**# Social Dance Event Chatbot Design Document**

**1. Overview**

**This document outlines the design and implementation plan for a chatbot that will help users retrieve social dance event information. The chatbot will:**

* **Interact with OpenAI's LLM (GPT-4o or equivalent) for natural language processing.**
* **Generate SQL queries dynamically based on user input.**
* **Execute those queries on a PostgreSQL database hosted on Supabase.**
* **Be deployed using Streamlit (frontend) and Render (backend API hosting).**
* **Be embedded into a WordPress website using iframed Streamlit components.**

**2. System Architecture**

**2.1 Components**

1. **User Interface (UI)**
   * **Technology: Streamlit, Iframe embed for WordPress**
   * **Functionality:** 
     + **Accepts user queries (e.g., "Are there any Salsa events this weekend?")**
     + **Displays chatbot responses and event data in a structured format**
2. **Backend API**
   * **Technology: FastAPI hosted on Render**
   * **Functionality:** 
     + **Routes user input to OpenAI's LLM for intent recognition**
     + **Generates SQL queries dynamically**
     + **Queries Supabase using SQLAlchemy and formats the response**
     + **Returns structured data to the Streamlit app**
3. **Database**
   * **Technology: PostgreSQL hosted on Supabase**
   * **Functionality:**
     + **Stores event data (name, location, date, organizer, links, etc.)**
     + **Allows query execution via Supabase API**
   * **Current Table Schema (Events Table):**
   * **CREATE TABLE IF NOT EXISTS events (**
   * **event\_id SERIAL PRIMARY KEY,**
   * **org\_name TEXT,**
   * **dance\_style TEXT,**
   * **url TEXT,**
   * **event\_type TEXT,**
   * **event\_name TEXT,**
   * **day\_of\_week TEXT,**
   * **start\_date DATE,**
   * **end\_date DATE,**
   * **start\_time TIME,**
   * **end\_time TIME,**
   * **price TEXT,**
   * **location TEXT,**
   * **address\_id INTEGER,**
   * **description TEXT,**
   * **time\_stamp TIMESTAMP**
   * **);**
4. **Chatbot Logic (LLM Integration)**
   * **Technology: OpenAI GPT-4o API**
   * **Functionality:** 
     + **Understands natural language input**
     + **Extracts relevant query parameters (e.g., event type, date, location)**
     + **Generates optimized SQL queries**
     + **Converts database responses into human-readable answers**

**3. Workflow**

1. **User interacts with Streamlit Chatbot UI → Enters query**
2. **UI sends query to Backend API (Render-hosted FastAPI server)**
3. **Backend sends query to OpenAI LLM → Extracts key parameters**
4. **Backend generates SQL query based on user input and query intent**
5. **SQL query is sent to Supabase (PostgreSQL) via SQLAlchemy**
6. **Supabase returns structured event data**
7. **Backend formats response and sends it back to the Streamlit UI**
8. **User sees the event results and chatbot's explanation**

**4. Implementation Plan**

**4.1 Pre-Development Setup**

**✅ Set up Supabase Database (Completed)**

**✅ Set up OpenAI API Access**

* **Get API key**
* **Test query extraction and LLM-generated SQL**

**✅ Set up Streamlit App on Local Machine**

* **Install Streamlit**
* **Create a simple chatbot interface**
* **Test embedding iframe into WordPress**

**✅ Set up Backend on Render**

* **Deploy FastAPI that connects Streamlit to OpenAI and Supabase**

**4.2 Development Phases**

**Phase 1: Database & Query Engine**

**✅ Define PostgreSQL schema for events (Completed) ✅ Implement SQL query generation logic using OpenAI ✅ Test API connectivity with Supabase using SQLAlchemy**

**Phase 2: Backend API Development**

**✅ Develop FastAPI server for handling chatbot requests ✅ Implement SQL query generation and execution logic using SQLAlchemy ✅ Deploy backend to Render and test API responses**

**Phase 3: Streamlit Frontend Development**

**✅ Develop chatbot UI ✅ Integrate API calls to backend ✅ Implement data visualization for event results ✅ Test UI with real database queries**

**Phase 4: Deployment & Integration**

**✅ Deploy Streamlit app to Render ✅ Embed chatbot into WordPress via iframe ✅ Test user interactions and refine chatbot responses**

**4.3 Post-Launch Enhancements**

**📌 Performance Optimization: Cache frequent queries for faster response times 📌 User Authentication: Allow user profiles and saved events 📌 Monetization Options: Paid event promotions, premium chatbot features**

**5. Tech Stack Summary**

| **Component** | **Technology Stack** |
| --- | --- |
| **UI** | **Streamlit + WordPress (iframe embed)** |
| **Backend API** | **FastAPI (Python) hosted on Render** |
| **Database** | **Supabase (PostgreSQL) + SQLAlchemy** |
| **AI Model** | **OpenAI GPT-4o API** |
| **Hosting** | **Render for backend & Streamlit app** |

**6. Risks & Mitigation**

| **Risk** | **Mitigation Strategy** |
| --- | --- |
| **Latency issues between WordPress (Bluehost), Supabase, and Render** | **Consider consolidating on the same cloud provider if latency becomes a problem** |
| **API rate limits (OpenAI, Supabase)** | **Implement caching and query batching to reduce API calls** |
| **Supabase query performance** | **Optimize indexes and limit query results to improve speed** |

**7. Next Steps**

* **Start backend API development on Render ✅**
* **Create Streamlit prototype and test queries ✅**
* **Plan WordPress integration and iframe setup ✅**

**social\_dance\_app/**

**│── config/ # Configuration files (e.g., credentials, settings)**

**│── data/ # Database exports, backups, sample data**

**│── src/ # Source code**

**│ ├── scraping/ # Web scraping scripts (existing code)**

**│ │ ├── spiders/ # Individual Scrapy spiders**

**│ │ ├── parsers/ # Parsing utilities**

**│ │ ├── utils/ # Helper functions (e.g., cleaning, deduplication)**

**│ │ └── \_\_init\_\_.py**

**│ ├── backend/ # FastAPI backend**

**│ │ ├── api/ # API routes (event queries, chatbot requests)**

**│ │ ├── services/ # Business logic (query generation, OpenAI calls)**

**│ │ ├── models/ # Database models (SQLAlchemy or Pydantic schemas)**

**│ │ ├── utils/ # Helper functions**

**│ │ ├── main.py # FastAPI entry point**

**│ │ └── \_\_init\_\_.py**

**│ ├── frontend/ # Streamlit UI**

**│ │ ├── pages/ # Multi-page Streamlit app (if applicable)**

**│ │ ├── components/ # Custom Streamlit widgets or UI elements**

**│ │ ├── app.py # Streamlit main entry point**

**│ │ └── \_\_init\_\_.py**

**│ ├── common/ # Shared utilities (e.g., logging, error handling)**

**│ └── \_\_init\_\_.py**

**│── tests/ # Unit tests**

**│── docs/ # Documentation**

**│── requirements.txt # Python dependencies**

**│── .env # Environment variables**

**│── .gitignore # Git ignore file**

**│── README.md # Project overview**