AURA AI Chatbot

Tagline: 'Your AI Assistant with Multimodal Intelligence'

Developed by: Varshith Kumar
Email: varshithkumar815@gmail.com
Technologies: Flask, MongoDB, Gemini AI, Python
Date: 2025

Abstract

The AURA AI Chatbot is a multimodal intelligent assistant designed to provide text, image, audio, video, and document analysis along with integrated web search. Built using Flask, MongoDB, SQLAlchemy, and Google Gemini AI models, AURA ensures secure authentication, efficient session handling, and detailed user interaction tracking. With modular architecture and rate limiting, the system is scalable, secure, and optimized for both educational and professional use cases.

Introduction

The rise of conversational AI has opened new opportunities for automation, learning, and productivity. AURA AI Chatbot was developed to provide a multimodal conversational experience powered by Gemini AI models. It can analyze not only text but also media formats like images, audio, video, and documents. It also integrates web search capabilities, making it a versatile digital assistant. The primary objective of AURA AI is to provide a secure, intelligent, and extensible chatbot solution for students, professionals, and businesses.

System Requirements

Software Requirements:
- Web Browser (Chrome, Firefox, Edge, Safari) - No heavy installation required

Hardware Requirements:
- Minimum: 4 GB RAM, 2-core CPU (sufficient) - Recommended: 8 GB RAM, 4-core CPU, GPU (optional)

Operating System Support:
- Windows, Linux, macOS, Android

External Requirements:
- Gemini API key - MongoDB Atlas or local instance

System Design (Architecture)

The AURA AI Chatbot follows a modular architecture. It consists of the following layers: 1. Frontend: HTML pages served by Flask. 2. Flask Backend: Handles routes, authentication, rate limiting, and file handling. 3. AI Service: Communicates with Gemini AI for multimodal tasks. 4. Database Layer: SQLite for structured user data, MongoDB for session/interactions. 5. Web Search Layer: DuckDuckGo and Wikipedia integrations.

Algorithm & Workflow

The workflow of the chatbot is structured as follows: A --> Initialize Libraries Import os, uuid, logging, bcrypt, requests, flask, flask_cors, flask_limiter, flask_sqlalchemy, pymongo, google.generativeai, etc. B --> Load Environment Load .env variables (e.g., GEMINI_API_KEY, MongoDB string) with load_dotenv(). C --> Set Up Logging Configure logging at INFO level with timestamp format. D --> Build Flask App Initialize Flask with static folder, CORS, secret key, SQLite URI, upload folder (Uploads), and 50MB limit. Set up SQLAlchemy for database. E --> Configure MongoDB Connect to MongoDB (aura_database) with MongoClient. Define MongoDBService with methods for sessions, interactions, and statistics. F --> Set Rate Limiting Use Flask-Limiter for 200/day, 50/hour limits. G --> Define File Validation Set allowed extensions and create upload directory. H --> Define User Model Create User model with id, name, email, password, created_at. Initialize SQLite tables. I --> Set Up Gemini AI Define GeminiAIService with API key, models, and methods. J --> Define Web Search Implement search_duckduckgo and search_wikipedia. K --> Define Auth Decorator Create require_auth to check user_id in session. L --> Define Routes Implement routes for /, /auth, /news, /register, /login, /logout, /chat, /analyze-image, /analyze-audio, /analyze-video, /analyze-document, /web-search, /user-stats, etc. M --> Run App Check pymongo, run Flask on 0.0.0.0:5000 in debug mode.

Database Design

The chatbot uses a hybrid database architecture: 1. SQLite (Relational Database): Stores user credentials and account details. - User Table: id, name, email, password, created_at 2. MongoDB (NoSQL Database): Stores session and interaction data. - Sessions Collection: user_id, session_id, start_time, end_time - Interactions Collection: session_id, message, response, timestamp This combination ensures secure authentication with structured data and efficient logging of interactions.

Features

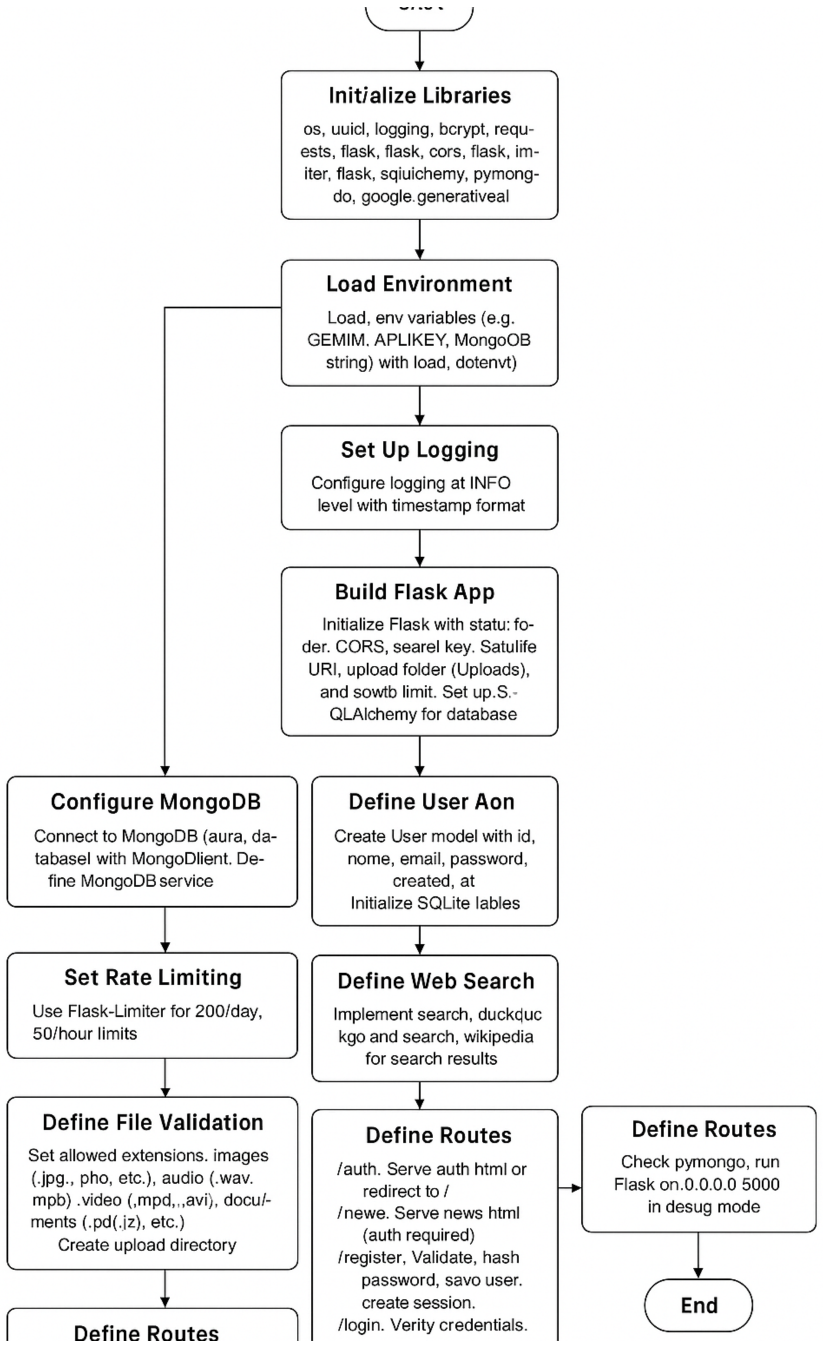
- Secure authentication with bcrypt - Session handling with MongoDB - Rate limiting to prevent abuse - Multimodal AI: text, image, audio, video, document analysis - Web search integration - User statistics and history tracking - Scalable modular design

Implementation Details

The backend is implemented using Flask. Security is ensured with bcrypt password hashing and Flask sessions. CORS is enabled for cross-origin requests. Flask-Limiter applies per-user request limits. Gemini AI integration is modular, supporting text and multimodal analysis. Uploads are stored temporarily and deleted after processing. MongoDB stores logs of interactions for analytics. SQLAlchemy handles relational user data. Logging is configured at INFO level for debugging and monitoring.

Conclusion & Future Enhancements

The AURA AI Chatbot demonstrates the integration of secure authentication, multimodal AI capabilities, and hybrid database architecture. It provides a scalable and extensible platform for conversational AI. **Future enhancements include:** - Integration with messaging platforms (WhatsApp, Telegram) - Real-time voice conversation - Personalization with memory and preferences - Analytics dashboard for administrators This project lays the foundation for next-generation intelligent assistants.


Initialize Libraries
os, uuicl, logging, bcrypt, requ-ests, flask, flask, cors, flask, im-iter, flask, sqiuichemy, pymong-do, google.generativeal
Load Environment
Load, env variables (e.g. GEMIM. APLIKEY, MongoOB string) with load, dotenvt)
Set Up Logging
Configure logging at INFO level with timestamp format
Build Flask App
Initialize Flask with statu: fo-der. CORS, searel key. Satulife URI, upload folder (Uploads), and sowtb limit. Set up.S.-QLAlchemy for database
Configure MongoDB
Connect to MongoDB (aura, da-tabasel with MongoDlient. De-fine MongoDB service
Define User Aon
Create User model with id, nome, email, password, created, at
Initialize SQLite lables
Set Rate Limiting
Use Flask-Limiter for 200/day, 50/hour limits
Define Web Search
Implement search, duckduc kgo and search, wikipedia for search results
Define File Validation
Set allowed extensions. images (.jpg., pho, etc.), audio (.wav. mpb) .video (,mpd,,avi), docu/-ments (.pd(.jz), etc.)
Create upload directory
Define Routes
/auth. Serve auth html or redirect to /
/newe. Serve news html (auth required)
/register, Validate, hash password, savo user. create session.
/login. Verity credentials.
Define Routes
Check pymongo, run Flask on.0.0.0.0 5000 in desug mode
End
Define Routes