AWS Machine Learning

Digital Egypt Pioneers

Digital Egypt Pioneers- project documentation

CAI1_AIS1_S2e

Project Title:

Nutrition Buddy

Course Instructors:

Eng. Abdelrhman Ahmed

Team Members:

Name	ID
Hossam Tarek Elsayed	21006898
Ali Elsayed Ali	21037198
Hatem Salem Abo sarea’	21034958

Project Overview:

The **Nutrition Buddy** chatbot is a Streamlit-based application integrated with MongoDB for user authentication and chat history storage. It provides a conversational AI interface, powered by the Langchain framework, where users can ask nutrition-related questions. The chatbot interacts with users through a session-based chat, and previous interactions are stored and retrieved using MongoDB.

Key Components:

1. **Libraries and Dependencies:**
    - **Streamlit (streamlit):** Used for building the interactive web app.
    - **PyMongo (pymongo):** Connects the app to a MongoDB database for user data and chat history.
    - **Langchain (langchain_core, langchain_community):** Provides language model capabilities to interact with users.
    - **YAML (yaml):** Loads configuration for MongoDB from an external file (config.yaml).
    - **Custom RAG Chain (return_rag_chain):** Retrieves responses based on provided questions and chat history.
2. **MongoDB Integration:**
    - **MongoDB Connection:** The app connects to a MongoDB database using credentials stored in a config.yaml file. It interacts with the users collection to manage user data (signup, login).
    - **MongoDBChatMessageHistory:** A customized class from Langchain MongoDB extension is used to store and retrieve chat history for each session.
3. **Authentication (Login/Signup):**
    - The app provides login and signup functionality, allowing users to securely create accounts and authenticate using credentials stored in MongoDB.
    - Upon successful login or signup, the user is redirected to the **Nutrition Buddy** chat interface.
4. **Chat Functionality:**
    - **Session-Based Chat:** Chat history is stored for each user session. Upon returning to the chat, previous interactions are displayed from the MongoDB database.
    - **Icons for User and Bot:** Custom icons are used to differentiate between user and bot messages.
    - **RAG Chain Integration:** The app uses a custom Retrieval-Augmented Generation (RAG) model to generate responses to user queries, providing accurate and context-aware answers.

Code Breakdown:

1. **Page Setup:**
   - st.set_page_config sets the page title for the web app.
   - Streamlit's session state is used to maintain the current user's login status, session, and page state.
2. **MongoDB Configuration:**
   - The **MongoDB connection** string is loaded from a config.yaml file, which contains credentials for the MongoDB database.
   - The users_collection is accessed for storing and retrieving user information.
3. **User Authentication (Login/Signup):**
   - **Login Page:** Users enter their username and password, which are validated against the database.
   - **Signup Page:** New users can create accounts. Input validation checks ensure proper usernames, matching passwords, and minimum password length.
4. **Chat Functionality:**
   - **open_chat(session_id):** This function retrieves the chat history for a specific user session from MongoDB and renders the conversation on the chat interface.
   - **RAG Chain (rag_chain)**: A custom chain used to generate contextually relevant responses to user queries.
   - **chat_input**: Captures user input and sends it to the RAG chain model, which generates and displays the response.
5. **Session State and Navigation:**
   - The app uses st.session_state to keep track of the current user's page, logged-in status, and navigation flow.
   - Navigation functions (go_to_login, go_to_signup, go_to_home, etc.) are used to control which page the user is on.
6. **Logout:**
   - **Logout Button:** Clears the user's session state and navigates back to the login page.

File Structure:

- **config.yaml:** Stores MongoDB credentials securely.
- **rag_mongo_v2.py:** Contains the custom function return_rag_chain that returns the Retrieval-Augmented Generation chain for answering user queries.
- **icons:** Stores the custom icons used for the chat interface.

Potential Improvements:

1. **Security Enhancements:**

    - Encrypt sensitive user information, such as passwords, before storing them in the MongoDB database.
    - Implement session expiration and re-authentication mechanisms for added security.
2. **Error Handling:**
    - Improve error handling in the open_chat function to manage potential database connection issues or missing chat history.
3. **User Experience:**
    - Provide visual feedback while the bot is generating a response (loading indicators).
    - Enhance input validation with stricter checks for username and password complexity.
    - Automatically create a session for the user on log in or sign up if he does not have a session already.

Deploying Your Application on AWS

---

1. Set Up Your AWS Account

- **Create an AWS Account**: If you don't have one, go to the AWS website and sign up for an account.
- **Configure IAM Users**: Set up Identity and Access Management (IAM) users with the necessary permissions to enhance security.

2. Choose Your Hosting Method

Depending on your application architecture, you can choose among several AWS services:

- **AWS Elastic Beanstalk**: An easy-to-use service for deploying and scaling web applications and services.
- **Amazon EC2**: A more manual approach where you can create and manage your own virtual servers.
- **AWS Lambda**: For serverless applications, where you can run code in response to events without provisioning or managing servers.
- **Amazon ECS/EKS**: If you are using containers, consider hosting on Elastic Container Service (ECS) or Elastic Kubernetes Service (EKS).

3. Deploy Your Application

Using AWS Elastic Beanstalk:

1. **Package Your Application**: Prepare your application code and any necessary configuration files.
2. **Create an Elastic Beanstalk Environment**:
    - Go to the Elastic Beanstalk console.
    - Create a new application and environment.
    - Upload your application package.

3. **Configure the Environment**: Set environment variables, scaling options, and other configurations.
4. **Deploy the Application**: Once everything is set up, deploy the application through the console.

Using Amazon EC2:

1. **Launch an EC2 Instance**:
    - Choose an Amazon Machine Image (AMI) that fits your tech stack (e.g., Ubuntu, Amazon Linux).
    - Select instance type based on your resource requirements.
2. **Configure Security Groups**: Allow inbound traffic on required ports (e.g., HTTP, HTTPS, and SSH).
3. **Connect to Your Instance**: Use SSH to access your instance and install necessary dependencies (e.g., Python, libraries, and MongoDB).
4. **Deploy Your Application**: Clone your repository or transfer files and run your application.

4. Set Up a Database (Optional)

If your application requires a database:

- **Amazon RDS**: Set up a managed relational database service for easy database management.
- **Amazon DynamoDB**: For NoSQL databases, use DynamoDB for scalability and high availability.
- **MongoDB Atlas**: If using MongoDB, consider deploying your database on MongoDB Atlas, which can be integrated with AWS.

5. Configure Networking

- **VPC (Virtual Private Cloud)**: Create a VPC to launch your resources in a secure network.
- **Route 53**: Use AWS Route 53 for domain registration and DNS management to route users to your application.

6. Set Up Load Balancing and Scaling

- **Elastic Load Balancer**: Set up a load balancer to distribute traffic across multiple instances if your application requires high availability.
- **Auto Scaling**: Configure auto-scaling to adjust the number of running instances based on traffic demands.

7. Monitor and Maintain Your Application

- **Amazon CloudWatch**: Use CloudWatch to monitor application performance and set up alerts for issues.
- **AWS CloudTrail**: Enable CloudTrail for logging and monitoring AWS API calls.

8. Set Up CI/CD (Optional)

For automated deployment:

- **AWS CodePipeline**: Set up a continuous integration and continuous deployment pipeline to automate the deployment process from your code repository to AWS.

9. Secure Your Application

- **SSL Certificates**: Use AWS Certificate Manager to provision SSL/TLS certificates for secure communication.
- **AWS WAF**: Consider setting up AWS Web Application Firewall (WAF) to protect against common web exploits.

10. Test Your Application

After deploying, ensure that your application is working correctly by performing thorough testing. Monitor logs and fix any issues that arise.

Conclusion:

This documentation provides a comprehensive understanding of the application's structure, database integration, and key functionalities. The **Nutrition Buddy** application is designed to offer a seamless, personalized user experience by combining AI-powered responses with a secure and scalable backend.