

Large Scale Question

Answering at Tourism Data

Project Architecture

Domain: NLP

Author : Debasish Mohanty

Date: 15.02.2024

**DOCUMENT VERSION CONTROL**

## Change Record:

|  |  |  |
| --- | --- | --- |
| **VERSION** | **DATE** | **AUTHOR** |
| 1 | 15- Feb - 2024 | **Debasish Mohanty** |
|  |  |  |
|  |  |  |
|  |  |  |

# CONTENTS

1. [Introduction 0](#_TOC_250005)3
   1. [What is Architecture Design Document? 0](#_TOC_250004)3
   2. [Scope 0](#_TOC_250003)3
2. [Architecture 0](#_TOC_250002)4
   1. Development Architecture 04
   2. Deployment Architecture 05
   3. Architecture Description 05
3. Deployment 06
   1. [Frontend with ReactJS 0](#_TOC_250001)6
   2. [Backend with FastAPI 0](#_TOC_250000)6
   3. Deployment on Cloud 06

# Introduction

## What is Architecture design document?

Any software needs the architectural design to represents the design of software. IEEE defines architectural design as “the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.” The software that is built for computer-based systems can exhibit one of these many architectures.

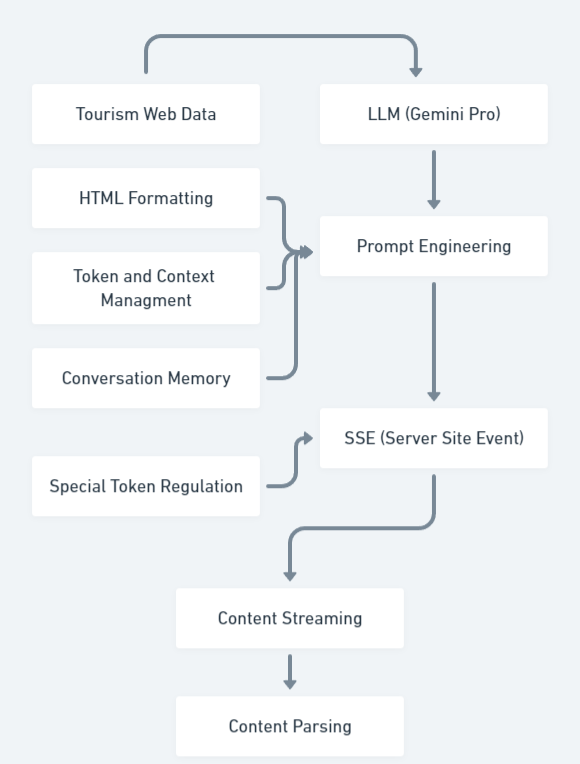
Each style will describe a system category that consists of :

* A set of components (e.g. : a database, computational modules) that will perform a function required by the system.
* The set of connectors will help in coordination, communication, and cooperation between the components.
* Conditions that how components can be integrated to form the system.
* Semantic models that help the designer to understand the overall properties of the system.

## Scope

Architecture Design Document (ADD) is an architecture design process that follows a step-by-step refinement process. The process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the design principles may be defined during requirement analysis and then refined during architectural design work.

1. **Architecture:**
   1. **Development Architecture**

****

* 1. **Deployment Architecture**

**FRONTEND**

**(React JS)**

**CORS**

**Configuration**

**LLM Endpoint**

**BACKEND**

**(FastAPI)**

**Cloud Deployment**

**(AWS EC2)**

* 1. **Architecture Description**
     + **Tourism Web Data**

LLMs trained on web data, which often includes tourism information, might implicitly learn tourism-related knowledge through their weights and biases. This opens up possibilities for either enriching the input with additional context or directly utilizing the LLM's internal tourism representation.

* + - **LLM (Gemini - Pro)**

For our tourism chatbot project, we selected a Google LLM specifically for its robust understanding of travel information, thanks to its superset training data. The easy-to-use API endpoints facilitated seamless integration into our development process

* + - **Prompt Engineering**

Although the LLM stores relevant information, we recognized the need for tailored formatting (e.g., summaries, lists) and clear state indications (e.g., "I'm still thinking"). Additionally, a context management system helps maintain conversation coherence, enhancing overall user experience.

* + - **Server-Site Events (SSE)**

Transformer-based LLMs excel at generating data in a stream, but minimizing user wait time requires specific architecture. We address this by utilizing backend generators for efficient response creation and leveraging Server-Sent Events (SSE) on the frontend for real-time data transfer, ensuring a seamless user experience.

* + - **Content Parsing:**

Presenting the LLM's output directly as text wouldn't provide the desired formatting and user experience. Therefore, we utilize HTML parsing on the frontend. This enables us to stream and interpret HTML tags embedded within the LLM response, ensuring proper formatting and preventing issues arising from unclosed tags.

1. **Deployment**
   1. **Frontend with ReactJS**

In this project, the user interface (UI) was developed using ReactJS, a JavaScript library. The front-end implementation involved the creation of a robust framework to facilitate the communication with the backend through API endpoints. To retrieve data from the server-side, a predominant utilization of the axios library was observed, enabling seamless and efficient API calls.

* 1. **Backend with FastAPI**

The backend API for this project was implemented using the FastAPI framework, renowned for its efficiency and performance. It served as the foundation for seamlessly integrating the machine learning model into the backend infrastructure. The FastAPI backend was responsible for receiving and processing user inputs, seamlessly feeding them into the underlying model for prediction, and subsequently delivering the corresponding results back to the frontend interface. It’s important to mention that some of the personal modules has been included in the code which performs certain tasks.

* 1. **Deployment on Cloud**

The fully operational code was obtained from the code repository using Git and deployed on an AWS EC2 instance. The necessary configuration was performed to enable Cross-Origin Resource Sharing (CORS) on the instance, allowing the service to be accessed via HTTP and HTTPS protocols. The ReactJS and FastAPI processes were seamlessly interconnected by appropriately configuring their respective ports, enabling smooth interaction between the frontend and backend via the API endpoints. Finally, the functionality of the application was tested by accessing it through the assigned IP address and the domain name provided by AWS, ensuring a seamless user experience.