Technical University of Kosice

Faculty of Electrical Engineering and Informatics

Implementation of Q&A bot using LLM application

2025 Lukianytsia, Sachenko, Levchyk, Tovtyn, Krysanova, Kostianets

Contents

Introduction **1**

1 Problem formulation **2**

2 Analysis **3**

2.1 Expected Functionality 3

2.2 Constraints and Assumptions 4

2.3 Dataset overview 4

3 Processing and storing data **6**

4 Models evaluation **7**

5 Local and Cloud hosted models **10**

5.1 Local hosted models 10

5.2 Cloud Hosted Models 11

6 Application flow **13**

6.1 Setup 13

6.2 User data extraction 13

6.3 Chat history 14

6.4 Context building and answering 15

6.5 Application view 17

7 Future work **19**

8 Conclusion **21**

Introduction

Integrating artificial intelligence into everyday applications has revolutionized how systems handle complex tasks. Among the most transformative advancements are Large Language Models (LLMs), which excel at understanding and generating natural language. In this project, we aim to create an intelligent Question and Answer (Q&A) system for healthcare, designed to process patient medication records stored in structured formats like CSV files. We aim to empower users, including patients and healthcare professionals, to interact with data seamlessly through natural language queries.

For example, users could ask, "What medications did I take in 2023?" or "For how long did I take Ibuprofen?" The system would interpret the queries, retrieve the relevant information, and provide meaningful, context-aware responses. This involves leveraging LLMs, advanced preprocessing techniques, and a hybrid data storage architecture combining SQL for structured queries with a vector database for semantic similarity matching. Additionally, the system features a user-friendly interface that ensures accessibility while maintaining the precision and relevance of its outputs.

Through this project, we aim to address the challenges of handling nuanced and sensitive healthcare data while pushing the boundaries of LLM applications in real-world scenarios.

1 Problem formulation

This study aims to test and implement an intelligent Q&A (Question & Answer) system using Large Language Models (LLM) technology. The work will focus on reviewing and testing existing LLM solutions available on the market, with to select the most suitable model for our specific needs. After thorough testing, the selected model will be fine-tuned on our data to increase its relevance and accuracy in the context of our domain. The process includes training the model on specific data so that it can answer questions related to information contained in these data. The work will result in a functional Q&A system, that can effectively interact with users and provide them with accurate answers to their questions.

2 Analysis

The implementation of this project utilized several technologies to ensure an efficient and user-friendly system. Python served as the primary programming language, used for the integration of large language models using the LangChain and Transformers frameworks. These tools were used for leveraging the capabilities of LLMs in processing natural language queries and generating context-aware responses.

The backend was developed using FastAPI, which allowed the creation of a lightweight, API service to handle requests and responses asynchronously and efficiently. For data storage, the Chroma vector database was employed to manage embeddings and enable similarity searches, ensuring accurate identification of users based on their names. Meanwhile, the frontend was built with React, offering a simple and intuitive user interface to access the system's functionalities.

This combination of technologies distributes responsibility between components for efficient query handling and a responsive user experience, ensuring the system's effectiveness and future scalability.

2.1 Expected Functionality

The Q&A system is designed to process patient medication records stored in a CSV file, leveraging Large Language Models for agentic generation of accurate and context-aware answers to user questions. The core functionality involves generating answers to user prompts derived from the dataset. Patients should be able to ask questions like, "Hi, my name is Arnold Kollár. What medications did I take in 2023?" or "Hello, I'm Hortenzia Skutecka, for how long did I take Ibuprofen?" The system should extract the patient's name, analyze questions, extract relevant context from the database, and generate answers, providing detailed and accurate responses.

The Q&A chatbot must process questions in English and respond in the same language. It should be able to handle ambiguous or incomplete questions by improving extracted context iteratively, or answering with predicted error messages like “I don’t know”, or “Please, provide more context.” An interactive user interface should allow users to input their questions and receive responses dynamically. The application must be well structured and consist of reusable components to offer a better integration experience for different environments. The web interface, written using the React framework, provides answers in a structured and user-friendly format. Additionally, the system should log questions and responses for debugging and future fine-tuning over gathered data.

2.2 Constraints and Assumptions

The current implementation uses a static dataset, so updates require manual application restart, which will cause database recreation.

Constraints include hardware limitations that may restrict the use of some models. The project prioritizes and investigates the use of open-source LLMs for text-generating purposes, but advanced techniques like text-to-SQL require access to the OpenAI models for more accurate results in less amount of time. Additionally, the CSV structure must remain consistent to ensure seamless data preprocessing.

2.3 Dataset overview

The dataset used for this Q&A system consists of patient medication records stored in a CSV file. It contains the following columns:

- patient_id: A unique identifier for each patient.
- meno and priezvisko: The first and last names of the patients.
- datum_narodenia: The date of birth of the patient.
- pohlavie: The gender of the patient.
- nazov_lieku: The name of the prescribed medication.
- cena: The price of the medication.

- mnozstvo: The quantity prescribed.
- datum_predpisu: The date the medication was prescribed.
- obdobie_lieku_dni: The duration for which the medication was prescribed (in days).
- datum_konca: The end date of the prescription.
- liek_vybral: Indicates whether the medication was collected (Yes/No).
- poistovna_zaplatila_spravne: Indicates whether the insurance covered the cost correctly (Yes/No).

3 Processing and storing data

Initially stored as a CSV file, the dataset undergoes several preprocessing steps to ensure consistency and easy semantic understanding. To ensure simplicity while working with the data, `liek_vybral` and `poistovna_zaplatila_spravne` fields are converted to the boolean format to clearer represent binary variance of data. Everything else seems to be fine, other concerning details would be date standardization: we verified they are stored in `YYYY-MM-DD` format, and missing values in some rows, that were not found during analysis.

The vector database stores embeddings generated from the names, each labeled with the patient's ID. This storage is crucial for handling scenarios where patients introduce themselves with minor errors or variations in their input. The embeddings allow the system to match user-provided data to the closest stored representation, even if it is not an exact match. For example, if a patient inputs “Arnald Kolár” instead of “Arnold Kollár”, the vector database uses semantic similarity to identify the intended patient correctly. This eliminates the need for strict exact matching and enables the system to identify patients smoothly and effectively.

The SQL database acts as the static store for structured patient data. This includes personal details, prescription records, medication costs, and insurance information. The SQL database is designed to handle precise, deterministic queries generated from the context of our patient's question. As an enhancement, it is good to have several tables and then select context data using more complex queries, or more queries, however, it would give an application unnecessary complexity while providing no notable impact.

By combining these two storage types and separating data early, we ensure that every part of our application uses the correct way of handling different situations, and the solution is scalable for adding new knowledge to the flow.

4 Models evaluation

We conducted a thorough evaluation of three large language models—OLLAMA 3.2, MISTRAL, and GEMMA 2—paired with ChatGPT for database queries. The assessment was carried out using Promptfoo as the benchmarking tool, with random test cases generated from our dataset through data analysis.

Evaluation Methodology

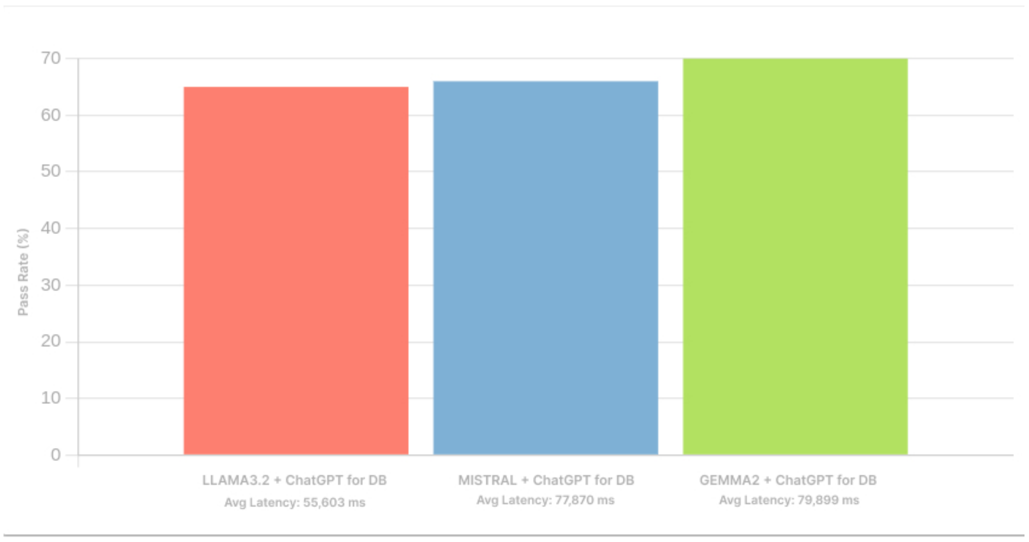
Using our dataset, we applied data analysis techniques to create diverse and randomized test cases. These cases represented a variety of real-world scenarios, including:

- Fact-based questions.
- Edge cases requiring reasoning and multi-step problem-solving.
- Ambiguous or incomplete queries to test the models' contextual understanding.

Benchmarking Framework: Promptfoo

We leveraged Promptfoo, a powerful tool for LLM evaluation, to:

- Compare the three models systematically across 100 randomly generated test cases.
- Visualize performance trends through metrics like pass rates, prompt scores, and latency.
- Analyze precision and robustness in the models' ability to generate accurate and relevant responses.



Obrázok 4 – 1 Evaluation Results

Evaluation Metrics

The models were evaluated based on the following metrics, with ChatGPT serving as the reference model to evaluate the responses of our Q&A LLM:

- Pass Rate: The percentage of queries answered correctly by the evaluated models, judged against the responses generated by ChatGPT as the benchmark for correctness and relevance.
- Latency: Average response time in milliseconds, measuring the efficiency and speed of each model.

Evaluation Results

Model	**Pass Rate (%)**	**Avg Latency(ms)**	**Key Insights**
GEMMA 2	70% (70/100)	79.9	Superior in precision and relevance, best for complex, multi-step queries.
MISTRAL	66% (66/100)	68.3	Balanced performance, ideal for general-purpose use.
OLLAMA 3.2	65% (65/100)	55.8	Fastest model, best for real-time applications, slight drop in accuracy for complex queries.

Table 4 − 1 Combined Analysis of Pass Rate, Latency, and Key Insights

5 Local and Cloud hosted models

Integrating large language models into our system involves a mix of local and cloud-hosted approaches, each with distinct benefits and considerations. This section explores these approaches, provides a comparative analysis, and describes the state of development of the models used in the project.

5.1 Local hosted models

Local hosting of models refers to deploying the LLMs on local hardware or private servers within the system's infrastructure. In our project, this approach was used for models such as Llama3.2 and Gemma, which are open-source and fine-tuned for general text generation and chat-oriented use cases.

Advantages:

1. **Data Privacy and Security**: local models ensure sensitive patient data remains within the organization's infrastructure, reducing exposure to external threats and enhancing compliance with privacy regulations.
2. **Lower Latency**: by eliminating the dependency on internet connectivity, local models provide faster response times, making them ideal for real-time applications.
3. **Cost Efficiency**: for systems with high query volumes, hosting models locally can be more cost-effective over time, avoiding recurring cloud service fees.
4. **Customization**: local deployment allows extensive customization, including fine-tuning and integrating additional domain-specific knowledge.

Disadvantages:

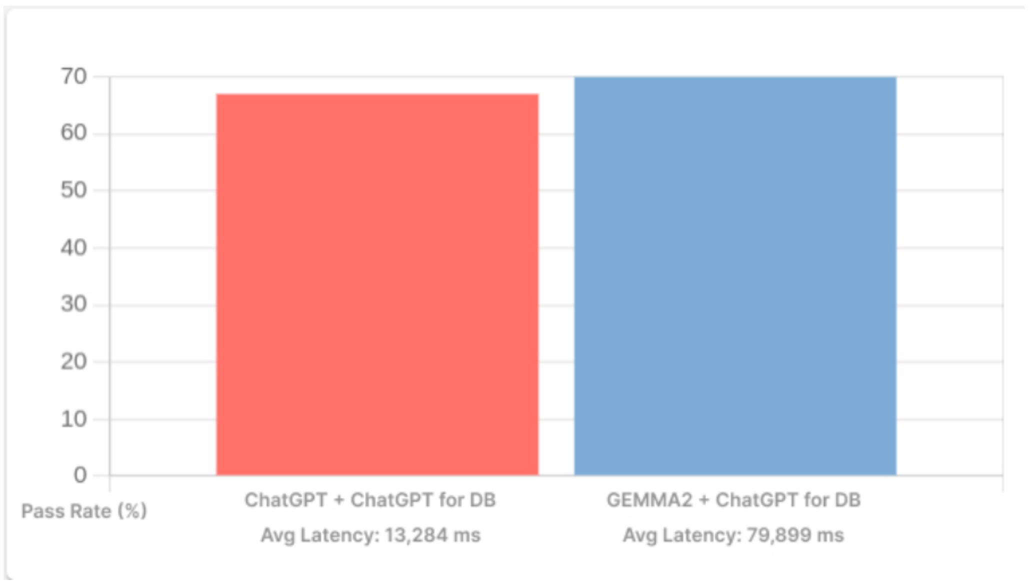
1. **High Resource Requirements**: running LLMs locally demands substantial computational resources, including high-performance GPUs and significant memory.
2. **Maintenance Overhead**: regular updates, optimization, and management of the infrastructure add complexity and operational costs.
3. **Limited Scalability**: scaling local models requires additional hardware investment, which may not be feasible for rapidly growing applications.

5.2 Cloud Hosted Models

Cloud-hosted models like ChatGPT (OpenAI) or Claude (Anthropic) are provided as APIs and managed entirely by cloud providers. These were considered for advanced capabilities, particularly for text-to-SQL tasks requiring higher accuracy and fewer constraints.

Advantages:

1. **Ease of Use**: cloud models provide plug-and-play functionality, allowing quick integration without significant setup effort.
2. **Scalability**: the elastic nature of cloud services enables handling large query volumes without additional infrastructure investment.
3. **Regular Updates**: providers maintain and optimize these models, ensuring access to the latest advancements and features.
4. **Advanced Performance**: many cloud models are pre-trained on extensive datasets, providing superior accuracy and generalization, especially for complex or ambiguous queries.



Obrázok 5 − 1 Comparison of local and cloud-based models

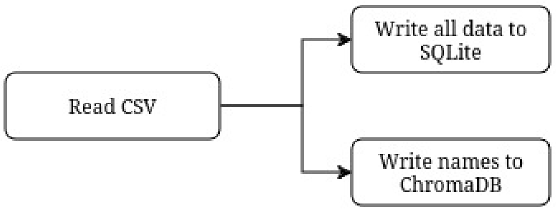
Disadvantages:

1. **Data Privacy Concerns**: sending sensitive data to cloud providers may raise security and compliance issues, particularly in healthcare applications.
2. **Dependency on Network Connectivity**: cloud models require a stable internet connection, which could introduce latency or downtime in case of connectivity issues.
3. **Recurring Costs**: usage-based pricing models can become expensive for applications with high query volumes, especially for real-time requirements.

6 Application flow

6.1 Setup

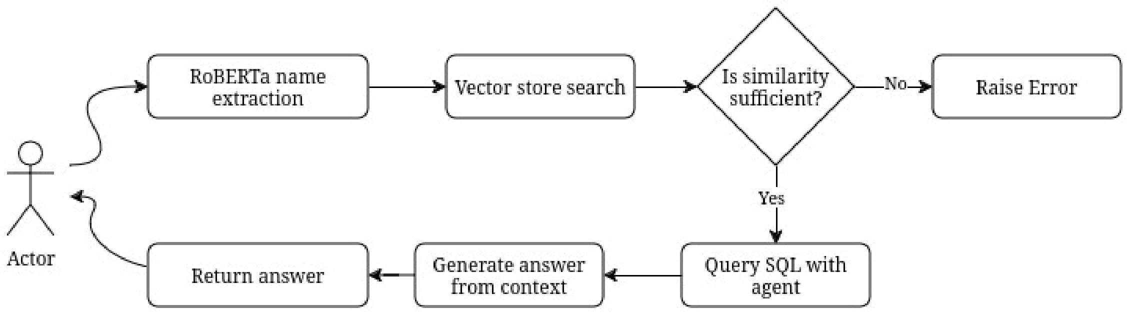
Our application starts working by processing the provided CSV dataset, which contains static patient information. The first step is to read and parse the CSV file, then, two key data stores are initialized and populated. The entire dataset is written into an SQLite database, which acts as a static store for patient information. At the same time, patients' full names and identifiers are extracted and stored in a ChromaDB vector database. The purpose of several storage systems is to provide modular query processing and error generation during user interactions. While the SQLite database serves as the main source for querying collected patient data, the vector database ensures that the system can perform similarity checks for the patients' names. Simple flow is shown on $6-1$



Obrázok 6 − 1 Data sources setup

6.2 User data extraction

Once the application is set up and databases are initialized, the primary functionality revolves around processing patient's prompts. When the endpoint for answer generation is called, as the first step, a name extraction is performed using a RoBERTa model to identify the patient's name and surname. The extracted name is then used to search the vector database for similarity matching. If the similarity score of the selected entity meets the predefined threshold, 0.3 in our case, we are using the patient_id for further processing. This validation ensures that patients who didn't



Obrázok 6 – 2 Application Flow

introduce themselves or did it incorrectly won't get random data as an output, in case of insufficient similarity, an error is thrown and returned to the patient.

The workflow advances to context data aggregation and building for questions with sufficient similarity of extracted names.

6.3 Chat history

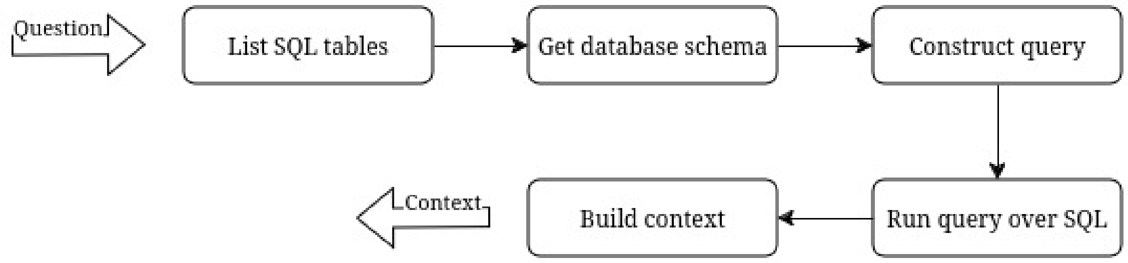
The chat history is maintained using a Redis-based storage system, ensuring fast and efficient conversation data retrieval. Each chat session is associated with a unique identifier (chat ID) used to store and retrieve the conversation history. This history includes the interactions between the user and the system, capturing user queries and system responses. The storage mechanism ensures that:

Each new query is appended to the session's existing history. Historical data can be analyzed for context when generating responses. Session continuity is maintained, even during multi-turn conversations. The system uses this stored data during each interaction to decide whether the current query can be answered using previously gathered information. If the context from earlier interactions suffices, a response is generated based on the stored data, eliminating the need for data extraction and processing.

6.4 Context building and answering

The agent component in our system is implemented using the LangGraph React Agent. This agent handles dynamically generating SQL queries based on the patient's input and building the context for accurate question resolution. The agent initiates the process by listing all available SQL tables. Once the tables are identified, the agent retrieves the database schema, this provides detailed information about the structure and attributes of each table, including column names and their data types, which helps to construct accurate SQL queries over existing data. The agent constructs an SQL query using the retrieved schema tailored to the user's input. This process involves understanding the user's question, aligning it with the available data, and translating it into a valid SQL statement. After constructing the query, the agent executes it against the database. The query result provides the raw data required to build a response.

The next step in the agent workflow is context generation. The raw query results are formatted to a more structured view. In the end, the initial question and created context are passed to the LLM, which generates meaningful textual answer based on the provided data. Backend code responsible for answering when `patient_id` is extracted:



Obrázok 6 – 3 Agent Flow

Listing 1 Python Code Example

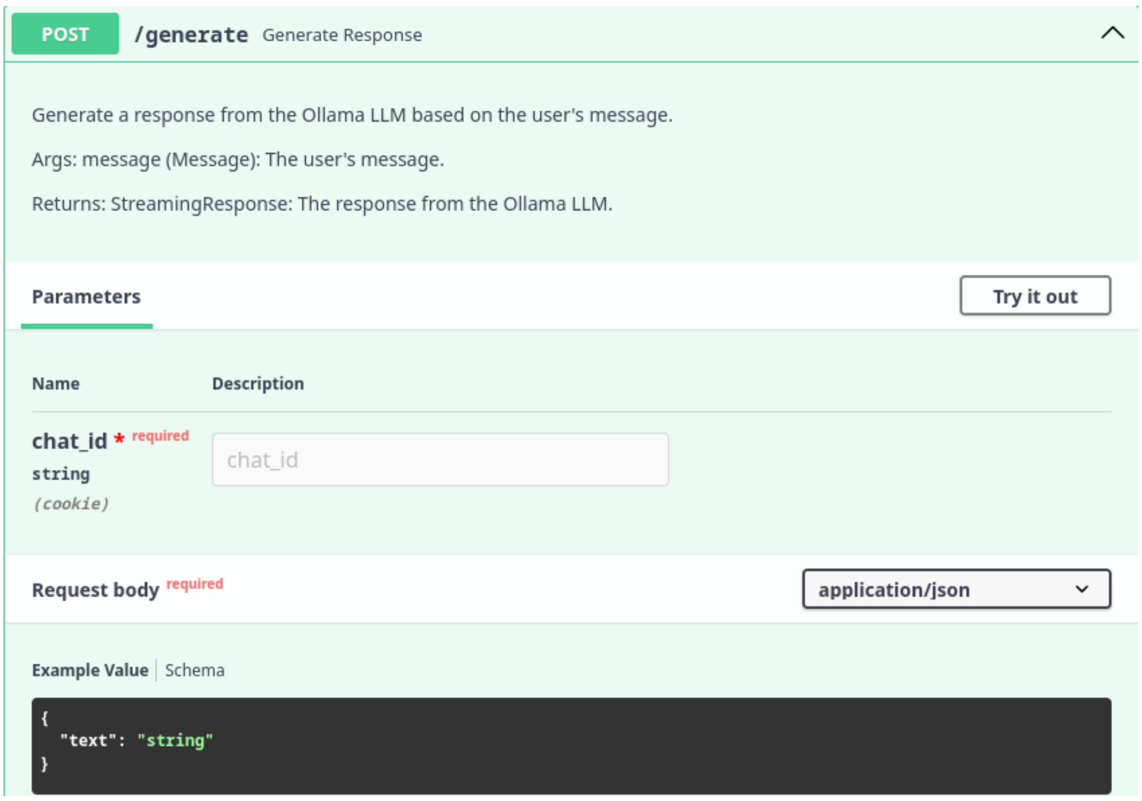
```python
is_info_available = await analyze_history(message.text,
                        chat_id)
if is_info_available:
    return StreamingResponse(
        get_answer_from_context(message.text, chat_id),
        media_type="text/event-stream",
    )
context = await get_patient_data_with_agent(message.text,
patient_id)
return StreamingResponse(
        generate_ollama_stream_response(message.text,
        context, chat_id),
        media_type="text/event-stream")
```

6.5 Application view

The application architecture integrates a FastAPI-based backend and a React-powered frontend, working cohesively to deliver a seamless user experience. The backend, built on FastAPI, is responsible for handling incoming requests and interfacing with the large language model to generate responses. Upon receiving a JSON payload containing a text field, the backend processes the input and streams the generated tokens back to the client as the output. This streaming mechanism ensures minimal latency and an interactive experience, as users can see responses being formed incrementally in real-time.

On the client side, the React frontend enables intuitive interaction with the system. It manages chats through cookies, which are generated and set to maintain session continuity and track individual conversations. The frontend processes the streamed output from the backend and displays the results in a chat-like interface, rendering the tokens sequentially to mimic the conversational flow. This approach not only enhances the interactivity of the application but also provides a user-friendly visualization of the model's responses, aligning with the expectations of modern chat-based interfaces.

Screenshots of the backend Swagger documentation and the frontend main page are provided to illustrate these components.



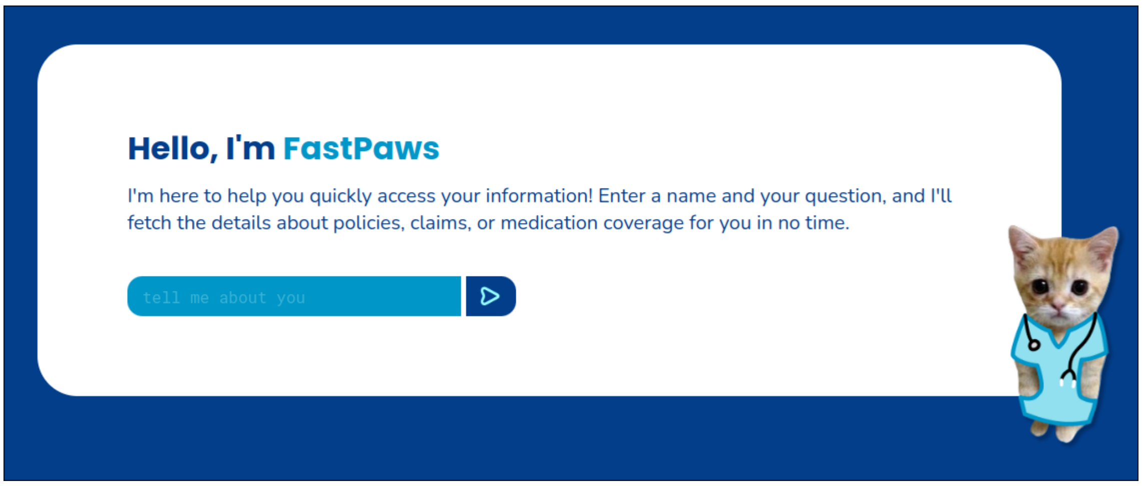
Generate a response from the Ollama LLM based on the user's message.

Args: message (Message): The user's message.

Returns: StreamingResponse: The response from the Ollama LLM.

```
{
  "text": "string"
}
```

Obrázok 6 – 4 Swagger docs screenshot



Hello, I'm FastPaws

I'm here to help you quickly access your information! Enter a name and your question, and I'll fetch the details about policies, claims, or medication coverage for you in no time.

Obrázok 6 – 5 Swagger docs screenshot

7 Future work

The current Q&A system represents significant progress in healthcare applications of artificial intelligence, while several potential enhancements could further advance its capabilities and practical utility.

Dynamic Dataset Integration

The transition from the current static dataset architecture to a dynamic integration model represents a crucial advancement opportunity. This evolution would enable real-time data updates through secure API connections to electronic health record systems and healthcare databases, eliminating the need for manual updates and system restarts. Such integration would require robust change-tracking mechanisms to maintain comprehensive audit trails and ensure data traceability across the system's operations.

Enhanced Multilingual Support

Expanding the system's linguistic capabilities through comprehensive multilingual support would significantly broaden its accessibility and utility. This enhancement would require fine-tuning the language models on domain-specific datasets across multiple languages, coupled with dynamic language detection capabilities. Additionally, incorporating regional and cultural nuances in medical terminology would significantly improve user comprehension and satisfaction across diverse healthcare settings, ensuring the system's effectiveness in various cultural contexts.

Broader Query Capabilities

The system's query processing capabilities could be substantially expanded to handle more sophisticated analytical requests. Advanced functionality would enable the processing of multi-step queries that combine multiple data points, such as correlating medication history with cost analysis. Cross-dataset query functionality would enable more comprehensive healthcare insights by integrating multiple data sources. Furthermore, incorporating visualization capabilities would enhance data interpretation through the automated generation of relevant charts and graphs in response to user queries.

Incorporation of Predictive Analytics

The integration of predictive analytics represents a transformative opportunity to evolve the system from a reactive query tool to a proactive healthcare resource. This advancement would enable the system to forecast potential medication adherence issues based on historical patterns, identify potential drug interaction risks, and generate personalized recommendations for prescription management. These predictive capabilities could significantly enhance the system's value in supporting both patient care and clinical decision-making processes.

Through these strategic enhancements, the Q&A system can evolve into an increasingly sophisticated and indispensable healthcare tool. The combination of real-time data integration, expanded linguistic capabilities, advanced query processing, and predictive analytics would create a comprehensive platform capable of delivering actionable insights while maintaining accessibility for both patients and healthcare professionals. These improvements align with the system's core objective of leveraging artificial intelligence to enhance healthcare data interaction and decision-making processes.

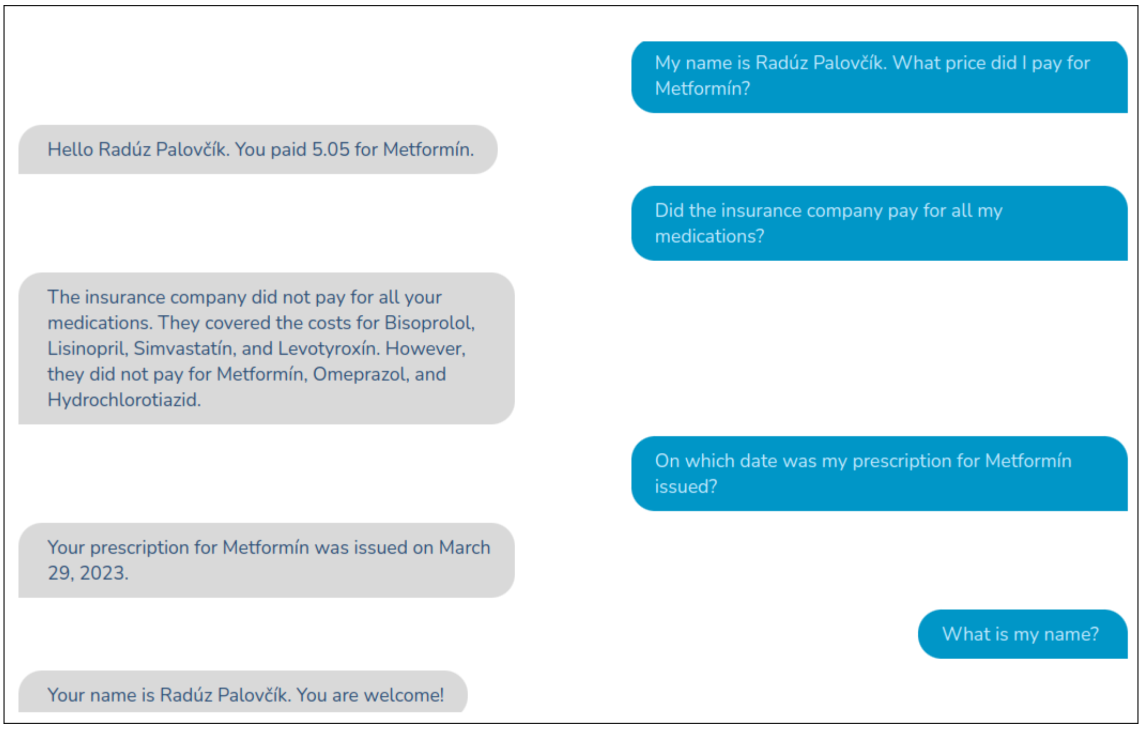
8 Conclusion

In this project, we successfully implemented a robust and intelligent Q&A system tailored for healthcare use. By harnessing the power of advanced LLMs and integrating them with a hybrid data storage architecture, the system demonstrated its ability to process natural language queries and provide accurate, context-aware responses.

The combination of SQL databases for structured data queries and vector databases for semantic similarity matching proved effective in handling both deterministic and ambiguous questions. Our system's capability to adapt to minor variations, such as spelling mistakes or name inconsistencies, enhances its usability and reliability. Furthermore, the React-based interface ensured a seamless user experience, making complex data accessible to a wide audience.

Throughout the project, we also extensively evaluated various LLMs to identify the best model for our application. This informed and data-driven approach ensured optimal performance in understanding and responding to diverse query types. By documenting the system's design, challenges, and solutions, we have created a blueprint that can be expanded for future healthcare applications.

Ultimately, this project highlights the potential of AI-driven systems to transform healthcare data interaction, improving decision-making and accessibility for patients and professionals alike. It stands as a testament to the ability of technology to bridge gaps in data-driven insights while maintaining user-centric design and ethical considerations.


My name is Radúz Palovčík. What price did I pay for Metformín?

Hello Radúz Palovčík. You paid 5.05 for Metformín.

Did the insurance company pay for all my medications?

The insurance company did not pay for all your medications. They covered the costs for Bisoprolol, Lisinopril, Simvastatín, and Levotyroxín. However, they did not pay for Metformín, Omeprazol, and Hydrochlorotiazid.

On which date was my prescription for Metformín issued?

Your prescription for Metformín was issued on March 29, 2023.

What is my name?

Your name is Radúz Palovčík. You are welcome!

Obrázok 8 – 1 Final Result