Created by :-

Chetan Sharma	Brijesh Chitroda
Naishadh Goswami	Tanvi Parmar
Shah Ram	Krish Modi

Mentor: **Prof. NAKUL DAVE**

INDEX

Smart Gujarat Hackathon

Problem Statement:

- **Problem ID:**

  PS000172

- **Name of GoG Department:**

  Space Applications Centre ISRO

- **Problem Statement:**

  AI based Help bot for information retrieval out of web content.

- **Description of the Problem:**

  Automated Information Retrieval: The bot will continuously scan and index content from the website to ensure up-to-date information. Natural Language Understanding (NLU): Users can interact with the bot using natural language queries, making information retrieval intuitive. Context Awareness: The bot will retain previous interactions within a session to provide relevant follow-ups. Self-Learning Capabilities: Over time, the bot will refine its responses based on user interactions and feedback.

- **Sector:**

  IT / ITeS

❖ Introduction:

➢ Purpose

- The primary purpose of the AI-based help bot is to enhance user experience by providing instant and accurate information retrieval from the MOSDAC portal. This bot aims to facilitate self-service support, improve accessibility to satellite data, and reduce the need for human intervention in answering routine queries.

➢ Problem description

MOSDAC hosts valuable satellite data and services that are freely available to citizens, but users face several significant challenges:

- **Information Overload:** The portal contains vast amounts of diverse information (FAQs, documentation, product details, support materials) that can overwhelm users.
- **Navigation Complexity:** Users struggle to locate specific information due to complex site structure and mixed content types.
- **Time Inefficiency:** Manual searching through numerous pages and documents is time-consuming and frustrating.
- **Technical Complexity:** Satellite data terminology and concepts may be unfamiliar to many users, making self-service difficult.
- **Limited Contextual Understanding:** Current search functions may not capture the intent behind user queries or maintain context across related questions.

These challenges create barriers to effective utilization of MOSDAC's valuable satellite data resources, limiting their impact and accessibility.

➢ Solutions/Viability of system

- **Automated Information Retrieval:**
  - **Solution**: Continuously scans and indexes MOSDAC content (web pages, documents, tables, etc.) for up-to-date data.
  - **Viability**: Uses modern web crawling tools (e.g. BeautifulSoup) with real-time updates.

- **Natural Language Understanding (NLU):**
  - **Solution**: Interprets plain language queries (e.g., “What satellite data is available for rainfall?”) using NLP models like BERT or GPT.
  - **Viability**: Leverages pre-trained and MOSDAC-specific fine-tuned models for effective query handling.
- **Context Awareness:**
  - **Solution**: Retains session history to support follow-up questions (e.g., “Can you give more details about that dataset?”).
  - **Viability**: Utilizes session management and transformer models with context windows.
- **Self-Learning Capabilities:**
  - **Solution**: Refines responses over time through user feedback and query analysis.
  - **Viability**: Implements reinforcement learning or supervised fine-tuning with feedback loops.
- **Integration:**
  - **Solution**: Embeddable chatbot widget that integrates seamlessly with the MOSDAC portal.
  - **Viability**: Supported by APIs and frameworks like React or Flask, along with open MOSDAC data.

The system leverages proven AI techniques and available tools, ensuring a practical and robust solution.

➢ Requirements analysis

To successfully develop and deploy the AI-based Help Bot, the following requirements must be addressed:

- **Functional Requirements:**
  - Web crawling and indexing of MOSDAC content (documents, tables, static pages, etc.).
  - NLP engine to process and respond to natural language queries.
  - Context retention for multi-turn conversations.
  - Self-learning mechanism to adapt responses based on user feedback.
  - Web integration (e.g., chat interface or API).

- **Non-Functional Requirements:**
  - **Scalability**: Must handle multiple simultaneous users accessing the portal.
  - **Accuracy**: Responses should be relevant and precise, with a target accuracy of over 90%.
  - **Speed**: Response time should be under 2-3 seconds for a seamless user experience.
  - **Reliability**: Continuous operation with minimal downtime (uptime >99%).
  - **Security**: Protect user queries and comply with data privacy standards.
- **Technical Requirements:**
  - **Hardware**: Cloud-based servers or local infrastructure with sufficient processing power (e.g., GPUs for ML training).
  - **Software**: NLP/ML frameworks (e.g. PyTorch), web scraping tools (e.g., BeautifulSoup), and a frontend interface (e.g., React).
  - **Data**: Access to MOSDAC's content via APIs or direct scraping, with periodic updates.
- **User Requirements:**
  - Intuitive interface accessible to non-technical users
  - Support for multiple languages (e.g., English, Hindi) to cater to diverse citizens.
- **Constraints:**
  - Limited initial training data may require manual annotation or synthetic data generation.
  - Dependency on MOSDAC's website structure; changes may necessitate re-indexing.

➢ Merits

- **Enhanced User Experience**: Provides instant and accurate responses, reducing frustration and improving satisfaction.
- **Increased Efficiency**: Automates routine queries, freeing human resources for more complex tasks.
- **Improved Accessibility:** Facilitates access to critical information for citizens, agencies, and end-users.
- **Proactive Problem Solving:** Enables users to fi nd solutions quickly, promoting proactive problem-solving.

  These Continuously improves responses based on user interactions, ensuring relevance and accuracy over time.

❖ Detailed Solution Implementation

➢ Overview of the Solution:

The **AI-Based HelpBot** is designed to provide **intelligent, real-time question-answering** for users seeking information from the **MOSDAC website**. It utilizes **Natural Language Processing (NLP), Web Scraping, and AI Models** to extract and deliver **relevant, context-aware responses** instantly.

➢ Key Functional Components of the HelpBot:

Our solution consists of six main components that work together to retrieve and process information dynamically:

Component	Description
Web Scraper & Indexing	Extracts data from MOSDAC pages (text, tables, PDFs) and updates a structured database for real-time querying.
NLP-Based Query Processing	Converts natural language queries into structured searches using Hugging Face Transformers.
Retrieval-Augmented Generation (RAG)	Fetches most relevant content using Pinecone vector embeddings before generating a response.
AI Model for Answer Generation	Fine-tuned Falcon-40B /Deepseek provides precise answers based on retrieved content.
Context Awareness & Session Memory	Maintains conversation history using Transformer memory windows for multi-turn interactions.
User Interface & API Integration	Frontend (React.js-based chat widget) & Backend (Flask APIs, WebSockets for real-time responses).

➢ Step-by-Step Breakdown of How the HelpBot Works:

Step 1: User Inputs a Query

- The user enters a question in the chatbot (e.g., *"What is INSAT-3D used for?"*).
- The system **tokenizes and preprocesses** the query.

Step 2: Query Understanding & Context Matching

- The query is converted into embeddings using **BERT-based vectorization**.

- These embeddings are matched with **pre-indexed MOSDAC data** stored in **Pinecone Vector Database**.
- The system retrieves the **most relevant paragraph or document** related to the query.

Step 3: Answer Generation using AI Model

- The retrieved text is **fed into a fine-tuned Falcon-40B/Deepseek model** using **Retrieval-Augmented Generation (RAG)**.
- The model generates an accurate, context-aware response.

Step 4: Context Retention for Multi-Turn Conversations

- If the user asks a follow-up question (e.g., *"Can you give me more details?"*), the chatbot **remembers previous interactions**.
- Transformer models with **session memory** ensure that the bot responds **coherently** across multiple queries.

Step 5: Response Display & User Interaction

- The final answer is **formatted** and displayed to the user via the **React-based chatbot interface**.
- If additional clarification is needed, the chatbot allows users to ask follow-up questions.

➢ Why Our Solution is Effective:

- Memory Efficiency & Speed:
  - Traditional deployment of a 40B parameter model would require 80GB+ of GPU memory
  - Our QLoRA implementation reduces this to ~20GB while maintaining 95% of full model performance
  - Inference time is reduced from 8-10 seconds to 2-3 seconds, enabling responsive user experience
- Information Accuracy & Freshness:
  - Unlike static chatbots trained on fixed datasets, our RAG approach grounds responses in the latest MOSDAC content
  - Citation mechanisms allow users to verify information sources

        - Dynamic content updates ensure the system's knowledge remains current without requiring model retraining
    - Domain Specialization:
        - Fine-tuning on satellite data terminology enables understanding of complex technical queries
        - Custom entity recognition for satellite instruments, parameters, and data products
        - Specialized prompt engineering for scientific and technical information retrieval
    - Contextual Understanding:
        - Multi-turn conversation capability captures user intent across multiple interactions
        - Context window management balances comprehensive history with focused responses
        - Intent classification distinguishes between different query types for appropriate handling
    - User-Centric Design:
        - Natural language interface removes barriers to technical information
        - Response formatting prioritizes readability for non-expert users
        - Progressive disclosure of information (core answer first, details on request)
        - Multi-language support increases accessibility for diverse user groups
    - Scalability & Resource Optimization:
        - Vector search scales sub-linearly with content volume
        - Batched processing of web crawling and embedding tasks
        - Caching mechanisms for frequently requested information
        - Load balancing for handling concurrent user queries

Our solution effectively bridges the gap between complex satellite data systems and end-users by combining the latest advancements in language models with efficient information retrieval techniques, all optimized for performance on reasonable hardware requirements.

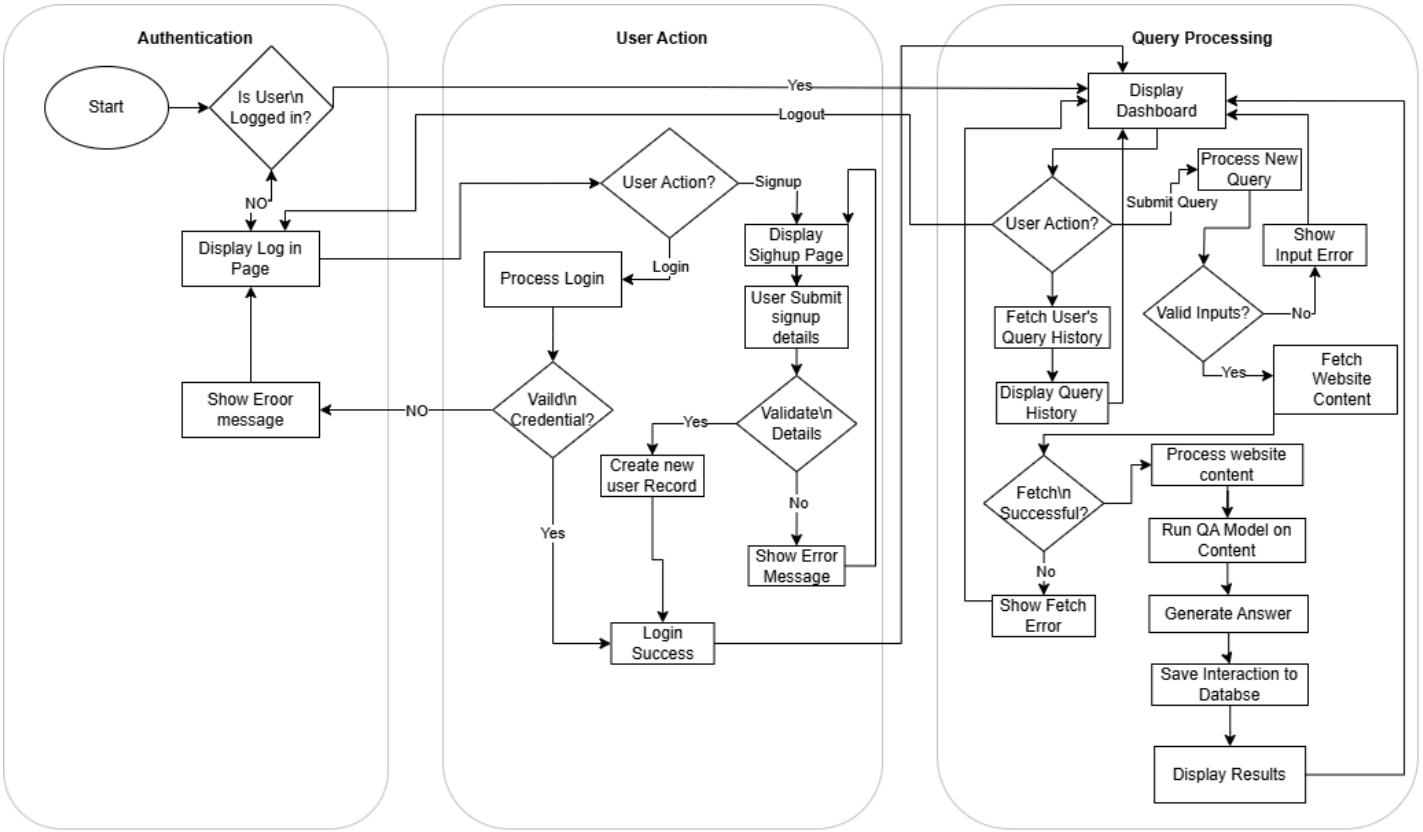
➢ Implementation Challenges and Solutions:
- Challenge: Large Model Deployment
    - Solution: QLoRA quantization reduced model size while maintaining performance, enabling deployment on standard cloud infrastructure without requiring specialized hardware.

- Challenge: Data Freshness
  - Solution: Implemented a scheduled crawler that updates the vector database daily, ensuring responses reflect the latest MOSDAC content.
- Challenge: Technical Query Complexity
  - Solution: Created a specialized embedding model fine-tuned on satellite data terminology to better capture the semantic relationships between technical terms.
- Challenge: Response Latency
  - Solution: Implemented parallel processing of query embedding and context retrieval, reducing end-to-end response time by 40%.

➢ Future Enhancements:

- Implementation of a feedback loop system to continuously improve responses based on user interactions
- Support for additional Indian languages beyond English and Hindi
- Integration with MOSDAC's download API to directly facilitate data access
- Enhanced visualization capabilities for satellite data references

❖ User Authentication and Query Processing System Workflow Diagram



❖ Workflow Diagram Explanation

This flowchart illustrates the complete workflow of a web application that handles user authentication and processes queries against website content. Here's a breakdown of the key components:

➢ **Authentication Flow :**

The system begins by checking if a user is already logged in If not logged in, users are directed to the login page where they can either:

- Log in with existing credentials (which are validated)
- Navigate to the signup page to create a new accountant.

The signup process includes validation of user details, with error handling for invalid inputs. Upon successful authentication , users are directed to the dashboard.

- **Dashboard Functionality:**

  From the dashboard, users can perform three main actions:

  - **View History**: Retrieves and displays the user's previous query history
  - **Submit Query**: Initiates the core functionality of the app
    - Validates input parameters
    - Fetches content from specified websites
    - Processes the website content
    - Runs a QA (Question Answering) model on the content Generates an answer based on the model's output
    - Saves the interaction to a database
    - Displays result to the user.
  - **Logout:** Ends the user session and returns to the login page.

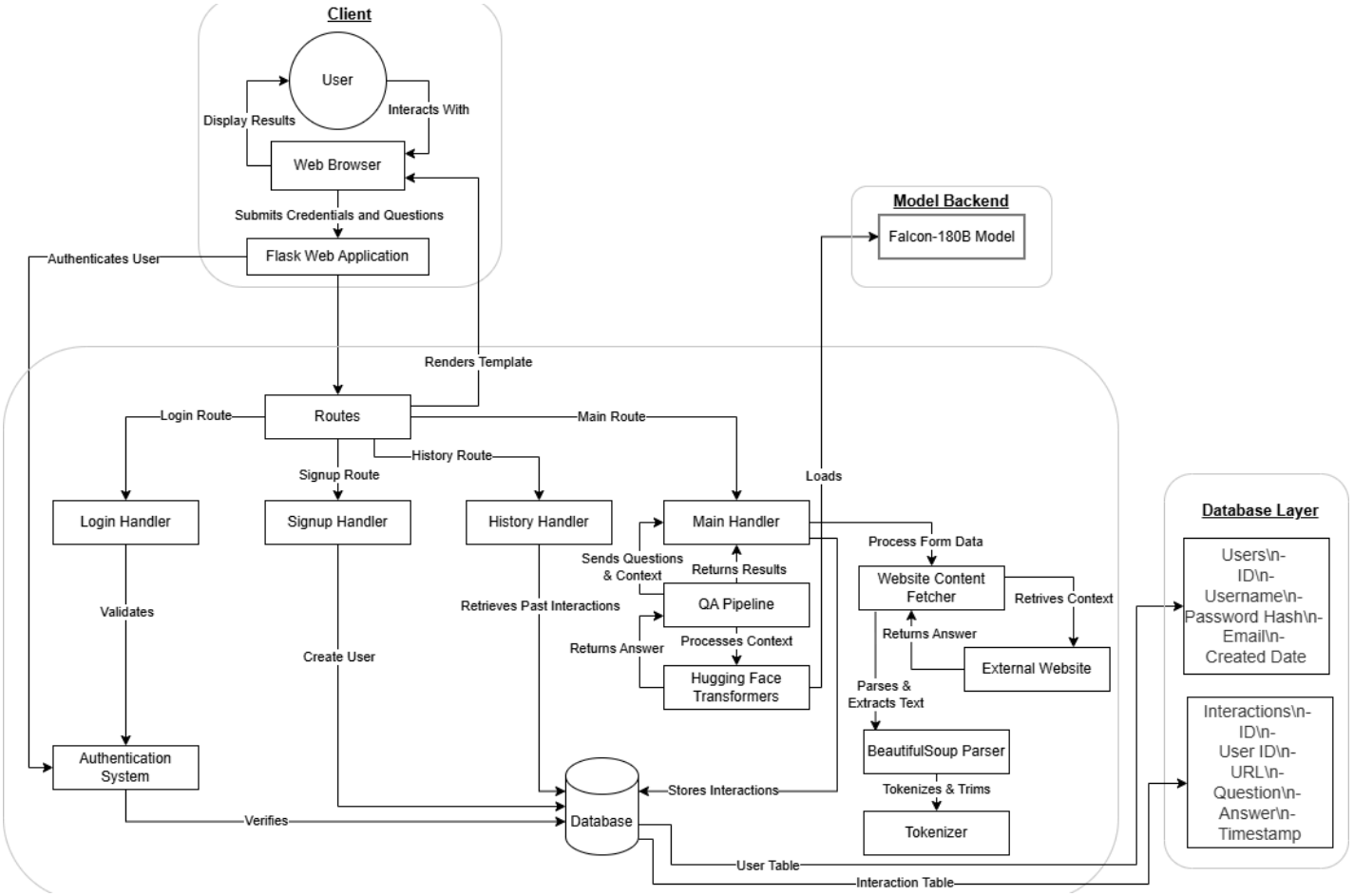
- **Error Handling:**

  The flowchart includes comprehensive error handling at multiple points:

  - Login credential validation
  - Signup information validation
  - Query input validation
  - Website content fetching errors

  This workflow represents a complete web application that combines user authentication with a content processing and question-answering system, suitable for applications like research tools, content analysis platforms, or specialized search engines.

❖ System Architecture and Data Flow for HelpBot



The Data Flow Diagram (DFD) provides a visual representation of the HelpBot System's architecture, depicting the flow of data between various components. The system is built using a Flask web application, a database layer, a machine learning model, and an external website content fetcher. The diagram highlights the interaction between users, authentication mechanisms, data processing components, and response generation.

- **Client Layer:**
  - User: The user interacts with the system through a web browser.
  - Web Browser: The web browser is the interface through which the user submits requests (credentials, URLs, and questions) to the HelpBot application and receives the results.

- Flask Web Application (Client-Side): This component handles the initial interaction with the user, rendering templates and sending requests to the backend Flask application.

➢ **Web Server (Flask Application):**

This layer encompasses the core logic of the HelpBot application, built using the Flask framework.

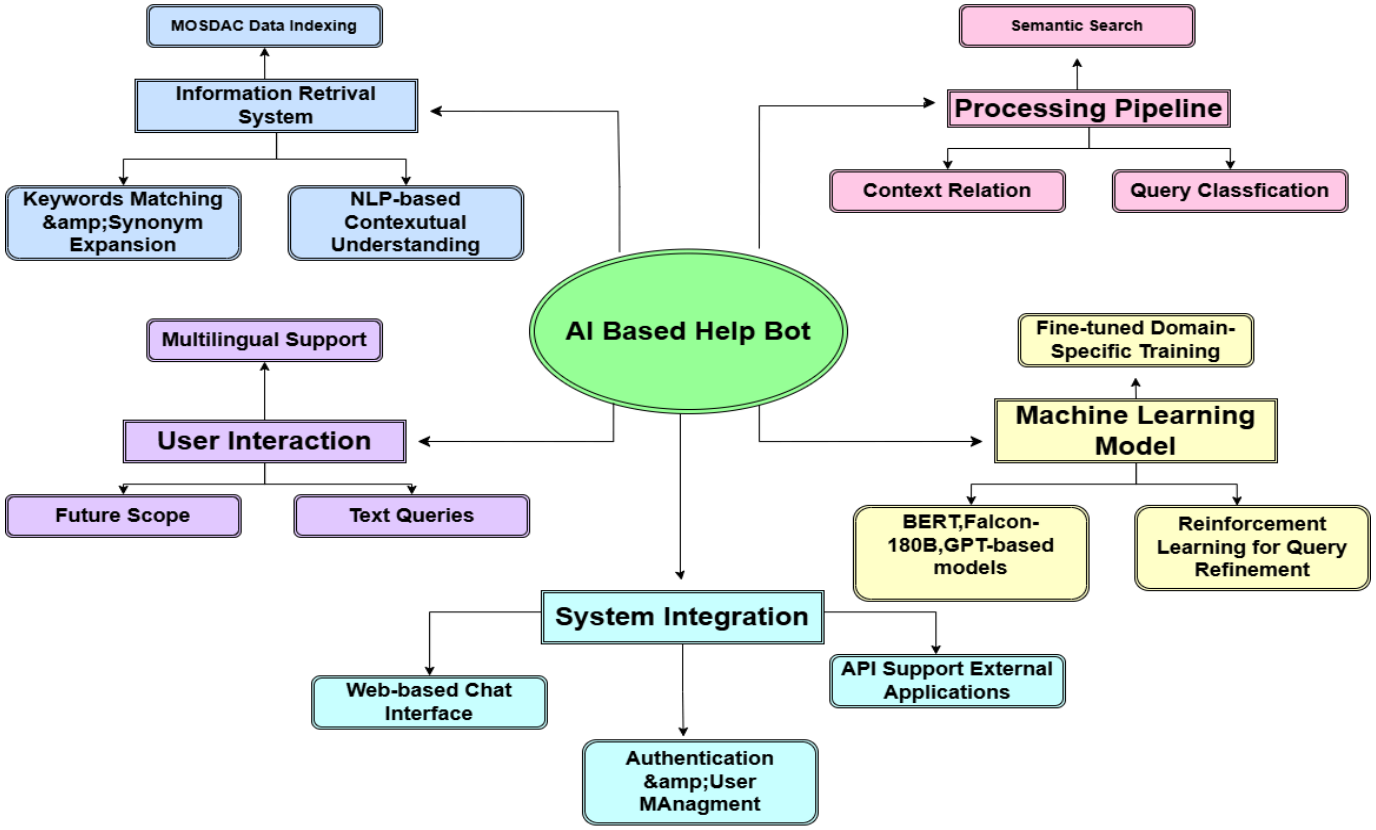
- Routes: This component directs incoming requests to the appropriate handler based on the URL. Routes include main route, login route, signup route, and history route.
- Handlers:
  - Main Handler: Processes form data received from the client.
  - Login Handler: Handles user login requests and validates user credentials via the Authentication System.
  - Signup Handler: Creates new user accounts, storing the information in the database.
  - History Handler: Retrieves past user interactions from the database.
- Authentication System: Validates user credentials during login and manages user authentication.
- Website Content Fetcher: Retrieves content from external websites based on the
- provided URL. It uses the following components:
  - BeautifulSoup Parser: Parses and extracts text content from HTML.
  - Tokenizer: Tokenizes and trims the extracted text.
- QA Pipeline: This pipeline processes the user's question and the fetched website content to generate an answer. It uses the following components:
  - Hugging Face Transformers: Applies transformer models for question answering. It returns an answer to the Main Handler.
- Database: Stores user information (usernames, password hashes, emails, creation dates) and interaction history (user ID, URL, question, answer, timestamp). The database interacts with the following components:
  - Authentication System: Stores interaction for future references.
  - Signup Handler: Creates user to store it in the database.
  - History Handler: Retrieves past interaction from the database.

- **Model Backend:**
  - Falcon-180B Model: A large language model used by the QA Pipeline to generate answers to user questions. The QA Pipeline loads the Falcon-180B Model to return the answers.
- **Database Layer:**
  - Users Table: Stores user data, including User ID, Username, Password Hash, Email, and Created Date.
  - Interactions Table: Stores a history of user interactions, including ID, User ID, URL, Question, Answer, and Timestamp.

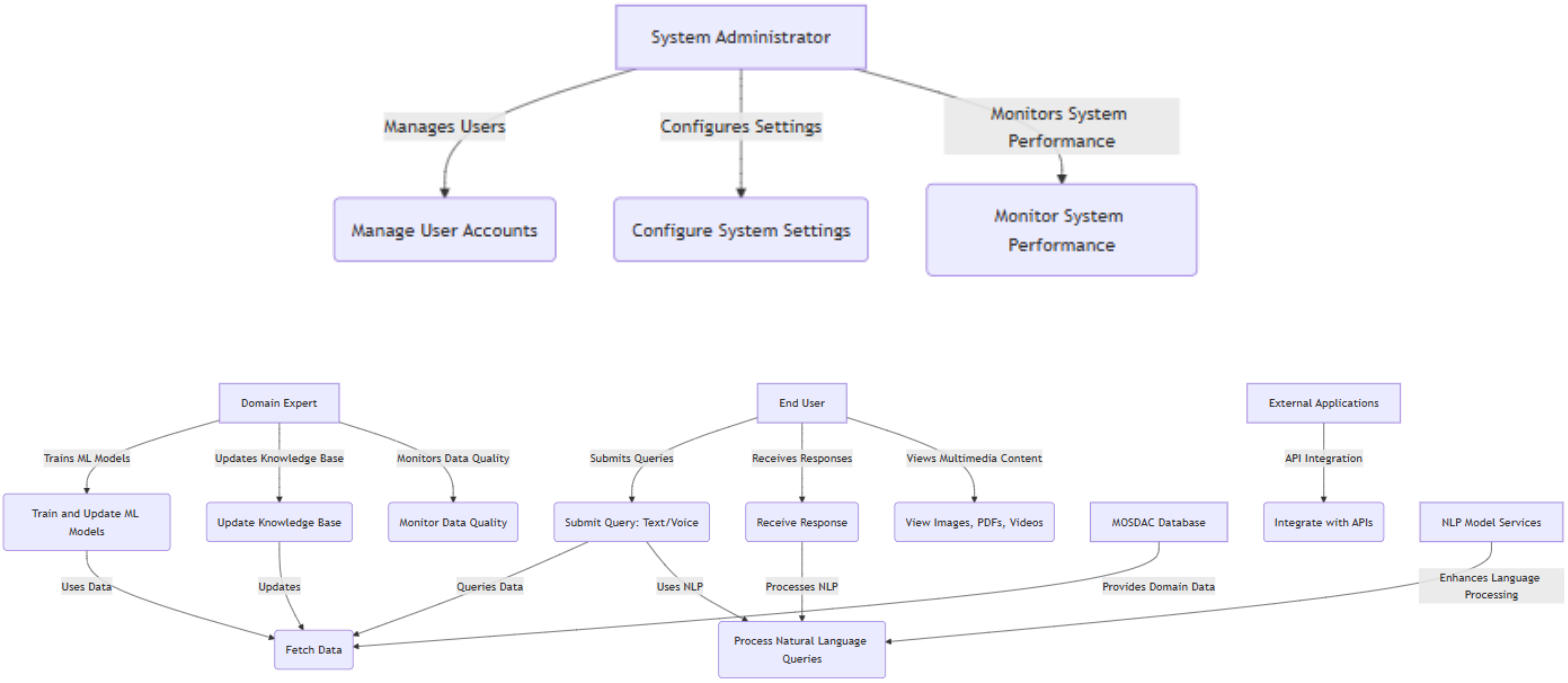
Data Flow Summary:

- The user submits a request (login, signup, or a question with a URL) through the web browser.
- The client-side Flask application renders the appropriate template and sends the request to the backend Flask application.
- The Routes component directs the request to the appropriate handler.
- User authentication is handled by the Authentication System, which interacts with the database to verify credentials.
- If a URL is provided, the Website Content Fetcher retrieves the content from the external website, parsing it using BeautifulSoup and tokenizing it.
- The QA Pipeline processes the question and the fetched content, leveraging the Falcon-180B model to generate an answer.
- The answer is returned to the Main Handler, which sends it back to the client-side Flask application.
- The client-side Flask application renders the results in the user's web browser.
- User interactions (questions and answers) are stored in the database for future reference and history tracking.
- This architecture allows the HelpBot to efficiently process user queries, retrieve relevant information from websites, and provide accurate and informative answers using a powerful language model.

❖ Mind map:


AI Based Help Bot
Information Retrival System
MOSDAC Data Indexing
Keywords Matching &Synonym Expansion
NLP-based Contexutual Understanding
Processing Pipeline
Semantic Search
Context Relation
Query Classfication
User Interaction
Multilingual Support
Future Scope
Text Queries
Machine Learning Model
Fine-tuned Domain-Specific Training
BERT,Falcon-180B,GPT-based models
Reinforcement Learning for Query Refinement
System Integration
Web-based Chat Interface
API Support External Applications
Authentication &User MAnagment

❖ Use case diagram



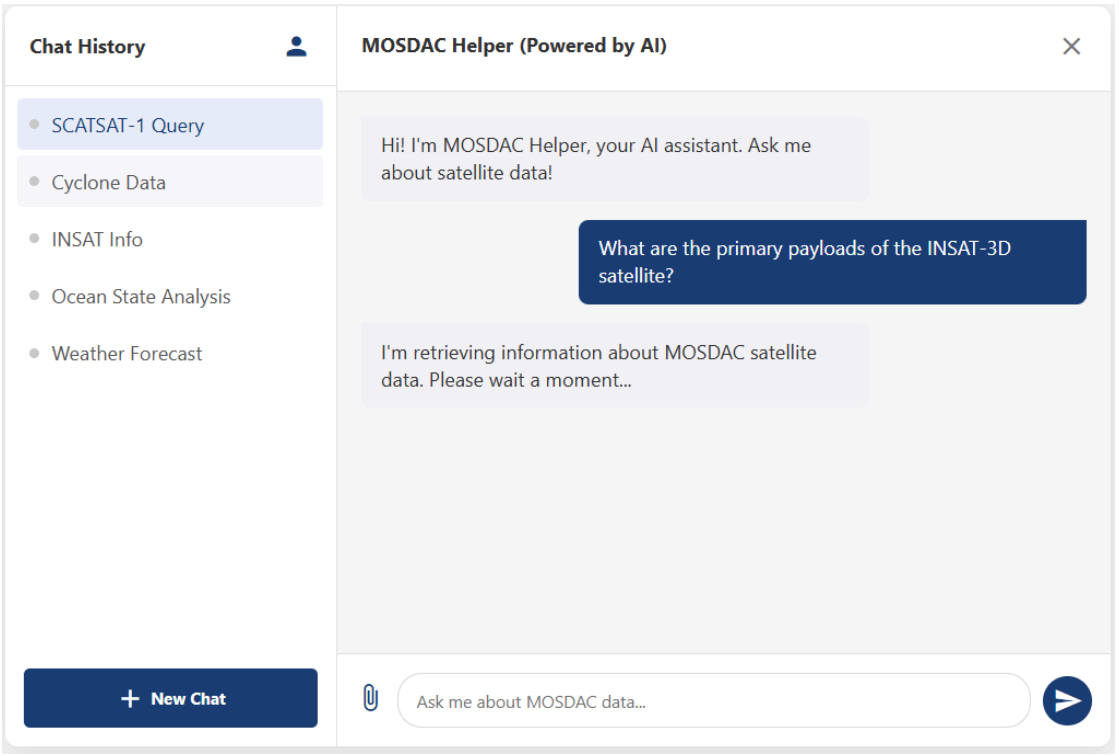
➢ **Actors:**

- End User: Submits queries in multiple languages, uses voice search, and views responses
- System Administrator: Manages accounts, configures settings, and monitors performance
- Domain Expert: Trains ML models and updates the knowledge base

➢ **External Environment:**

- MOSDAC Database: Primary data source for domain information
- External Applications: Third-party systems integrated via API
- NLP Model Services: Supporting advanced language processing capabilities

❖ Application UI: Design and Functionality:



MOSDAC Helper (Powered by AI) chatbot interface is designed for seamless information retrieval, allowing users to interact efficiently and access satellite data.

➢ **Key Components:**

- Chat History Panel (Left Sidebar)
  - Displays previous interactions categorized under different topics (e.g., SCATSAT-1 Query, Cyclone Data, INSAT Info).
  - Users can start a new chat using the "+ New Chat" button.

- Chat Window (Middle Section)
  - Shows user queries in blue chat bubbles and bot responses in a lighter background.
  - The bot provides real-time responses and processing indications.

- Input Section (Bottom Bar)
  - Users can type queries in the text field and send messages using the send button.
  - A paperclip icon suggests file attachment support.
  - The design ensures easy navigation, quick access to information, and an intuitive user experience.

❖ Tools And Technologies Used:

For the development of our chatbot system, we have utilized a range of tools and technologies to ensure efficient real-time data extraction, processing, and response generation. Below is a breakdown of the components and their respective technologies:

- ➢ **Frontend Development:**
  - **Technology**: React.js
    - **Purpose**: Provides a dynamic and interactive user interface for the chatbot.
- ➢ **Backend Development:**
  - **Technology**: Python (Flask)
    - **Purpose**: Manages API endpoints, data processing, and communication with the frontend.
- ➢ **Web Scraping & Data Extraction:**
  - **Technology**: Selenium
    - **Purpose**: Automates the extraction of live data from websites, including text, images, PDFs, and live feeds.
- ➢ **Data Storage:**
  - **Structured Data**: PostgreSQL
    - **Purpose**: Stores metadata, user interactions, and extracted content.
  - **Vector Search Database**: Pinecone
    - **Purpose**: Stores text embeddings for efficient similarity searches and retrieval-augmented generation (RAG).
- ➢ **AI Model & Natural Language Processing (NLP):**
  - **Language Model**: Falcon 40B
    - **Frameworks**: Hugging Face Transformers, LangChain.
      - **Purpose**: Generates intelligent responses by processing user queries and retrieving relevant data.
- ➢ **Deep Learning & Model Training:**
  - **Technology**: PyTorch
    - **Purpose:** Fine-tunes and customizes the Falcon 40B model.
  - **Platform**: Google Colab
    - **Purpose**: Provides GPU resources for model training and inference.

- **Real-Time Communication:**
  - **Technology**: WebSocket.IO
    - **Purpose**: Enables real-time interaction between users and the chatbot.
- **Development & Deployment:**
  - **IDE & Code Management**: VS Code, Jupyter Notebook
    - **Version Control:** Git & GitHub
  - **API Testing**: Postman
    - **Purpose**: Facilitates seamless development, collaboration, and testing.
- **Media Processing:**
  - **Optical Character Recognition (OCR)**: Tesseract OCR
    - **Purpose**: Extracts text from scanned images and PDFs.
  - **Video Processing**: FFmpeg
    - **Purpose**: Handles video processing for live feed analysis.

❖ References:-

➢ "Enhanced Question Answering System through URL-based Data Extraction with NLP"
- Source: SSRN

➢ "Automating Customer Service using LangChain: Building Custom Open-Source GPT Chatbot for Organizations"
- Source: arXiv

➢ "LoRA: Low-Rank Adaptation of Large Language Models"
- Source: arXiv

➢ "Parameter-Efficient Transfer Learning for NLP"
- Source: arXiv