# Chatbot Vexere - Design Doc

### 1. Context and Objectives

Vexere, là một nền tảng đặt vé toàn diện, đang tìm cách nâng cao trải nghiệm khách hàng thông qua việc triển khai một AI Chatbot đa kênh. Tài liệu này mô tả kiến trúc của một hệ thống Chatbot Proof-of-Concept (POC) được thiết kế để giải quyết nhu cầu này.

Mục tiêu chính của Chatbot là hỗ trợ người dùng ở ba cấp độ năng lực:

* **L-1: FAQ:** Trả lời các câu hỏi thường gặp về chính sách hoàn vé, hành lý, và các dịch vụ khác.
* **L-2: Dịch vụ sau bán hàng (After-Service):** Xử lý các yêu cầu như đổi giờ, hủy vé. PoC này tập trung vào luồng **đổi giờ vé**.
* **L-3: Đặt vé:** Hướng dẫn người dùng qua quy trình đặt vé hoàn chỉnh.

Ngoài ra, kiến trúc phải được thiết kế để có thể mở rộng, hỗ trợ các kênh tương tác đa phương thức như **Voice** và **Image** trong tương lai.

### 2. Overall System Architecture

**Sơ đồ kiến trúc:**

[Giao diện người dùng (Text, Image, Voice)]  
 ↕  
[Lõi Chatbot (Chatbot Core - src/main.py)]  
 |  
 ├─→ [Lớp xử lý L1: RAG cho FAQ] ───→ [Knowledge Base (faq\_vexere.csv)]  
 |  
 ├─→ [Lớp xử lý L2/L3: Quản lý luồng hội thoại] ───→ [API Backend Vexere]  
 |  
 └─→ [Lớp xử lý: Dịch vụ Đa phương tiện (Chờ triển khai)]

**Main Workflow:**

1. **Bộ điều phối trung tâm (Chatbot class trong src/main.py):** Tiếp nhận tất cả đầu vào từ người dùng.
2. **Phân loại ý định:** Dựa trên các từ khóa (đổi giờ, đặt vé) hoặc trạng thái hội thoại hiện tại, bộ điều phối sẽ kích hoạt luồng xử lý phù hợp.
3. **Luồng L1 - FAQ:** Nếu không có luồng nghiệp vụ nào đang diễn ra, yêu cầu của người dùng sẽ được xử lý bởi chuỗi **RAG (Retrieval-Augmented Generation)**. Hệ thống tìm kiếm thông tin liên quan từ cơ sở dữ liệu vector (được tạo từ faq\_vexere.csv) và sử dụng LLM để tạo ra câu trả lời tự nhiên. [cite: src/main.py]
4. **Luồng L2/L3:** Khi một luồng được kích hoạt, **State Machine** trong src/state.py sẽ tiếp quản. Nó theo dõi trạng thái hiện tại (current\_flow, current\_state) và thu thập thông tin cần thiết (slots) như mã vé, tuyến đường, để hoàn thành tác vụ. [cite: src/state.py]
5. **Lớp Dịch vụ (src/services.py):** Các tác vụ như đổi vé hoặc tạo đặt chỗ sẽ gọi các hàm trong lớp này. Lớp này hoạt động như một lớp trừu tượng, giả lập việc gọi đến API của Vexere, giúp tách biệt logic chatbot khỏi các hệ thống bên ngoài. [cite: src/services.py]

### 3. Technology Stack and Rationale

| **Components** | **Techs** | **Reason** |
| --- | --- | --- |
| **Framework LLM** | LangChain | Đơn giản hóa việc xây dựng các chuỗi RAG, quản lý prompt và tương tác với LLM. |
| **Models LLM** | Google Gemini 1.5 Flash | Cung cấp khả năng suy luận tốt, xử lý hội thoại nhiều lượt và tối ưu về tốc độ/chi phí. |
| **Embedding** | HuggingFace (hiieu/halong\_embedding) | Mô hình embedding mã nguồn mở, chất lượng cao cho tiếng Việt, phù hợp cho PoC mà không tốn chi phí. |
| **Vector DataBase** | ChromaDB | Cơ sở dữ liệu vector nhẹ, dễ dàng cài đặt và sử dụng tại local, lý tưởng cho giai đoạn PoC. |
| **Testing** | Unittest (trong test\_flows.py) | Framework kiểm thử tiêu chuẩn của Python để xây dựng các kịch bản kiểm thử toàn diện. [cite: test\_flows.py] |
| **CI/CD** | GitHub Actions | Tự động chạy kiểm thử mỗi khi có thay đổi trên nhánh main, đảm bảo chất lượng code. [cite: agondise/vexere-chatbot/Vexere-chatbot-a960ca616adf261a5214ee6c2c1b6b58d914091a/.github/workflows/ci.yml] |

### 4. Testing & Continuous Integration (CI) Pipeline

* **Testing:** Test end-to-end trong test\_flows.py. Các bài test này mô phỏng toàn bộ cuộc hội thoại cho từng luồng chính (FAQ, Đổi vé, Đặt vé) để đảm bảo tất cả các thành phần hoạt động chính xác với nhau.
* **Pipeline CI:** Tệp .github/workflows/ci.yml định nghĩa một pipeline trên GitHub Actions. Pipeline này tự động:
  1. Kích hoạt khi có push hoặc pull\_request vào nhánh main.
  2. Thiết lập môi trường Python.
  3. Cài đặt các thư viện từ requirements.txt.
  4. Chạy bộ kiểm thử test\_flows.py.
  5. Thông báo lỗi nếu có bất kỳ bước nào thất bại, ngăn chặn việc tích hợp code lỗi vào nhánh chính.

### 5. Limitations and Future Work

Kiến trúc PoC này được thiết kế để dễ dàng mở rộng thành một hệ thống production hoàn chỉnh.

* **Tích hợp Đa phương tiện:**
  + **Image:** Chức năng process\_ticket\_image trong services.py có thể được triển khai bằng các dịch vụ OCR/IDP như Google Cloud Vision để tự động trích xuất mã vé từ ảnh người dùng gửi lên.
  + **Voice:** Các hàm speech\_to\_text và text\_to\_speech sẽ được tích hợp với các API như Google Speech-to-Text hoặc các mô hình như Whisper của OpenAI để cho phép người dùng tương tác bằng giọng nói.
* **Khả năng mở rộng (Scalability):**
  + **Vector Database:** Thay thế ChromaDB cục bộ bằng một giải pháp được quản lý như Pinecone, Weaviate, hoặc Vertex AI Vector Search để xử lý lượng dữ liệu lớn hơn.
  + **Quản lý trạng thái:** Chuyển từ việc lưu trạng thái trong bộ nhớ sang một kho lưu trữ bền vững như Redis hoặc DynamoDB để quản lý các phiên hội thoại của nhiều người dùng cùng lúc.
* **Cải thiện NLU:** Có thể thêm một module Nhận dạng Ý định (Intent Recognition) và Trích xuất Thực thể (Entity Extraction) rõ ràng hơn để phân loại yêu cầu của người dùng chính xác hơn thay vì chỉ dựa vào từ khóa.