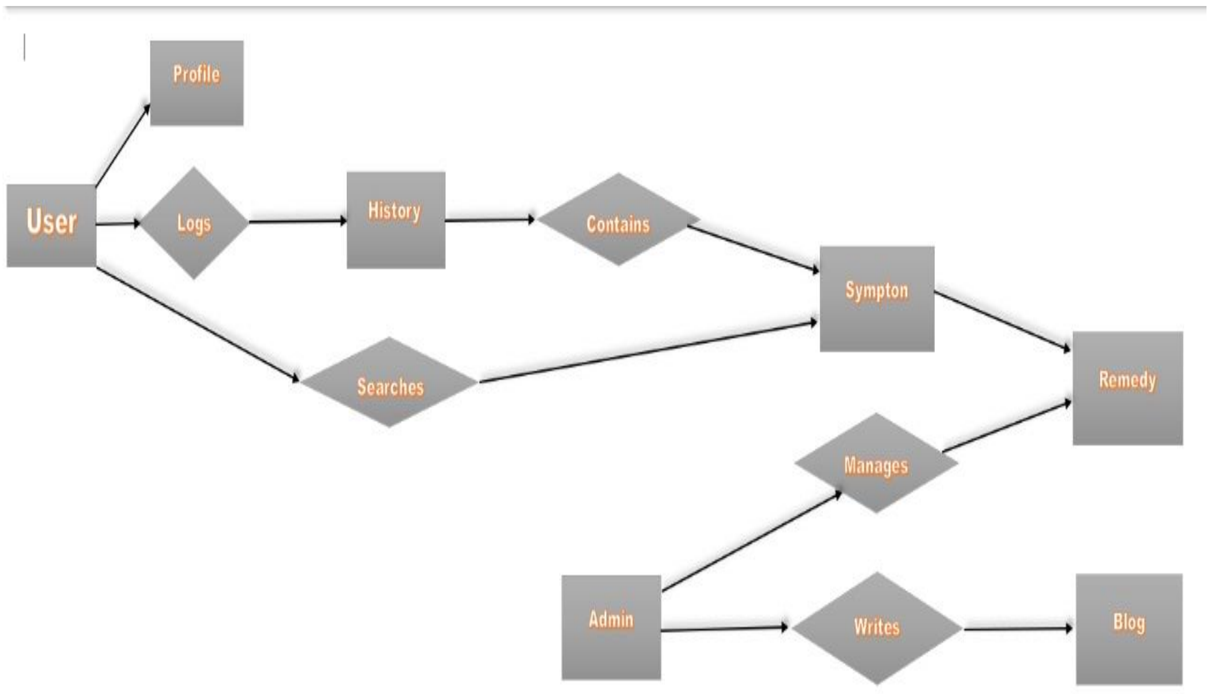

START
Input Phase: Users enter their symptoms via a simple interface.
Processing Phase: The system analyses the input and retrieves the most relevant remedies from the database.
Output Phase: Users are presented with a list of remedies along with detailed instructions
END

ER diagram

The Entity-Relationship (ER) diagram highlights the relationships between Wellnessify's core components:



DFD (Level 1, 2, and 3)



Source Code

Wellnessify's codebase is modular and scalable, with the following key components:

1. **Frontend:** Built using HTML, CSS, and JavaScript to provide an interactive and responsive user interface.

aboutUs.jsx

```
import React from 'react';

import bgImage from 'C:/project2/Willnessly-Project/Frontend/src/assets/aboutus.png'; // Adjust the path to your image file



const AboutUs = () => {

  return (

    <div style={{...styles.pageContainer, backgroundImage: url(${bgImage})}}>




      <div style={styles.content}>

        {/* About Wellnessify Section */}

        <section style={styles.section}>

          <h2 style={styles.sectionTitle}>About Wellnessify</h2>

          <p style={styles.sectionText}>

            Wellnessify is a platform dedicated to promoting holistic health through personalized natural remedies. We aim to empower individuals to take control of their health and well-being using safe and natural solutions.

          </p>

        </section>
```

```
{/* Our Mission Section */}

<section style={styles.section}>

  <h2 style={styles.sectionTitle}>Our Mission</h2>

  <p style={styles.sectionText}>

    Wellnessify is dedicated to providing personalized, natural health remedies to help people live healthier lives. We believe in the power of nature and aim to make holistic wellness accessible to everyone.

  </p>

</section>



{/* Our Vision Section */}

<section style={styles.section}>

  <h2 style={styles.sectionTitle}>Our Vision</h2>

  <p style={styles.sectionText}>

    Our vision is to become a trusted platform for natural healthcare, empowering individuals to take control of their well-being with safe, natural solutions. We strive to bridge the gap between modern healthcare and traditional natural remedies.

  </p>

</section>



{/* Our Team Section */}

<section style={styles.section}>

  <h2 style={styles.sectionTitle}>Our Team</h2>

  <p style={styles.sectionText}>
```

```
            We are a group of healthcare enthusiasts, naturopaths, and tech professionals committed to providing reliable, research-based natural remedies. Our team is passionate about promoting a healthier world through nature.
          </p>
        </section>
      </div>
    </div>
  );
};


// Styles for a light, health-themed color palette
const styles = {
  pageContainer: {
    fontFamily: "'Arial', sans-serif",
    margin: 0,
    padding: 0,
    backgroundColor: '#e3f8f4', // Soft Teal background as fallback
    backgroundSize: 'cover', // Ensures the background image covers the full viewport
    backgroundPosition: 'center', // Centers the image
    backgroundRepeat: 'no-repeat', // Prevents the background image from repeating
    minHeight: '100vh', // Full height of the screen
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
    justifyContent: 'flex-start', // Keeps content at the top
  },
  header: {
    backgroundColor: '#f8f9fa', // White Smoke background for header
    padding: '20px',
    textAlign: 'center',
    width: '100%',
    boxShadow: '0 2px 5px rgba(0, 0, 0, 0.1)',
  },
  headerTitle: {
    fontFamily: "'Pacifico', cursive",
    margin: '0',
    fontSize: '2.8rem',
    color: '#8fd3f4', // Sky Blue for header title
  },
  content: {
    maxWidth: '800px',
    margin: '40px auto',
    padding: '30px',
    backgroundColor: 'rgba(255, 255, 255, 0.8)', // White background with opacity to allow
background image to show
    boxShadow: '0 4px 8px rgba(0, 0, 0, 0.1)',
    borderRadius: '8px',
  },
  section: {
    marginBottom: '30px',
    paddingBottom: '20px',
    borderBottom: '1px solid #f0f0f0', // Adds a subtle separation line between sections
  },
  sectionTitle: {
    textAlign: 'center',
    color: '#333333', // Dark Slate for section titles
    fontSize: '2rem',
    marginBottom: '10px', // Adds space below the title
  },
  sectionText: {
    fontSize: '1.1rem',
    color: '#5a5a5a', // Medium Gray for section text
    lineHeight: '1.8', // More comfortable line-height for better readability
    textAlign: 'justify', // Justifies the text for a clean, professional look
  },
};

export default AboutUs;
```

EditProfile.js

```
// src/components/EditProfile.js


import React, { useEffect, useState } from 'react';

import styled from 'styled-components';


const FormContainer = styled.div`

  max-width: 400px;

  margin: 20px auto;

  padding: 20px;

  background: #fff;

  border-radius: 10px;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

`;


const Input = styled.input`

  width: 100%;

  padding: 10px;

  margin: 10px 0;

  border: 1px solid #ccc;

  border-radius: 5px;

`;
```

```
const Button = styled.button`
  padding: 10px;
  width: 100%;
  background: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;

  &:hover {
    background: #0056b3;
  }
`;

const EditProfile = () => {
  const [user, setUser] = useState({ name: '', email: '' });
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchProfile = async () => {
      const token = localStorage.getItem('token'); // Get the token from local storage

      try {
            const response = await fetch('http://localhost:5000/api/auth/profile', {
                method: 'POST',
                headers: {
                    'Authorization': Bearer ${token}, // Include the token in the request
                },
            });

            if (!response.ok) {
                const errorData = await response.text(); // Get the error response as text
                throw new Error(errorData || 'Error fetching profile'); // Throw an error with the
response text
            }

            const data = await response.json(); // Parse JSON response
            setUser(data.user); // Set user data
        } catch (err) {
            setError(err.message); // Set error message
        }
    };

    fetchProfile(); // Call the function to fetch the profile
}, []); // Empty dependency array means this effect runs once when the component mounts

const handleChange = (e) => {
    const { name, value } = e.target;

    setUser({ ...user, [name]: value }); // Update user state

  };


  const handleSubmit = async (e) => {

    e.preventDefault(); // Prevent page reload

    const token = localStorage.getItem('token');


    try {

      const response = await fetch('http://localhost:5000/api/auth/profile', { // Use the correct endpoint

        method: 'PUT', // Update method

        headers: {

          'Authorization': Bearer ${token},

          'Content-Type': 'application/json',

        },

        body: JSON.stringify(user), // Send updated user data

      });


      if (!response.ok) {

        const errorData = await response.text();

        throw new Error(errorData || 'Error updating profile');

      }
```

```
        alert('Profile updated successfully!'); // Success message
    } catch (err) {
        setError(err.message);
    }
};


return (
    <FormContainer>
        <h1>Edit Profile</h1>
        {error && <div style={{ color: 'red' }}>Error: {error}</div>}
        <form onSubmit={handleSubmit}>
            <Input
                type="text"
                name="name"
                value={user.name}
                onChange={handleChange}
                placeholder="Name"
                required
            />
            <Input
                type="email"
                name="email"
```

```
          value={user.email}
          onChange={handleChange}
          placeholder="Email"
          required
        />
        <Button type="submit">Save Changes</Button>
      </form>
    </FormContainer>
  );
};

export default EditProfile;
```

HomePage.jsx

```
// HomePage.jsx
import React from 'react';
import { useNavigate } from 'react-router-dom';
import bgImage from 'C:/project2/Willnessly-Project/Frontend/src/assets/bg1.jpeg';
const HomePage = () => {
  const navigate = useNavigate();

  const handleSubmit = (event) => {
    event.preventDefault();
  // Check if the user is logged in by verifying the token
  const token = localStorage.getItem('token');

  if (token) {
    // User is logged in; navigate to the remedies page
    navigate('/remedies');
  } else {
    // User is not logged in; navigate to the login page
    navigate('/login');
  }
};

return (
  <>
    {/* <Navbar /> */}
    <div style={styles.pageContainer}>







      <main style={styles.mainContent}>
```

```
          <div style={styles.formContainer}>
            <h2 style={styles.heading}>Find the Right Remedy</h2>
            <p style={styles.description}>Describe your symptoms, and we’ll suggest suitable
remedies.</p>
            <form id="symptom-form" onSubmit={handleSubmit} style={styles.form}>
              <input
                type="text"
                id="symptoms"
                name="symptoms"
                placeholder="e.g., headache, fatigue"
                style={styles.input}
              />
              <button type="submit" style={styles.button}>Get Remedies</button>
            </form>
          </div>
        </main>
      </div>
    </>
  );
};


const styles = {
  pageContainer: {
    fontFamily: "'Arial', sans-serif",
  color: "#333333",
  backgroundImage: url(${bgImage}),
  backgroundSize: 'cover',
  backgroundPosition: 'center',
  backgroundRepeat: 'no-repeat',
  height: '84.2vh',
  width: '100vw',
  overflow: 'hidden',
  display: 'flex',
  flexDirection: 'column',
  alignItems: 'center',
  justifyContent: 'center',
  position: 'relative',
  margin: 0,
  padding: 0,
},
header: {
  textAlign: 'center',
  marginBottom: '2rem',
},
title: {
  fontSize: '2.5rem',
  fontWeight: 'bold',
  color: '#0077b6',
},
subtitle: {
  fontSize: '1.2rem',
  color: 'black',
},
mainContent: {
  width: '100%',
  maxWidth: '600px',
  padding: '1rem',
  textAlign: 'center',
},
formContainer: {
  backgroundColor: '#f1f1f1',
  padding: '2rem',
  borderRadius: '10px',
  boxShadow: '0px 4px 8px rgba(0, 0, 0, 0.1)',
},
heading: {
  fontSize: '1.8rem',
  color: '#0077b6',
  marginBottom: '1rem',
},
description: {
  color: '#666666',
  marginBottom: '1.5rem',
},
form: {
  display: 'flex',
  flexDirection: 'column',
},
input: {
  padding: '0.8rem',
  fontSize: '1rem',
  borderRadius: '5px',
  border: '1px solid #f1f1f1',
  marginBottom: '1rem',
},
button: {
  padding: '0.8rem',
  fontSize: '1rem',
  color: '#ffffff',
  backgroundColor: '#00b4d8',
  border: 'none',
  borderRadius: '5px',
  cursor: 'pointer',
    transition: 'background-color 0.3s ease',
  },
};




export default HomePage;
```

Login.jsx

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

const Login = () => {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [errorMessage, setErrorMessage] = useState('');
    const navigate = useNavigate(); // Use useNavigate for navigation

    const handleLogin = async (event) => {
        event.preventDefault(); // Prevent form submission

        try {
```

```
const response = await fetch('http://localhost:5000/api/auth/login', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({ email, password }),
});

// Check for successful login
if (!response.ok) {
    const errorData = await response.json();
    setErrorMessage(errorData.message || 'Error logging in. Please try again.');
    return;
}

const data = await response.json();

// Store token and user ID in local storage
localStorage.setItem('token', data.token); // Store the JWT token
localStorage.setItem('userId', data.user.id); // Store the user ID

// Navigate to the remedies page
navigate('/remedies');
```

```
    } catch (error) {
        console.error('Error during login:', error);
        setErrorMessage('An unexpected error occurred. Please try again.');
    }
};

return (
    <div style={styles.pageContainer}>
        <div style={styles.container}>

            <h2 style={styles.headerTitle}>Wellnessify Login</h2>
            <form id="loginForm" onSubmit={handleLogin}>
                <div style={styles.inputGroup}>
                    <label htmlFor="username" style={styles.label}>Email</label>
                    <input
                        type="email"
                        id="username"
                        placeholder="Enter Email"
                        value={email}
                        onChange={(e) => setEmail(e.target.value)}
                        required
                        style={styles.input}
                    />
```

```
          </div>
          <div style={styles.inputGroup}>
            <label htmlFor="password" style={styles.label}>Password</label>
            <input
              type="password"
              id="password"
              placeholder="Enter Password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              required
              style={styles.input}
            />
          </div>
          {errorMessage && <div style={styles.errorMessage}>{errorMessage}</div>}
          <button type="submit" style={styles.button}>Login</button>
          <p style={styles.text}>
            Don't have an account? <a href="/signup" style={styles.link}>Sign up</a>
          </p>
        </form>
      </div>
    </div>
  );
};
```

```
// Styles for the CareConnect theme
const styles = {
  pageContainer: {
    display: 'flex',
    flexDirection: 'column',
    minHeight: '100vh',
    backgroundColor: '#E8F8F5', // Background color to match the theme
    justifyContent: 'center',
    alignItems: 'center',
  },
  container: {
    background: '#FFFFFF',
    padding: '50px',
    borderRadius: '10px',
    boxShadow: '0 4px 8px rgba(0, 0, 0, 0.1)',
    width: '350px',
    textAlign: 'center',
  },
  logo: {
    width: '150px',
    marginBottom: '20px',
  },
headerTitle: {
    color: '#2980B9',
    marginBottom: '15px',
    fontFamily: "'Pacifico', cursive",
},
inputGroup: {
    width: '100%',
    marginBottom: '15px',
},
label: {
    display: 'block',
    marginBottom: '5px',
    fontWeight: 'bold',
    color: '#333',
},
input: {
    width: '100%',
    padding: '10px',
    border: '1px solid #ccc',
    borderRadius: '5px',
},
button: {
    backgroundColor: '#2980B9',
        color: 'white',
        padding: '10px',
        width: '100%',
        border: 'none',
        borderRadius: '5px',
        cursor: 'pointer',
        marginTop: '10px',
        fontWeight: 'bold',
    },
    errorMessage: {
        color: 'red',
        marginBottom: '10px',
    },
    text: {
        color: '#333',
        marginBottom: '20px',
    },
    link: {
        color: '#ffcc33',
        textDecoration: 'none',
    },
};
```

```
export default Login;
```

MyProfile.jsx

```
// src/components/ProfilePage.js

import React, { useEffect, useState } from 'react';
import styled from 'styled-components';
import ProfileCard from '../components/ProfileCard'; // Import the ProfileCard component

const PageContainer = styled.div`
   background: #f8f9fa; /* Light background color */
   min-height: 70vh; /* Full height */
   display: flex;
   flex-direction: column;
   align-items: center;
   justify-content: center;
   padding: 20px 10px; /* Padding for spacing */
`;

const Title = styled.h1`
   font-size: 2.5rem;
   color: #1d3557; /* Dark blue from CareConnect theme */
   margin-bottom: 10px; /* Space below the title */
```

```
`;

const ErrorMessage = styled.div`
    color: #e63946; /* Error color */
    font-size: 1.2rem;
    margin: 10px; /* Space around the error message */
`;

const LoadingMessage = styled.div`
    font-size: 1.2rem;
    color: #343a40; /* Dark gray */
`;

const ProfilePage = () => {
    const [user, setUser] = useState(null); // State to hold user data
    const [error, setError] = useState(null); // State to hold error messages

    useEffect(() => {
        const fetchProfile = async () => {
            const token = localStorage.getItem('token'); // Get the token from local storage

            try {
                const response = await fetch('http://localhost:5000/api/auth/profile', {
```

```
          method: 'POST',

          headers: {

            'Authorization': Bearer ${token}, // Include the token in the request

          },

        });


        if (!response.ok) {

          const errorData = await response.text(); // Get the error response as text

          throw new Error(errorData || 'Error fetching profile'); // Throw an error with the
response text

        }


        const data = await response.json(); // Parse JSON response

        setUser(data.user); // Set user data

      } catch (err) {

        setError(err.message); // Set error message

      }

    };


    fetchProfile(); // Call the function to fetch the profile

  }, []); // Empty dependency array means this effect runs once when the component mounts


  // Render loading state, error messages, or user data

  if (error) return <ErrorMessage>Error: {error}</ErrorMessage>;
```

```
    if (!user) return <LoadingMessage>Loading...</LoadingMessage>;

    return (
        <PageContainer>
            <Title>Profile Page</Title>
            <ProfileCard user={user} /> {/* Render the ProfileCard with user data */}
        </PageContainer>
    );
};

export default ProfilePage;
```

NaturalRemedies.jsx

```
import React, { useState } from 'react';

const NaturalRemedies = () => {
  const [symptom, setSymptom] = useState('');
  const [remedies, setRemedies] = useState([]);

  // Sample remedies data
  const remediesData = {
    headache: ['Drink plenty of water', 'Rest in a dark room', 'Apply a cold compress'],
    cold: ['Drink warm fluids', 'Use steam inhalation', 'Ginger tea with honey'],
    stomachache: ['Peppermint tea', 'Ginger ale', 'Warm compress'],
    fatigue: ['Stay hydrated', 'Take short breaks', 'Get a good night’s sleep'],
  };

  const handleSearch = () => {
    const foundRemedies = remediesData[symptom.toLowerCase()] || [];
    setRemedies(foundRemedies);
  };

  return (
    <div style={styles.container}>
      <h2 style={styles.title}>Natural Remedies</h2>
      <input
        type="text"
        placeholder="Enter your symptom"
        value={symptom}
        onChange={(e) => setSymptom(e.target.value)}
        style={styles.input}
      />
      <button onClick={handleSearch} style={styles.button}>Find Remedies</button>

      <p id="symptom-text" style={styles.symptomText}>
        {symptom ? Remedies for: ${symptom} : ''}
        </p>

        <div id="remedies-output">
          {remedies.map((remedy, index) => (
            <div key={index} className="remedy-box" style={styles.remedyBox}>
              {remedy}
            </div>
          ))}
          {remedies.length === 0 && symptom && (
            <div className="remedy-box" style={styles.remedyBox}>
              No remedies found for {symptom}.
            </div>
          )}
        </div>
      </div>
    );
  };

  // Styles
  const styles = {
    container: {
      fontFamily: 'Arial, sans-serif',
      padding: '20px',
  },
  title: {
    fontSize: '24px',
    color: '#333',
  },
  input: {
    fontSize: '16px',
    padding: '10px',
    marginRight: '10px',
    border: '1px solid #ccc',
    borderRadius: '5px',
    width: '300px',
  },
  button: {
    padding: '10px 20px',
    backgroundColor: '#28a745',
    color: 'white',
    border: 'none',
    borderRadius: '5px',
    cursor: 'pointer',
  },
  symptomText: {
    fontSize: '18px',
    color: '#555',
    marginTop: '15px',
  },
  remedyBox: {
    backgroundColor: '#f9f9f9',
    padding: '15px',
    marginBottom: '20px',
    borderRadius: '8px',
    boxShadow: '0 4px 8px rgba(0, 0, 0, 0.1)',
  },
};

export default NaturalRemedies;
```

Remedies.jsx

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';

const Remedies = ({ setRemedies }) => {
    const [symptoms, setSymptoms] = useState('');
    const [error, setError] = useState('');
const navigate = useNavigate();

const handleChange = (e) => {
  setSymptoms(e.target.value);
};

const handleFetchRemedies = async (e) => {
  e.preventDefault();

  if (!symptoms) {
    setError('Please enter a symptom.');
    return;
  }

  try {
    const userName = localStorage.getItem('name');

    // Save the symptom to the backend
    await axios.post('http://localhost:5000/api/history/symptoms', {
      name: symptoms,
      userName,
    });
```

```
        // Fetch remedies based on the symptom

        const                    response                    =                    await
axios.get(http://localhost:5000/api/remedy/remedies?symptom=${symptoms});


        // Check if the response contains data

        if (Array.isArray(response.data)) {

          setRemedies(response.data); // Update the remedies state

          navigate('/remedies-list'); // Navigate to the remedies list

        } else {

          setError('Unexpected data format received.');

        }

      } catch (error) {

        console.error('Error fetching remedies:', error);

        setError('Error fetching remedies. Please try again.');

      }

    };


    return (

      <div style={styles.pageContainer}>

        <div style={styles.container}>


          <h2 style={styles.h2}>Get Remedies for Your Symptoms</h2>

          {error && <p style={styles.error}>{error}</p>}
```

```
            <form onSubmit={handleFetchRemedies} style={styles.form}>
              <div style={styles.inputGroup}>
                <label htmlFor="symptoms" style={styles.label}>Enter Your Symptoms:</label>
                <input
                  id="symptoms"
                  name="symptoms"
                  type="text"
                  value={symptoms}
                  onChange={handleChange}
                  required
                  style={styles.input}
                />
              </div>
              <button type="submit" style={styles.fetchBtn}>Get Remedies</button>
            </form>
          </div>
          <footer style={styles.footer}>
            <p style={styles.footerText}>By using this service, you agree to our <a href="#" style={styles.link}>Privacy Policy</a> and <a href="#" style={styles.link}>Terms of Service</a>.</p>
          </footer>
        </div>
  );
};
```

```
// Inline styles for the component
const styles = {
  pageContainer: {
    display: 'flex',
    flexDirection: 'column',
    justifyContent: 'space-between',
    height: '100vh', // Full viewport height
    backgroundColor: '#E6E6FA',
    padding: '20px',
  },
  container: {
    background: '#FFFFFF',
    padding: '40px',
    borderRadius: '10px',
    boxShadow: '0 4px 10px rgba(0, 0, 0, 0.15)',
    width: '100%',
    maxWidth: '400px',
    textAlign: 'center',
    display: 'flex',
    flexDirection: 'column',
    justifyContent: 'center',
    alignItems: 'center',
    flex: '1',
    margin: 'auto',
},
logoImage: {
    width: '150px',
    marginBottom: '20px',
},
h2: {
    color: '#003366',
    marginBottom: '15px',
    fontFamily: 'Arial, sans-serif',
},
error: {
    color: '#FF6B6B',
    marginBottom: '10px',
    fontSize: '0.9rem',
},
form: {
    width: '100%',
},
inputGroup: {
    width: '100%',
    marginBottom: '15px',
},
label: {
   display: 'block',
   marginBottom: '5px',
   fontWeight: 'bold',
   color: '#333',
   fontSize: '1rem',
},
input: {
   width: '100%',
   padding: '10px',
   border: '1px solid #A8D8E7',
   borderRadius: '5px',
   fontSize: '1rem',
   fontFamily: 'Arial, sans-serif',
},
fetchBtn: {
   backgroundColor: '#003366',
   color: 'white',
   padding: '10px',
   width: '100%',
   border: 'none',
   borderRadius: '5px',
        cursor: 'pointer',
        marginTop: '10px',
        fontWeight: 'bold',
        fontSize: '1rem',
    },
    footer: {
        textAlign: 'center',
        padding: '20px',
        backgroundColor: '#003366',
        color: 'white',
        width: '100%',
        bottom: '0',
    },
    footerText: {
        margin: '0',
        fontSize: '0.9rem',
    },
    link: {
        color: '#FFCC33',
        textDecoration: 'none',
    },
};
```

```
export default Remedies;
```

RemediesList.jsx

```
import React from 'react';


const RemediesList = ({ remedies }) => {
  // Ensure remedies is defined and is an array
  if (!remedies || remedies.length === 0) {
    return (
      <div style={styles.container}>
        <h2 style={styles.h2}>No Remedies Found</h2>
      </div>
    ); // Show a message when no remedies are available
  }


  return (
    <div style={styles.pageContainer}>
      <div style={styles.container}>
        <h2 style={styles.h2}>Suggested Remedies</h2>
        <ul style={styles.list}>
          {remedies.map((remedy, index) => (
            <li key={index} style={styles.remedyItem}>
              <h3 style={styles.remedyName}>{remedy.name}</h3>
```

```
                    <p><strong>Ingredients:</strong> {remedy.ingredients.join(', ')}</p>

                    <p><strong>Instructions:</strong> {remedy.instructions}</p>

                    {remedy.contraindications && (

                      <p><strong>Contraindications:</strong>
{remedy.contraindications}</p>

                    )}

                  </li>

                ))}

              </ul>

            </div>

            <footer style={styles.footer}>

              <p style={styles.footerText}>

                By using this service, you agree to our <a href="#" style={styles.link}>Privacy
Policy</a> and <a href="#" style={styles.link}>Terms of Service</a>.

              </p>

            </footer>

          </div>

  );

};



// Inline styles for the component

const styles = {

  pageContainer: {

    display: 'flex',
    flexDirection: 'column',
    justifyContent: 'space-between', // Space between header, content, and footer
    height: '100vh', // Full viewport height
    backgroundColor: '#E8F8F5', // Light background for the entire page
    padding: '20px', // Padding for the page
},
container: {
    background: '#FFFFFF',
    padding: '40px',
    borderRadius: '10px',
    boxShadow: '0 4px 10px rgba(0, 0, 0, 0.15)',
    width: '100%', // Full width within its container
    maxWidth: '600px', // Max width of the container
    textAlign: 'center',
    margin: 'auto', // Center container horizontally
    flex: '1', // Allow the container to grow and take available space
},
h2: {
    color: '#003366', // Darker blue for headings
    marginBottom: '20px',
    fontFamily: 'Arial, sans-serif', // Clean font
},
list: {
    listStyleType: 'none', // Remove default list styling
    padding: 0,
    margin: 0,
    textAlign: 'left', // Align text to the left for readability
},
remedyItem: {
    marginBottom: '20px',
    borderBottom: '1px solid #ccc', // Add a separator for clarity
    paddingBottom: '10px',
},
remedyName: {
    color: '#003366', // Darker color for remedy names
},
footer: {
    textAlign: 'center',
    padding: '20px',
    backgroundColor: '#003366', // Footer color
    color: 'white',
    width: '100%',
    bottom: '0', // Stick to the bottom
},
footerText: {
    margin: '0', // Remove default margin
    fontSize: '0.9rem', // Slightly smaller font for the footer text
  },
  link: {
    color: '#FFCC33', // Accent color for links
    textDecoration: 'none',
  },
};

export default RemediesList;
```

SearchHistory.jsx

```
import React, { useEffect, useState } from 'react';

const SearchHistory = () => {
  const [searchHistory, setSearchHistory] = useState([]); // State to store symptoms

  // Fetch symptoms from the backend
  const fetchSymptoms = async () => {
    try {
      const response = await fetch('http://localhost:5000/api/history/symptoms');
      if (!response.ok) {
            throw new Error(HTTP error! status: ${response.status});
        }
        const data = await response.json();
        setSearchHistory(data); // Store the fetched data
    } catch (error) {
        console.error('Error fetching symptoms:', error);
    }
};

useEffect(() => {
    fetchSymptoms(); // Call fetchSymptoms on component mount
}, []);

return (
    <div style={styles.pageContainer}>
        <div style={styles.container}>
            <h2 style={styles.h2}>Search History</h2>
            <ul style={styles.list}>
                {searchHistory.map((item) => (
                    <li key={item._id} style={styles.historyItem}>
                        <strong>Symptom:</strong> {item.name}
                        <br />
                        <strong>Date:</strong>                    {new
Date(item.dateEntered).toLocaleDateString()}
```

```
                    </li>
                  ))}
                </ul>
              </div>
              <footer style={styles.footer}>
                <p style={styles.footerText}>
                  By using this service, you agree to our <a href="#" style={styles.link}>Privacy
Policy</a> and <a href="#" style={styles.link}>Terms of Service</a>.
                </p>
              </footer>
            </div>
  );
};


// Inline styles for the component with light theme
const styles = {
  pageContainer: {
    display: 'flex',
    flexDirection: 'column',
    justifyContent: 'space-between', // Space between header, content, and footer
    height: '100vh', // Full viewport height
    backgroundColor: '#ffffff', // Clean White for the background
    padding: '20px', // Padding for the page
  },
container: {
    background: '#f1f1f1', // Light Gray for the container background
    padding: '40px',
    borderRadius: '10px',
    boxShadow: '0 4px 10px rgba(0, 0, 0, 0.1)', // Soft shadow for depth
    width: '100%', // Full width within its container
    maxWidth: '600px', // Max width of the container
    textAlign: 'center',
    margin: 'auto', // Center container horizontally
    flex: '1', // Allow the container to grow and take available space
},
h2: {
    color: '#0077b6', // Calm Blue for headings
    marginBottom: '20px',
    fontFamily: 'Arial, sans-serif', // Clean font
},
list: {
    listStyleType: 'none', // Remove default list styling
    padding: 0,
    margin: 0,
    textAlign: 'left', // Align text to the left for readability
},
historyItem: {
        marginBottom: '20px',

        borderBottom: '1px solid #ccc', // Light border for separation

        paddingBottom: '10px',

        textAlign: 'left', // Align text to the left

    },

    footer: {

        textAlign: 'center',

        padding: '20px',

        backgroundColor: '#0077b6', // Calm Blue for footer background

        color: 'white',

        width: '100%',

        bottom: '0', // Stick to the bottom

    },

    footerText: {

        margin: '0', // Remove default margin

        fontSize: '0.9rem', // Slightly smaller font for the footer text

    },

    link: {

        color: '#00b4d8', // Soft Green for links

        textDecoration: 'none',

    },

};
```

```
export default SearchHistory;
```

Settings.jsx

```
import React from 'react';
import { useNavigate } from 'react-router-dom';

const Settings = () => {
  const navigate = useNavigate();

  const handleLogout = () => {
    // Perform logout actions here (e.g., clearing user data)
    localStorage.removeItem('loggedInUser'); // Example: clearing user data
    navigate('/login'); // Redirect to login page after logout
  };

  return (
    <div style={styles.container}>
      <div style={styles.header}>
        <h1 style={styles.title}>Settings</h1>
      </div>

      <div style={styles.settingsContainer}>
    <table style={styles.table}>
      <thead>
        <tr>
          <th style={styles.th}>Option</th>
          <th style={styles.th}>Action</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td style={styles.td}>Profile Settings</td>
          <td style={styles.td}><a href="/myprofile" style={styles.link}>Edit
Profile</a></td>
        </tr>
        <tr>
          <td style={styles.td}>Notifications</td>
          <td style={styles.td}><a href="#notifications" style={styles.link}>Manage
Notifications</a></td>
        </tr>
        <tr>
          <td style={styles.td}>Privacy Settings</td>
          <td style={styles.td}><a href="#privacy" style={styles.link}>Update
Privacy</a></td>
        </tr>
        <tr>
          <td style={styles.td}>Account Security</td>
```

```
              <td style={styles.td}><a href="#security" style={styles.link}>Change Password</a></td>
            </tr>
            <tr>
              <td style={styles.td}>Log Out</td>
              <td style={styles.td}>
                <button style={styles.logoutBtn} onClick={handleLogout}>Log Out</button>
              </td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  );
};


// Styles
const styles = {
  container: {
    fontFamily: 'Arial, sans-serif',
    backgroundColor: '#f1f1f1',
    margin: 0,
    padding: 0,
  },
  header: {
    backgroundColor: '#333',
    color: 'white',
    textAlign: 'center',
    padding: '20px',
  },
  title: {
    margin: 0,
    fontSize: '2rem',
  },
  settingsContainer: {
    margin: '20px auto',
    width: '80%',
    maxWidth: '600px',
    backgroundColor: 'white',
    padding: '20px',
    borderRadius: '10px',
    boxShadow: '0 4px 10px rgba(0, 0, 0, 0.1)',
  },
  table: {
    width: '100%',
    borderCollapse: 'collapse',
  },
  th: {
    padding: '15px',
    textAlign: 'left',
    borderBottom: '1px solid #ddd',
    backgroundColor: '#333',
    color: 'white',
  },
  td: {
    padding: '15px',
    textAlign: 'left',
    borderBottom: '1px solid #ddd',
  },
  link: {
    color: '#003366',
    textDecoration: 'none',
    fontWeight: 'bold',
  },
  logoutBtn: {
    padding: '10px',
    backgroundColor: '#ff4c4c',
    color: 'white',
    border: 'none',
    borderRadius: '5px',
    cursor: 'pointer',
    width: '100%',
    textAlign: 'center',
  },
};

export default Settings;
```

Signup.jsx

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom'; // For navigation
import axios from 'axios'; // For API calls

const Signup = () => {
    const navigate = useNavigate();

    // State for form inputs
    const [formData, setFormData] = useState({
        name: '',
        email: '',
        password: '',
    confirmPassword: '',
    role: 'user', // Default role
});

// State for error messages
const [error, setError] = useState('');

// Handle input change
const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
};

// Handle form submission
const handleSignup = async (e) => {
    e.preventDefault();
    const { name, email, password, confirmPassword, role } = formData;

    // Validate passwords match
    if (password !== confirmPassword) {
        setError("Passwords do not match.");
        return;
    }
    try {
        // API call to register the user
        const response = await axios.post('http://localhost:5000/api/auth/register', {
            name,
            email,
            password,
            role, // Include role
        });

        if (response.status === 201) {
            alert('User registered successfully!');
            navigate('/login'); // Redirect to login page
        }
    } catch (err) {
        console.error(err);
        setError(err.response?.data?.message || 'Error signing up. Please try again.');
    }
};

return (
    <div style={styles.pageContainer}>
        <div style={styles.container}>
```

```
<h2 style={styles.h2}>Join Wellnessify</h2>
<p>Create your account to access personalized health management tools.</p>

{error && <p style={styles.error}>{error}</p>}

<form onSubmit={handleSignup} style={styles.form}>
  <div style={styles.inputGroup}>
    <label htmlFor="name" style={styles.label}>Full Name</label>
    <input
      type="text"
      id="name"
      name="name"
      value={formData.name}
      onChange={handleChange}
      required
      style={styles.input}
    />
  </div>

  <div style={styles.inputGroup}>
    <label htmlFor="email" style={styles.label}>Email</label>
    <input
```

```
        type="email"
        id="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
        required
        style={styles.input}
    />
</div>

<div style={styles.inputGroup}>
    <label htmlFor="password" style={styles.label}>Password</label>
    <input
        type="password"
        id="password"
        name="password"
        value={formData.password}
        onChange={handleChange}
        required
        style={styles.input}
    />
</div>
```

```
          <div style={styles.inputGroup}>
            <label htmlFor="confirmPassword" style={styles.label}>Confirm
Password</label>
            <input
              type="password"
              id="confirmPassword"
              name="confirmPassword"
              value={formData.confirmPassword}
              onChange={handleChange}
              required
              style={styles.input}
            />
          </div>

          <div style={styles.inputGroup}>
            <label htmlFor="role" style={styles.label}>Role</label>
            <select
              id="role"
              name="role"
              value={formData.role}
              onChange={handleChange}
              required
              style={styles.input}
            >
```

```
                <option value="user">User</option>

                <option value="admin">Admin</option>

              </select>

            </div>


            <button type="submit" style={styles.signupBtn}>Sign Up</button>

          </form>


          <p style={styles.footer}>

            Already have an account? <a href="/login" style={styles.link}>Login</a>

          </p>

        </div>

      </div>

  );

};


const styles = {

  pageContainer: {

    display: 'flex',

    justifyContent: 'center',

    alignItems: 'center',

    height: '100vh',

    backgroundColor: '#E8F8F5',
  },
  container: {
    background: '#fff',
    padding: '30px',
    borderRadius: '10px',
    boxShadow: '0 4px 8px rgba(0, 0, 0, 0.1)',
    maxWidth: '400px',
    width: '100%',
    textAlign: 'center',
  },
  logoImage: {
    width: '150px',
    marginBottom: '15px',
  },
  h2: {
    color: '#003366',
    marginBottom: '15px',
  },
  error: {
    color: 'red',
    marginBottom: '15px',
  },
  inputGroup: {
    marginBottom: '15px',
  },
  label: {
    display: 'block',
    marginBottom: '5px',
    fontWeight: 'bold',
  },
  input: {
    width: '100%',
    padding: '10px',
    border: '1px solid #ccc',
    borderRadius: '5px',
  },
  signupBtn: {
    backgroundColor: '#003366',
    color: '#fff',
    padding: '10px',
    width: '100%',
    border: 'none',
    borderRadius: '5px',
    fontWeight: 'bold',
    cursor: 'pointer',
  },
  footer: {
    marginTop: '15px',
  },
  link: {
    color: '#003366',
    textDecoration: 'none',
  },
};

export default Signup;
```

app.jsx

```
import React, { useState } from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import './App.css';
import AboutUs from './pages/aboutUs';
import HomePage from './pages/homePage';
import Login from './pages/Login';
import ProfilePage from './pages/MyProfile';
import NaturalRemedies from './pages/NaturalRemedies';
import SearchHistory from './pages/SearchHistory';
import Signup from './pages/Signup';
import NavBar from './components/NavBar';
import Remedies from './pages/Remedies';
import RemediesList from './pages/RemediesList'; // Make sure to import RemediesList
import EditProfile from './pages/EditProfile';



function App() {
  const [remedies, setRemedies] = useState([]); // Initialize remedies state

  return (
    <Router>
      <NavBar />
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/aboutus" element={<AboutUs />} />
        <Route path="/searchhistory" element={<SearchHistory />} />
        <Route path="/profile" element={<ProfilePage />} />
        <Route path="/login" element={<Login />} />
        <Route path="/natural-remedies" element={<NaturalRemedies />} />
        <Route path="/signup" element={<Signup />} />
        <Route path="/remedies" element={<Remedies setRemedies={setRemedies} />}
/>
        <Route path="/remedies-list" element={<RemediesList remedies={remedies} />}
/>
        <Route path="/edit-profile" element={<EditProfile />} />
      </Routes>
    </Router>
  );
}

export default App;
```

index.css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

 html, body {
  height: 100%;
  width: 100%;
 }
```

main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

PostForm.jsx

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate, useParams } from 'react-router-dom';

const PostForm = () => {
    const [title, setTitle] = useState('');
    const [content, setContent] = useState('');
    const navigate = useNavigate();
    const { id } = useParams();

    // Get userId from localStorage
    const getUserId = () => {
```

```
    const userId = localStorage.getItem('userId');  // Assuming userId is stored in localStorage
    return userId;
  };

  const userId = getUserId();  // Fetch the userId from localStorage

  // Fetch post for editing
  useEffect(() => {
    if (id) {
      const fetchPost = async () => {
        try {
          const response = await axios.get(http://localhost:5000/api/blog/posts/${id});
          setTitle(response.data.title);
          setContent(response.data.content);
        } catch (error) {
          console.error('Error fetching post:', error);
        }
      };

      fetchPost();
    }
  }, [id]);
```

```
const handleSubmit = async (e) => {
  e.preventDefault();
  const postData = { title, content, author: userId }; // Pass userId in the request data
  try {
    if (id) {
      // Update post
      await axios.put(http://localhost:5000/api/blog/posts/${id}, postData, {
        headers: {
          Authorization: Bearer ${localStorage.getItem('token')},
        },
      });
    } else {
      // Create post
      await axios.post('http://localhost:5000/api/blog/posts/', postData, {
        headers: {
          Authorization: Bearer ${localStorage.getItem('token')},
        },
      });
    }
    navigate('/');  // Redirect to home page after submit
  } catch (error) {
    console.error('Error submitting post:', error);
  }
```

```
};


return (
  <div>
    <h2>{id ? 'Edit Post' : 'Create Post'}</h2>
    <form onSubmit={handleSubmit}>
      <label>Title:</label>
      <input
        type="text"
        value={title}
        onChange={(e) => setTitle(e.target.value)}
        required
      />
      <br />
      <label>Content:</label>
      <textarea
        value={content}
        onChange={(e) => setContent(e.target.value)}
        required
      />
      <br />
      <button type="submit">{id ? 'Update' : 'Create'} Post</button>
    </form>
```

```
        </div>
    );
};

export default PostForm;
```

PostDetails.jsx

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useParams, useNavigate } from 'react-router-dom';
import { FaEdit, FaTrashAlt } from 'react-icons/fa'; // Import edit and delete icons

const PostDetail = () => {
    const [post, setPost] = useState(null);
    const [isAdmin, setIsAdmin] = useState(false); // State to track if the user is an admin
    const { id } = useParams(); // To get the post id from the URL
    const navigate = useNavigate(); // To navigate programmatically

    useEffect(() => {
        // Fetch the post by id from the backend
        const fetchPost = async () => {
            try {
```

```
            const response = await axios.get(http://localhost:5000/api/blog/posts/${id});
            setPost(response.data);
        } catch (error) {
            console.error('Error fetching post:', error);
        }
    };

    // Check the user's role from localStorage
    const checkAdminRole = () => {
        const role = localStorage.getItem('role'); // Retrieve role from localStorage
        setIsAdmin(role === 'admin'); // Set isAdmin to true if the role is 'Admin'
    };

    fetchPost();
    checkAdminRole();
}, [id]);

// Function to delete the post
const handleDelete = async () => {
    try {
        await axios.delete(http://localhost:5000/api/blog/posts/${id}, {
            headers: {
                Authorization: Bearer ${localStorage.getItem('token')},
            },
        });
        navigate('/'); // Redirect to the homepage after deletion
    } catch (error) {
        console.error('Error deleting post:', error);
    }
};

// Redirect to update form
const handleUpdate = () => {
    navigate(/update/${id});
};

if (!post) return <p>Loading...</p>;

const formattedDate = new Date(post.createdAt).toLocaleDateString('en-US', {
    year: 'numeric',
    month: 'long',
    day: 'numeric',
});

// Styling for the page
const styles = {
```

```
postDetailCard: {
   width: '80%',
   margin: '20px auto',
   padding: '20px',
   backgroundColor: '#f9f9f9',
   borderRadius: '10px',
   boxShadow: '0 4px 8px rgba(0, 0, 0, 0.1)',
},
postDetailContainer: {
   textAlign: 'center',
},
title: {
   fontSize: '2rem',
   color: '#333',
   marginBottom: '20px',
},
content: {
   fontSize: '1rem',
   color: '#555',
   margin: '10px 0',
},
actionButtons: {
   marginTop: '20px',
    display: 'flex',
    justifyContent: 'center',
    gap: '20px',
},
actionButton: {
    backgroundColor: '#1abc9c',
    color: 'white',
    border: 'none',
    padding: '10px 20px',
    fontSize: '1rem',
    borderRadius: '5px',
    cursor: 'pointer',
    display: 'flex',
    alignItems: 'center',
    gap: '5px',
    transition: 'background-color 0.3s ease, color 0.3s ease',
},
actionButtonHover: {
    backgroundColor: '#16a085',
},
deleteButton: {
    backgroundColor: '#e74c3c',
},
    deleteButtonHover: {
        backgroundColor: '#c0392b',
    },
    contentWrapper: {
        marginBottom: '20px',
    },
};

return (
    <div style={styles.postDetailCard}>
        <div style={styles.postDetailContainer}>
            <h1 style={styles.title}>{post.title}</h1>
            <p><strong>Author:</strong> {post.author}</p>
            <p><strong>Created on:</strong> {formattedDate}</p>

            <div style={styles.contentWrapper}>
                <p style={styles.content}>{post.content}</p>
            </div>

            {/* Conditionally render action buttons only for Admin users */}
            {isAdmin && (
                <div style={styles.actionButtons}>
                    <button
```

```
                        style={{ ...styles.actionButton, ...styles.actionButtonHover }}

                        onClick={handleUpdate}

                    >

                        <FaEdit size={20} /> Edit

                    </button>

                    <button

                        style={{                    ...styles.actionButton,                    ...styles.deleteButton,
...styles.deleteButtonHover }}

                        onClick={handleDelete}

                    >

                        <FaTrashAlt size={20} /> Delete

                    </button>

                </div>

            )}

        </div>

    </div>

  );

};


export default PostDetail;
```

PostList.jsx

```
import React, { useState, useEffect } from 'react';

import { Link } from 'react-router-dom';

import axios from 'axios';


const PostList = () => {

  const [posts, setPosts] = useState([]); // Initialize posts as an empty array

  const [isAdmin, setIsAdmin] = useState(false); // State to track if the user is an admin


  useEffect(() => {

    // Fetch posts from the backend

    const fetchPosts = async () => {

      try {

        const response = await axios.get('http://localhost:5000/api/blog/posts');

        console.log(response.data); // For debugging: Log the full response

        setPosts(response.data.posts); // Set posts to the array from response.data.posts

      } catch (error) {

        console.error('Error fetching posts:', error);

      }

    };


    // Check the user's role from localStorage

    const checkAdminRole = () => {

      const role = localStorage.getItem('role'); // Retrieve role from localStorage
      setIsAdmin(role === 'admin'); // Set isAdmin to true if the role is 'Admin'
    };

    fetchPosts();
    checkAdminRole();
  }, []);

  return (
    <div style={styles.container}>
      <h1 style={styles.header}>All Posts</h1>

      {/* Conditionally render Add Post button for Admin users */}
      {isAdmin && (
        <Link to="/create">
          <button style={styles.button}>Add Post</button>
        </Link>
      )}

      {/* List of posts */}
      <div style={styles.postsContainer}>
        {Array.isArray(posts) && posts.length > 0 ? (
          posts.map((post) => (
            <Link to={/posts/${post._id}} key={post._id} style={styles.card}>
```

```
                    <h3 style={styles.title}>{post.title}</h3>
                    <p   style={styles.content}>{post.content.substring(0,   100)}...</p>   {/*
Show a preview */}
                  </Link>
                ))
              ) : (
                <p>No posts available</p>
              )}
            </div>
          </div>
  );
};


const styles = {
  container: {
    width: '80%',
    margin: '0 auto',
    padding: '20px',
    fontFamily: 'Arial, sans-serif',
  },
  header: {
    fontSize: '2rem',
    color: '#333',
    marginBottom: '20px',
},
button: {
  margin: '10px 0',
  padding: '10px 20px',
  backgroundColor: '#1abc9c',
  color: 'white',
  border: 'none',
  borderRadius: '5px',
  cursor: 'pointer',
},
postsContainer: {
  display: 'grid',
  gridTemplateColumns: 'repeat(auto-fill, minmax(300px, 1fr))',
  gap: '20px',
},
card: {
  border: '1px solid #ddd',
  borderRadius: '5px',
  padding: '15px',
  backgroundColor: '#fff',
  boxShadow: '0 4px 6px rgba(0, 0, 0, 0.1)',
  textDecoration: 'none',
  color: '#333',
    },
    title: {
        fontSize: '1.5rem',
        color: '#333',
        marginBottom: '10px',
    },
    content: {
        fontSize: '1rem',
        color: '#555',
    },
};

export default PostList;
```

2. **Backend:** Developed in Python, leveraging frameworks like Flask or Django for seamless symptom analysis and remedy matching.

   authController.js

   ```
   // controllers/authController.js
   const User = require('../models/User');
   const bcrypt = require('bcryptjs');
   const jwt = require('jsonwebtoken');
   ```

```
// Register User
exports.register = async (req, res) => {
  const { name, email, password } = req.body;

  try {
    // Check if the user already exists
    let user = await User.findOne({ email });
    if (user) return res.status(400).json({ message: "User already exists" });

    // Hash the password before saving
    const hashedPassword = await bcrypt.hash(password, 10);

    // Create a new user
    user = new User({ name, email, password: hashedPassword });
    await user.save();

    // Optionally return the created user's information
    res.status(201).json({
      message: "User registered successfully",
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
            createdAt: user.createdAt,
          },
        });
    } catch (err) {
        res.status(500).json({ message: "Error registering user", error: err.message });
    }
};

// Login User
exports.login = async (req, res) => {
    const { email, password } = req.body;

    try {
        // Check if the user exists
        const user = await User.findOne({ email });
        if (!user) return res.status(400).json({ message: "User not found" });

        // Compare passwords
        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) return res.status(400).json({ message: "Invalid credentials" });

        // Generate a JWT token
        const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, { expiresIn:
"1h" });
```

```
        // Return the token and user information
        res.json({
            token,
            user: {
                id: user._id,
                name: user.name,
                email: user.email,
            },
        });
    } catch (err) {
        res.status(500).json({ message: "Error logging in", error: err.message });
    }
};


// Get User Profile
exports.getProfile = async (req, res) => {
    const userId = req.user.userId; // Assuming you are using middleware to extract user ID
from token


    try {
        // Find the user by ID
        const user = await User.findById(userId).select('-password'); // Exclude the password
field
```

```
        if (!user) return res.status(404).json({ message: "User not found" });

        // Return the user's information
        res.json({
            user: {
                id: user._id,
                name: user.name,
                email: user.email,
                createdAt: user.createdAt,
            },
        });
    } catch (err) {
        res.status(500).json({ message: "Error retrieving profile", error: err.message });
    }
};



// Update User Profile
exports.updateProfile = async (req, res) => {
    const { name, email, password } = req.body;
    const userId = req.user.userId; // Assuming you are using middleware to extract user ID
from token
```

```
try {
    // Find the user by ID
    let user = await User.findById(userId);
    if (!user) return res.status(404).json({ message: "User not found" });

    // Update the user fields
    if (name) user.name = name;
    if (email) user.email = email;

    // If a new password is provided, hash it and update
    if (password) {
        const hashedPassword = await bcrypt.hash(password, 10);
        user.password = hashedPassword;
    }

    // Save the updated user information
    await user.save();

    // Return the updated user information
    res.json({
        message: "Profile updated successfully",
        user: {
            id: user._id,
            name: user.name,
            email: user.email,
          },
        });
    } catch (err) {
        res.status(500).json({ message: "Error updating profile", error: err.message });
    }
};

// controllers/authController.js
exports.editProfile = async (req, res) => {
    const { name, email } = req.body; // Get new data from the request body
    const userId = req.user.userId; // Get the user ID from the token

    try {
        // Find the user by ID
        const user = await User.findById(userId);
        if (!user) return res.status(404).json({ message: "User not found" });

        // Update user fields
        if (name) user.name = name; // Update name if provided
        if (email) user.email = email; // Update email if provided
        await user.save(); // Save changes to the database

        // Return the updated user information
        res.json({
          message: "Profile updated successfully",
          user: {
            id: user._id,
            name: user.name,
            email: user.email,
            createdAt: user.createdAt,
          },
        });
    } catch (err) {
        res.status(500).json({ message: "Error updating profile", error: err.message });
    }
};
```

remedyController.js

```
const Remedy = require('../models/Remedy'); // Make sure to import your Remedy model

// Function to get remedies by symptom
const remedy = async (req, res) => {
  try {
    const symptomName = req.query.symptom;

    if (!symptomName) {
      return res.status(400).json({ message: 'Symptom name is required.' });
    }

    const remedies = await Remedy.find({
      symptoms: { $elemMatch: { name: symptomName } }
    });

    console.log('Found remedies:', remedies); // Log remedies for debugging

    if (remedies.length === 0) {
      return res.status(404).json({ message: 'No remedies found for the provided symptom.' });
    }

    res.json(remedies);
  } catch (error) {
    console.error("Error retrieving remedies:", error);
    res.status(500).json({ message: 'Error retrieving remedies', error: error.message });
  }
};
```

```
// Function to add remedies

const addRemedies = async (req, res) => {

  try {

    const { name, ingredients, instructions, contraindications, source_reference,
symptoms } = req.body;



    // Check that required fields are provided

    if (!name || !ingredients || !instructions || !symptoms) {

      return res.status(400).json({ message: 'Please provide all required fields.' });

    }



    // Create a new remedy instance

    const remedy = new Remedy({

      name,

      ingredients,

      instructions,

      contraindications,

      source_reference,

      symptoms // This should be an array of objects with a name property

    });



    // Save the remedy to the database

    await remedy.save();
```

```
        // Respond with the created remedy
        res.status(201).json(remedy);
    } catch (error) {
        console.error("Error creating remedy:", error); // Log the error
        res.status(400).json({ message: 'Error creating remedy', error: error.message });
    }
};

module.exports = { remedy, addRemedies };
```

symptomController.js

```
// controllers/symptomController.js

const Symptom = require('../models/symptom'); // Import the Symptom model

// POST controller to add a new symptom
const createSymptom = async (req, res) => {
    const { name, description, userId } = req.body;

    const newSymptom = new Symptom({
        name,
        description,
        userId, // Optional if you want to associate the symptom with a user
    });

    try {
        const savedSymptom = await newSymptom.save();
        res.status(201).json(savedSymptom); // Return the saved symptom
    } catch (error) {
        res.status(400).json({ error: 'Error saving symptom: ' + error.message });
    }
};

// DELETE controller to remove a symptom by ID
const deleteSymptom = async (req, res) => {
    const { id } = req.params; // Extract ID from the request parameters

    try {
        const deletedSymptom = await Symptom.findByIdAndDelete(id);
        if (!deletedSymptom) {
            return res.status(404).json({ message: 'Symptom not found.' });
        }

        res.status(200).json({ message: 'Symptom deleted successfully.' });
    } catch (error) {
        res.status(400).json({ error: 'Error deleting symptom: ' + error.message });
    }
};

const getAllSymptoms = async (req, res) => {
    try {
        const symptoms = await Symptom.find(); // Retrieve all symptoms from the database
        res.status(200).json(symptoms); // Return the list of symptoms
    } catch (error) {
        res.status(500).json({ error: 'Error retrieving symptoms: ' + error.message });
    }
};

module.exports = {
    createSymptom,
    deleteSymptom,
    getAllSymptoms,
};
```

auth.js

```
// middleware/auth.js
const jwt = require('jsonwebtoken');

const auth = (req, res, next) => {
   const token = req.header('Authorization');
   console.log(token); // Log the token for debugging
   if (!token) return res.status(401).json({ message: "No token, authorization denied" });

   try {
      const decoded = jwt.verify(token.split(" ")[1], process.env.JWT_SECRET); // Split to
get the token part
      req.user = decoded;
      next();
   } catch (err) {
      console.log(err); // Log the error for debugging
      res.status(401).json({ message: "Invalid token" });
   }
};
```

```
module.exports = auth;
```

auth.js

```
// routes/auth.js
const auth = require('../middleware/auth')
const express = require('express');
const router = express.Router();
const { register, login, editProfile, getProfile} = require('../controllers/authController');

router.post('/register', register);
router.post('/login', login);
router.put('/profile', auth, editProfile);
router.post('/profile',auth, getProfile);


module.exports = router;
```

remedyRoute.js

```
const express = require('express');
const router = express.Router();
const {remedy, addRemedies} = require('../controllers/remedyController')

router.get('/remedies', remedy)

router.post('/add', addRemedies)


module.exports = router;
```

symptomRoute.js

```
// routes/symptom.js

const express = require('express');

const router = express.Router();

const { createSymptom, deleteSymptom, getAllSymptoms } = require('../controllers/symptomController'); // Import the controllers


// POST endpoint to add a new symptom

router.post('/symptoms', createSymptom);


router.get('/symptoms', getAllSymptoms);


// DELETE endpoint to remove a symptom by ID

router.delete('/symptoms/:id', deleteSymptom);
```

```
module.exports = router;
```

3. **Database:** A relational database (e.g., MySQL or PostgreSQL) storing user data, symptoms, and remedies.

Remedy.js

```
const mongoose = require('mongoose');


// Define the Symptoms subdocument schema
const SymptomSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    index: true // Indexing for faster symptom lookup
  }
});


// Define the Remedies schema with embedded Symptoms
const RemedySchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  ingredients: {
    type: [String], // Array of ingredients for the remedy
    required: true
  },
  instructions: {
    type: String,
    required: true
  },
  contraindications: {
    type: String
  },
  source_reference: {
    type: String
  },
  symptoms: {
    type: [SymptomSchema], // Embedding SymptomSchema as an array
    required: true
  }
});

// Create the Remedy model
const Remedy = mongoose.model('Remedy', RemedySchema);

// Export the Remedy model
```

```
module.exports = Remedy;
```

symptom.js

```
// models/symptom.js

const mongoose = require('mongoose');

// Define the schema for storing symptoms
const symptomSchema = new mongoose.Schema({
   name: {
      type: String,
      required: true, // Name is required
      trim: true, // Remove whitespace from the beginning and end
   },
   description: {
      type: String,
      trim: true,
   },
   dateEntered: {
      type: Date,
      default: Date.now, // Automatically set the date when the document is created
   },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User', // Reference to a User model, if you want to associate symptoms with a specific user
  },
});

// Create the model
const Symptom = mongoose.model('Symptom', symptomSchema);

module.exports = Symptom;
```

User.js

```
// models/User.js
const mongoose = require('mongoose');

// Define the user schema
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
        type: String,
        required: true,
        unique: true, // Ensure that email is unique
        lowercase: true, // Convert email to lowercase
        trim: true, // Remove extra whitespace
    },
    password: {
        type: String,
        required: true,
    },
}, { timestamps: true }); // Automatically manage createdAt and updatedAt timestamps

// Create a model from the schema
const User = mongoose.model('User', userSchema);
module.exports = User;
```

postController.js

```
const Post = require('../models/Post'); // Assuming the Post model is in the models directory
const Comment = require('../models/Comment');

// Create a new blog post
const createPost = async (req, res) => {
    try {
```

```
// Destructure the incoming request body
const { title, content, author } = req.body;

// Check if required fields are provided
if (!title || !content || !author) {
  return res.status(400).json({
    message: 'Title, content, and author are required'
  });
}

// Create a new post instance
const newPost = new Post({
  title,
  content,
  author,
  createdAt: new Date(),
  updatedAt: new Date()
});

// Save the new post to the database
await newPost.save();

// Send back the created post as a response
```

```
        res.status(201).json({
            message: 'Post created successfully',
            post: newPost
        });
    } catch (error) {
        console.error('Error creating post:', error);  // Log the error for debugging
        res.status(500).json({
            message: 'Error creating post',
            error: error.message
        });
    }
};

// Controller to get all posts
const getAllPosts = async (req, res) => {
    try {
        // Fetch all posts from the database
        const posts = await Post.find();

        // If no posts are found, return a 404 response
        if (posts.length === 0) {
            return res.status(404).json({
                message: 'No posts found'
            });
        }


        // Send back the list of posts
        res.status(200).json({
            message: 'Posts fetched successfully',
            posts: posts
        });
    } catch (error) {
        console.error('Error fetching posts:', error);  // Log the error for debugging
        res.status(500).json({
            message: 'Error fetching posts',
            error: error.message
        });
    }
};


// Controller to get a particular post by ID
// Backend controller example
const getPostById = async (req, res) => {
    try {
        const post = await Post.findById(req.params.id);
        if (!post) {
            return res.status(404).json({ message: 'Post not found' });
        }
        console.log(post.createdAt);  // Debugging line
        res.json(post);
    } catch (error) {
        res.status(500).json({ message: 'Server error' });
    }
};


// Controller to update a post by ID
const updatePost = async (req, res) => {
    try {
        const postId = req.params.id;
        const { title, content } = req.body;

        if (!title && !content) {
            return res.status(400).json({
                message: 'Title or content is required to update the post'
            });
        }

        const updatedPost = await Post.findByIdAndUpdate(
            postId,
            {
                title,
                content,
                updatedAt: new Date()
            },
            { new: true }
        );

        if (!updatedPost) {
            return res.status(404).json({
                message: 'Post not found'
            });
        }

        res.status(200).json({
            message: 'Post updated successfully',
            post: updatedPost
        });
    } catch (error) {
        console.error('Error updating post:', error); // Log the full error for debugging
        res.status(500).json({
            message: 'Error updating post',
            error: error.message // Send the error message in the response
        });
    }
  };

  // Controller to delete a post by ID
  const deletePost = async (req, res) => {
    try {
        // Get the post ID from the request parameters
        const postId = req.params.id;

        // Find and delete the post by its ID
        const deletedPost = await Post.findByIdAndDelete(postId);

        // If no post is found, return a 404 response
        if (!deletedPost) {
            return res.status(404).json({
                message: 'Post not found'
            });
        }

        // Send back a success message
        res.status(200).json({
            message: 'Post deleted successfully',
            post: deletedPost
        });
    } catch (error) {
        console.error('Error deleting post:', error);  // Log the error for debugging
        res.status(500).json({
            message: 'Error deleting post',
            error: error.message
        });
    }
};

// Controller to add a comment to a post
// const addComment = async (req, res) => {
//     const { content } = req.body;  // Get the comment content
//     const postId = req.params.id;   // Get the post ID from the URL
//     const userId = req.user.id;     // Assuming the user ID is available from authentication

//     if (!content) {
//         return res.status(400).json({ message: 'Comment content is required' });
//     }

//     try {
```

```
//         // Create and save the new comment
//         const newComment = await Comment.create({
//             content,
//             author: userId,
//             postId
//         });

//         // Add the comment ID to the post's comments array
//        await Post.findByIdAndUpdate(postId, { $push: { comments: newComment._id } });

//         // Respond with the added comment
//                   res.status(201).json({ message: 'Comment added successfully', comment: newComment });
//     } catch (error) {
//         console.error('Error adding comment:', error);
//         res.status(500).json({ message: 'Error adding comment', error: error.message });
//     }
// };



// Export the controller functions
module.exports = {
    createPost,
    getAllPosts,
    getPostById,
    updatePost,
    deletePost,
    // likePost,
    // addComment,
};
```

Comment.js

```
const mongoose = require('mongoose');


// Define the Comment Schema
const CommentSchema = new mongoose.Schema({
    content: { type: String, required: true }, // Content of the comment
    author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true }, // Reference to the User model for the comment's author
    postId: { type: mongoose.Schema.Types.ObjectId, ref: 'Post', required: true }, // Reference to the Post model to associate the comment with a post
    createdAt: { type: Date, default: Date.now } // Timestamp when the comment was created
});


// Define and export the Comment model
const Comment = mongoose.model('Comment', CommentSchema);
module.exports = Comment;
```

Post.js

```
const mongoose = require('mongoose');

// Define the Post Schema
const PostSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  content: {
    type: String,
    required: true
  },
  author: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',  // Reference to the User model
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
    },
    updatedAt: {
        type: Date,
        default: Date.now
    },
});

// Define and export the Post model
const Post = mongoose.model('Post', PostSchema);
module.exports = Post;
```

postRoute.js

```
const express = require('express');
const   router = express.Router();
const postController = require('../controllers/postController');
// CRUD Operations
router.post('/posts', postController.createPost);
router.get('/posts', postController.getAllPosts);
router.get('/posts/:id', postController.getPostById);
router.put('/posts/:id', postController.updatePost);
router.delete('/posts/:id', postController.deletePost);
// router.post('/posts/:id/comments', postController.addComment);
module.exports = router;
```

Screenshot of Running Project

Key features displayed in the screenshots include:

Image-1

Login Page: Secure user authentication interface.

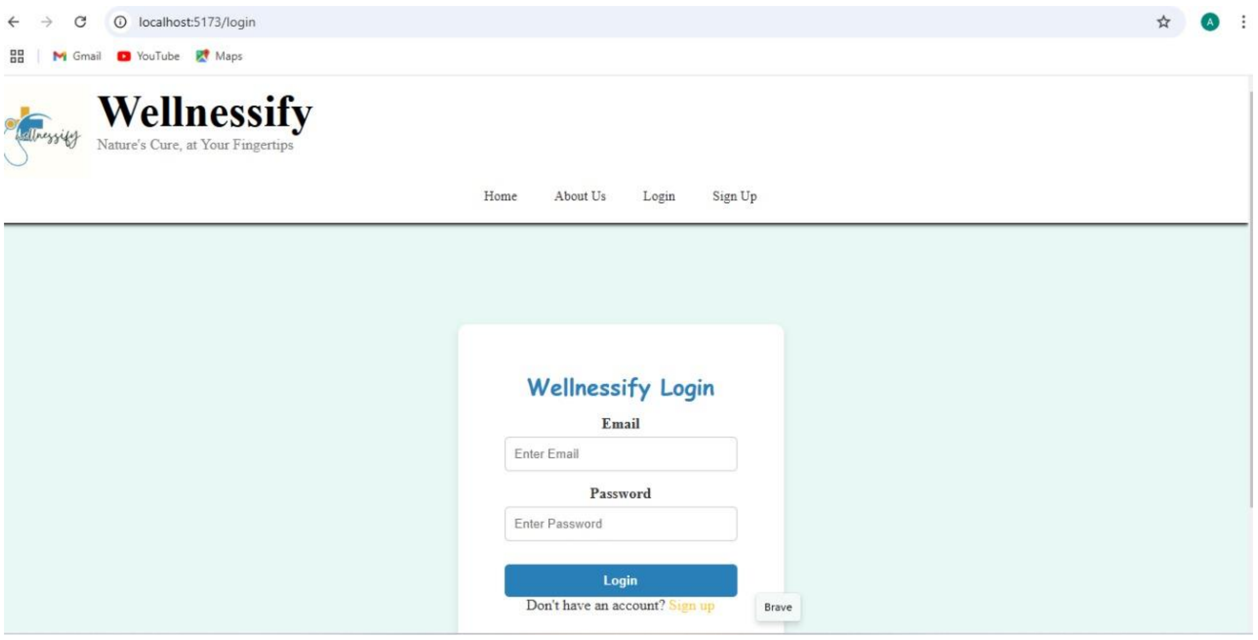


Image-2

Symptom Input Page: A clean and intuitive form for entering health symptoms.

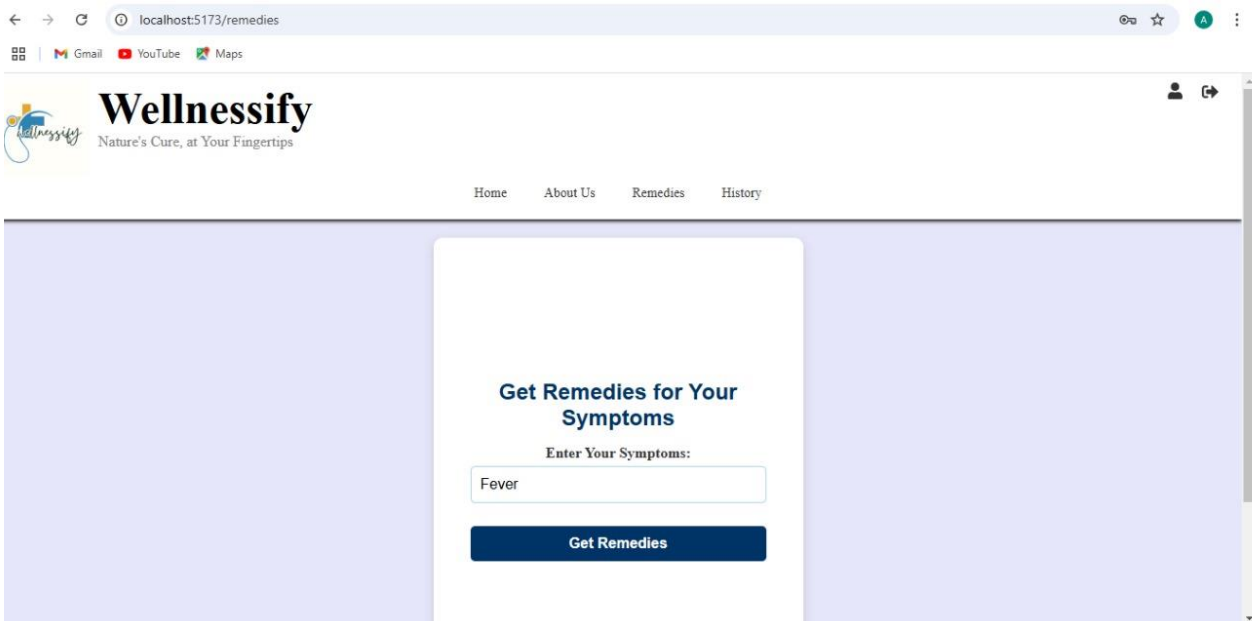
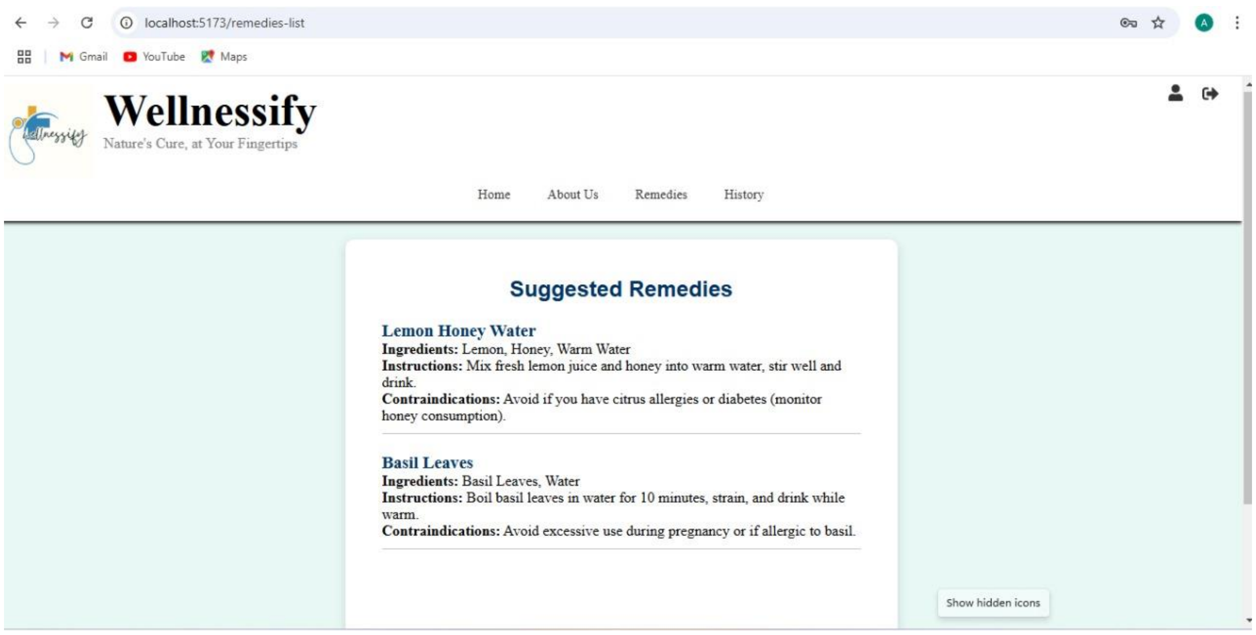


Image-3

Remedy Results Page: Detailed results with step-by-step instructions for home remedies.



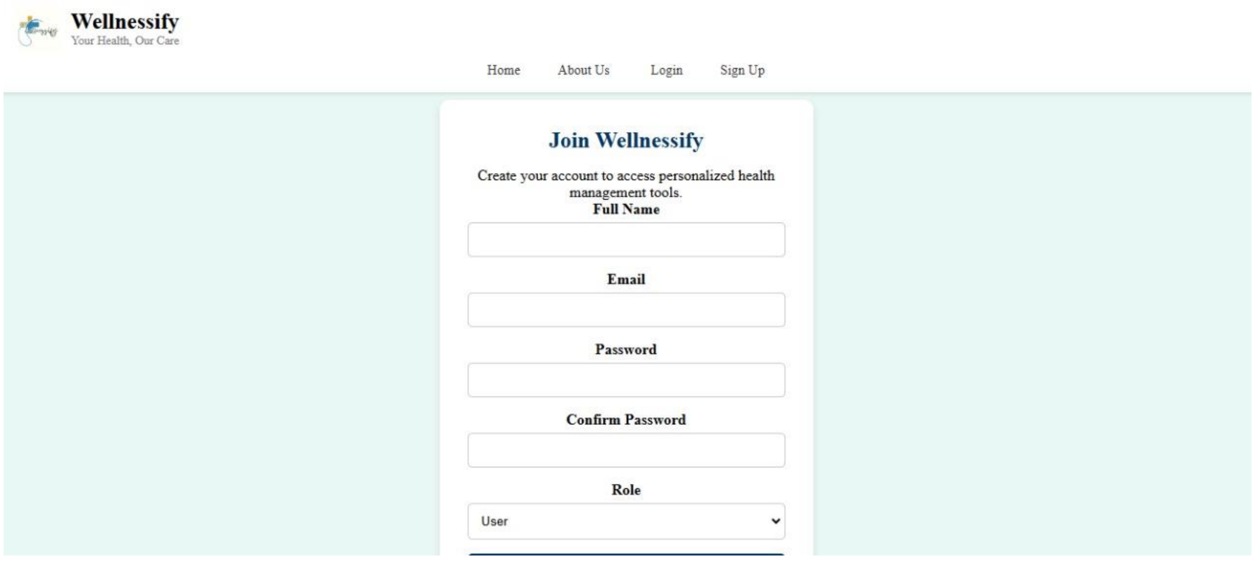
Suggested Remedies

Lemon Honey Water
Ingredients: Lemon, Honey, Warm Water
Instructions: Mix fresh lemon juice and honey into warm water, stir well and drink.
Contraindications: Avoid if you have citrus allergies or diabetes (monitor honey consumption).

Basil Leaves
Ingredients: Basil Leaves, Water
Instructions: Boil basil leaves in water for 10 minutes, strain, and drink while warm.
Contraindications: Avoid excessive use during pregnancy or if allergic to basil.

Image-4

Signup page



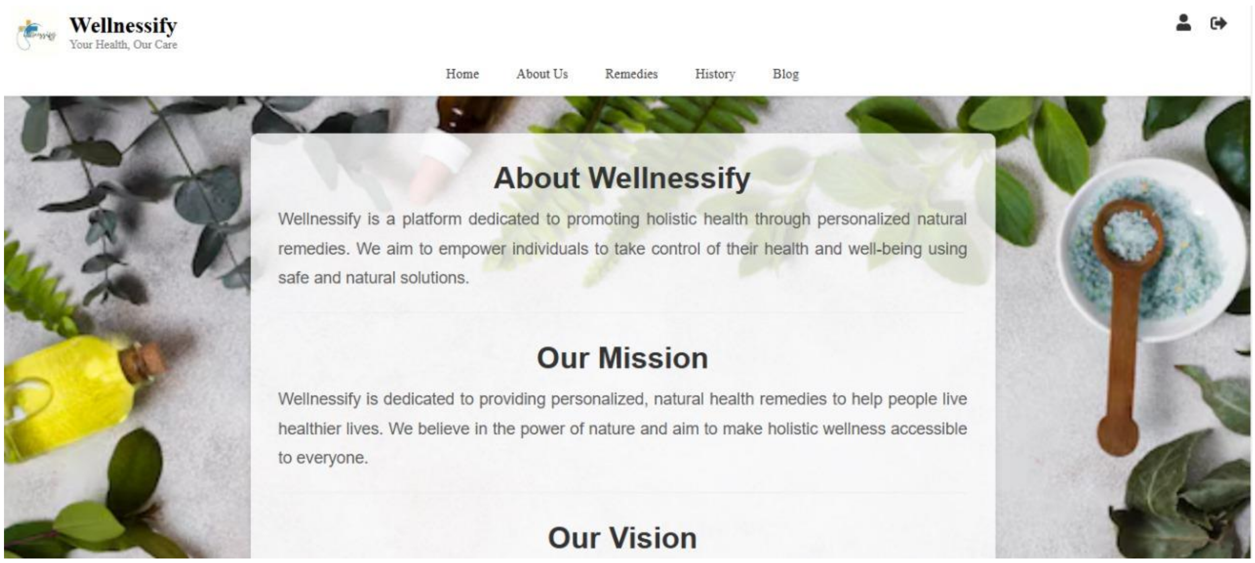
Join Wellnessify

Create your account to access personalized health management tools.

Full Name	
Email	
Password	
Confirm Password	
Role	User

Image-5

About us page



About Wellnessify

Wellnessify is a platform dedicated to promoting holistic health through personalized natural remedies. We aim to empower individuals to take control of their health and well-being using safe and natural solutions.

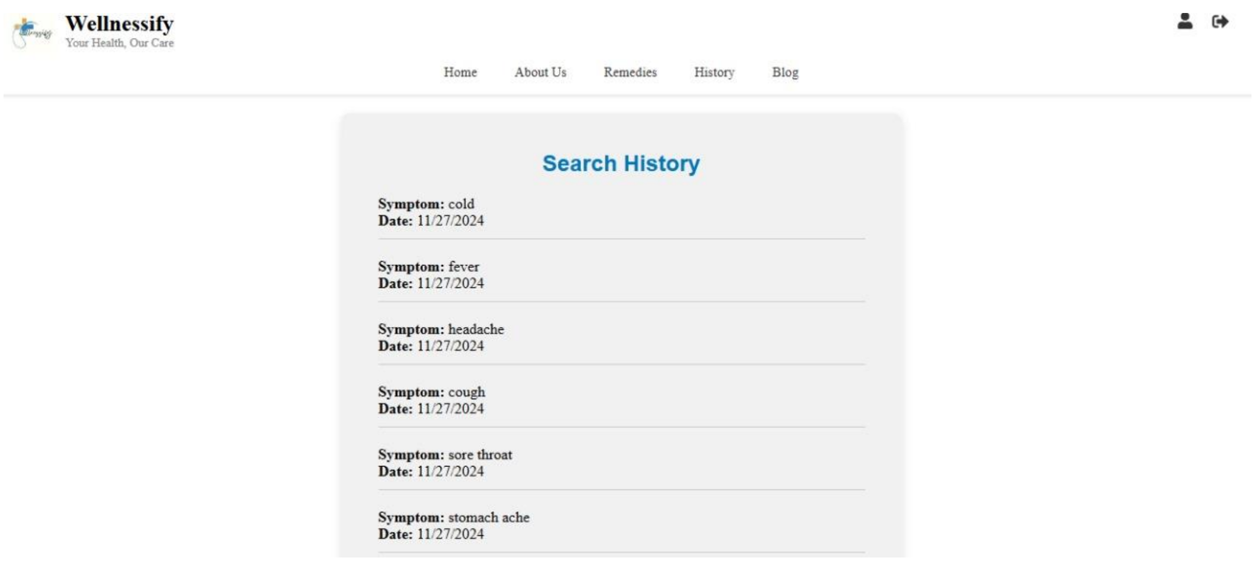
Our Mission

Wellnessify is dedicated to providing personalized, natural health remedies to help people live healthier lives. We believe in the power of nature and aim to make holistic wellness accessible to everyone.

Our Vision

Image-6

History page



Search History

Symptom: cold
Date: 11/27/2024

Symptom: fever
Date: 11/27/2024

Symptom: headache
Date: 11/27/2024

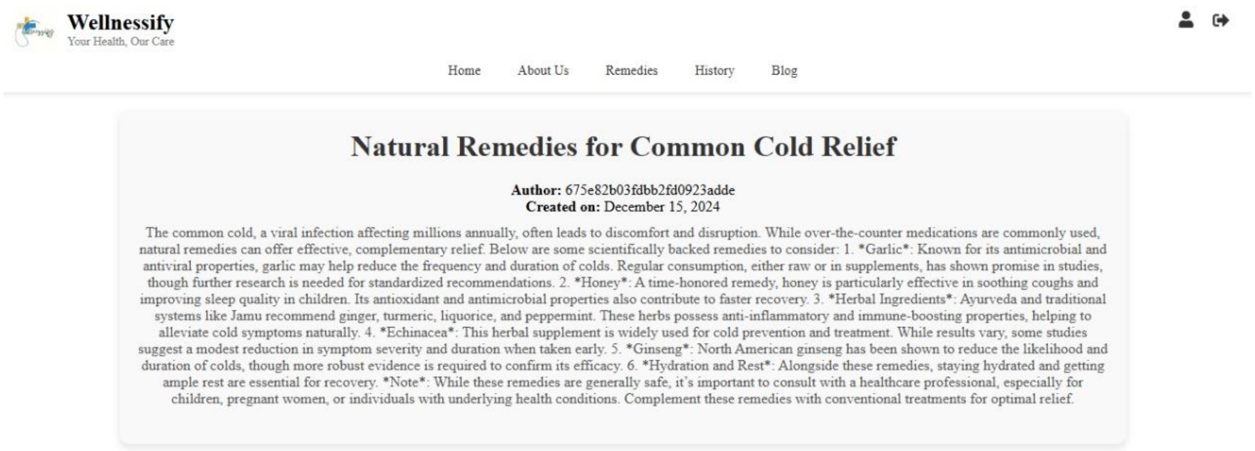
Symptom: cough
Date: 11/27/2024

Symptom: sore throat
Date: 11/27/2024

Symptom: stomach ache
Date: 11/27/2024

Image-7

Blog page



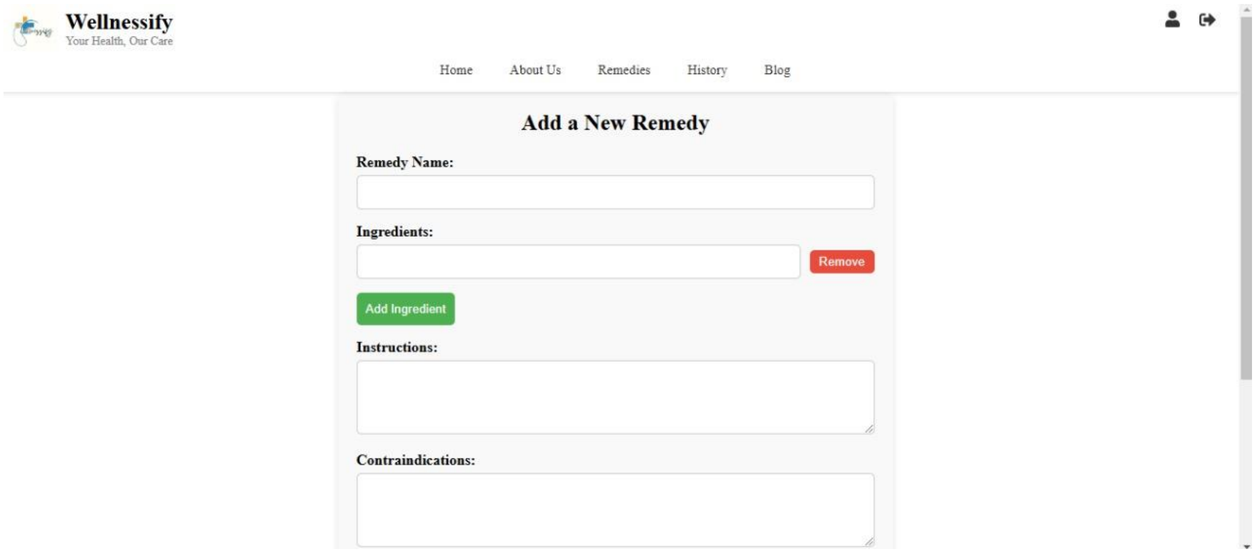
Natural Remedies for Common Cold Relief

Author: 675e82b03fdbb2fd0923adde
Created on: December 15, 2024

The common cold, a viral infection affecting millions annually, often leads to discomfort and disruption. While over-the-counter medications are commonly used, natural remedies can offer effective, complementary relief. Below are some scientifically backed remedies to consider: 1. *Garlic*: Known for its antimicrobial and antiviral properties, garlic may help reduce the frequency and duration of colds. Regular consumption, either raw or in supplements, has shown promise in studies, though further research is needed for standardized recommendations. 2. *Honey*: A time-honored remedy, honey is particularly effective in soothing coughs and improving sleep quality in children. Its antioxidant and antimicrobial properties also contribute to faster recovery. 3. *Herbal Ingredients*: Ayurveda and traditional systems like Jamu recommend ginger, turmeric, liquorice, and peppermint. These herbs possess anti-inflammatory and immune-boosting properties, helping to alleviate cold symptoms naturally. 4. *Echinacea*: This herbal supplement is widely used for cold prevention and treatment. While results vary, some studies suggest a modest reduction in symptom severity and duration when taken early. 5. *Ginseng*: North American ginseng has been shown to reduce the likelihood and duration of colds, though more robust evidence is required to confirm its efficacy. 6. *Hydration and Rest*: Alongside these remedies, staying hydrated and getting ample rest are essential for recovery. *Note*: While these remedies are generally safe, it's important to consult with a healthcare professional, especially for children, pregnant women, or individuals with underlying health conditions. Complement these remedies with conventional treatments for optimal relief.

Image -8

Admin remedy input page



Add a New Remedy

Remedy Name:

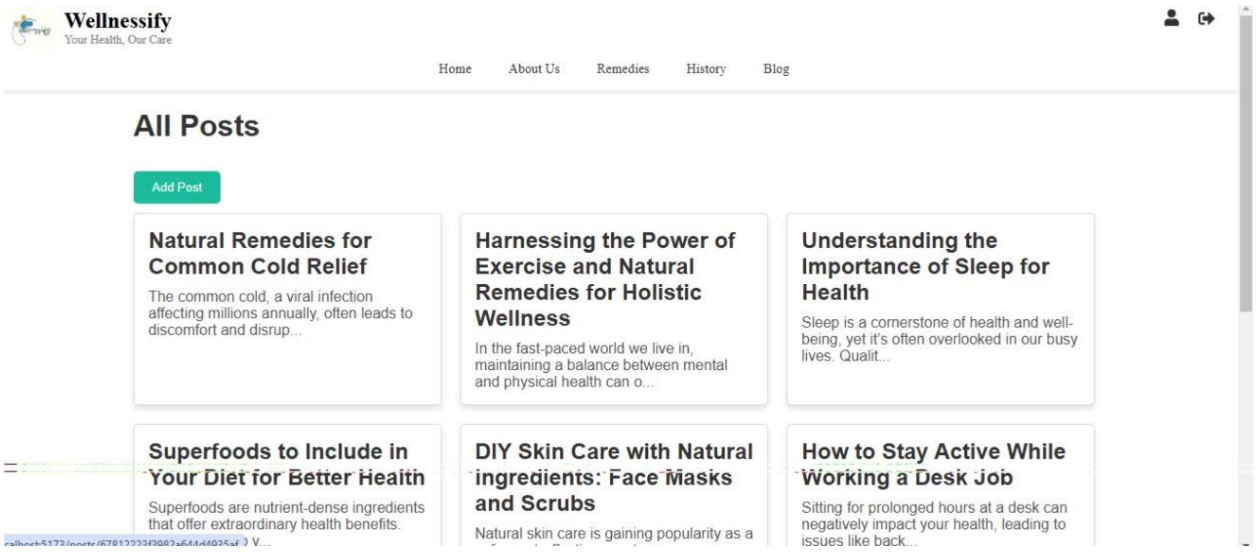
Ingredients:

Instructions:

Contraindications:

Image-9

Admin blog input page



All Posts

Natural Remedies for Common Cold Relief
The common cold, a viral infection affecting millions annually, often leads to discomfort and disrup...

Harnessing the Power of Exercise and Natural Remedies for Holistic Wellness
In the fast-paced world we live in, maintaining a balance between mental and physical health can o...

Understanding the Importance of Sleep for Health
Sleep is a cornerstone of health and well-being, yet it's often overlooked in our busy lives. Qualit...

Superfoods to Include in Your Diet for Better Health
Superfoods are nutrient-dense ingredients that offer extraordinary health benefits.

DIY Skin Care with Natural Ingredients: Face Masks and Scrubs
Natural skin care is gaining popularity as a

How to Stay Active While Working a Desk Job
Sitting for prolonged hours at a desk can negatively impact your health, leading to issues like back...

Project Outcome

1.High User Satisfaction and User Engagement:

The platform receives favourable reviews due to its unique user centered design. Due to the self-explanatory interface, even non-technical users are able to quickly learn how to navigate the available tools and resources. Considerable sentiments were also expressed towards the platform in relation to the ease of use that enables efficient provision of tailored healthcare solutions. It transcends among the tech-savvy to the less acquainted with digital health solutions as there is a consistent use and computer and mobile literacy contributes to the ease of use.

Focusing on the engagement metrics like active user retention, session length and usage rates, all these have increased drastically due to the above user engagement factors. This indicates that such participants did not just abandons their participation after the initial engagement however, they consistently revisit the site after engaging and this is a major indicator of long-term satisfaction and further site value.

2. Scalability :

Flexible Infrastructure: The easy to scale architecture of the platform makes it easy to add any additional medical procedures, offerings and custom solutions. When introducing new treatment options for instance, the system is easily able to incorporate such additions amongst other features without compromising the overall user experience. The site encourages users to explore and adopt more of a wholesome and healthy lifestyle. In support of broader health demands which are based on care, avoidance and sustainability, this could help reduce the future need for drugs.

Health And Community Education: Besides offering solutions, the site is a learning platform which empowers individuals to manage their wellness. It also provides educational resources on the benefits of various modalities, how they work and their role in the prevention and management of disease. It is in this aspect that the education component enhances the knowledge and health seeking behaviour of the users by increasing the awareness of health issues and encouraging users to live healthier lives. This expanded version offers a better detailed interpretation of the findings, providing insight into the relationships between user interaction and scalability.

Regional Particularities: Another characteristic that testifies to the platform of the system to be expanded in further regions is the ability to customize the platform in accordance with the specifics of different countries or regions. The platform allows for localization and regional customization because certain health problems, available treatment methods and health laws are often specific for particular regions. These might encompass unique medications, cultural languages, or localized health material. This flexibility assures that a range of user groups can be accommodated, thus increasing the potential of the platform to network across different geographical borders.

Result

The targets of the Wellnessify platform are achieved through offering necessary features that effectively welcome customer's concerns of health and wellness management. These include:

1. Accurate Advice Based on Symptoms Presented by Patients:

Personalized Cures: The recognition and usefulness of the platform are mainly through its ability to provide patients individualized health advice that correlates to the health problems that patients report. Users may experience difficulties and excessive frustration when trying to fill in their health problems. The platform is therefore designed to process the data input in a way that facilitates suitable answers to the user's health issues. To give users personalized recommendations, the system uses sophisticated algorithms that take into account the specific medical, social and environmental history of each user.

Suggestions: The Wellnessify recommendation system prevents misapplication of the treatments by ensuring that the suggestions are closely related to the user's current health status. This increases the chances of good outcomes and also reduces the uncertainty that is sometimes associated with conventional methods of self-medication.

Custom Health Profiles: Through the assistance of its health profile system, Wellnessify is able to track the progress of a user over time. The platform can modify the next recommendation based on previously provided feedback, when users enter their symptoms and what results they achieved which makes recommendations even more precise and individualized. Such an individualized approach progressively increases the level of engagement and the level of trust towards systems' recommendations.

2. Extensive Collection of Remedies:

Chosen and Evidence Based Remedies: A team of professionals routinely reviews and updates Wellnessify's portfolio of home remedies which includes only the vetted and tried out remedies. The website combines modern science and traditional medicine so that any proposal is effective and not only.

With data from clinical trials, research and expert's insights, every treatment in the database is trustworthy and has been clinically validated.

Many Treatments: The database contains an exhaustive list of procedures for the treatment of some common problems such as over-stress, skin diseases, weaknesses in digestion, and several others. To make the selection of the appropriate remedy easier, remedies are classified by the ailment and preference of the user. The platform also includes detailed descriptions on how to prepare and use each remedy, possible side effects and some expected outcomes.

Author Contribution

Team Composition and Responsibilities

1. **Team Lead: Aditi Kumari**
    - Played a pivotal role in overseeing the entire project development process.
    - Ensured that project milestones were achieved on time, maintaining the alignment with the set objectives.
    - Actively facilitated innovative solutions by promoting creative brainstorming sessions within the team.
    - Acted as the primary liaison between team members, ensuring smooth communication and resolving any challenges encountered during the development.
2. **Frontend Developers: Aastha Singh, Aditi Kumari, Abhismita Nayak**
    - Collaboratively designed and implemented an intuitive, user-friendly interface for the platform.
    - Focused on ensuring that the user experience was seamless and visually appealing.
    - Integrated responsive design principles, making the platform accessible across multiple devices and screen sizes.
    - Worked on creating dynamic elements to enhance interactivity and improve usability for end-users.
3. **Backend Developers: Guide Help (Mentor)**
    - Developed robust algorithms for accurate symptom matching and efficient retrieval of appropriate home remedies.
    - Focused on optimizing server-side logic for faster response times and seamless integration with the frontend.
    - Ensured the scalability and reliability of the backend systems to handle user queries effectively.
4. **Database Specialists: Aastha Singh, Aditi Kumari**
    - Took responsibility for curating an extensive and reliable database of home remedies.
    - Ensured the accuracy, consistency, and integrity of the data stored within the platform.
    - Regularly updated the database to include new remedies and refine existing entries based on user feedback.
    - Implemented efficient indexing techniques to improve query performance.
5. **Researchers: Abhismita Nayak**
    - Conducted comprehensive research to validate the remedies incorporated into the platform.
    - Ensured that all remedies were backed by credible sources and aligned with safety standards.
    - Collaborated with the database specialists to ensure accurate representation of the researched information.
    - Stayed updated with the latest advancements in health and wellness to enhance the database continuously.

References

articles

1. Pal, S.K., & Shukla, Y. (2002). Herbal Medicine: Current Status and the Future. Asian Pacific Journal of Cancer Prevention.

2. Winslow, L.C., & Kroll, D.J. (1998). Herbs as Medicine. Archives of Internal Medicine.

3. Vickers, A., & Zollman, C. (1999). ABC of Complementary Medicine: Herbal Medicine. BMJ.

4. Gupta, L.M., & Raina, R. (1998). Side Effects of Some Medicinal Plants. Current Science.

journels

5. Kamboj, V.P. (2000). Herbal Medicine. Current Science**.**

6. World Health Organization (1998). Quality Control Methods for Medicinal Plants Materials. Geneva.

7. Boullata, J.I., & Nace, A.M. (2000). Safety Issues with Herbal Medicine. Pharmacotherapy.

8. Murray, M.T., & Pizzorno, J.E. (2000). Botanical Medicine - A Modern Perspective. Churchill Livingstone.

9. Ernst, E. (2000). Herbal Medicines: Where is the Evidence? BMJ.

10. Cox, P.A. (2000). Will Tribal Knowledge Survive the Millennium? Science.

links

https://pmc.ncbi.nlm.nih.gov/articles/PMC8642800/

https://link.springer.com/chapter/10.1007/978-3-030-58975-2_1#:~:text=The%20resurgence%20in%20recognizing%20medicinal%20plants%20as,mainstream%20world%20over%20during%20the%20last%20few

https://pmc.ncbi.nlm.nih.gov/articles/PMC7114233/