AI Web Chat Platform with Web Crawler & Analytics

A full-stack AI-powered web application that crawls predefined websites, stores their content, allows users to query the data through a chat interface, and displays usage analytics on a dashboard. Users can also personalize the chat experience by submitting their name and photo.

🚀 Tech Stack

Frontend

Next.js: Provides a powerful full-stack framework that supports API routes, server components, and frontend integration — ideal for building both UI and backend logic in one place.

Tailwind CSS: Provides rapid, utility-first styling that keeps UI clean, consistent, and easy to maintain.

Backend

Node.js (Next.js API routes): Enables writing scalable backend logic including the web crawler and AI communication with Ollama.

Database

MongoDB: Its native support for vector search via Atlas or local plugin makes it an ideal store for embedding-based retrieval systems.

AI Models

Local inference allows unlimited queries, reduces cost, and enables privacy. The combo of embedding + language model provides a production-capable retrieval-augmented generation (RAG) system.

- ollama: Local LLM for inference
  - llama3.2:3b – for answering user queries
  - nomic-embed-text – for generating vector embeddings from website content and user queries

---

🛠 Installation & Setup

💻 Recommended System Specs:

- 8 GB RAM (minimum)
- 10 GB+ Disk space (for model storage)
- Node.js v18+ and MongoDB 7.0+ (with vectorSearch enabled)
- Ollama installed locally (https://ollama.com/download)

1. Clone the repository

git clone https://github.com/your-username/ai-webchat-app.git cd ai-webchat-app

2. Install dependencies

npm install

3. Set up environment variables

Create a .env.local file and add:

MONGODB_URI=mongodb://localhost:27017/your-db-name

- Ensure MongoDB is running and has vector indexing enabled.
- Make sure Ollama is running locally.

4. Install Ollama

Install from ollama

After installation

ollama run llama3.2

ollama pull nomic-embed-text

5. Start the development server

npm run dev

6. Run the web crawler

npm run crawl

This will crawl the default URLs configured in src/lib/crawler.js and store vector embeddings in MongoDB.

🔧 Assignment Implementation Breakdown

✅ Part 1: Web Crawler

- **File**: src/lib/crawler.js
- **Run Command**: npm run crawl

Uses **Puppeteer** to:

- Open predefined URLs
- Extract text from HTML
- Chunk text and get embeddings using nomic-embed-text
- Store content + embeddings in **MongoDB**

Each website is tracked with **crawl statuses**:

- Pending

- In Progress
- Completed
- Failed

✅ Part 2: AI Component

- **Powered by**: Ollama (runs locally)
- **Embeddings via**: nomic-embed-text
- **Query answering via**: llama3.2:3b

Flow:

1. User submits a query.
2. The query is embedded.
3. Top 2 most relevant content chunks are retrieved from MongoDB using **vector similarity**.
4. A prompt is created using:
   - User name
   - Retrieved content
   - User query
5. Prompt is sent to the local llama3.2 model.
6. Response is shown via the **Chat UI**.

✅ Part 3: Chat System & Dashboard

💬 Chat UI

- Chat interface available at: /chat
- Website dropdown allows selecting a target site
- Queries are **personalized** if the user has submitted their **name and photo**

👤 User Form

- Route: /user
- Fields: Name, Email, Photo
- If a user exists (based on email), their data is returned. New photo uploaded, then updates profile pic
- Otherwise, a new record is created

📊 Dashboard

- Route: /

Displays:

- **Total Users** (excluding guests)
- **Total Queries** (aggregated from all users)

Real-time analytics powered by MongoDB aggregations