

Large Scale Question

Answering at Tourism Data

Low Level Design

Domain: NLP

Author : Debasish Mohanty

Date: 15.02.2024

**Document Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Date issued | Version | Description | Author |
| Feb 15th , 2024 | 1.1 | First Draft | Debasish Mohanty |

CONTENTS

1. [Introduction](#_Toc110595994) 03

1.1[What is Low-Level Design Document?](#_Toc110595995) 03

1.2[Scope](#_Toc110595996) 03

2.Architecture 04

2.1Model Development Architecture 04

2.2Model Deployment Architecture 05

2.3Architecture Description 05

3.Deployment 06

3.1Frontend with ReactJS 06

3.2Backend with FastAPI 06

3.3Deployment on Cloud 07

4. Unit Cases 08

# Introduction:

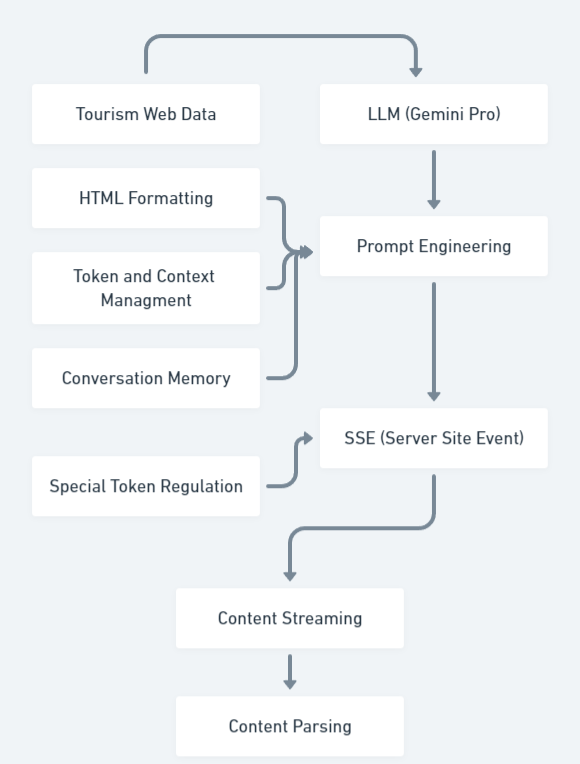
## 1.1 What is Low-Level Design Document?

The goal of LLD or a low-level design document is to give the internal logical of the actual program code for LC50 toxicity value prediction. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli.

## 1.2 Scope:

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

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

****

* 1. **Model Deployment Architecture**

**FRONT-END**

**(React JS)**

**CORS and Ports**

**Configuration**

**LLM Endpoint**

**BACKEND-END**

**(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.

* 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.

# Unit cases

|  |  |  |
| --- | --- | --- |
| Test Case Description | Pre-Requisite | Expected Result |
| Verify whether the Application URL is  accessible to the user | Application URL should be defined | Application URL should be accessible to the user |
| Verify whether the Application loads completely for the user when the URL is accessed | 1. Application URL is accessible   2. Application is deployed | The Application should load completely for the user when the URL is accessed |
| Verify whether user is able to see input fields. | Application is accessible | User should be able to see input fields |
| Verify whether user is able to edit all input fields | Application is accessible | User should be able to edit all input fields |
| Verify whether user gets Submit button to submit the inputs | Application is accessible | User should get Submit button to submit the inputs |
| Verify whether user is presented with results on clicking submit | Application is accessible | User should be presented with results on clicking submit |
| Verify whether the results are in accordance to the selections user made | Application is accessible | The results should be in accordance to the selections user made |