SG_JB Dental Chatbot: Complete Code Analysis

Your chatbot represents a sophisticated **multi-brain agentic AI architecture** with semantic search, contextual filtering, and session memory management. Here's a comprehensive breakdown of each key segment:

Core Architecture Overview

The system follows a **4-stage processing pipeline** with specialized AI "brains" that work together:

1. **Factual Brain** - Entity extraction and intent understanding
2. **Deterministic Planner** - Filter merge/replace logic
3. **Ranking Brain** - Sentiment-based prioritization
4. **Response Generation** - Contextual answer formulation

[1] [2] [3]

Environment Setup & Dependencies

```
# Environment and client configuration
load_dotenv()
gemini_api_key = os.getenv("GEMINI_API_KEY")
genai.configure(api_key=gemini_api_key)
supabase_url = os.getenv("SUPABASE_URL")
supabase_key = os.getenv("SUPABASE_KEY")
supabase: Client = create_client(supabase_url, supabase_key)
```

Function: Establishes connections to Google's Gemini AI models for natural language processing and Supabase for vector database operations. This follows best practices for API key management using environment variables.[4] [5]

AI Model Architecture

```
factual_brain_model = genai.GenerativeModel('gemini-1.5-flash-latest')
ranking_brain_model = genai.GenerativeModel('gemini-1.5-flash-latest')
embedding_model = 'models/embedding-001'
generation_model = genai.GenerativeModel('gemini-1.5-flash-latest')
```

Function: Implements a **specialized multi-brain approach** where each AI model serves a distinct purpose:

- **Factual Brain**: Extracts dental services and locations from user queries
- **Ranking Brain**: Determines sentiment-based priorities for clinic ranking
- **Embedding Model**: Converts text to numerical vectors for semantic search
- **Generation Model**: Produces final conversational responses

This architecture aligns with emerging **agentic AI patterns** that use specialized agents for different cognitive tasks. [6] [7] [8]

Data Models & Schema Validation

```
class ChatMessage(BaseModel):
    role: str
    content: str

class UserQuery(BaseModel):
    history: List[ChatMessage]
    applied_filters: Optional[dict] = Field(None)
    candidate_pool: Optional[List[dict]] = Field(None)
```

Function: Implements **session state management** through Pydantic models that maintain conversation history and filter persistence across interactions. The `applied_filters` and `candidate_pool` enable **stateful conversations** where the bot remembers previous search criteria and results. [9] [10] [11]

Stage 1A: Factual Brain with Two-Prompt Safety Net

```
# Attempt 1: Tool-based extraction
factual_response = factual_brain_model.generate_content(prompt_text, tools=[UserIntent])

# Attempt 2: JSON-based safety net
if not current_filters:
    safety_net_prompt = f"""
    Analyze the user's query and extract information into a JSON object...
    """
```

Function: Implements a **dual-extraction strategy** to maximize entity recognition accuracy. The primary method uses structured tool calling, while the fallback uses JSON parsing. This approach addresses common LLM reliability issues in entity extraction tasks. [12] [13]

Stage 1B: Deterministic Filter Management

```
user_wants_to_reset = any(keyword in latest_user_message for keyword in RESET_KEYWORDS)

if user_wants_to_reset:
    final_filters = current_filters
    candidate_clinics = []
else:
```

```
    final_filters = previous_filters.copy()
    final_filters.update(current_filters)
```

Function: Manages **conversation state transitions** by detecting reset intentions and implementing filter merge logic. This enables users to refine searches incrementally or start fresh, maintaining conversation continuity. [14] [11]

Stage 1C: Ranking Brain for Sentiment Analysis

```
ranking_prompt = f"""
For complex services ('implant', 'braces', 'root canal'), prioritize 'sentiment_dentist_s
For cosmetic services ('whitening', 'veneers'), prioritize 'sentiment_cost_value'.
For location queries ('near', 'in'), prioritize 'sentiment_convenience'.
"""
```

Function: Implements **context-aware ranking** that adjusts clinic prioritization based on the type of dental service requested. This sophisticated approach goes beyond simple ratings to consider user intent and service complexity. [15] [7]

Stage 2: Conditional Semantic Search

```
if not candidate_clinics:
    search_text = latest_user_message if not final_filters else json.dumps(final_filters)
    query_embedding_response = genai.embed_content(model=embedding_model, content=search_
    query_embedding_list = query_embedding_response['embedding']

    db_response = supabase.rpc('match_clinics_simple', {
        'query_embedding_text': query_embedding_text,
        'match_count': 75
    }).execute()
```

Function: Performs **semantic similarity search** using vector embeddings only when needed, optimizing for performance. The system converts user queries into high-dimensional vectors and finds clinics with similar semantic content. The conditional execution prevents unnecessary database calls when using existing candidate pools. [16] [17] [1]

Stage 3: Multi-Layer Filtering System

Quality Gate Filtering

```
for clinic in candidate_clinics:
    if clinic.get('rating', 0) >= 4.5 and clinic.get('reviews', 0) >= 30:
        quality_gated_clinics.append(clinic)
```

Factual Filtering

```
if final_filters.get('township') and final_filters.get('township').lower() not in clinic.
    match = False
if final_filters.get('services'):
    for service in final_filters.get('services'):
        if not clinic.get(service, False):
            match = False
```

Function: Implements a **progressive filtering pipeline** that first ensures quality thresholds (4.5+ rating, 30+ reviews) then applies user-specific criteria. This approach balances recommendation quality with personalized requirements.[18] [19]

Stage 3B: Advanced Ranking Algorithms

Sentiment-First Ranking

```
if ranking_priorities:
    ranking_keys = ranking_priorities + ['rating', 'reviews']
    unique_keys = list(dict.fromkeys(ranking_keys))
    ranked_clinics = sorted(qualified_clinics, key=lambda x: tuple(x.get(key, 0) or 0 for
```

Objective Weighted Scoring

```
else:
    norm_rating = (clinic.get('rating', 0) - 1) / 4.0
    norm_reviews = np.log1p(clinic.get('reviews', 0)) / np.log1p(max_reviews)
    clinic['quality_score'] = (norm_rating * 0.65) + (norm_reviews * 0.35)
```

Function: Provides **dual ranking strategies**:

- **Sentiment-first**: Prioritizes aspect-based sentiments (skill, cost, convenience) based on service type
- **Objective-first**: Uses normalized weighted scoring of ratings and review counts

The logarithmic transformation of review counts prevents high-volume clinics from dominating results purely on quantity.[20] [18]

Stage 4: Context-Aware Response Generation

```
context = ""
if top_clinics:
    clinic_data_for_prompt = []
    for clinic in top_clinics:
        clinic_info = {
            "name": clinic.get('name'),
            "address": clinic.get('address'),
            "rating": clinic.get('rating'),
```

```
            "reviews": clinic.get('reviews'),
            "website_url": clinic.get('website_url'),
            "operating_hours": clinic.get('operating_hours')
        }
        clinic_data_for_prompt.append(clinic_info)
    context = json.dumps(clinic_data_for_prompt, indent=2)
```

Function: Constructs a **structured context payload** containing only the most relevant clinic information. This approach follows RAG (Retrieval-Augmented Generation) principles by grounding the AI's response in factual data while preventing hallucinations.[13] [21]

Session Memory & State Persistence

```
return {
    "response": final_response.text,
    "applied_filters": final_filters,
    "candidate_pool": candidate_clinics
}
```

Function: Implements **session-level memory persistence** by returning the conversation state along with the response. The `applied_filters` maintain user preferences across turns, while `candidate_pool` enables efficient reuse of search results for follow-up queries.[3] [22] [23]

Why This Architecture Excels

Multi-Brain Specialization

Each AI component has a focused responsibility, improving accuracy and maintainability compared to monolithic prompt approaches.[7] [6]

Progressive Filtering

The quality gate → factual filter → ranking pipeline ensures high-quality, relevant results while maintaining user control.[19] [18]

Semantic Understanding

Vector embeddings capture intent better than keyword matching, especially for medical terminology and conversational queries.[17] [24] [1]

Conversation Continuity

The system maintains context across interactions while providing reset mechanisms, creating natural conversational flow.[11] [25] [9]

Current Limitations for Booking Integration

The architecture is **read-only** for recommendations but lacks:

- **Appointment scheduling APIs**
- **Calendar integration systems**
- **Booking confirmation workflows**
- **Real-time availability checking**

To add booking functionality, you'd need additional components for appointment management and integration with clinic management systems.

This represents a **sophisticated agentic AI implementation** that combines semantic search, multi-brain reasoning, and session management to create an intelligent dental clinic recommendation system with conversation memory.

⁂

1. https://www.pickl.ai/blog/semantic-search-with-embedding-models/
2. https://dev.to/vipascal99/building-a-full-stack-ai-chatbot-with-fastapi-backend-and-react-frontend-51ph
3. https://solace.com/blog/long-term-memory-agentic-ai-systems/
4. https://euclideanai.substack.com/p/fastapi-supabase-template-for-llm
5. https://www.youtube.com/watch?v=PlZcgIMk3aw
6. https://www.dailydoseofds.com/ai-agents-crash-course-part-12-with-implementation/
7. https://markovate.com/blog/agentic-ai-architecture/
8. https://www.linkedin.com/pulse/architecture-agentic-ai-key-components-explained-abhijit-kakhandiki-ni6uc
9. https://milvus.io/ai-quick-reference/how-does-langchain-manage-state-and-memory-in-a-conversation
10. https://google.github.io/adk-docs/sessions/
11. https://optiblack.com/insights/ai-chatbot-session-management-best-practices
12. https://arxiv.org/abs/2208.11018
13. https://coralogix.com/ai-blog/how-to-build-a-rag-chatbot-from-scratch-with-minimal-ai-hallucinations/
14. https://discuss.streamlit.io/t/question-regarding-session-management-for-chatbots-with-multiple-users/69117
15. https://learn.microsoft.com/en-us/azure/redis/overview-vector-similarity
16. https://blog.maximeheckel.com/posts/building-magical-ai-powered-semantic-search/
17. https://www.techtarget.com/searchenterpriseai/tip/Embedding-models-for-semantic-search-A-guide
18. https://docs.databricks.com/aws/en/generative-ai/vector-search
19. https://www.oracle.com/sg/database/vector-search/
20. https://milvus.io/blog/how-to-filter-efficiently-without-killing-recall.md
21. https://thepythoncode.com/article/build-rag-chatbot-fastapi-openai-streamlit

22. https://www.jit.io/resources/devsecops/its-not-magic-its-memory-how-to-architect-short-term-memory-for-agentic-ai
23. https://www.reddit.com/r/AI_Agents/comments/1htzozg/how_do_you_handle_ai_agents_memory_between/
24. https://www.elastic.co/search-labs/blog/enhancing-chatbot-capabilities-with-nlp-and-vector-search-in-elasticsearch
25. https://www.pinecone.io/learn/series/langchain/langchain-conversational-memory/