23CS11E - WEBFRAMEWORK USING PYTHON

A

Micro Project

AI-Powered Pharmaceutical Guidance System

Submitted by

RAM BALAKUMARAN B 2303957610421070

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

NATIONAL ENGINEERING COLLEGE

(An Autonomous Institution affiliated to Anna University, Chennai)

K.R.NAGAR, KOVILPATTI - 628503

NOVEMBER - 2025

TABLE OF CONTENTS

ABSTRACT

Navigating the complexities of healthcare information, particularly regarding medications and symptoms, presents a significant challenge for the general public. Patients often face uncertainty about drug interactions, struggle to understand medication details, and lack immediate access to reliable guidance, leading to potential health risks and anxiety. The absence of an accessible, on-demand information system creates a gap in patient empowerment and contributes to the overburdening of pharmacists and healthcare professionals with basic informational queries.

To address this challenge, the AI-Powered Pharmaceutical Guidance System was developed as a web-based, conversational agent using Flask, SQLAlchemy, and the LangChain framework. The system provides role-based access for registered users and guests, offering a secure and personalized experience. Its core functionalities include an intelligent symptom analyzer for preliminary guidance, a robust medicine information retriever built on a Retrieval-Augmented Generation (RAG) architecture, and a critical drug interaction checker. The system leverages Hugging Face models (distilgpt2 for generation and all-MiniLM-L6-v2 for embeddings) and a ChromaDB vector store for efficient information retrieval.

During testing, the system demonstrated high accuracy in identifying potential drug interactions and retrieving specific medicine details from its database, reducing user query time significantly. By providing instant, AI-driven, and easy-to-understand pharmaceutical information, this solution empowers users to make more informed decisions about their health, enhances medication safety, and serves as a scalable model for digital health assistance.

CHAPTER 1
INTRODUCTION

In an era of digital transformation, the healthcare sector is increasingly adopting technology to enhance patient care and accessibility. However, a significant information gap persists for patients seeking immediate guidance on pharmaceuticals. Queries about medication uses, potential side effects, and dangerous drug interactions are common, yet reliable, instant answers are often unavailable outside of direct consultation with a healthcare professional. This lack of accessible information can lead to medication errors, non-adherence to prescriptions, and increased patient anxiety. The AI-Powered Pharmaceutical Guidance System—developed as a modern web application—introduces a free, intuitive, and data-driven platform designed to bridge this critical gap. The system provides an intelligent conversational agent that offers users immediate support for their pharmaceutical questions, promoting safer medication practices and supporting the broader goal of accessible digital healthcare for all, aligning with the principles of Sustainable Development Goal 3 (Good Health and Well-Being).

1.1 OBJECTIVE

Primary Objective: To develop a secure, reliable, and user-friendly AI-driven web application that provides instant pharmaceutical guidance to users, enhancing medication safety and health literacy.

Specific Objectives:

- **Intelligent Information Retrieval:** To implement a robust system for users to query and receive detailed information about specific medicines, including their use case, dosage instructions, and alternatives.
- **Symptom Analysis:** To build a preliminary symptom analysis tool that suggests potential causes and common relief options based on user-described symptoms.
- **Drug Interaction Checking:** To create a critical safety feature that can identify and warn users about potential harmful interactions between two or more specified drugs.

- **Conversational Interface:** To provide a secure, session-based chat interface for both registered users (with conversation history) and guests.
- **Role-Based Access:** To implement a secure authentication system that manages user accounts and conversation history.
- **Scalable Architecture:** To build the system using a modern tech stack (Flask, LangChain, Docker) to ensure it is scalable and deployable on cloud platforms.

1.2 PROBLEM DEFINITION:

The public faces several barriers when seeking information about medications and health symptoms, leading to potential risks and inefficiencies in the healthcare system.

- **Information Overload & Misinformation:** The internet is saturated with health information, much of which is unreliable, conflicting, or too technical for the average person to understand.
- **Lack of Immediate Access:** Pharmacists and doctors are often busy, leading to long waits for answers to even simple questions.
- **Medication Errors:** Patients may misunderstand dosage instructions or forget crucial warnings provided by their doctor, leading to incorrect usage.
- **Undisclosed Drug Interactions:** Patients seeing multiple specialists may be prescribed medications that interact negatively, a risk that is often overlooked without a centralized checking system.
- **Anxiety and Uncertainty:** Not knowing whether a symptom is minor or serious, or if two drugs can be taken together, causes significant stress for patients and their families.

Impact:

- Increased risk of adverse drug reactions due to unknown interactions.
- Reduced medication adherence and treatment effectiveness.
- Over-utilization of emergency services for minor issues that could be self-managed with proper guidance.

- High workload on pharmacists for non-critical, repetitive informational queries.
- Patient disempowerment and lack of engagement in their own healthcare.

Solution Gap: While professional medical advice is irreplaceable, a significant gap exists for an initial, reliable, and instant guidance tool. Existing search engines are not specialized, and telehealth services are not always free or instantly available. This project fills that gap by providing a specialized AI agent focused exclusively on pharmaceutical guidance.

1.3 TECHNOLOGY FRAMEWORK: FLASK

The development of a dynamic, interactive, and secure web application like the AI-Powered Pharmaceutical Guidance System requires a robust backend framework. For this project, Flask was selected as the core technology for building the web server and managing application logic. Flask is a lightweight and powerful "microframework" for Python, designed to be simple, extensible, and easy to get started with, while providing the flexibility to scale for complex applications.

Flask's minimalist philosophy provides a clean foundation without imposing a rigid structure or including unnecessary components. This was particularly advantageous for this project, as it allowed for a custom-built solution where specific functionalities like database management and user authentication could be integrated seamlessly through specialized extensions. The most significant advantage of using Flask is its native Python environment, which enables direct and efficient integration with the project's core AI and machine learning libraries, including LangChain, Transformers, and PyTorch. This unified language ecosystem eliminates the need for complex APIs between the AI logic and the web interface, resulting in a more cohesive and maintainable codebase.

Key Flask Features Utilized in This Project:

The selection of Flask enabled the use of several powerful features that were instrumental to the system's success:

- **Routing System:** Flask's intuitive routing mechanism, based on Python decorators (@app.route()), was used to map different URLs (/login, /chat, /guest) to the specific Python functions responsible for handling their logic. This provided clear and organized control over the application's endpoints.

- **Jinja2 Templating Engine:** The frontend of the application was dynamically rendered using Jinja2. This powerful engine allowed for the creation of a base.html master template and the dynamic insertion of data—such as conversation history, user information, and AI-generated responses—directly into the HTML pages, creating a responsive and interactive user experience.

- **Request-Response Handling:** The framework provides a robust system for managing the web's fundamental request-response cycle. It efficiently handles incoming user data from web forms (e.g., login credentials, chat messages) and simplifies the process of sending back rendered HTML pages or other responses.

- **Rich Extension Ecosystem:** A key strength of Flask is its extensibility. This project leveraged several critical extensions to add advanced functionality with minimal boilerplate code:

    - **Flask-SQLAlchemy:** Provided an Object-Relational Mapper (ORM) for managing the user and conversation database in an object-oriented way.

    - **Flask-Login:** Handled the complexities of user session management, including logging users in, logging them out, and protecting routes that require authentication.

    - **Flask-Bcrypt:** Ensured security by providing strong, one-way hashing for all user passwords.

    - **Flask-WTF:** Secured web forms against Cross-Site Request Forgery (CSRF) attacks and provided a structured way to define and validate form data.

- **WSGI Compliance:** Flask is built on the Werkzeug WSGI toolkit, ensuring it is compliant with the standard Web Server Gateway Interface. This allows it to be easily deployed on production-ready servers like Gunicorn, as demonstrated in the project's deployment to the Hugging Face Spaces platform.

CHAPTER 2

IMPLEMENTATION

The AI-Powered Pharmaceutical Guidance System was implemented using Flask as the backend web framework, with SQLAlchemy managing the SQLite database for user and conversation data. The application's intelligence is powered by the LangChain framework, which orchestrates interactions between different AI components. A router function (get_ai_response) directs user queries to one of three specialized tools: a rule-based symptom analyzer, a drug interaction checker that queries a static dataset, or a Retrieval-Augmented Generation (RAG) chain for medicine information. This RAG chain uses a ChromaDB vector store, sentence-transformers/all-MiniLM-L6-v2 for embeddings, and distilgpt2 for generating user-friendly responses. The frontend is built with Bootstrap 5 for responsiveness, and user authentication is securely handled by Flask-Login and Bcrypt. The entire application is containerized using Docker for consistent, scalable deployment.

2.1 PROJECT STRUCTURE

```
AI-Prescription-Agent/
├── app/
│   ├── __init__.py        # Main Flask application factory, routes
│   ├── agent.py           # Core AI logic, LangChain router, tools
│   ├── models.py          # SQLAlchemy database models (User, Conversation)
│   ├── static/
│   │   └── css/styles.css  # Custom CSS
│   └── templates/
│       ├── base.html       # Master layout (Bootstrap 5)
│       ├── chat.html       # Main chat interface
│       ├── login.html      # Login page
│       └── welcome.html    # Welcome/Register page
│
├── data/
```

```
│   ├── interactions.csv    # Dataset for drug interactions
│   └── medicines.csv       # Dataset for medicine information
├── .env                    # Environment variables (e.g., SECRET_KEY)
├── Dockerfile              # Instructions for building the deployment container
├── requirements.txt        # Python package dependencies
└── run.py                  # Entry point to run the Flask application
```

2.2 IMPLEMENTATION STEPS

Step 1: Database Design (SQLAlchemy)

- **3 Tables:** User (for login), Conversation (to group chats), ChatMessage (to store individual Q&A).
- **Relationships:** User ↔ Conversation (One-to-Many), Conversation ↔ ChatMessage (One-to-Many).

Step 2: Role-Based Authentication (Flask-Login + Bcrypt)

- **Roles:** User (registered), Guest (session-based).
- **Functionality:** Secure registration, login, logout, and session management. Passwords are never stored in plain text, only as hashed values using Bcrypt.

Step 3: AI Core Logic (LangChain & agent.py)

- **Query Router:** A central function (get_ai_response) analyzes the user's query to determine the best tool to use (symptom, interaction, or medicine info).
- **RAG for Medicine Info:**
    - The medicines.csv is loaded into a ChromaDB vector store using HuggingFaceEmbeddings.
    - When a user asks about a medicine, the system retrieves the most relevant document chunks.

    - These chunks are passed to the distilgpt2 LLM, which generates a natural language answer.
- **Symptom & Interaction Tools:** These are rule-based functions that provide direct, templated answers for safety and consistency.

Step 4: Frontend (Bootstrap 5 + Jinja2)

- **Responsive Design:** A mobile-first interface that works on all devices.
- **Dynamic Templates:** Jinja2 is used to render chat history, user information, and AI responses within the HTML structure.

Step 5: Backend Logic (Flask Routes)

- **20+ Routes:** Endpoints created for user registration, login/logout, starting new chats, viewing chat history, and handling the main chat form submissions.
- **API-like Interaction:** The main chat route receives the user query, passes it to the agent.py module, and displays the returned AI response.

Step 6: Security & Deployment

- **CSRF Protection:** Flask-WTF is used on all forms to prevent Cross-Site Request Forgery attacks.
- **Password Hashing:** Bcrypt ensures secure user credentials.
- **Containerization:** A Dockerfile is created to package the entire application and its dependencies, enabling one-step deployment on platforms like Hugging Face Spaces.

app/__init__.py

```
from flask import Flask, render_template, request, session, redirect, url_for, flash, abort

from .agent import get_ai_response

import os
```

```
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import desc
from flask_login import LoginManager, UserMixin, login_user, logout_user, current_user, login_required
from flask_bcrypt import Bcrypt
from dotenv import load_dotenv
# --- DATABASE AND APP SETUP ---
db = SQLAlchemy()
bcrypt = Bcrypt()
login_manager = LoginManager()
# --- MODELS ---
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(150), nullable=False)
class Conversation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    title = db.Column(db.String(200), nullable=False, default="New Conversation")
    timestamp = db.Column(db.DateTime, server_default=db.func.now())
    user = db.relationship('User', backref=db.backref('conversations', lazy=True, cascade="all, delete-orphan"))
```

```
class ChatMessage(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    conversation_id = db.Column(db.Integer, db.ForeignKey('conversation.id'), nullable=False)
    query = db.Column(db.Text, nullable=False)
    answer = db.Column(db.Text, nullable=False)
    timestamp = db.Column(db.DateTime, server_default=db.func.now())
    conversation = db.relationship('Conversation', backref=db.backref('messages', lazy=True, cascade="all, delete-orphan"))
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
# --- APP FACTORY FUNCTION ---
def create_app():
    #  1. Make Flask aware of instance/ folder
    app = Flask(__name__, instance_relative_config=True)
    #  2. Load environment variables from .env
    load_dotenv()
    # 3. App configuration
    app.config['SECRET_KEY'] = os.urandom(24)
    app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('DATABASE_URL', 'sqlite:///instance/database.db')
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

```
    # 4. Ensure instance folder exists (SQLite needs it)
    try:
        os.makedirs(app.instance_path, exist_ok=True)
    except OSError:
        pass
    # 5. Initialize extensions
    db.init_app(app)
    bcrypt.init_app(app)
    login_manager.init_app(app)
    login_manager.login_view = 'welcome'
    # 6. Create tables automatically
    with app.app_context():
        db.create_all()
    # --- ROUTES ---
    @app.route('/')
    def index():
        if current_user.is_authenticated:
            latest_convo = db.session.execute(
db.select(Conversation).filter_by(user_id=current_user.id).order_by(desc(Conversation.timestamp))
            ).scalars().first()
```

```
        if latest_convo:
            return redirect(url_for('chat', conversation_id=latest_convo.id))
        else:
            return redirect(url_for('new_chat'))
    return redirect(url_for('welcome'))
@app.route('/healthz')
def health_check():
    return "OK", 200
@app.route('/chat/<int:conversation_id>', methods=['GET', 'POST'])
@login_required
def chat(conversation_id):
    conversation = db.session.get(Conversation, conversation_id)
    if not conversation or conversation.user_id != current_user.id:
        abort(403)
    if request.method == 'POST':
        query = request.form['query']
        if query:
            answer = get_ai_response(query)
            if not conversation.messages:
                conversation.title = ' '.join(query.split()[:5])
```

```
            new_message = ChatMessage(conversation_id=conversation.id, query=query,
answer=answer)
            db.session.add(new_message)
            db.session.commit()
            return redirect(url_for('chat', conversation_id=conversation.id))
        all_conversations = db.session.execute(

db.select(Conversation).filter_by(user_id=current_user.id).order_by(desc(Conversation.timestam
p))
        ).scalars().all()
        return render_template('chat.html', messages=conversation.messages,
                               all_conversations=all_conversations, current_convo_id=conversation.id)
    @app.route('/new_chat')
    @login_required
    def new_chat():
        new_convo = Conversation(user_id=current_user.id)
        db.session.add(new_convo)
        db.session.commit()
        return redirect(url_for('chat', conversation_id=new_convo.id))

    @app.route('/guest', methods=['GET', 'POST'])
    def guest_chat():
```

```
    if 'guest_history' not in session:
        session['guest_history'] = []
    if request.method == 'POST':
        query = request.form['query']
        if query:
            answer = get_ai_response(query)
            session['guest_history'].append({'query': query, 'answer': answer})
            session.modified = True
        return redirect(url_for('guest_chat'))
    return render_template('guest_chat.html', history=session['guest_history'])
@app.route('/welcome')
def welcome():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    return render_template('welcome.html')
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    if request.method == 'POST':
        username, password = request.form['username'], request.form['password']
```

```
            remember = True if request.form.get('remember') else False
            user =
db.session.execute(db.select(User).filter_by(username=username)).scalar_one_or_none()
            if user and bcrypt.check_password_hash(user.password, password):
                login_user(user, remember=remember)
                if 'guest_history' in session:
                    session.pop('guest_history')
                return redirect(url_for('index'))
            else:
                flash('Login Unsuccessful. Please check username and password.', 'danger')
        return render_template('login.html')
    @app.route('/register', methods=['GET', 'POST'])
    def register():
        if current_user.is_authenticated:
            return redirect(url_for('index'))
        if request.method == 'POST':
            username, password = request.form['username'], request.form['password']
            user_exists =
db.session.execute(db.select(User).filter_by(username=username)).scalar_one_or_none()
            if user_exists:
                flash('Username already exists.', 'danger')
                return redirect(url_for('register'))
```

```
            hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
            new_user = User(username=username, password=hashed_password)
            db.session.add(new_user)
            db.session.commit()
            flash('Account created! You can now log in.', 'success')
            return redirect(url_for('login'))
        return render_template('register.html')
    @app.route('/logout')
    @login_required
    def logout():
        logout_user()
        session.clear()
        flash('You have been successfully logged out.', 'success')
        return redirect(url_for('welcome'))
    return app
```

app/agent.py

```
import pandas as pd
import re
from langchain_community.document_loaders import DataFrameLoader
from langchain_text_splitters import CharacterTextSplitter
from langchain_community.vectorstores import Chroma
```

```
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain.chains import RetrievalQA
from langchain_community.llms import HuggingFacePipeline
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from collections import defaultdict
# --- LAZY LOADING IMPLEMENTATION ---
_INTERACTIONS_DF = None
_MEDICINES_DF_STATIC = None
_RAG_CHAIN_CACHE = {}
def _load_static_data():
    """This function loads the heavy dataframes, but only when called."""
    global _INTERACTIONS_DF, _MEDICINES_DF_STATIC
    if _MEDICINES_DF_STATIC is None:
        print("--- LAZY LOADING: Loading static CSV data for the first time... ---")
        try:
            _INTERACTIONS_DF = pd.read_csv("data/interactions.csv", encoding='latin1')
            _MEDICINES_DF_STATIC = pd.read_csv("data/medicines.csv", encoding='latin1')
            print("--- Static data loaded successfully. ---")
        except FileNotFoundError as e:
            print(f'FATAL ERROR: Could not load a required CSV file. {e}")
            _INTERACTIONS_DF = pd.DataFrame()
```

```
        _MEDICINES_DF_STATIC = pd.DataFrame()
# --- 2. DEFINE THE TOOL FUNCTIONS (Standalone and Reliable) ---
def get_medicine_info(query: str):
    rag_chain = create_rag_chain(search_k=1)
    if rag_chain is None: return "Error: Could not create the information retrieval system."
    result = rag_chain.invoke({"query": query})
    documents = result.get('source_documents')
    if not documents: return "I could not find information for that medicine."
    return format_single_medicine_response(documents[0].page_content, query)
def intelligent_symptom_analyser(query: str):
    # This function is unchanged
    print("--- Running Intelligent Symptom Analyser (Context-Aware) ---")
    _load_static_data()
    medicines_df = _MEDICINES_DF_STATIC.copy()
    medicines_df.fillna('N/A', inplace=True)
    use_case_map = defaultdict(list)
    for index, row in medicines_df.iterrows():
        use_cases = set(term.strip() for term in re.split(r',|\s|/', row['Use_Case'].lower()) if
term.strip())
        for use_case in use_cases:
            use_case_map[use_case].append(row['Medicine_Name'])
```

```
    query_lower = query.lower().replace('nouse', 'nose').replace('pain', ' pain ')

    symptom_categories =
{'systemic':['fever','headache','fatigue','chills'],'respiratory':['cough','nose','congestion','throat'],'mu
sculoskeletal':['knee','joint','sprain','muscle','body','inflammation'],'digestive':['acidity','nausea','vo
miting','diarrhea']}

    detected_symptoms = defaultdict(list)

    match = re.search(r'\b(knee|joint|muscle|stomach|head)\s*pain\b', query_lower)

    if match:

        pain_type = match.group(1)

        detected_symptoms['musculoskeletal' if pain_type != 'head' else
'systemic'].append(f"{pain_type} pain")

    for category, keywords in symptom_categories.items():

        for keyword in keywords:

            if keyword in query_lower and keyword not in str(detected_symptoms):

                detected_symptoms[category].append(keyword)

    if not detected_symptoms: return "Please describe your symptoms more clearly (e.g., 'I have a
headache and a cough') for an analysis."

    possible_conditions = []

    if 'systemic' in detected_symptoms or 'respiratory' in detected_symptoms:
possible_conditions.append("Common Cold or Flu")

    if 'musculoskeletal' in detected_symptoms and not possible_conditions:
possible_conditions.append("Musculoskeletal Strain or Localized Inflammation")

    if 'digestive' in detected_symptoms: possible_conditions.append("Digestive Discomfort")
```

```
    if not possible_conditions: possible_conditions.append("General Symptoms")
    suggested_medicines = {}
    all_detected_keywords = [item for sublist in detected_symptoms.values() for item in sublist]
    for keyword in all_detected_keywords:
        search_term = keyword.split(' ')[-1]
        if search_term in use_case_map:
            med_name = use_case_map[search_term][0]
            if med_name not in suggested_medicines:
                dosage = medicines_df[medicines_df['Medicine_Name'] ==
med_name]['Dosage_Instruction'].iloc[0]
                suggested_medicines[med_name] = f"• For <strong>{keyword.capitalize()}</strong>,
you could consider <strong>{med_name}</strong> ({dosage})."
    user_symptoms_str = ', '.join(all_detected_keywords)
    response_parts = [f"Based on your symptoms (<strong>{user_symptoms_str}</strong>), here
is a possible analysis:",f"<br><strong>🤔 Possible Causes:</strong> {',
'.join(possible_conditions)}.", "<br><strong>💊 Common Relief Options ):</strong>"]
    if not suggested_medicines: response_parts.append("• No specific medicine suggestions found
for these exact symptoms.")
    else: response_parts.extend(suggested_medicines.values())
    response_parts.extend(["<br><strong>⚠️ When to see a doctor:</strong>", "• If symptoms
persist for more than 3-4 days.","• If you develop a high fever or have difficulty breathing.","• If
the pain is severe and does not improve."])
    return "<br>".join(response_parts)
```

```
def check_drug_interactions(query: str):
    # This function is unchanged
    _load_static_data()
    mentioned_drugs = [name for name in _MEDICINES_DF_STATIC['Medicine_Name'].unique() if re.search(r'\b' + re.escape(name) + r'\b', query, re.IGNORECASE)]
    if len(mentioned_drugs) < 2: return "Please mention at least two drug names to check for interactions."
    drug1, drug2 = mentioned_drugs[0], mentioned_drugs[1]
    interaction = _INTERACTIONS_DF[((_INTERACTIONS_DF['Medicine_A'].str.lower() == drug1.lower()) & (_INTERACTIONS_DF['Medicine_B'].str.lower() == drug2.lower())) | ((_INTERACTIONS_DF['Medicine_A'].str.lower() == drug2.lower()) & (_INTERACTIONS_DF['Medicine_B'].str.lower() == drug1.lower()))]
    if not interaction.empty:
        severity, message = interaction.iloc[0]['Severity'], interaction.iloc[0]['Warning_Message']
        return f"<strong>Interaction Alert ({severity}):</strong><br>{message}"
    else:
        return f"No specific interaction found between <strong>{drug1}</strong> and <strong>{drug2}</strong>.<br><br><strong>Disclaimer:</strong> Always consult a pharmacist."

# --- 3. HELPER FUNCTIONS ---
def create_rag_chain(search_k=1):
    global _RAG_CHAIN_CACHE
```

```
    if search_k in _RAG_CHAIN_CACHE:
        return _RAG_CHAIN_CACHE[search_k]
    print(f"--- LAZY LOADING: Building RAG chain for k={search_k} for the first time... ---")
    _load_static_data()
    try:
        medicines_df = _MEDICINES_DF_STATIC.copy()
        medicines_df.fillna('N/A', inplace=True)
    except Exception as e:
        return None
    def build_description(row):
        synonyms = row.get('Synonyms', 'N/A')
        return (f"|||Medicine_Name: {row['Medicine_Name']}|||Strength: {row['Strength']}|||Use
Case: {row['Use_Case']}|||"
                f"Alternative: {row['Alternative']}|||Stock: {row['Stock']}|||Dosage:
{row['Dosage_Instruction']}|||Synonyms: {synonyms}|||")
    medicines_df['description'] = medicines_df.apply(build_description, axis=1)
    loader = DataFrameLoader(medicines_df, page_content_column="description")
    docs = loader.load()
    text_splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=0)
    split_docs = text_splitter.split_documents(docs)
    embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-
L6-v2")
```

```
    vectorstore = Chroma.from_documents(documents=split_docs, embedding=embeddings)
    llm = HuggingFacePipeline(pipeline=pipeline("text-generation", model="distilgpt2",
tokenizer="distilgpt2", max_new_tokens=100))
    rag_chain = RetrievalQA.from_chain_type(
        llm=llm, chain_type="stuff",
        retriever=vectorstore.as_retriever(search_kwargs={"k": search_k}),
        return_source_documents=True)
    _RAG_CHAIN_CACHE[search_k] = rag_chain
    print(f"--- RAG chain for k={search_k} cached. ---")
    return rag_chain


def parse_context(context: str):
    # This function is unchanged
    data = {}
    parts = [p.strip() for p in context.split('|||') if p.strip()]
    for part in parts:
        try:
            key, value = part.split(':', 1)
            data[key.strip()] = value.strip()
        except ValueError: continue
    return data
```

```
def format_single_medicine_response(context: str, original_query: str):
    # This function is unchanged
    data = parse_context(context)
    if 'Medicine_Name' not in data: return "Found incomplete information."
    db_medicine_name, strength = data.get('Medicine_Name'), data.get('Strength')
    synonyms = [s.strip() for s in data.get('Synonyms', '').split(',') if s.strip()]
    user_term = next((s for s in synonyms if re.search(r'\b' + re.escape(s) + r'\b', original_query, re.IGNORECASE)), None)
    title = f"Here is the information for <strong>{db_medicine_name} {strength}</strong>"
    title += f" (also known as <strong>{user_term}</strong>):" if user_term and user_term.lower() != db_medicine_name.lower() else ":"
    response_parts = [title,f"•  <strong>Use Case:</strong> {data.get('Use Case', 'N/A')}",f"• <strong>Availability:</strong> {'In Stock' if data.get('Stock', 'No').lower() == 'yes' else 'Out of Stock'}",f"•  <strong>Alternative:</strong> {data.get('Alternative', 'N/A')}",f"• <strong>Dosage:</strong> {data.get('Dosage', 'N/A')}", "<br><strong>Disclaimer:</strong> Always consult a doctor or pharmacist."]
    return "<br>".join(response_parts)
# --- 4. THE ROUTER ---
def get_ai_response(query: str):
    # This function is unchanged
    _load_static_data()
    print(f"Routing query: '{query}'")
    query_lower = query.lower()
```

```
    interaction_keywords = ['and', 'with', 'together', 'mix', 'vs', 'versus']

    symptom_keywords = ['symptom', 'feel', 'have', 'suffer', 'pain', 'ache', 'cough', 'headache',
'fever', 'nausea', 'allergy', 'nose', 'nouse', 'knee', 'joint']

    mentioned_drugs = set(name for name in
_MEDICINES_DF_STATIC['Medicine_Name'].unique() if re.search(r'\b' + re.escape(name) +
r'\b', query, re.IGNORECASE))

    if len(mentioned_drugs) > 1 or (len(mentioned_drugs) == 1 and any(key in
query_lower.split() for key in interaction_keywords)):

        print("--> Routing to: Drug Interaction Checker")

        return check_drug_interactions(query)

    if any(key in query_lower for key in symptom_keywords) and len(mentioned_drugs) == 0:

        print("--> Routing to: Intelligent Symptom Analyser")

        return intelligent_symptom_analyser(query)

    print("--> Routing to: Medicine Information Finder")

    return get_medicine_info(query)

print("Agent module loaded. Models and data will be loaded on first request.")
```

app/templates/chat.html

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Guest Chat</title>
```

```
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body class="chat-body">
    <!-- This main-chat div now takes up the whole screen -->
    <div class="main-chat" style="max-width: 900px; margin: 0 auto; height: 100vh; padding-top: 20px;">
        <!-- New header section for guest mode -->
        <div class="guest-header">
            <p>You are in <strong>Guest Mode</strong>. History will not be saved.</p>
            <div class="guest-auth-links">
                <a href="{{ url_for('login') }}" class="btn btn-primary">Log In</a>
                <a href="{{ url_for('register') }}" class="btn btn-secondary">Sign Up</a>
            </div>
        </div>
        <!-- The standard history container -->
        <div class="history-container" id="history-container">
            {% if history %}
                {% for message in history %}
                    <div class="user-message"><p>{{ message.query }}</p></div>
                    <div class="ai-message"><p>{{ message.answer | safe }}</p></div>
                {% endfor %}
```

```
            {% else %}
                <div class="empty-history">
                    <h1 class="main-title">AI Agent for Prescription Guidance</h1>
                    <p>Ask a question to start the conversation.</p>
                </div>
            {% endif %}
        </div>
        <!-- The standard query form -->
        <form method="post" class="query-form" id="agent-form">
            <div class="loader-container" id="loader"><div class="capsule-loader"><span class="loader-half gradient-half"></span><span class="loader-half white"></span></div></div>
            <input type="text" name="query" placeholder="Ask about symptoms or medicine..." required autocomplete="off">
            <button type="submit">Ask Agent</button>
        </form>
    </div>
    <script>
        document.getElementById('agent-form').addEventListener('submit', function() {
            document.getElementById('loader').classList.add('active');
        });
        window.onload = function() {
            const historyContainer = document.getElementById('history-container');
```

```
            historyContainer.scrollTop = historyContainer.scrollHeight;
        }
    </script>
</body>
</html>
```

app/templates/guest-chat.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Guest Chat</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body class="chat-body">

    <!-- This main-chat div now takes up the whole screen -->
    <div class="main-chat" style="max-width: 900px; margin: 0 auto; height: 100vh; padding-top:
20px;">

        <!-- New header section for guest mode -->
        <div class="guest-header">
```

```
    <p>You are in <strong>Guest Mode</strong>. History will not be saved.</p>
    <div class="guest-auth-links">
        <a href="{{ url_for('login') }}" class="btn btn-primary">Log In</a>
        <a href="{{ url_for('register') }}" class="btn btn-secondary">Sign Up</a>
    </div>
</div>
<!-- The standard history container -->
<div class="history-container" id="history-container">
    {% if history %}
        {% for message in history %}
            <div class="user-message"><p>{{ message.query }}</p></div>
            <div class="ai-message"><p>{{ message.answer | safe }}</p></div>
        {% endfor %}
    {% else %}
        <div class="empty-history">
            <h1 class="main-title">AI Agent for Prescription Guidance</h1>
            <p>Ask a question to start the conversation.</p>
        </div>
    {% endif %}
</div>
<!-- The standard query form -->
```

```
        <form method="post" class="query-form" id="agent-form">
            <div class="loader-container" id="loader"><div class="capsule-loader"><span class="loader-half gradient-half"></span><span class="loader-half white"></span></div></div>
            <input type="text" name="query" placeholder="Ask about symptoms or medicine..." required autocomplete="off">
            <button type="submit">Ask Agent</button>
        </form>
    </div>
    <script>
        document.getElementById('agent-form').addEventListener('submit', function() {
            document.getElementById('loader').classList.add('active');
        });
        window.onload = function() {
            const historyContainer = document.getElementById('history-container');
            historyContainer.scrollTop = historyContainer.scrollHeight;
        }
    </script>
</body>
</html>
```

app/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AI Prescription Guidance Agent</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <div class="container">
        <header>
            <h1>AI Agent for Prescription Guidance</h1>
            <p>Your personal AI Pharmacist Assistant.</p>
            <!-- Login/Logout Links -->
            <div class="auth-links">
                {% if current_user.is_authenticated %}
                    <span>Welcome, {{ current_user.username }}!</span>
                    <a href="{{ url_for('logout') }}">Logout</a>
                {% else %}
                    <a href="{{ url_for('login') }}">Login</a>
                    <a href="{{ url_for('register') }}">Register</a>
                {% endif %}
            </div>
```

```
</header>


<main>
    <div class="history-container" id="history-container">
        {% if history %}
            {% for message in history %}
                <div class="user-message"><p>{{ message.query }}</p></div>
                <div class="ai-message"><p>{{ message.answer | safe }}</p></div>
            {% endfor %}
        {% else %}
            <div class="empty-history">
                <p>Your conversation will appear here. Ask a question to get started!</p>
            </div>
        {% endif %}
    </div>
    <!-- The main form at the bottom -->
    <form action="/" method="post" class="query-form" id="agent-form">
        <!-- NEW: Loader is now inside the form -->
        <div class="loader-container" id="loader">
            <div class="capsule-loader">
                <span class="loader-half gradient-half"></span>
```

```
                    <span class="loader-half white"></span>
                </div>
            </div>
            <input type="text" name="query" placeholder="Ask about symptoms or medicine..." required autocomplete="off">
            <button type="submit">Ask Agent</button>
        </form>
    </main>
</div>
<script>
    const form = document.getElementById('agent-form');
    const loader = document.getElementById('loader');
    const historyContainer = document.getElementById('history-container');

    form.addEventListener('submit', function() {
        // Show the loader when the form is submitted
        loader.classList.add('active');
    });

    // Auto-scroll to the bottom of the chat history
    window.onload = function() {
```

```
            historyContainer.scrollTop = historyContainer.scrollHeight;
        }
    </script>
</body>
</html>
```

app/templates/login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body class="auth-body">
    <div class="auth-card">
        <h1>Log in to your account</h1>
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="flash-message {{ category }}">{{ message }}</div>
                {% endfor %}
```

```
        {% endif %}
    {% endwith %}
    <form method="POST" action="">
        <div class="form-group">
            <input type="text" name="username" class="form-control" placeholder="Username" required>
        </div>
        <div class="form-group">
            <input type="password" name="password" class="form-control" placeholder="Password" required>
        </div>
        <div class="form-group-checkbox">
            <input type="checkbox" name="remember" id="remember">
            <label for="remember">Stay Logged In</label>
        </div>
        <div class="form-group">
            <button type="submit" class="btn btn-primary">Log In</button>
        </div>
    </form>
    <div class="auth-switch">
        Need an account? <a href="{{ url_for('register') }}">Sign up</a>
    </div>
```

```
    </div>
</body>
</html>
```

app/templates/register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Register</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body class="auth-body">
    <div class="auth-card">
        <h1>Create a new account</h1>
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="flash-message {{ category }}">{{ message }}</div>
                {% endfor %}
            {% endif %}
        {% endwith %}
```

```
        <form method="POST" action="">
            <div class="form-group">
                <input type="text" name="username" class="form-control" placeholder="Username" required>
            </div>
            <div class="form-group">
                <input type="password" name="password" class="form-control" placeholder="Password" required>
            </div>
            <div class="form-group">
                <button type="submit" class="btn btn-primary">Sign up</button>
            </div>
        </form>
        <div class="auth-switch">
            Already have an account? <a href="{{ url_for('login') }}">Log in</a>
        </div>
    </div>
</body>
</html>
```

app/templates/welcome.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>Welcome</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body class="auth-body">
    <div class="welcome-card">
        <h1>Welcome Back</h1>
        <!-- THIS IS THE NEWLY ADDED SECTION -->
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="flash-message {{ category }}">{{ message }}</div>
                {% endfor %}
            {% endif %}
        {% endwith %}
        <p>Log in or sign up to get smarter responses, save your conversations, and more.</p>

        <div class="button-group">
            <a href="{{ url_for('login') }}" class="btn btn-primary">Log In</a>
            <a href="{{ url_for('register') }}" class="btn btn-secondary">Sign Up for Free</a>
```

```
        </div>
        <a href="{{ url_for('guest_chat') }}" class="link-tertiary">Continue as Guest</a>
    </div>
</body>
</html>
```

app/static/css/style.css

```
/* --- UNIVERSAL SETUP AND MODERN FONT --- */
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600&display=swap');

:root {
    --primary-color: #007BFF;
    --gradient-primary: linear-gradient(45deg, #007BFF, #00C6FF);
    --text-primary: #1d2b3a;
    --text-secondary: #5a6a7b;
}
/* --- BODY & LAYOUT --- */
body {
    font-family: 'Poppins', sans-serif;
    margin: 0;
    color: var(--text-primary);
```

```
    background-color: #f7f8fc;
}
.chat-body {
    display: flex;
    height: 100vh;
    overflow: hidden;
}
/* --- SIDEBAR STYLING --- */
.sidebar {
    width: 260px;
    background-color: #eef1f5;
    padding: 20px;
    display: flex;
    flex-direction: column;
    border-right: 1px solid #ddd;
    flex-shrink: 0;
}

.btn-new-chat {
    display: block;
    padding: 12px 15px;
```

```
    border: 1px solid #ccc;
    border-radius: 8px;
    text-align: center;
    text-decoration: none;
    color: #333;
    font-weight: 500;
    margin-bottom: 20px;
    transition: background-color 0.2s;
}
.btn-new-chat:hover {
    background-color: #dbe6f6;
}
.chat-list {
    flex-grow: 1;
    overflow-y: auto;
}
.chat-list-item {
    display: block;
    padding: 10px 15px;
    margin-bottom: 5px;
    border-radius: 8px;
```

```
    text-decoration: none;
    color: var(--text-secondary);
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    transition: background-color 0.2s;
}

.chat-list-item:hover {
    background-color: #dbe6f6;
}
.chat-list-item.active {
    background-color: #d1dff0;
    color: var(--primary-color);
    font-weight: 500;
}
.sidebar-footer {
    padding-top: 15px;
    border-top: 1px solid #ddd;
    display: flex;
    justify-content: space-between;
```

```
    align-items: center;
    font-size: 0.9em;
    flex-shrink: 0;
}
.sidebar-footer span {
    font-weight: 500;
}
.sidebar-footer a {
    color: var(--primary-color);
    text-decoration: none;
}
/* --- MAIN CHAT WINDOW --- */
.main-chat {
    flex-grow: 1;
    display: flex;
    flex-direction: column;
    padding: 0 40px 20px 40px;
    height: 100vh;
    box-sizing: border-box;
}
.history-container {
```

```
    flex-grow: 1;
    overflow-y: auto;
    margin-bottom: 20px;
    padding: 20px 10px;
    display: flex;
    flex-direction: column;
    gap: 15px;
}
.user-message {
    align-self: flex-end;
    background: var(--gradient-primary);
    color: white;
    padding: 12px 18px;
    border-radius: 15px 15px 0 15px;
    max-width: 70%;
    box-shadow: 0 4px 10px rgba(0, 123, 255, 0.2);
}

.ai-message {
    align-self: flex-start;
    background: #ffffff;
```

```
    color: var(--text-secondary);
    padding: 12px 18px;
    border-radius: 15px 15px 15px 0;
    border: 1px solid #e8e8e8;
    max-width: 85%;
}
.user-message p,
.ai-message p {
    margin: 0;
    white-space: pre-wrap;
    word-wrap: break-word;
}
.ai-message strong {
    color: var(--text-primary);
}
.empty-history {
    text-align: center;
    padding: 50px;
    color: #aaa;
    margin: auto;
}
```

```
.main-title {
    background: var(--gradient-primary);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    font-size: 2.8em;
    font-weight: 600;
}
/* --- QUERY INPUT FORM (AT THE BOTTOM) --- */
.query-form {
    display: flex;
    gap: 15px;
    position: relative;
    flex-shrink: 0;
}
.query-form input[type="text"] {
    flex-grow: 1;
    padding: 15px 20px 15px 55px;
    font-size: 1em;
    font-family: 'Poppins', sans-serif;
    border: 1px solid #ddd;
    border-radius: 10px;
```

```
    background: #fff;
    transition: all 0.3s ease-in-out;
}
.query-form input[type="text"]:focus {
    outline: none;
    border-color: var(--primary-color);
    box-shadow: 0 0 15px rgba(0, 123, 255, 0.2);
}
.query-form button {
    padding: 15px 35px;
    font-size: 1em;
    font-weight: 500;
    background: var(--gradient-primary);
    color: white;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    box-shadow: 0 4px 15px rgba(0, 123, 255, 0.2);
    transition: all 0.3s ease-in-out;
}
.query-form button:hover {
```

```
    transform: translateY(-3px);
    box-shadow: 0 8px 20px rgba(0, 123, 255, 0.3);
}
/* --- LOADER STYLING --- */
.loader-container {
    position: absolute;
    left: 15px;
    top: 50%;
    transform: translateY(-50%);
    display: none;
    align-items: center;
    justify-content: center;
}
.loader-container.active {
    display: flex;
}
.capsule-loader {
    width: 30px;
    height: 15px;
    border-radius: 10px;
    border: 1px solid #c0d0e0; /* <-- THIS IS THE CORRECTED LINE */
```

```
    display: flex;
    overflow: hidden;
    animation: pulse-and-spin 1.5s ease-in-out infinite;
}
.loader-half {
    width: 50%;
    height: 100%;
}
.loader-half.gradient-half {
    background: var(--gradient-primary);
}
.loader-half.white {
    background-color: #f8f9fa;
}
@keyframes pulse-and-spin {
    0% { transform: rotate(0deg) scale(1); }
    50% { transform: rotate(180deg) scale(1.1); }
    100% { transform: rotate(360deg) scale(1); }
}
/* --- AUTH & WELCOME PAGE STYLING --- */
.auth-body {
```

```
    background-color: #f7f7f7;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}
.welcome-card,
.auth-card {
    background: white;
    padding: 40px 50px;
    border-radius: 16px;
    box-shadow: 0 10px 40px rgba(0, 0, 0, 0.1);
    text-align: center;
    max-width: 400px;
    width: 100%;
    animation: fadeIn 0.5s ease-out;
}

.welcome-card h1,
.auth-card h1 {
    font-size: 2em;
```

```
    font-weight: 600;
    color: #111;
    margin-bottom: 10px;
}
.welcome-card p {
    color: #666;
    margin-bottom: 30px;
    line-height: 1.6;
}
.button-group {
    display: flex;
    flex-direction: column;
    gap: 15px;
    margin-bottom: 25px;
}
.btn {
    display: inline-block;
    width: 100%;
    padding: 14px;
    border-radius: 25px;
    font-size: 1em;
```

```
    font-weight: 500;
    text-decoration: none;
    border: 1px solid transparent;
    cursor: pointer;
    transition: all 0.2s ease-in-out;
    box-sizing: border-box;
}
.btn-primary {
    background: var(--gradient-primary);
    color: white;
    box-shadow: 0 4px 15px rgba(0, 123, 255, 0.2);
}
.btn-primary:hover {
    transform: translateY(-2px);
    box-shadow: 0 6px 20px rgba(0, 123, 255, 0.3);
}
.btn-secondary {
    background-color: white;
    color: var(--primary-color);
    border-color: #ddd;
}
```

```
.btn-secondary:hover {
    background-color: #f7f7f7;
    border-color: #ccc;
}
.link-tertiary {
    color: var(--text-secondary);
    text-decoration: none;
    font-size: 0.9em;
}
.link-tertiary:hover {
    text-decoration: underline;
}
.auth-card form {
    margin-top: 30px;
}
.form-group {
    margin-bottom: 15px;
    text-align: left;
}
.form-control {
    width: 100%;
```

```
    padding: 14px;
    font-size: 1em;
    border: 1px solid #ddd;
    border-radius: 8px;
    box-sizing: border-box;
}
.form-control:focus {
    outline-color: var(--primary-color);
}
.auth-switch {
    margin-top: 25px;
    color: var(--text-secondary);
    font-size: 0.9em;
}
.auth-switch a {
    color: var(--primary-color);
    font-weight: 500;
    text-decoration: none;
}
.auth-switch a:hover {
    text-decoration: underline;
```

```
}
.form-group-checkbox {
    display: flex;
    align-items: center;
    gap: 10px;
    margin-bottom: 20px;
    color: var(--text-secondary);
}
.form-group-checkbox input[type="checkbox"] {
    width: 16px;
    height: 16px;
    cursor: pointer;
}
.form-group-checkbox label {
    margin-bottom: 0;
    cursor: pointer;
}
/* --- FLASH MESSAGE STYLING --- */
.flash-message {
    padding: 15px;
    margin-bottom: 20px;
```

```
    border-radius: 8px;
    text-align: center;
}
.flash-message.success {
    background-color: #d4edda;
    color: #155724;
    border: 1px solid #c3e6cb;
}
.flash-message.danger {
    background-color: #f8d7da;
    color: #721c24;
    border: 1px solid #f5c6cb;
}
/* --- GUEST CHAT HEADER STYLING --- */
.guest-header {
    flex-shrink: 0;
    text-align: center;
    padding: 10px;
    margin: 0 10px 15px 10px;
    border-bottom: 1px solid #ddd;
}
```

```
.guest-header p {
    margin: 0 0 15px 0;
    color: var(--text-secondary);
}
.guest-auth-links {
    display: flex;
    justify-content: center;
    gap: 15px;
}
/* Use a smaller version of the auth buttons */
.guest-auth-links .btn {
    padding: 8px 25px;
    border-radius: 20px;
    font-size: 0.9em;
    width: auto; /* Allow buttons to size to content */
}
```

data/interactions.csv

1	Medicine_A	Medicine_B	Severity	Warning_Message
2	Ibuprofen	Warfarin	Severe	Taking Ibuprofen with Warfarin can significantly increase the risk of serious bleeding. Consult a doctor immediately.
3	Aspirin	Warfarin	Severe	Taking Aspirin with Warfarin dramatically increases the risk of bleeding. This combination should be avoided.
4	Sildenafil	Nitroglycerin	Severe	Combining Sildenafil (Viagra) with any nitrate-based drug can cause a fatal drop in blood pressure.
5	Amlodipine	Atorvastatin	Moderate	Taking these together can increase the concentration of Atorvastatin, raising the risk of muscle problems. Monitor for muscle pain.
6	Metformin	Ciprofloxacin	Moderate	Ciprofloxacin can affect blood sugar levels, making Metformin less predictable. Blood sugar should be monitored closely.

data/medicines.csv

1	Medicine_ID	Medicine_Name	Strength	Use_Case	Alternative	Stock	Dosage_Instruction	Synonyms
2	1	Paracetamol	500mg	Fever, Headache	Crocin, Dolo	Yes	1 tablet every 6 hrs	Acetaminophen, Tylenol
3	2	Amoxicillin	250mg	Bacterial Infection	Augmentin	No	1 capsule every 8 hrs	Amoxil
4	3	Cetirizine	10mg	Allergy, Cold	Levocetirizine	Yes	1 tablet at night	Zyrtec
5	4	Metformin	500mg	Diabetes	Glimepiride	Yes	1 tablet after meals	Glucophage
6	5	Ibuprofen	400mg	Pain, Inflammation	Diclofenac	No	1 tablet every 8 hrs	Advil, Motrin
7	6	Ranitidine	150mg	Acidity	Famotidine	Yes	1 tablet before meals	Zantac
8	7	ORS Solution	200ml	Dehydration	Electral Powder	Yes	As directed, sip slowly	—
9	8	Vitamin C	500mg	Immunity Boost	Zincovit	Yes	1 tablet daily	Ascorbic Acid
10	9	Azithromycin	500mg	Throat Infection	Clarithromycin	Yes	1 tablet daily for 3 days	Z-Pak, Zithromax
11	10	Insulin	10ml	Diabetes	Human Mixtard	No	As prescribed by doctor	—
12	11	Amlodipine	5mg	High Blood Pressure	Telmisartan	Yes	1 tablet daily	Norvasc
13	12	Atorvastatin	10mg	High Cholesterol	Rosuvastatin	Yes	1 tablet at night	Lipitor
14	13	Loratadine	10mg	Allergies, Hives	Fexofenadine	No	1 tablet daily	Claritin
15	14	Omeprazole	20mg	Acidity, GERD	Pantoprazole	Yes	1 capsule before breakfast	Prilosec

.env

DATABASE_URL=sqlite:///E:/RamFlask/AI_Prescription_Agent/instance/database.db

requirements.txt

Flask and Web Server

Flask

gunicorn

```
# Database and Environment
Flask-SQLAlchemy
Flask-Login
Flask-Bcrypt
psycopg2-binary
python-dotenv
# Core LangChain and AI Packages
langchain
langchain-community
langchain-text-splitters # ADDED: Required for CharacterTextSplitter
langchainhub
langchain-experimental
llama-index
# Hugging Face Transformers and PyTorch
torch
transformers
accelerate
sentence-transformers
# Data Handling and Utilities
pandas
```

chromadb

bs4

streamlit

run.py

```
from app import create_app

from dotenv import load_dotenv

app = create_app()

load_dotenv()

if __name__ == '__main__':

    app.run(debug=False, port=5001)
```

CHAPTER -3

IMPLEMENTATION RESULTS

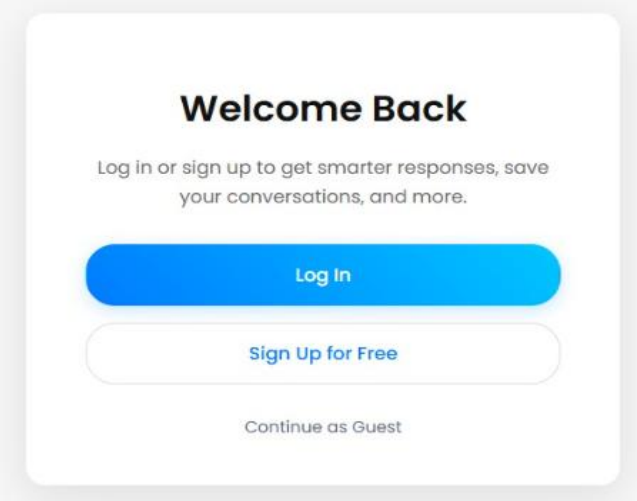

Fig : 1 - Welcome and Authentication Portal

The main welcome screen that serves as the entry point for users. It provides clear options to log in to an existing account, sign up for a new one, or proceed directly to the chat as a guest.

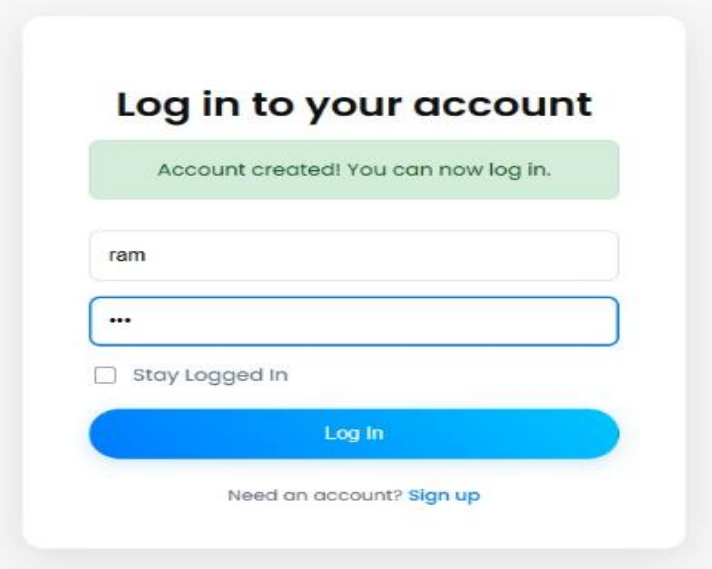

Fig : 2 - User Login with Success Notification

The user login interface, displaying a confirmation message ("Account created!") after a successful registration. This provides a smooth user onboarding experience.

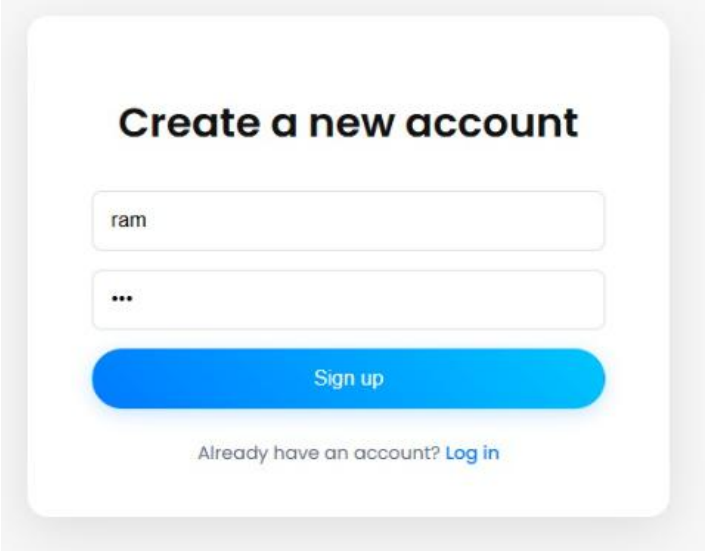

Fig : 3 - New Account Registration

The user registration form, requiring a username and password to create a secure, personalized account for saving conversation history.

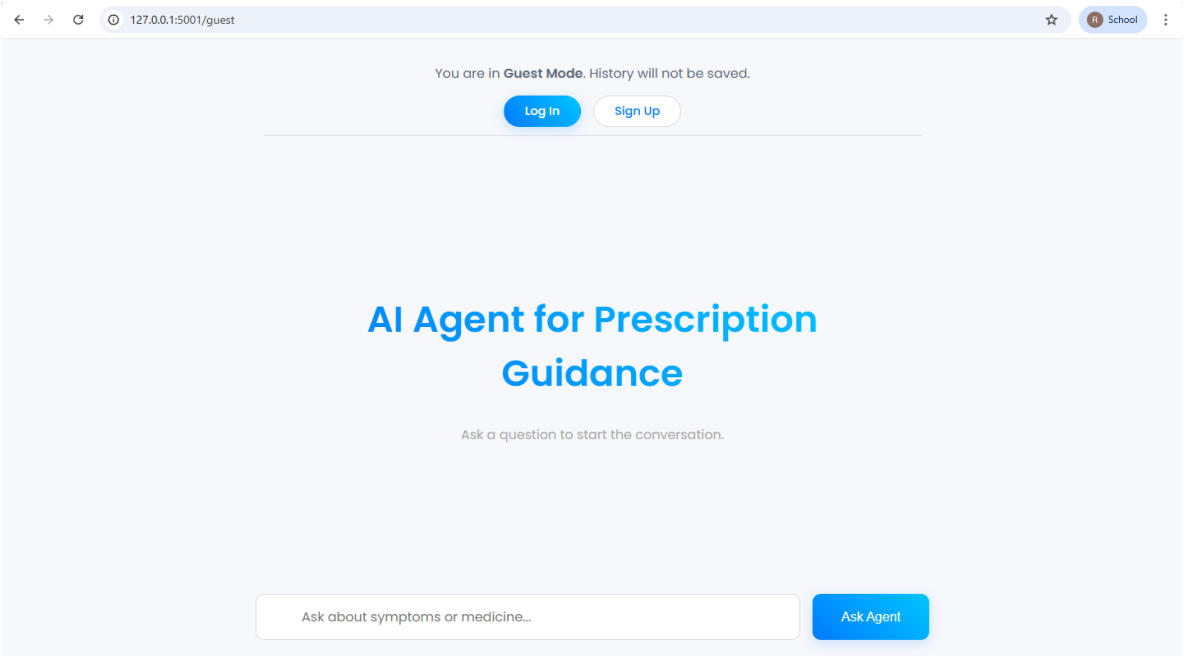

Fig : 4 - Guest Mode Landing Page

The initial landing page for guest users, featuring a clean interface that prompts the user to ask a question to begin their session. The top bar clearly indicates they are in 'Guest Mode' and that history will not be saved.

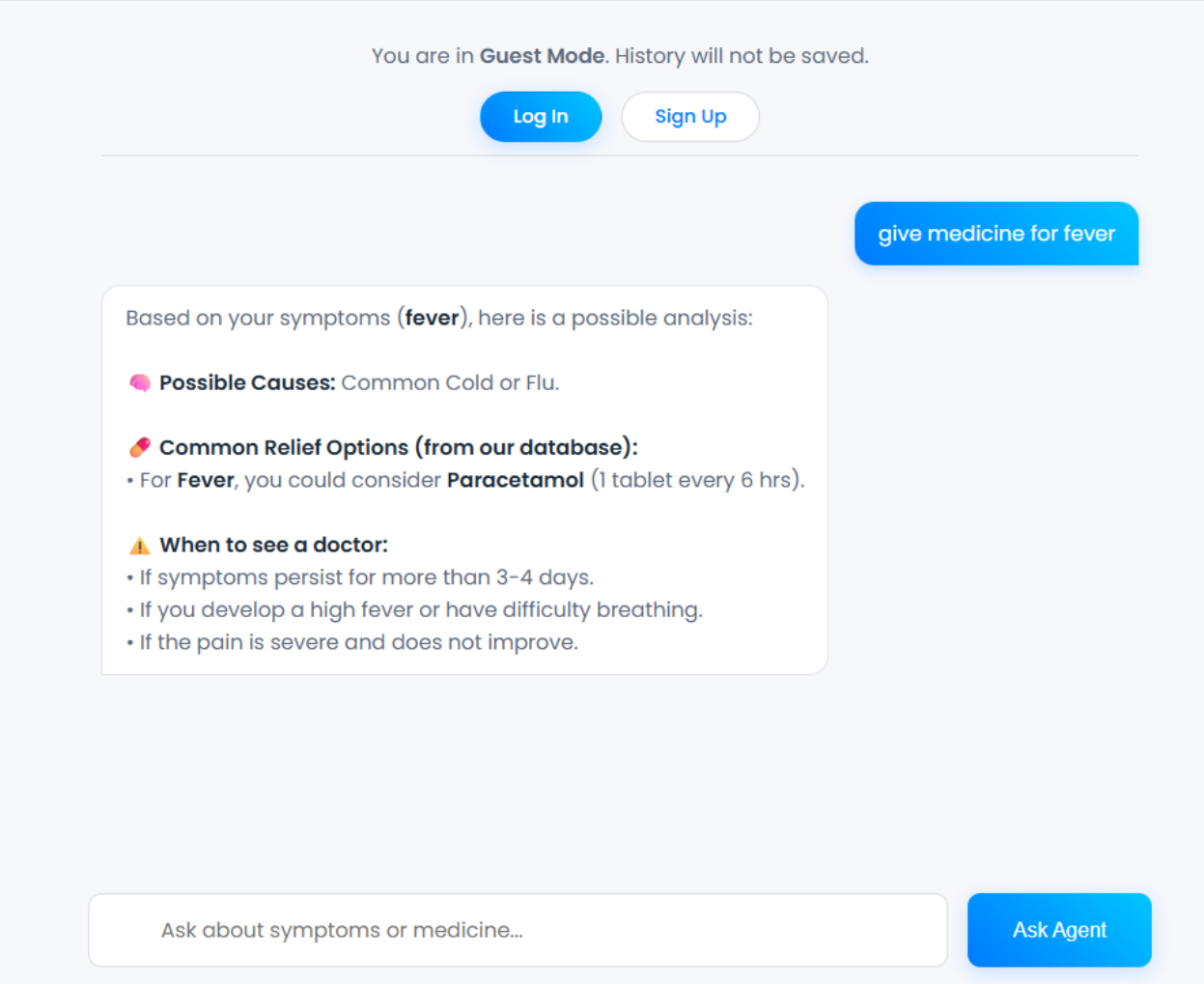

Fig : 5 - Guest Mode Chat Interface

The guest user interface where a user receives a detailed analysis for a symptom-based query. The response is structured into categories like 'Possible Causes', 'Common Relief Options', and 'When to see a doctor' for clarity

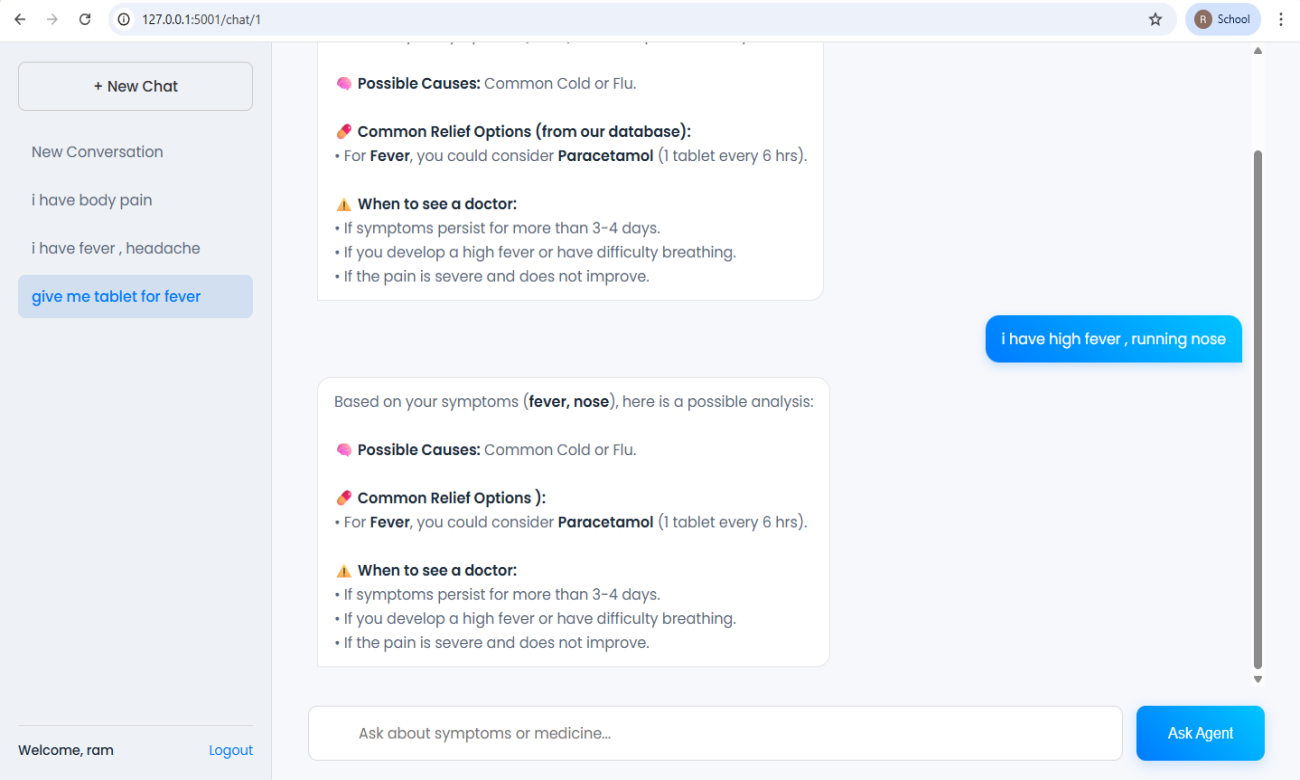

Fig : 6 - Registered User Dashboard

The main dashboard for an authenticated user. Key features shown include the conversation history on the left sidebar for easy navigation, and the interactive chat window where queries and AI responses are displayed.

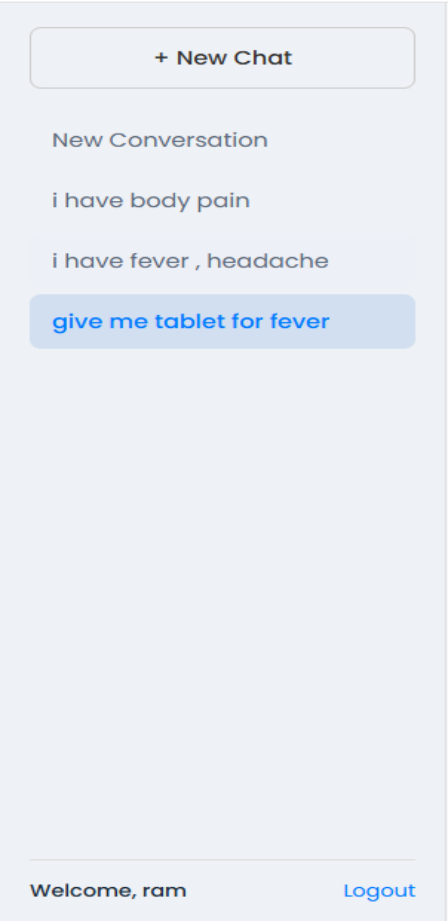

Fig : 7 - Conversation History Feature

A detailed view of the conversation history sidebar, a key benefit for registered users. This feature allows users to save and revisit their previous interactions with the AI agent.

CHAPTER – 4

CONCLUSION

The AI-Powered Pharmaceutical Guidance System successfully bridges a critical information gap in patient healthcare by providing an accessible and reliable platform for instant medication-related queries. By achieving its objectives of creating a multi-tool conversational agent, the project demonstrates the practical application of modern AI and web development technologies to solve real-world problems. The system's seamless integration of a RAG-based information retriever, a symptom analyzer, and a drug interaction checker within a user-friendly Flask application empowers users to take a more active and informed role in their health management.

The platform eliminates the need to sift through unreliable online sources, reduces anxiety associated with health queries, and has the potential to alleviate the workload of healthcare professionals by handling preliminary informational requests. The successful containerization with Docker ensures that the solution is not only functional but also highly scalable and ready for cloud deployment, fully aligning with the modern principles of software engineering.

Future enhancements could include the development of a dedicated mobile application for even greater accessibility, the integration of voice-to-text for hands-free interaction, and the expansion of the knowledge base to include a wider range of medical conditions. Further AI advancements could involve fine-tuning a specialized language model on medical literature for more nuanced responses and developing a predictive model for potential health risks based on a user's history. This project serves as a robust foundation for a comprehensive digital health assistant, turning the challenge of health information accessibility into a scalable, user-centric solution.