Ecole Supérieure d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

Université Internationale de Rabat
THE INNOVATIVE UNIVERSITY

Rihla

AI-Powered Morocco Travel Assistant

Project Report

Réalisé par:

Ihssane IBRAHIMI

et

Siham Hamou

Encadré par :

Mr Hamza Gamouh et

Mr Hakim Hafidi

Table of Contents

1. Introduction

Planning a trip to Morocco can be challenging due to the diversity of destinations, cultural richness, and lack of personalized travel planning tools. Travelers often rely on static travel guides that do not adapt to individual needs such as available time or budget.

Rihla is an AI-powered Morocco Travel Assistant that provides conversational travel recommendations and generates personalized itineraries. The system combines modern AI techniques such as Retrieval-Augmented Generation (RAG), vector databases, and Large Language Models (LLMs) to deliver accurate and contextual responses.

---

2. Problem Definition

The main problems addressed by this project are: - **Lack of personalized travel planning solutions**. - **Difficulty accessing structured and reliable information about Moroccan destinations**. - Inefficient manual itinerary planning based on days and budget.

The goal is to design an intelligent system capable of understanding user queries and providing relevant, context-aware travel information.

---

3. Project Objectives

The objectives of the Rihla project are: - Develop an AI-based conversational travel assistant. - Integrate a vector database for semantic search. - Generate personalized itineraries based on user constraints. - Build a complete frontend-backend architecture. - Deliver a functional Minimum Viable Product (MVP).

---

4. Project Requirements and Constraints

- Team size: **2 members**.
- Mandatory use of AI models via APIs.
- Integration of a vector database.
- Implementation of a frontend and backend.
- Clear documentation and presentation of results.

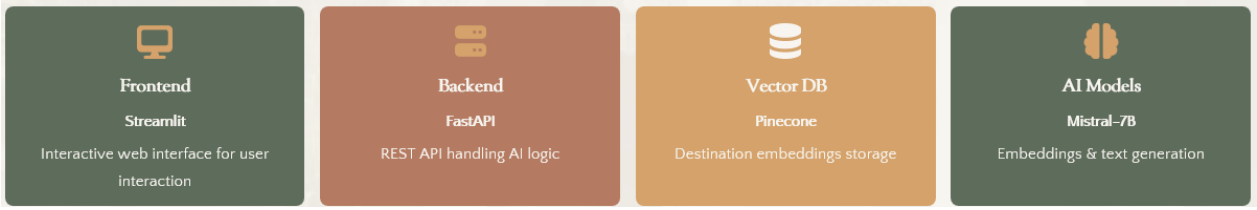
---

5. System Architecture

The system follows a client-server architecture with clear separation of responsibilities.

Components: - **Frontend:** Streamlit web interface. - **Backend:** FastAPI REST API. - **Vector Database:** Pinecone. - **AI Models:** Sentence-Transformers for embeddings and Mistral-7B via Hugging Face.

System architecture diagram showing Frontend → FastAPI → Pinecone → LLM.



---

6. AI Techniques and Models Used

6.1 Retrieval-Augmented Generation (RAG)

Rihla uses the Retrieval-Augmented Generation approach. When a user submits a query, the system retrieves relevant destination data from a vector database and injects this context into the prompt sent to the LLM.

Code snippet showing query embedding and Pinecone similarity search.

```
query_embedding = embedding_model.encode(message.text).tolist()
results = index.query(
    vector=query_embedding,
    top_k=5,
    include_metadata=True
)

RELEVANCE_THRESHOLD = 0.35
relevant_matches = [m for m in results.matches if m.score >= RELEVANCE_THRESHOLD]
```

6.2 Embedding Model

- Model: **`sentence-transformers/all-MiniLM-L6-v2`.**
- Purpose: Convert user queries and destination descriptions into numerical vectors for semantic similarity search.

Code snippet showing `embedding_model.encode()` usage.

```
embedding_model = SentenceTransformer(EMBEDDING_MODEL)
print(" Embedding model loaded")
```

```
    query_embedding = embedding_model.encode(message.text).tolist()
```

6.3 Large Language Model (LLM)

- Model: **`mistralai/Mistral-7B-Instruct-v0.2`.**
- Accessed via **Hugging Face Inference API.**
- Generates conversational and human-like responses.

Code snippet showing Hugging Face `text_generation()` call.

```
    response = hf_client.text_generation(
        prompt,
        model=LLM_MODEL,
        max_new_tokens=500,
        temperature=0.7,
        top_p=0.9,
        repetition_penalty=1.1,
        return_full_text=False
    )
```

---

7. Vector Database Integration (Pinecone)

Pinecone is used to store vector embeddings of Moroccan destinations.

Features: - Fast similarity search using cosine similarity. - Retrieval of top-k relevant destinations.

Code snippet showing `index.query()` with `top_k=5`.

```
results = index.query(
    vector=query_embedding,
    top_k=5,
    include_metadata=True
)
```

---

8. Backend Implementation (FastAPI)

The backend is implemented using FastAPI and handles all AI-related logic.

Main endpoints: - `/chat`: Handles conversational queries. - `/itinerary`: Generates personalized itineraries. - `/health`: Service status check. - `/docs`: API documentation.

Code snippet showing `/chat` endpoint implementation.

```
@app.post("/chat", response_model=ChatResponse)
async def chat(message: Message):
```

```
response_text = generate_llm_response(message.text, context_data)
```

---

9. Frontend Implementation (Streamlit)

The frontend provides a clean and interactive user interface.

Features: - Chat-based interaction. - Sidebar destination shortcuts. - Itinerary generation form.

Code snippet showing frontend sending POST request to `/chat`.

```
    response = requests.post(
        "http://127.0.0.1:8000/chat",
        json={"text": last_message["content"]},
        timeout=30
    )
```

Code snippet showing itinerary form (`number_input`, `selectbox`).

```
days = st.number_input(
    "Number of days",
    min_value=1,
    max_value=14,
    value=5
)

budget = st.selectbox(
    "Budget",
    ["budget", "moderate", "luxury"]
)
```

---

10. Itinerary Generation Logic

The itinerary generator uses predefined templates based on the number of days selected by the user.

Parameters: - Number of days. - Budget level (budget, moderate, luxury).

Code snippet showing itinerary template selection logic.

```
response = requests.post(
    "http://127.0.0.1:8000/itinerary",
    json={
        "days": days,
        "budget": budget
    },
    timeout=30
)
```

---

11. Application Workflow

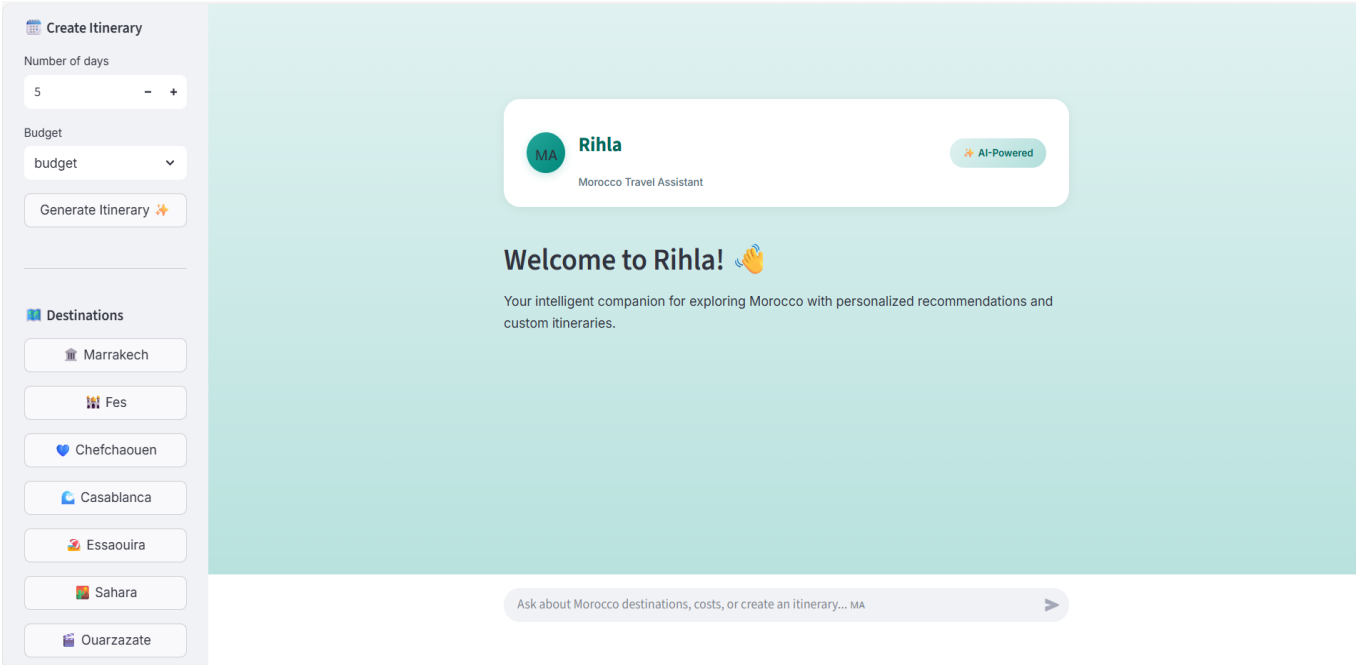
1. User submits a question or itinerary request.
2. Frontend sends the request to FastAPI backend.
3. Backend encodes query and retrieves relevant data from Pinecone.
4. Retrieved context is sent to the LLM.
5. AI-generated response is returned to the frontend.

---

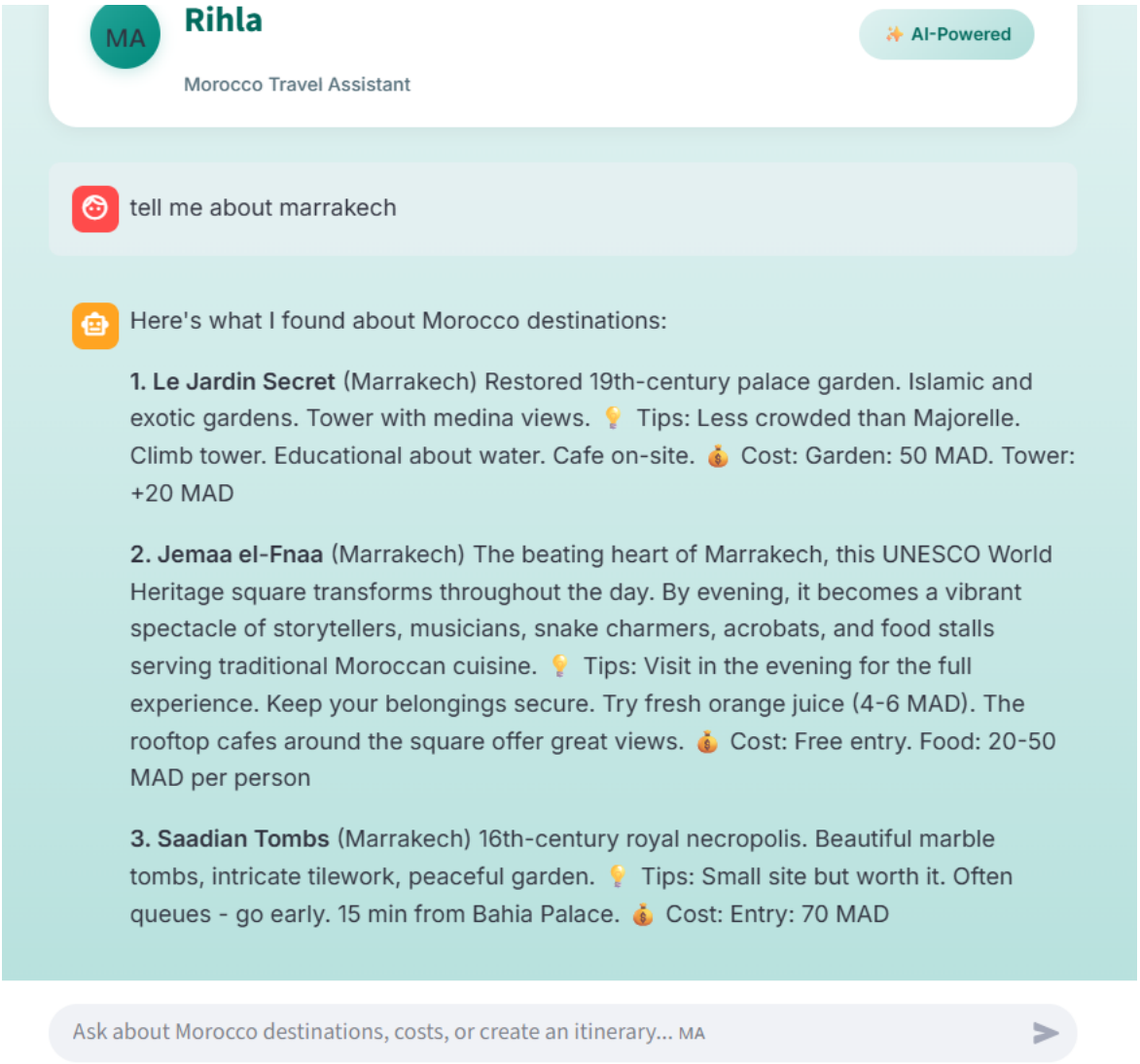
12. Results and Demonstration

The application successfully demonstrates: - Accurate AI-generated travel recommendations. - Personalized itinerary generation. - Stable frontend-backend communication.

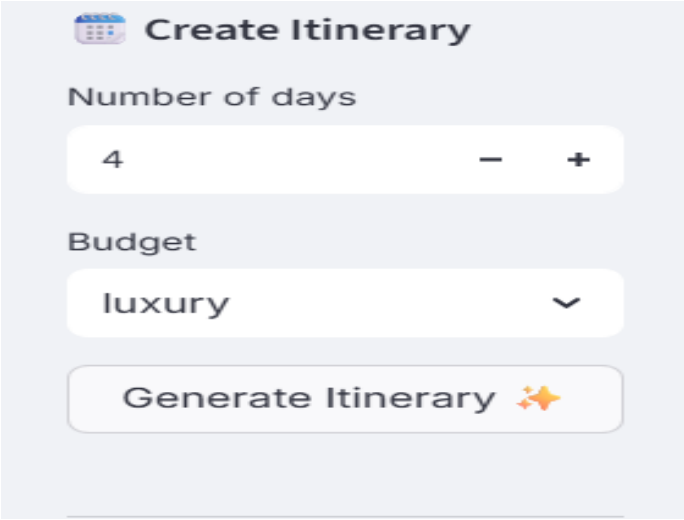
Home page of the application.



Chat interaction example (“Tell me about Marrakech”).



Generated itinerary displayed in the chat.



---

13. Teamwork and Task Distribution

The project was completed by a **team of 2 members**.

Task	Member 1	Member 2
Backend & AI Logic	**yes**	
Vector DB Integration	**yes**	
Frontend Development		**yes**
UI/UX Design		**yes**
Testing & Documentation	**yes**	**yes**

---

14. Challenges and Solutions

Challenge: AI fallback responses when context was insufficient. - **Solution:** Improved relevance filtering and prompt engineering.

Challenge: Frontend-backend synchronization. - **Solution:** Unified API contracts and error handling.

---

15. Future Improvements

- User authentication and profiles.
- Interest-based itinerary customization.
- Online deployment.
- Multilingual support.

---

16. Conclusion

Rihla demonstrates a complete AI-powered application combining Retrieval-Augmented Generation, vector databases, and modern web technologies. The project fulfills all academic requirements and highlights effective teamwork and applied AI skills.