Uppsala University

# Empowering Employees through Domain-Specific Chat Assistance

**Authors**
Jesper Borglund Vannebäck
Lukas Bygdell Malmstig
Martin Gundhus
Viktor Kangasniemi
Axel Olsson
David Ohanjanian
Benjamin Strandberg

**Supervisors**
Aletta Nylén
Mats Daniels
Diletta Goglia

**External Supervisors**
Carl Enlund
Douglas Karlsson Hedström

**Abstract**

This report outlines the development and implementation of *Dr. Upps*, a specialized chatbot system designed for *Drupps*, a water innovation company, specializing in atmospheric water extraction. *Dr. Upps* streamlines the retrieval and generation of domain-specific information and knowledge of the company's operations.

Providers of standalone Large Language Models, such as Meta and OpenAI, offer chatbots but with the lack of domain specific data. *Dr. Upps* aims to bridge this gap of knowledge by creating a system with access to the company data.

This is achieved by the creation of a modularized system composed of three parts; an web-based frontend, backend and the use of Google's Vertex AI Agent, offering the generation of contextually relevant responses.

The system presents itself as functioning. It has an intuitive user interface and sub 10 second response time. However, the system is only accurate if the questions passed to it are descriptive and specific. Because of this, the system may be less suited for new employees.

# Contents

# 1 Introduction

This report details the development and implementation of *Dr. Upps*, a chatbot system customized for Drupps, a water innovation company specializing in atmospheric water extraction[1]. The chatbot, with access to the company's internal documents, allows employees to ask domain specific questions, easing the access of information from their internal document database and essential physical and chemical processes crucial to the company's operations.

## 1.1 Background

AI[1] is becoming increasingly relevant in today's world and is being used everywhere; from the apps on our phones to the cars we drive. More specifically LLMs [2], are commonly used in chatbots to answer a broad range of questions. Questions can pertain to any subject offering almost unlimited information to the user.

When an employee is new to a company it usually takes time until they reach their full productivity. A metric for this time is called *Time to Productivity*, and measures the time between the first day of hire for an employee to when the employee has reached their full productivity[2]. Companies want to make the time to productivity as low as possible since employees who are not fully productive will cost the company and increases the workload of other employees. This time differs from every company but generally it takes 8 to 10 months for a newly hired employee to reach their full productivity.

AI can be leveraged to expedite this process by providing chat assistance to new employees when they encounter questions or require clarification on company policies, procedures, or any other related information. By introducing an LLM based chatbot, new hires can quickly access answers to their questions, reducing time spent searching for information.

[1]AI: Artificial Intelligence

[2]LLM: A large language model is a type of program that can recognize and generate text, among other tasks.

## 1.2 Research Question

This report aims to answer the question: **How can AI-driven solutions be integrated to provide domain-specific chat assistance within a web-based user interface?** Leveraging natural language processing, the chatbot simplifies information retrieval, allowing new hires to ask questions in natural language and receive responses.

## 1.3 Related Works

Similar systems are likely to exist, but are inaccessible due to its reliance on private data. Other platforms offer solutions for training LLMs with domain-specific data. For instance, OpenAI's *GPT*[3], with the ability to understand and generate responses based on data from attached files, it enables the creation of chatbots customized for specific industries or companies.

Another similar system is Google's *Dialogflow*[4] that offers tools for building chatbots and conversational agents tailored to specific domains or industries. *Dialogflow* can connect to a database utilizing the contents, improving the accuracy and relevance of its responses.

# 2 Theory

The system design and implementation requires knowledge of numerous crucial concepts. This section will explore the theory relevant to the system and compare different options.

## 2.1 Artificial Intelligence

Artificial Intelligence refers to the mimicking of human intelligence by computer programs[5]. This includes various tasks such as image recognition, chess bots and, understanding and generating human language. The underlying theory behind the functionality of AI are neural networks employed with deep learning. A neural network is a processing network that processes data by passing it through multiple layers of neurons, each of which transforms the data. Deep learning is a way of tuning the network, improving its accuracy.

### 2.1.1 Vectorization of words

Vectorization, converting data from text and images to vectors, is central to how data is handled within AI. In the case of LLMs, such as *CHATGPT-3*, each token is converted to a 12288-dimensional vector[6]. How the values of each vector are calculated is complex, but the end result is that words with similar meanings have similar vectors. For example, words like "big" and "large" end up with similar vectors. This can be used to construct a vector database which are effective at querying for similarity. A vector database queries by finding the K-nearest neighbors[3] to the searched term. However, these searches, and in extension vector databases, are significantly more resource intensive than other databases such as MySQL[7]. But when unknown input is given, a similarity search is necessary.

### 2.1.2 Large Language Model

Large Language Models are a subset of generative AIs that can create contextual text from input[8]. They are neural networks implemented with deep learning that work by predicting the next token from a given text. How the LLM predicts the next word depends on the data it is trained on.

The text that the user writes to the LLM, called a prompt, is the text used as a starting point for the prediction. But that is not the only text provided to the LLM. A preprompt is a text that is prepended to the user's prompt. This is used to guide the LLM's response in order to get more appropriate responses. The preprompt is usually invisible to the users, set by the developer and highly programmable. Examples of preprompts include: "Answer as if you are an expert assistant", "Be very polite" and "Ask the user to change your name". Setting up and training an LLM is a significant undertaking. Therefore, it is common practice to utilize an existing LLM in applications.

LLMs can be hosted either on a local machine or accessed over the web through an API[4] from providers like OpenAI or Google. Generally, the larger the model, the better or more human-like responses it can generate. Smaller models typically have around 7 billion parameters, while the largest can have more than 1 trillion[9]. Consequently, the size of the model crucially affects the need for RAM and CPU resources.

[3]K-nearest neighbors: a machine learning algorithm.

[4]API: Application Programming Interface

### 2.1.3 Extending the knowledge of a Large Language Model

In order to provide a domain-specific chatbot, integrating information into the LLM is essential. There are two primary methods for adding new specific knowledge to an LLM[10]. One method involves retraining the model on the additional data. This can be achieved in various ways, but an effective approach is by adding new layers to the LLM and hyper tuning it with the specific information. However, this method has several drawbacks; it is time consuming and challenging to execute[10]. Moreover, it alters the original model away from its trained maxima, increasing the chance of hallucination[5] if not done properly.

Another method involves using RAG[6]. This approach constructs a vector-based database, containing the new informatio[11]. When a user prompts the LLM, the input is first passed to the RAG, which retrieves relevant segments from the database. These segments are added to the preprompt and passed to the LLM. This eliminates the need to retraining the LLM, making it easy to add and change information, and provide different access levels to the database for different users. Additionally, the RAG can refer to the source of the information and operate independently from the LLM, making the system modular.

### 2.1.4 Alternatives

There are many different LLMs to choose from. OpenAI's *ChatGPT* boasts several benefits. It is known for its advanced language understanding and generation capabilities, enabling it to handle a wide range of tasks, from answering questions to understanding attached files. The model can be accessed through a simple and powerful API. Additionally, *ChatGPT* benefits from continuous improvements, enhancing performance and expanding its functionalities based on user feedback and the latest advancements in AI research. However, *ChatGPT* also has some drawbacks. Utilizing the model, especially for large-scale applications, can be costly due to the high computational requirements and OpenAI's pricing structure. There are also data privacy concerns since the processing occurs on OpenAI's servers, raising concerns regarding the security of proprietary information. Moreover, despite

[5]Hallucination: the generation of content that is incorrect, nonsensical, or not based on the input or data the model was trained on.

[6]RAG: Retrieval Augmented Generation

its capabilities, *ChatGPT* can sometimes produce inaccurate or nonsensical responses, necessitating human oversight in critical applications to ensure accuracy and reliability[12].

Google's *Gemini* offers notable benefits, including integration with Google's extensive ecosystem of services, making it appropriate for businesses already using Google products. However, this is also a drawback since the integration can be complex, requiring technical expertise. Leveraging Google's leading AI research, *Gemini* benefits from advancements in natural language processing making it able to handle attached files and speech. The latest *Gemini* model 1.5 pro features a prompt window of 1 million tokens[13], making it capable of ingesting a large context for its answer. Using Google's ecosystem incurs a financial cost, but this is comparable to the other alternatives.

IBM's *Watsonx*[14] is designed with a focus on enterprise use, offering tools for data analytics, business intelligence, and AI-driven insights. It provides customization options, enabling businesses to fine-tune the system for specific use cases and industries. However, implementing and maintaining *Watsonx* can be expensive, which makes it less relevant for small to medium-sized businesses. The complexity and range of features come with a learning curve, requiring substantial training and expertise. Additionally, despite their robustness and reliability, IBM's AI solutions is perceived as less innovative compared to newer entrants in the market[11].

When implementing RAG in an application there are options to consider. Using prebuilt RAG solutions from providers such as Google, Microsoft, and Hugging Face offers several advantages. These prebuilt models can significantly shorten development time, only requiring integration with their existing APIs. This reduces the need for local processing power and allows developers to focus on building the application's features rather than the underlying AI infrastructure. One issue is that these solutions have limited customisation. Many applications requires a more tailored RAG in order to get the relevant context. This can be done by building a custom RAG. The Python libraries *Lang-chain* and *Llama-index* are tools to implement RAG[11].

Finally, another approach is to use AI Agents[15]. An AI Agent is a software program that interacts with its environment, gathers data, and uses this data to perform tasks independently to achieve set goals. Therefore, they usually incorporate LLM and RAG functionalities in order to handle such tasks. One option is Google's *Vertex AI Agent*[16], an Agent which incorporates *Gemini* as its LLM. As a part of *Google Cloud Services*[17], it easily integrates with other Google services, making it particularly advantageous if other components of the system are within the same ecosystem.

## 2.2 Frontend

The frontend will consist of a web application with a relatively small amount of features. This should be taken into consideration when exploring what frameworks to be used.

### 2.2.1 Framework

When it comes to web development, *React*, *Angular* and *Vue* are the most prominent. *React*, originating from Facebook, is renowned for its component-based architecture, offering exceptional flexibility and performance. With a vast ecosystem and strong community support, *React* allows developers to efficiently build dynamic user interfaces[18]. *Angular*, maintained by Google, distinguishes itself as a comprehensive framework built on *TypeScript*, providing a holistic solution for constructing large-scale applications. Its opinionated structure promotes maintainability and scalability, making it ideal for enterprise-level projects [19]. *Vue*, meanwhile, shares *React's* component-based approach but opts for a balance between simplicity and ease of use. However, it does have a smaller community and does not scale as well compared to both *React* and *Angular*[20]. Additionally, in the *React* ecosystem, *Next.js* is a powerful extension. Leveraging *React's* strengths, *Next.js* simplifies server-side rendering and routing, offering enhanced capabilities for building robust web applications[21].

When styling websites the language CSS[7] is used. There exists several frameworks that allow streamlining the process of writing CSS. The main two are *Bootstrap* and *Tailwind CSS. Bootstrap*, developed by Twitter, is a comprehensive *CSS* framework that offers a wide range of predesigned components, utilities and JavaScript components for interactive elements [22]. On the other hand, *Tailwind* provides a utility-first methodology, allowing developers to style directly with utility classes. This offers more flexibility and control in exchange for a steeper learning curve[23].

[7]CSS: Cascading Style Sheets

## 2.3 Backend

The backend will hold the main functionalities of the system, which the frontend will connect to. Given the variety of implementation options available, a decision must be made to determine the most suitable approach for the system.

### 2.3.1 REST API

REST[8] is as an architectural style for crafting networked applications, particularly web services. It emphasizes a stateless client-server interaction where clients request resources and servers provide representations of those resources. RESTful systems typically use standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources, and a web API following the REST architectural style is called a REST API[24].

Numerous frameworks are available for building REST APIs, one of them being *Flask*[9]. *Flask* is a framework designed for developing web applications written in Python, known for its lightweight nature and ease of implementation, making it particularly suitable for simpler systems. However, one of the primary drawbacks of employing *Flask* for building a REST API lies in its minimalist nature. As due to its lightweight design, *Flask* lacks built-in support for many common functionalities required in larger and more complex systems. Consequently, developers often find themselves either relying on third-party extensions or creating custom implementations for essential features, such as authentication systems.

Another framework worth considering for creating the REST API is *Django*[10]. Unlike *Flask*, *Django* comes with a large set of built-in features, making it suitable for handling larger and more intricate complex systems. The built-in features gives developers the necessary resources for implementing various functionalities demanded by such systems. However, as *Django* is better suited for more complex systems, a steeper learning curve follows for new developers. Thus, for a more simple solution when constructing a REST API, *Django* might not be the optimal choice.

[8]REST: Representational State Transfer

[9]https://flask.palletsprojects.com/en/3.0.x/

[10]https://www.djangoproject.com/

### 2.3.2 Database

The system will be required to store data, including chat history, data generated by the Agent, such as the vectorized database used for the RAG, and AI-generated responses. All this data will be stored in a database. The choice of the database model depends on the type of data the system handles. The following section will explore various existing database models and discuss each respective area of application.

Relational databases are well-suited for structured data storage, making them a viable choice when it comes to managing user profiles, preferences and chat histories. There are several options when it comes to relational databases, such as MySQL[11] and PostgreSQL[12], both of which are easy to use. These databases offer a user friendly management of relational data, providing a strong foundation for handling structured data efficiently.

NoSQL databases, as opposed to relational databases, are designed to handle large volumes of unstructured or semi-unstructured data as well as structured data[25]. Instead of storing the data in tables with predefined columns with constraints, the data is stored in documents, usually *JSON*, *BSON* or *XML*. A NoSQL database could therefore be a suitable implementation to use as a data repository for RAG system since it accommodates vector representation.

## 2.4 Development Environment with Docker

Docker is a platform and toolset designed to simplify the process of building, distributing and managing applications using containers[26]. Containers are isolated environments that contain everything needed to run the application, such as dependencies, code, resources and libraries. Because of this, an advantage of using Docker containers is their consistency and reproducibility across different environments. The frontend, backend and database modules each reside within their own container.

To containerize a codebase, a Dockerfile is crafted to encapsulate the application into a Docker Image, a snapshot of the application. Prebuilt Docker images are available for common applications, such as databases to download from Dockerhub[27]. To build a collection of containers a Docker Compose file is written. This file serves as a blueprint for building a docker

[11]https://www.mysql.com/

[12]https://www.postgresql.org/

network where all of the relationships between the containers and external entities are configured.

# 3 Method

To achieve results aligned with the problem description, the system development process contains four stages: requirements, design, implementation, and evaluation. These stages are used to achieve results and this section describes the methods used during these stages of development.

## 3.1 Requirements

In the requirements stage, the first step involved specifying requirements with the client. The requirements obtained were to have a response time less than 10 seconds, an easily navigated user interface, and correctly generating answers from given contexts. The requirements serves as the foundation for the subsequent phases of the development process.

## 3.2 Design

In this stage, designing focuses on taking the problem given in the previous stage and design a system that solves it and fulfills all the requirements. This is crucial since the implementation will be easier if the design is well thought out. This stage involves creating figures and sketches regarding the system design, such as the system architecture and entity-relational diagram for the database structure. An interactive prototype was made using *Figma*[28] to have a template to follow when proceeding to the next stage.

## 3.3 Implementation

The implementation revolves around actualizing the system design. Modularizing the system achieved many benefits, such as minimized dependency and the implementation can be split into parts that can be worked on individually and in parallel. Therefore, the implementation is split into the modules frontend, backend, and AI. The frontend were implemented in connection with the backend being developed. For the backend, the database had to be set up and a REST API had to be written with functions to handle the user database. The AI, had to be adapted to match our requirements. Finally, these modules needed to be interconnected allowing the backend to send and receive messages to the AI and frontend.

## 3.4 Evaluation

For evaluation user testing was conducted, specifically a participatory heuristic evaluation[29]. For all user tests a script was used, shown in Figure 1. This script has been translated, original can be found in Figure 8. It contains instructions for the person doing the testing as well as the tasks and questions for the tester. In order to broaden the test pool, user testing will be done on both company employees and students. Testing involving students will focus on user experience, while testing with employees will prioritize technical accuracy and informational correctness.

Additional to this stage, performance testing was done, measuring both response time and accuracy. The response time was measured by prompting the system three different questions, each with varying intent, to measure the consistency of the response time for each respective prompt. In order to increase credibility, questions needed to include a variety of lengths and source of information, each potentially resulting in varying response times.

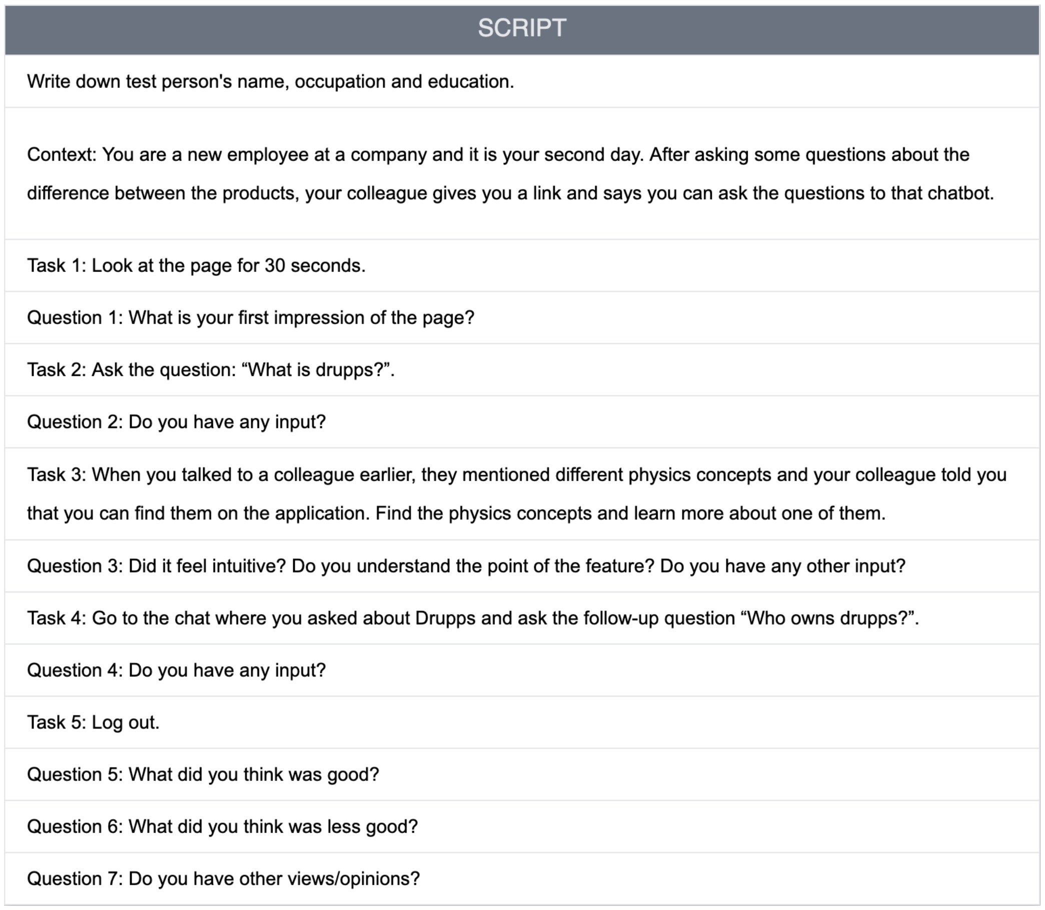
To measure the response accuracy, questions with altering degrees of difficulty were asked to the chatbot. The answers were then manually judged if they were correct enough or not based on the clients data. However, the data is confidential and can therefore not be shown in this report.

# 4 System Design

This section intends to give a brief overview of the system. Including the system architecture, the intended data and user flow, and the structure of the user database.

## 4.1 System Architecture

The system has three principal components: the frontend, the backend and the agent. To enhance modularity and separate concerns, these components are further divided into five smaller modules, as depicted in Figure 2. This



| SCRIPT |
|---|
| Write down test person's name, occupation and education. |
| Context: You are a new employee at a company and it is your second day. After asking some questions about the difference between the products, your colleague gives you a link and says you can ask the questions to that chatbot. |
| Task 1: Look at the page for 30 seconds. |
| Question 1: What is your first impression of the page? |
| Task 2: Ask the question: “What is drupps?”. |
| Question 2: Do you have any input? |
| Task 3: When you talked to a colleague earlier, they mentioned different physics concepts and your colleague told you that you can find them on the application. Find the physics concepts and learn more about one of them. |
| Question 3: Did it feel intuitive? Do you understand the point of the feature? Do you have any other input? |
| Task 4: Go to the chat where you asked about Drupps and ask the follow-up question “Who owns drupps?”. |
| Question 4: Do you have any input? |
| Task 5: Log out. |
| Question 5: What did you think was good? |
| Question 6: What did you think was less good? |
| Question 7: Do you have other views/opinions? |

Figure 1: Translated script for user testing

modular approach contributes to several advantages, such as testing for reliability and functionality independently. Additionally, it enables replacement of modules, if necessary[30].

The first module is a simplified web application enabling user interaction with the chatbot. To attend to modularity, it is designed to handle as little logic as possible.

The backend consists of two modules, one for handling user authentication and the other is a REST API. The purpose of the REST API is to act as an intermediary for all communication between the frontend and AI module. It includes a database intended to store history and data about said communication.

The last component consists of the AI module and company database. The company data base holds all documents and unstructured data from the company. The AI module contains a RAG that creates a context from unstructured data from the database and pass to the LLM to further process it and generate responses.

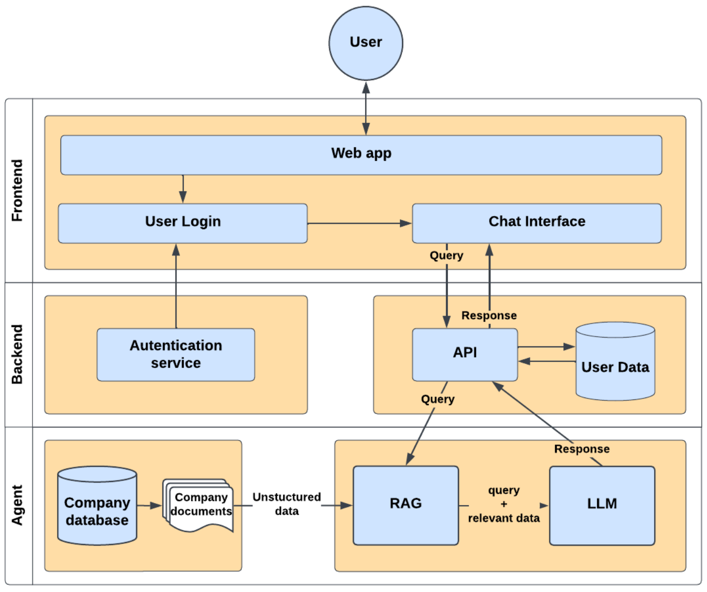


Figure 2: System Architecture

## 4.2 Data Flow

When a user submits a question through the frontend, it is forwarded as a query to the REST API. Acting as an middleman, the REST API then relays this query to the AI module The RAG, located in the AI module, retrieves data from the company database to establish a context relevant to the query. This context, in addition to the original query, is then transmitted to the LLM. The LLM processes this input and generates a response based on the context, which is relayed back to the REST API. Finally, the REST API forwards this response back to the frontend for the user to receive.

## 4.3 User Flow

The desired user flow of the system, visualized in Figure 3, showing user actions and how the user progresses through the application. After logging in, the user will enter the homepage, where numerous actions are available. One action is the ability to ask a question and receive an answer. Another action is to retrieve an old session and continue asking questions there. The user will also be able to choose from predetermined topics and choose a related question without having to write any question by themself.

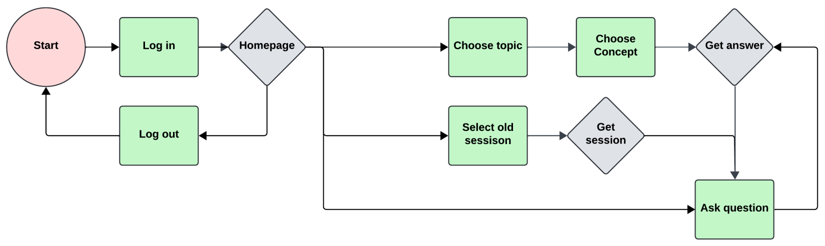


Figure 3: User flow

## 4.4 Entity-Relation Model

The entity-relation diagram, shown in Figure 4, represents the database tables for users, sessions and messages. The message table has a message_id as primary key and a session_id as a foreign key to the session table. The message table also contains the message itself, user_id which also indicates if the message is a question from the user or an answer from the chatbot, and finally timestamp for the message. The session table has session_id as primary key and user_id as a foreign key to the User table. The session table also contains title of the session and timestamp. The User table has user_id as primary key. Finally, both relations are one-to-many, meaning messages can only belong to one session, but a session can have multiple messages. Sessions can only belong to one user, but a user can have multiple sessions.

Furthermore, the database contains tables for topics and concepts, shown in Figure 5. The concepts table has a concept_id as primary key and a topic_id as a foreign key to the Topics table. It also contains name of the concept, a synonym to the concept, description and is_internal. The topics table has a topic_id as primary key and a title. Additionally, the relation between both tables is one-to-many, meaning concepts can only belong to a topic but a topic can have multiple concepts.

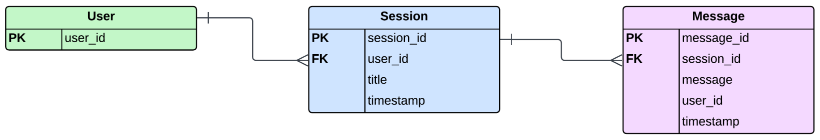


Figure 4: ER-diagram of Users, Sessions, and Messages

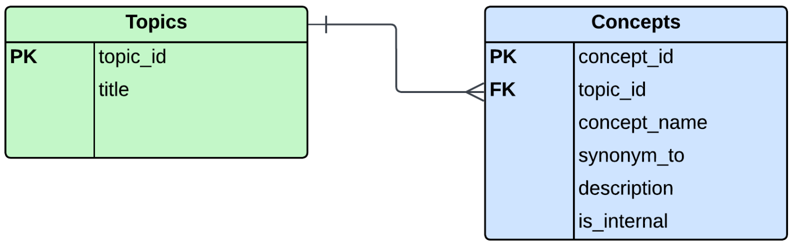


Figure 5: ER-diagram of Topics and Concepts

# 5 Implementation

This section inquires into the practical implementation of the system, providing insights into the specific technologies and methodologies implemented to realize its functionalities.

## 5.1 AI Agent

An AI Agent was used to implement LLM and RAG functionality into *Dr. Upps*. The use of an AI agent significantly shortens development time and simplifies maintenance. In the implementation, Google's Vertex AI Agent was selected, which provides options for deploying different machine learning models, including LLMs like text bison and Gemini. The main reason for this is to integrate within Google's ecosystem, which is well-suited since the data used within the RAG is located in Google Drive.

Gemini was selected as the LLM. The setup process starts with creating the AI agent in the Google Cloud Console tab *Agent Builder*. To provide the agent with the domain data, the data was stored in Cloud Storage Buckets[13]. To make the bucket work, there can be no folders in the bucket; all documents have to be stored in the root-directory. Furthermore, the cloud console also allows for testing the agent before integrating it with the rest of the system, providing a chat window for testing its answers and verifying the sources it uses.

The connection between the agent and the backend was established through an API managed by *Dialogflow CX*[14], Google's natural language understanding platform. Dialogflow is a framework for designing conversational interfaces and integrating them with various systems. But for the implementation, the functionalities of Dialogflow was disregarded as it was only used for accessing the AI Agent. In order to access the Agent, a Google service account has to be created, which will provide the necessary API keys.

[13]https://www.logicmonitor.com/blog/what-is-a-bucket-in-gcp-gcp-buckets-explained

[14]https://cloud.google.com/dialogflow/docs

## 5.2 Frontend

The frontend was developed using *React* and *Next.js*, complemented by *Tailwind CSS* for styling, in order to leverage the powerful component-based architecture provided by *React*, facilitating efficient development and maintenance of complex user interfaces. *Next.js* was chosen for its server-side rendering capabilities, enabling faster page loads and improved SEO [15] performance, while *Tailwind CSS* offers a utility-first approach to styling, allowing for rapid prototyping and easy customization without the need for writing custom CSS.

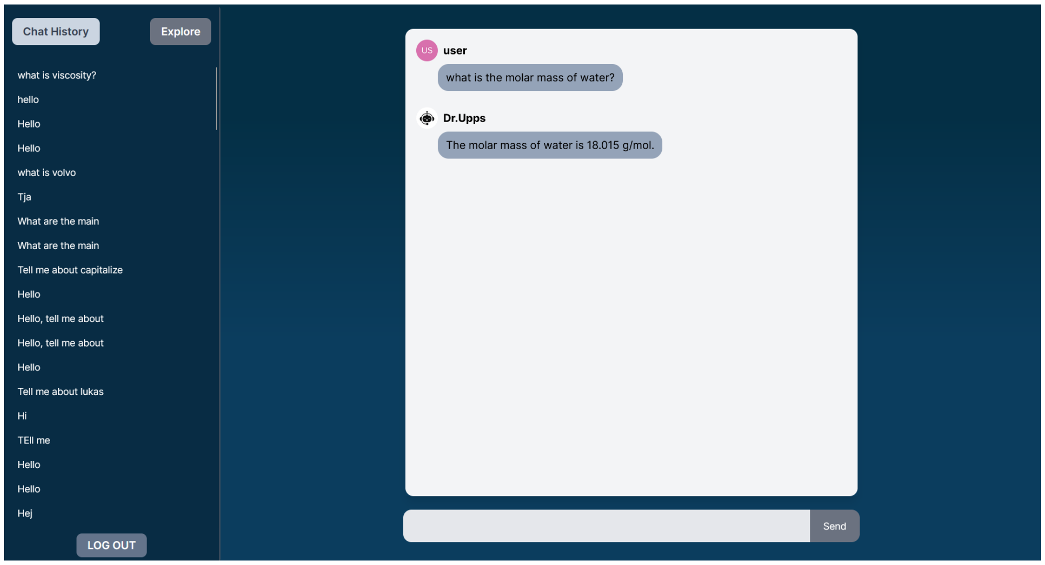
The user interface adopts a lightweight web application design. *Dr. Upps* consists of one page featuring a sidebar and chat, as seen in Figure 6. A cohesive experience is achieved through a general color theme of dark blue with white and gray as complimentary colors, ensuring consistency across interactions with *Dr. Upps.*

The login functionality in *Dr. Upps* is handled by Auth0, which is an authentication and authorization service that allows for secure access to applications and devices[31]. Auth0 helps manage and protect user identities, ensuring security and across different platforms and devices. When using the application users are directed to Auth0 to access *Dr. Upps.*

The sidebar, positioned on the left side of the web application, see Figure 6, enables users to conveniently access previous chat sessions or topics to explore, depending on the selection of what to display made via using the respective topmost buttons. The topics was made colorful so as to draw attention. Each of the topics have a subset of concepts that are displayed if a topic is chosen, see Figure 7. In turn, users can select to initiate a new chat session with one of the concepts. Additionally, the sidebar includes a button for logging out.

The chat interface comprises both an input box and a chat log, see Figure 6. The input box is where users can enter text to communicate with the chatbot, while the chat log displays previous messages and responses.

[15]SEO: Search Engine Optimization

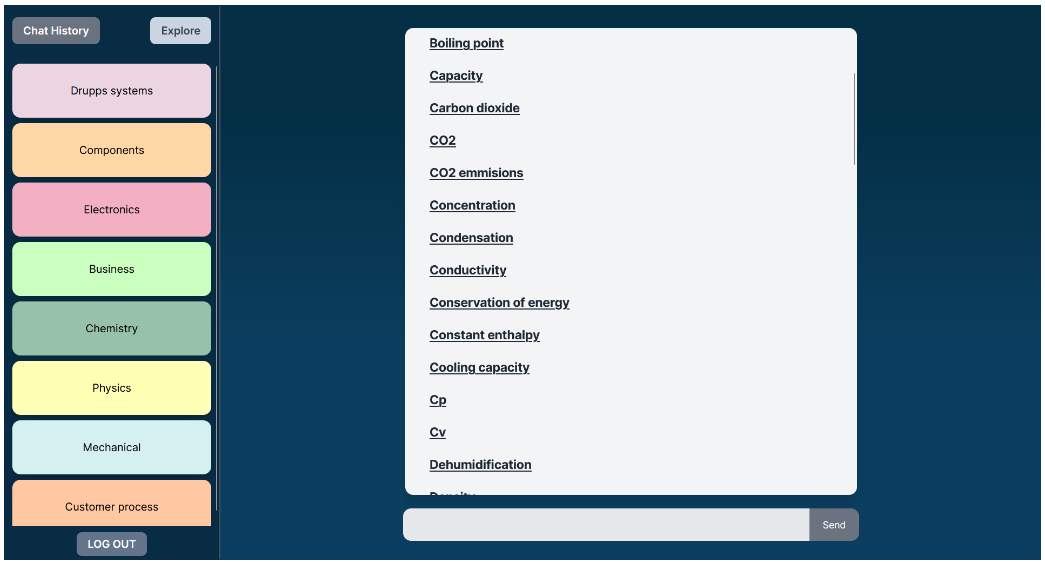


what is viscosity?
hello
Hello
Hello
what is volvo
Tja
What are the main
What are the main
Tell me about capitalize
Hello
Hello, tell me about
Hello, tell me about
Hello
Tell me about lukas
Hi
TEll me
Hello
Hello
Hej

US user
what is the molar mass of water?

Dr.Upps
The molar mass of water is 18.015 g/mol.

Figure 6: Home page of the frontend



Boiling point
Capacity
Carbon dioxide
CO2
CO2 emmisions
Concentration
Condensation
Conductivity
Conservation of energy
Constant enthalpy
Cooling capacity
Cp
Cv
Dehumidification

Figure 7: Concepts of the frontend

## 5.3 Backend

The main part of the backend is the REST API, which is built with *Flask*, a suitable choice due to the system's simple structure, as seen in Section 2.3.1. This module handles database storage and facilitates connectivity with the Agent API. It offers multiple methods for frontend interaction, primarily categorized into two types: HTTP requests, utilized to retrieve data stored in the database, and websockets, implemented with *Socket.io*[32], to establish and maintain reliable real-time communication with the server. All the endpoints that the frontend can connect to, seen in Table 4, are designed to access essential data relative to a specific task. This includes actions such as initiating a new chat session or loading previous ones. The system stores user data in a PostgreSQL database, as the data is well-suited for relational models. The data stored includes user information, chat records, and the predefined topics and concepts used in the system, as seen in Figures 4 & 5.

## 5.4 Authentication service

To ensure authentication and authorization processes for company employees accessing the system, Auth0 is utilized[31]. Auth0 is a platform that provides developers with the tools to authenticate and authorize users across various applications.

## 5.5 Docker Network Integration

Throughout the development stage of the project, a self hosted Docker Network was utilized[26]. This coupled with a deployment pipeline consisting of *Github Actions*[33] and *Dockerhub*[27] ensured easy deployment of newly developed code for testing.

The Docker container architecture is designed to isolate every module of the system. This facilitates updates to specific parts without impacting the entire application, thereby minimizing downtime. If one module crashes, it does not bring down the rest of the application. The Docker Network is configured in a way so that only the frontend container externally accessible. The backend and database containers are only accessible from within the frontend container.

As presented in the theory chapter above, all of the necessary dependencies, resources and other various prerequisites needed for the application to run are encapsulated within the containers. This allows the system to run on any platform that supports Docker including Linux, Windows and macOS. Additionally, Docker's isolation mechanisms ensures consistent performance across different platforms, minimizing the likelihood of conflicts with other applications or system components.

# 6 Results

This section displays the results of various tests, described in Section 3.4, which evaluate the system's performance and overall quality.

## 6.1 User Tests

The participatory heuristic evaluation was performed, shown in Table 1.

| Participant | Q.1 | Q.2 | Q.3 | Q.4 | Q.5 | Q.6 | Q.7 |
|---|---|---|---|---|---|---|---|
| 1 | Nice, like the color theme | Want to see it loading | Yes. Yes. Like the colors on topics, but the concept page is a bit boring | Would have liked it to show what's newest in the chat log and division over time | Easy to navigate with the side menu | Lack of timestamps in chat and that it doesn't show when it's loading | - |
| 2 | Good that it's dark, but the colors on the topics are a bit much | Would have liked everything to be written out as it goes along | Pretty much. Kind of. Ask dr.upps doesn't stand out enough | The icons look strange and would have liked some friction | That it's simple. | Friction | - |

Table 1: Translation of the results from the heuristic evaluation. Original can be found in Table 3 in the appendix

## 6.2 Agent Response Accuracy

Measuring the accuracy of AI responses poses a challenge due to various factors influencing the evaluation process. An answer may be correct but not comprehensible or extensive enough. *Dr.Upps* AI module is specifically designed to respond only to questions that can be constructed using the information stored in the RAG. Any question out of context will be denied. This means that the accuracy solely relies on the power of Google's *Vertex AI* itself and how detailed of a prompt it is given. The questions need to be rather specific for the chatbot to be able to answer. If the questions are specific enough, the answers have proven to be accurate as evaluated with user testing.

## 6.3 Query Response Time

Table 2 presents the response time of the system for 3 different questions given 10 times.

| No. | Q.1 Time [sec] | Q.2 Time [sec] | Q.3 Time [sec] |
|---|---|---|---|
| 1 | 3.56 | 9.51 | 9.51 |
| 2 | 3.03 | 8.86 | 9.15 |
| 3 | 2.84 | 8.35 | 9.39 |
| 4 | 2.91 | 8.48 | 9.24 |
| 5 | 2.68 | 9.93 | 10.12 |
| 6 | 3.02 | 7.91 | 9.02 |
| 7 | 2.72 | 8.18 | 8.85 |
| 8 | 3.07 | 8.43 | 8.97 |
| 9 | 2.68 | 9.33 | 9.5 |
| 10 | 2.84 | 8.48 | 9.55 |
| **Average:** | **2.935** | **8.746** | **9.33** |

Table 2: Table of response times. Q.1: "hello", Q.2: "What is Atmos?", Q.3: "What are the main modules of Atmos?"

# 7 Discussion

This section serves as a discussion on prevalent topics in software development. Data integrity refers to the accuracy, reliability and consistency of data throughout the system. The section on ethics encompasses a broad range of considerations related to the responsible and ethical use of technology.

## 7.1 Evaluation from Results

Regarding the user tests, the overall feedback was positive, with only minor details showing room for improvement. This indicates that the flow and ease of use of the system are satisfactory. These minor details include positive friction and timestamps in the chat log. However, the small sample size of the test raises concerns about the interpretation of the results. Therefore, it is crucial to emphasize the necessity of conducting a larger test to reinforce the reliability of the result.

In terms of response time, seen in Table 2, the results exceeded the initial goal of 10 seconds per query, which is important since an inadequate response time would negatively impact the user experience. The variations in the response time largely depends on the complexity of the question. Some questions requires the AI to summarize or deduce relevant text, which takes longer than providing an answer that naturally occurs in the text.

When it comes to the accuracy of the responses, there is room for improvement. Successful answers are always accurate but it relies on the questions being relatively specific. This requirement may present challenges for newly hired employees, as it assumes a certain level of familiarity with the company and its products.

The combined results of our conducted tests are mostly positive. The primary drawback identified from the tests is the response accuracy, particularly in handling ambiguous or out-of-scope queries. Further development and fine-tuning of the AI could improve its ability to provide more accurate and relevant responses in such cases.

## 7.2 Data Integrity

*Dr. Upps* is built upon a data repository created by the company. Some of this data is sensitive to the company and needs to be handled with care. The RAG and its data are hosted on Google Cloud which offers robust security measures for keeping your data safe[34].

Given that the data is stored securely on Google Cloud, ensuring the security of the application's access points becomes crucial for data protection. The containerized structure of the application forms a central layer of security. To access the RAG data, one would need access to either the API endpoints that connects to the RAG, or to the database where chat histories could be exploited to extract data. The Docker network consisting of the database, frontend and REST API is configured to restrict access. Specifically, the database and REST API are isolated, only accessible from within the frontend container. The chatbot is only accessible by authenticated users via Auth0 authentication. This means that only employees with the correct authentication credentials can access the chatbot.

## 7.3 Ethical considerations of the chatbot

Understanding and mitigating the ethical ramifications of chatbots is important to reduce the harm and increase the acceptance of them; making them more likely to complement the well being of society. Most ethical considerations in this project relate to the LLM that need to be addressed.

Firstly, one needs to consider the capability of the chatbot with regards to harmfully content generation. To reduce the risk, many LLM providers have filters to restrict harmful and inappropriate generated responses. It is also possible to add additional filters to an application. Google's AI Agent, used in *Dr. Upps*, provides functionality to ban phrases. This enables developers to further restrict the response. To choose a suitable filter level, one has to find a balance between useful, but dangerous, and safe results, which largely depends on the type of user and the chatbot's purpose. This application will be used by employees working with dangerous chemicals and machines, therefore minimal filter level is desired[35] [36].

Secondly, LLMs occasionally hallucinate, which raises ethical concerns. Hallucinations are presented with the same certainty as correct statements, making them difficult to differentiate. With the wisdom that ill-informed decisions are worse than uninformed in consideration, its likely that hallucinations undermine the positive ramifications of the chatbot [37]. The most important way of addressing this is with extensive training of the LLM which all advanced LLMs do. Moreover, most LLMs have a variable temperature, where a lower temperature produce more certain results. Hallucinations can also be introduced by the RAG system; if it retrieves the wrong or incomplete information, the chatbot's response will inevitably be incorrect. Another aspect of the RAG is the source information on the domain. A basic implementation of RAG treats all documents equally, regardless of the quality and accuracy of its content. The hallucinations of the RAG and LLM can be diminished by more advanced systems, but not eliminated. One option is to provide a link to the source information the RAG retrieves and an accuracy score of the LLM, enabling the users to double check the system. Finally, educating the users about the limitations of the system is also important, giving them a better understanding of what the chatbot can handle and where its prone to error [38]. The AI Agent utilized by *Dr. Upps* provides built in functionality that allows for gauging the confidence that all information in the response made by the LLM, is grounded by information collected by its RAG.

# 8 Future Work

This segment explores possible directions for future research and development within the scope of the project. While research and development, made in this project, has made noteworthy contributions, there are still various limitations and chances for enhancement to be addressed.

## 8.1 API key

One important aspect of future development involves the integration of API key authentication for the REST API endpoints. By requiring clients to authenticate using an API key, the security of the system is enhanced by ensuring that only authorized users and applications have access to the services. To achieve this, aspects such as key generation and management, monitoring, logging and implementation strategies have to be considered.

## 8.2 System administration

Another aspect is the process and execution of system administration. By forwarding functionality to handle tasks such as updating data, managing user privileges, and database management to the frontend, usability is enhanced and the technical threshold for administrators is reduced. Achieving this would require extending the REST API to allow external access to management functions, as well as implementing a protected admin interface in the frontend.

## 8.3 LLMs and RAG

There is a native risk of the chatbot providing unverified answers, which could reduce its credibility. This risk can be mitigated by integrating functionality for the chatbot to provide links and references to context-based answers generated from data in the document database, thereby expanding its scope to function as a search engine.

Another important aspect is the ability of the chatbot to adapt answers based on the user's education and company role, or both. Achieving this would require assigning roles to clients, and implementing functionality to manage those roles. It is important to acknowledge that there may be limitations in what the AI service provider can support by default, and alternative providers may have to be considered.

## 9 Conclusion

The development and integration of *Dr. Upps*, demonstrates the potential of using AI solutions to generate domain-specific chat assistance, in a web based interface. By using Large Language Models and integrating them with the company's internal data, *Dr. Upps* shows results of efficient information retrieval, thus reducing the time to productivity, especially for new employees. While the system shows good results on generating accurate responses, its effectiveness can be held back by the need of specific user queries. This limitation implies that further development is necessary to improve its usability for users who may lack knowledge of the company's domain. Future work should focus on enhancing the system's ability to handle less specific queries. Overall, *Dr. Upps* is an example of how AI-driven solutions can be used to meet the specific needs of a company.

## References

[1] *Redefining the value of water*, (Accessed 2024-05-22). [Online]. Available: https://drupps.com/.

[2] D. Aduszkiewicz, *How to structure onboarding to speed time to productivity*, Accessed: 2024-01-17), Oct. 25, 2021. [Online]. Available: https://humanpanel.com/onboarding-and-new-hire-time-to-productivity/#timetoproductivity.

[3] *Chat gpt by openai*, (Accessed 2024-05-16). [Online]. Available: https://platform.openai.com/docs/introduction.

[4] *Dialogflow documentation*, (Accessed 2024-05-16). [Online]. Available: https://cloud.google.com/dialogflow/docs.

[5] *What is artificial intelligence (ai)?* (Accessed 2024-05-22). [Online]. Available: https://www.ibm.com/topics/artificial-intelligence.

[6] N. Reimers, *Openai gpt-3 text embeddings - really a new state-of-the-art in dense text embeddings?* (Accessed 2024-05-08), Jan. 28, 2022. [Online]. Available: https://medium.com/@nils_reimers/openai-gpt-3-text-embeddings-really-a-new-state-of-the-art-in-dense-text-embeddings-6571fe3ec9d9.

[7] *The top 5 vector databases*, (Accessed 2024-05-22). [Online]. Available: https://www.datacamp.com/blog/the-top-5-vector-databases.

[8] *What are large language models (llms)?* (Accessed 2024-05-22). [Online]. Available: https://www.ibm.com/topics/large-language-models.

[9] deparente, *Small but mighty: The difference between small language models (slms) and large language models (llms)*, (Accessed 2024-05-11), Apr. 29, 2024. [Online]. Available: https://medium.com/@dparente/small-but-mighty-the-difference-between-small-language-models-slms-and-large-language-models-07c80db0043a.

[10] M. D. Koninck, *Leveraging llms on your domain-specific knowledge base*, (Accessed 2024-04-21), May 8, 2023. [Online]. Available: https://blog.ml6.eu/leveraging-llms-on-your-domain-specific-knowledge-base-4441c8837b47.

[11] *Promptenginnering website*, (Accessed 2024-05-15). [Online]. Available: https://www.promptingguide.ai/.

[12] *Openai*, (Accessed 2024-04-21). [Online]. Available: https://openai.com/api/pricing/.

[13] *Gemini general info*, (Accessed 2024-04-21). [Online]. Available: https://deepmind.google/technologies/gemini/.

[14] *Ibm watsonx*, (Accessed 2024-04-21). [Online]. Available: https://www.ibm.com/watson.

[15] *What are ai agents?* (Accessed 2024-05-22). [Online]. Available: https://aws.amazon.com/what-is/ai-agents/.

[16] *Google vertex ai agent builder*, (Accessed 2024-05-22). [Online]. Available: https://cloud.google.com/products/agent-builder.

[17] *Google cloud services*, (Accessed 2024-05-22). [Online]. Available: https://cloud.google.com/.

[18] *React*, (Accessed 2024-04-02). [Online]. Available: https://react.dev/.

[19] *Angular*, (Accessed 2024-04-02). [Online]. Available: https://angular.io/features.

[20] *Vue*, (Accessed 2024-04-02). [Online]. Available: https://vuejs.org/.

[21] *Next.js*, (Accessed 2024-04-02). [Online]. Available: https://nextjs.org/.

[22] *Bootstrap*, (Accessed 2024-04-02). [Online]. Available: https://getbootstrap.com/.

[23] *Tailind css*, (Accessed 2024-04-02). [Online]. Available: https://tailwindcss.com/.

[24] L. Gupta, *What is rest?* (Accessed 2024-04-17), Dec. 12, 2023. [Online]. Available: https://restfulapi.net/.

[25] *Aws*, (Accessed 2024-05-16). [Online]. Available: https://aws.amazon.com/nosql/.

[26] *Docker overview*, (Accessed 2024-05-03). [Online]. Available: https://docs.docker.com/get-started/overview/.

[27] *Dockerhub*, (Accessed 2024-05-16 ). [Online]. Available: https://hub.docker.com.

[28] *Create realistic experiences, fast*, (Accessed 2024-03-27). [Online]. Available: https://www.figma.com/prototyping/.

[29] *Participatory heuristic evaluation*, Sep. 1998. [Online]. Available: `https://dl.acm.org/doi/pdf/10.1145/285213.285219`.

[30] *Coupling and cohesion – software engineering*, (Accessed 2024-04-16), Mar. 3, 2024. [Online]. Available: `https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/`.

[31] *Auth0*, (Accessed 2024-05-07). [Online]. Available: `https://auth0.com/`.

[32] *Socket.io*, (Accessed 2024-05-07). [Online]. Available: `https://socket.io/`.

[33] *Github actions*, (Accessed 2024-05-16). [Online]. Available: `https://github.com/features/actions`.

[34] K. Dhanagopal, *Infrastructure for a rag-capable generative ai application using gke*, (Accessed 2024-05-07), Apr. 2, 2024. [Online]. Available: `https://cloud.google.com/architecture/rag-capable-gen-ai-app-using-gke#security_privacy_and_compliance`.

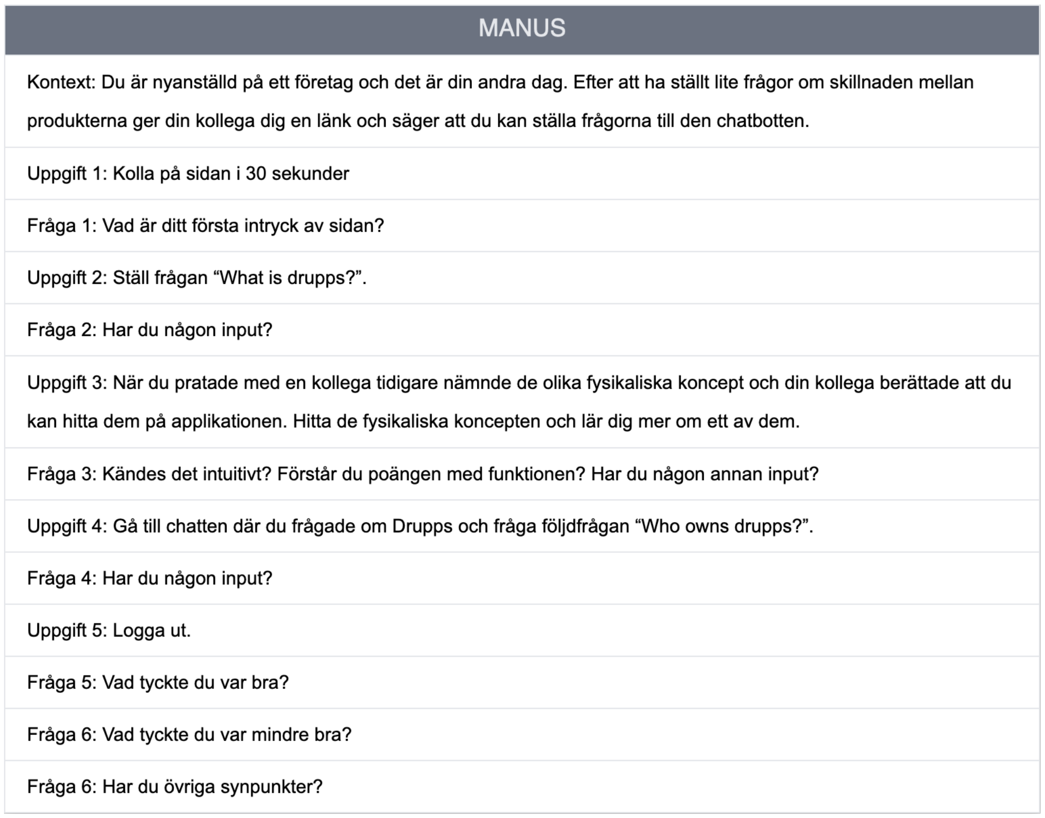
[35] M. Phute, *Llm self defense: By self examination,?* (Accessed 2024-05-14), May 2, 2024. [Online]. Available: `https://arxiv.org/pdf/2308.07308`.

[36] A. team, *Detecting chatgpt with ai: Enhanced security and moderation efforts*, (Accessed 2024-05-02), Nov. 6, 2023. [Online]. Available: `https://aicontentfy.com/en/blog/detecting-chatgpt-with-ai-enhanced-security-and-moderation-efforts`.

[37] P. J. Schulz and K. Nakamoto, *Worse than ignorance*, (Accessed 2024-05-03), Apr. 4, 2024. [Online]. Available: `https://www.cambridge.org/core/elements/worse-than-ignorance/42CE26781DD44AB85486905724D17DA3`.

[38] IBM, *What are ai hallucinations?* (Accessed 2024-05-02). [Online]. Available: `https://www.ibm.com/topics/ai-hallucinations`.

# 10 Appendix



| MANUS |
|---|
| Kontext: Du är nyanställd på ett företag och det är din andra dag. Efter att ha ställt lite frågor om skillnaden mellan produkterna ger din kollega dig en länk och säger att du kan ställa frågorna till den chatbotten. |
| Uppgift 1: Kolla på sidan i 30 sekunder |
| Fråga 1: Vad är ditt första intryck av sidan? |
| Uppgift 2: Ställ frågan "What is drupps?". |
| Fråga 2: Har du någon input? |
| Uppgift 3: När du pratade med en kollega tidigare nämnde de olika fysikaliska koncept och din kollega berättade att du kan hitta dem på applikationen. Hitta de fysikaliska koncepten och lär dig mer om ett av dem. |
| Fråga 3: Kändes det intuitivt? Förstår du poängen med funktionen? Har du någon annan input? |
| Uppgift 4: Gå till chatten där du frågade om Drupps och fråga följdfrågan "Who owns drupps?". |
| Fråga 4: Har du någon input? |
| Uppgift 5: Logga ut. |
| Fråga 5: Vad tyckte du var bra? |
| Fråga 6: Vad tyckte du var mindre bra? |
| Fråga 6: Har du övriga synpunkter? |

Figure 8: Script for user testing

| Participant | Q.1 | Q.2 | Q.3 | Q.4 | Q.5 | Q.6 | Q.7 |
|---|---|---|---|---|---|---|---|
| 1 | Snygg, gillar färgtemat | Vill se att det laddas | Ja. Ja. Gillar färgerna på topics, men konceptsidan är lite tråkig | Hade velat att det stod vart som är nyast i chatlog och uppdelning i tiden | Lätt att navigera med sidmenyn | Avsaknad av tidsmärkningar i chat och att det inte visar när det laddar | - |
| 2 | Bra att det är mörkt, men färgerna på topics är lite mycket | Hade velat ha att det skrivs ut allt eftersom | Ganska. Typ. Ask dr.upps sticker ut för lite | Ikonerna ser konstiga ut och hade velat ha friktion | Att det är simpelt. | Friktion | - |

Table 3: Results from the heuristic evaluation.

| **Endpoint** | **Usage** |
|---|---|
| /load_topics | Returns JSON of all the topics |
| /load_concepts | Retuns JSON of all the concepts related to the topic |
| /get_sessions | Returns JSON containing all sessions connected to a user_id |
| /get_session_id | Returns a session_id when creating a new session |
| /load_session | Returns the chat history of the given session_id, sorted by timestamp |
| /delete_session | Deletes a specific session along with its connected messages |
| /load_profile | Gets user profile data given user_id |
| /sign_in | Signs in the user, creates new if user does not exist |
| /message | Main function Handling messages from frontend, returns LLM answer through an established socket connection and stores necessary data in the database. |
| /connect | Called when a client connects to the socket |
| /disconnect | Called when a client disconnects from the socket |

Table 4: REST API endpoints & Usage