# 🛒 Zepto Chatbot – AI-Powered Grocery Assistant

**By Fiza G | Quality Engineer**

---

## 📘 Project Overview

I created the Zepto Chatbot as a **self-learning project** to strengthen my technical profile beyond manual testing and explore **AI-driven automation concepts**.
My goal was to design a smart chatbot that mimics Zepto's customer assistant — answering grocery-related queries, checking FAQs, suggesting offers, and providing fallback AI-based responses when data isn't available.

This project started purely as a learning experiment, but it turned into a working, fully deployed chatbot with a clean UI, real-time interaction, and analytics.

🔗 **Live App Link:** https://zepto-chatbot-demogit-ajbvzf5zvv975bhfzxth6s.streamlit.app/

---

## 🎯 Objective

Coming from a QA background, I wanted to showcase that I could:

- Design and test complete applications — from logic to UI.
- Integrate **AI models**, manage **APIs**, and handle **automation workflows**.
- Build something functional that highlights curiosity and real implementation skills.

---

## ⚙️ Tech Stack

- **Language:** Python
- **Frontend:** Streamlit
- **Backend Logic:** Python functions (rule-based + AI fallback)
- **AI Model:** LLaMA 3.2 via Hugging Face API
- **Data Management:** JSON, CSV, Pandas
- **Libraries Used:** rapidfuzz, huggingface_hub, dotenv, pandas, streamlit

---

## 🧩 Architecture & Flow

1. **User Interface:** Built with Streamlit to create a simple, mobile-friendly chat experience.
2. **Input Processing:** Cleans and normalizes user messages using Regex.
3. **FAQ & Product Matching:**
    - Uses *RapidFuzz* for fuzzy matching to handle typos or similar phrases.
    - Checks local JSON data (chat_data.json, zepto_data.json) for product info and prices.
4. **Festival Module:** Displays active offers or greetings if the date matches any event.
5. **AI Fallback:**
    - Initially used **Hugging Face Inference API**, but later improved with **LLaMA** for more accurate and faster responses.
    - Helps answer new or unlisted queries naturally.
6. **Logging:**
    - All user–bot conversations are saved in **JSON** and **CSV** formats.
    - This data is used later for dashboard analytics.
7. **Dashboard (Streamlit Page):**
    - Tracks daily chat volume, frequent words, and last 20 messages.
    - Gives insights into user behavior and chatbot usage.

---

## 🧠 Tools and Why I Used Them

| Tool | Purpose | Why I Chose It |
|---|---|---|
| **Python** | Main programming language | Easy to integrate APIs and handle logic cleanly |
| **Streamlit** | Web interface & dashboard | Simple, interactive, and no HTML/CSS required |
| **RapidFuzz** | Text similarity | Handles spelling mistakes and fuzzy matches faster than fuzzywuzzy |
| **Hugging Face / LLaMA** | AI response generation | Free and lightweight alternative to OpenAI for testing |
| **dotenv** | Secret key management | Keeps API keys safe instead of hardcoding them |
| **Pandas** | Chat log storage and analytics | Makes CSV management and visual charts easy |
| **JSON** | FAQ & data storage | Lightweight and readable, no database setup needed |

---

## 🧮 Testing Approach

As a QA Engineer, I applied testing concepts even while developing:

- **Functional Testing:** Validated all user message flows and responses.
- **Negative Testing:** Checked how the bot handles incomplete or irrelevant questions.
- **Regression Testing:** Ensured previous logic (FAQ, item checks) still worked after adding AI.
- **UI Testing:** Verified layout, message alignment, and responsive design in Streamlit.
- **Data Validation:** Cross-checked JSON data loading, file creation, and logging accuracy.

This helped me simulate both **QA and developer roles** in one project — from design to deployment.

---

## 🚀 Deployment

- Project hosted on **GitHub**.
- Deployed on **Streamlit Cloud** using GitHub integration.
- Streamlit automatically pulls updates from the repository on every commit.
- All data and logs remain local to ensure privacy.

---

## 🧾 Key Features

- Real-time chat interface with human-like responses.
- Handles FAQs, products, and festival offers.
- AI fallback using LLaMA model.
- Automatic chat logging in JSON & CSV.
- Password-protected admin view for logs.
- Streamlit dashboard for analytics & word frequency charts.

---

## ⚡ Challenges & Fixes

- **Slow and irrelevant responses** from the first Hugging Face API version → switched to **LLaMA model** for better accuracy and speed.
- **Session timeout** issues in Streamlit → optimized with state handling and smaller wait time.
- **Data mismatches** → used fuzzy logic (RapidFuzz) to improve matching accuracy.
- **UI freezing** → added controlled rerun mechanism in Streamlit for smooth refresh.

---

## 📊 Dashboard Insights

The dashboard page gives:

- Total message count
- Unique user count
- Daily usage trends
- Most common query words

- Last 20 user messages (real-time table view)

This feature shows my ability to not only build, but also **analyze and test user data behavior** — a key QA mindset.

---

**Drawbacks & Limitations**

Even though the chatbot performs well for a demo project, it has a few practical limitations:

- **Streamlit Sleep Mode:** Since it's hosted on Streamlit Cloud's free tier, the app automatically goes to sleep after inactivity. This can delay the first response when it restarts.
- **Limited Session Memory:** The chatbot doesn't store long-term context; once the page refreshes, chat history resets.
- **Response Time Variability:** LLaMA API sometimes responds slower or gives off-topic answers depending on server load.
- **Static Data:** Product and festival data are stored locally in JSON files, so updates must be done manually instead of fetching live data.
- **No Real Database:** All logs are saved locally (CSV/JSON), which limits scalability if many users interact at once.
- **Basic UI Flow:** The design focuses on functionality; advanced features like user login, voice input, or real-time updates are yet to be added.
- **Limited Error Handling:** Network or API failures aren't fully handled with retry logic or fallbacks yet.

---

**💡 Learning Outcomes**

- Strengthened my skills in **Python automation**, **AI integration**, and **UI testing**.
- Learned to handle **API errors**, **data parsing**, and **state management**.
- Improved my understanding of **end-to-end testing** in AI-driven applications.
- Built confidence in explaining complete workflow logic during technical interviews.

---

**🌱 Future Improvements**

- Add **database (SQLite or Firebase)** instead of local files.
- Train chatbot with custom Zepto-like data for higher accuracy.

- Introduce **voice input** or **multilingual support**.
- Connect with **real Zepto APIs** for live inventory and offers.
- Add **user authentication** for personalized interaction.

---

## 👩‍💻 About Me

I'm **Fiza G**, a **Quality Engineer** with 3.7 years of experience in **manual and automation testing** (Selenium, Python, Robot Framework, API Testing).
I'm passionate about building reliable, testable, and user-friendly systems.
This project is a reflection of my curiosity, self-learning approach, and interest in combining QA with AI and automation.

📍 Davangere, Karnataka, India
📧 fizag2901@gmail.com | 📞 **+91-9353055673**
🔗 **LinkedIn:** www.linkedin.com/in/fiza-g-9138781a8