Implementation Choices & Challenges Report

Implementation Choices

1. **Framework Selection**:
    - FastAPI was chosen due to its asynchronous capabilities, ease of use, and support for dependency injection. It also provides automatic generation of interactive API documentation.
2. **Data Storage & Retrieval**:
    - ChromaDB was used for embedding storage and retrieval. It provides fast querying capabilities and is compatible with the `SentenceTransformer` model for embedding generation.
    - Pandas was used for handling CSV data for analytics.
3. **Embedding Model**:
    - The `SentenceTransformer` model was used for generating embeddings of user queries. These embeddings are then queried against the stored embeddings in ChromaDB to retrieve relevant context.
4. **Hugging Face Inference API**:
    - Integrated for handling natural language generation using LLMs. The access token has already been added if it does not work use a new Hugging face token and ensure that the Hugging Face token is properly configured with all required permissions enabled. If authentication issues persist, generating a new token with the necessary permissions may resolve the problem.
5. **API Endpoints**:
    - `/analytics` (POST): Provides analytical data based on the year and month specified by the user.
    - `/ask` (POST): Processes user queries, generates embeddings, and retrieves relevant context before querying the LLM.
    - `/health` (GET): Checks the health of various components to ensure proper connectivity.

Challenges Faced

1. **Handling Different Query Types**:
    - Implementing a generalized query handling mechanism was challenging, particularly for distinguishing between analytical queries and LLM-based queries.
2. **Embedding Storage Management**:
    - Efficiently storing and retrieving embeddings from ChromaDB required careful consideration of indexing and querying mechanisms.
3. **API Integration with Hugging Face**:
    - Ensuring consistent and low-latency communication with the Hugging Face Inference API posed some challenges.

API Endpoints & Example Inputs/Outputs

/analytics (POST)

Input:

```
{
    "year": 2017,
    "month": 7
}
```

Output:

```
{
    "total_revenue": 759959.6699999999,
    "average_booking_price": 143.03776962168266,
    "num_bookings": 5313,
    "most_popular_hotels": {
        "City Hotel": 3559,
        "Resort Hotel": 1754
    },
    "monthly_revenue": {
        "2017-07-31T00:00:00": 759959.67
    },
    "yearly_revenue": {
        "2017-12-31T00:00:00": 759959.67
    },
    "cancellation_rate": 37.342367777150386
}
```

/ask (POST)

Input:

```
{
    "question": "What is the average price of a hotel booking?"
}
```

Output:

```
{
    "answer": "Context: Booking at Resort Hotel on 2016-11-11
00:00:00.000000 with price 85.0. Booking at Resort Hotel on 2016-11-11
00:00:00.000000 with price 85.0. Booking at Resort Hotel on 2016-09-10
00:00:00.000000 with price 85.62. Booking at Resort Hotel on 2016-11-12
00:00:00.000000 with price 85.0. Booking at Resort Hotel on 2016-11-12
00:00:00.000000 with price 85.0.\nQuestion: what is the average price of a
hotel booking?\nAnswer: The total amount of all bookings / the number of
bookings, which is 503.12 / 6 = 83.852. So the average price of a booking
is $83.85."
}
```

/health (GET)

Output:

```
{
    "database": "✅ Database connection successful",
    "chroma_db": "✅ ChromaDB connection successful",
```

```
    "embedding_model": "✅ Embedding model loaded successfully",
    "huggingface_api": "✅ Hugging Face Inference API connected
successfully"
}
```

The `/health` endpoint ensures that all essential services are connected and functioning properly.

Conclusion

The implementation successfully integrates analytics computation and natural language query handling with robust health-checking mechanisms.