# Factory Simulation

Faruk Kardovic

July 06, 2025

The project I propose is a simulation of a Chat GPT named Future Assistant . The agent in my simulation is a text prompt who will be able to answer the questions asked by customer. Future Assistant is an agent designed as a personal assistant that will make everyday tasks and obtaining information easier for future users. Currently, four functionalities have been implemented, and English language is supported, but the possibilities of this agent are unlimited and in the future it is planned to develop new functionalities according to the needs of target users. Technically, this agent is a generic LLM that is the basis of low-code platforms where the reusability of such applications is very high.

# Contents



# 1 Goals/Requirements

The goal of this project is to:

- create a simulation of a personal assistant who will behave like the real Chat GPT counterpart in providing help to the user
- implementing the simulation using Flask and other technologies and showing it in a GUI.
- testing the system to make sure it works as expected.

## 1.1 Scenarios

Users are expected to ask questions about available functions described in the introduction of the document and have his answers presented through the GUI.

# 2 Background

Many aspects of intelligent systems were taken into account in order to produce the software for the project or were shown in the behavior of the agent, some of the aspects that are required to understand the overall project are the ones discussed as follow:

The concept of autonomy has different aspects; it is about using the knowledge of the APIs and therefore the world to autonomously answer users questions and prompts.

- **Autonomy**: agents encapsulate control along with criteria to govern it, making them self-regulating entities. They can also control only what is set inside the boundaries defined by the functions of the agent. They are also self governing showing by using the ability to get real time data and present it to the user.

- **Reasoning:** The agent is using epistemic reasoning that is directed towards gaining new knowledge by replacing old information (which is outdated) with new information in order to create more effective reasoning behavior.

- **Intentions:** The agent is looking at intention of a customer to find the right response, there is no predefined orders of questions that customer can ask , so it will take from the query the intention of the customer and analyze it .

# 3. Requirement Analysis

**Functional requirements**

- Weather forecast (available languages: eng)

Weather forecasting is implemented within get_weather function.
The user asks the prompt „What is the weather for {City}“ . The LLM then processes the question and sends it to API. The API to get real time weather report used is api.open-meteo.com . The response is received from the API and then displayed to the user.

- Calculator (available languages: eng)

This functionality is implemented within the calculate() function. It supports all the basic operations and through OpenAI the linguistic ability to use it in two languages is provided.

- Search for news from the Internet (available languages: eng)

This functionality is implemented in the get_news() function. The function is directly connected to NewsAPI to have access to the latest news at any time. Due to the OpenAI API, the language ability to ask questions in multiple languages as well as get answers is provided.

- Calculator for BMI (body mass index)

The functionality is implemented in calculate_bmi ().
The application has a option to calculate the body mass index when weight, height and age are provided through the UI.

**Non functional requirements**

- **Reliability**:

1. Handles exceptions to prevent crashes.
2. Returns error messages on API failure.

- **Security**:

1. Uses .env file to securely store API keys.
2. Avoids hardcoding sensitive data.

- **Maintainability**:

1. Modular function design with clear responsibilities.
2. Uses docstrings for documentation and clarity.

- **Scalability**:

1. Structure allows easy extension or integration into larger systems.
2. Can be scaled with features like caching, batching, or queuing.

- **Usability (Developer Usability)**:

1. Simple, reusable functions with default parameters.
2. Code is clean and easy to read.

- **Portability**:

1. Runs on any platform with Python and required libraries.
2. Environment configuration handled through standard .env files.

- **Performance**:

1. Depends on OpenAI API latency.
2. No optimization (e.g., caching) yet, but structure allows for it.

- **Extensibility**:

1. Easy to add support for additional models or features.
2. Functions can be extended to include logging, retries, or more complex workflows.

# 3 Design

## 3.1 System structure

It is developed using Flask, Javascript and CSS. With special care, the animation of the clock was developed, where with the help of two images, one static, the other that moves, the illusion is created that the clock is really moving.

## 3.2 Domain entities

In this application , the domain is around interacting with a language model (e.g., GPT). Here's a breakdown of the main domain entities

- **Prompt**
    - Representing the user's input (query) sent to the language model.
    - Used in both generate_text_basic and generate_text_with_conversation.
- **System Prompt**
    - A special message that guides the model's behavior.
    - Domain concept for configuring model personality or behavior.
- **Message**
    - A single entry in a conversation ({"role": ..., "content": ...}).
    - Part of a conversation structure used in the OpenAI Chat API.
    - Appears as a list of messages in generate_text_with_conversation.
- **Model**
    - Represents the version of the language model being used (e.g., "gpt-3.5-turbo").
    - Can be considered a domain entity if multiple models are supported or selected.
- **Response**
    - The result returned by the model.
    - A key output of the system that drives its business value.

- **OpenAI API Client**
    - Although not defined as a class or object in your code, it's an implicit domain service that handles text generation.

### 3.3 Behaviour

The software agents programmed are autonomous and encapsulate their control logic, only by sharing data between them, they are able to cooperate and take actions accordingly.

## 4 Technologies Used

Different technologies, like the ones described below, where used to develop this agent:

- Python – the main programing language
- Python libraries – libraries of the main programing language
- OpenAI – API to provide language processing
- Flask - WSGI web application framework
- Javascript – language used
- Css – language used for making stylish changes
- CSRF Protect - Cross-site request forgery (also known as CSRF) is a web security feature that will prevent an attacker to induce users to perform actions that they do not intend to perform.
- Datasets accessible via open/paid APIs from the internet.

## 5 Validation

There was no need for validation because the system is connected to intelligent APIs

## 6 Run project instructions

In order to successfully run the project please execute the following commands:

```
pip install python-dotenv
pip install openai
pip install flask
pip install Flask-WTF
```

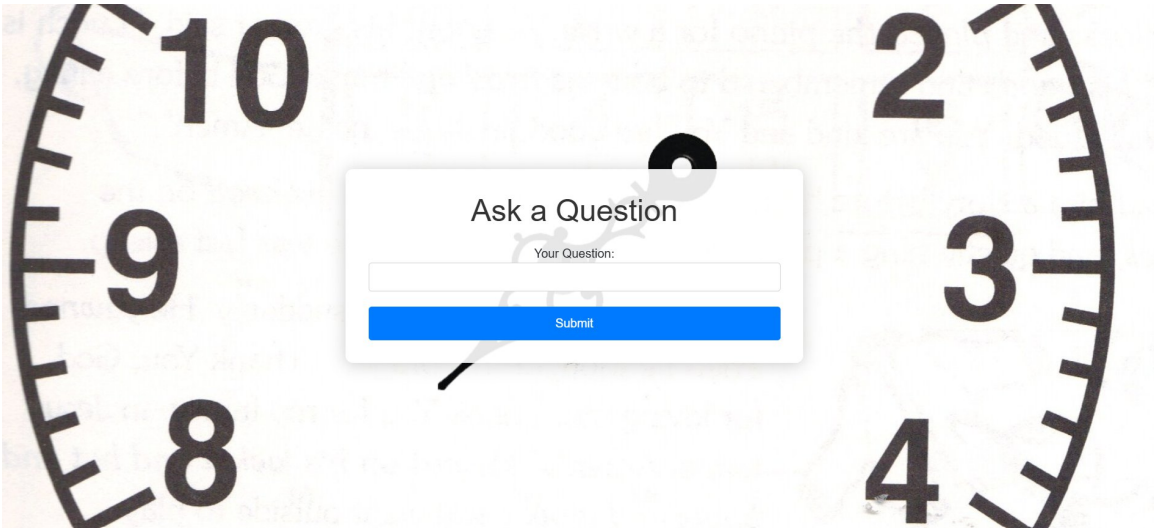
```
pip install newsapi-python
cd projectCode
python app.py
```

Also because the .env was forbidden to be uploaded to git , you need to provide following value of API and add .env file before running the application :

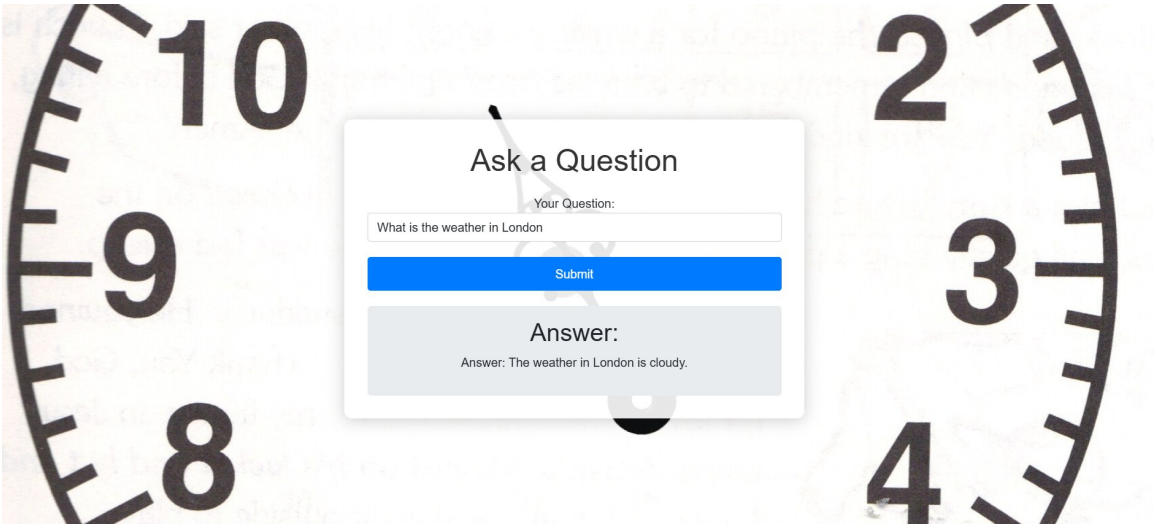
```
OPENAI_API_KEY="your key"
SECRET_KEY=your secret key
```

# 7 Usage example

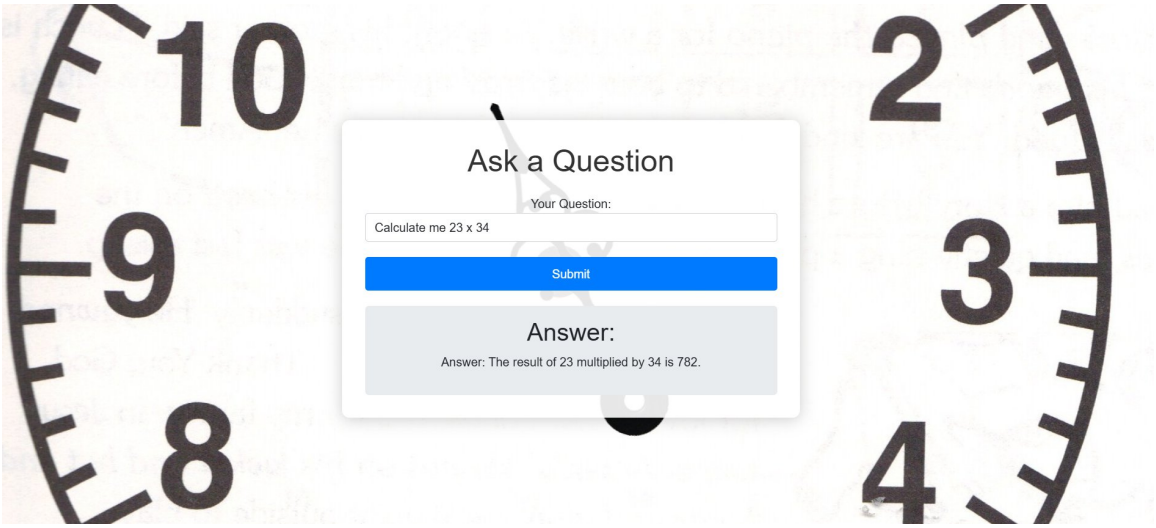
Please see the screenshots below for example of application look:



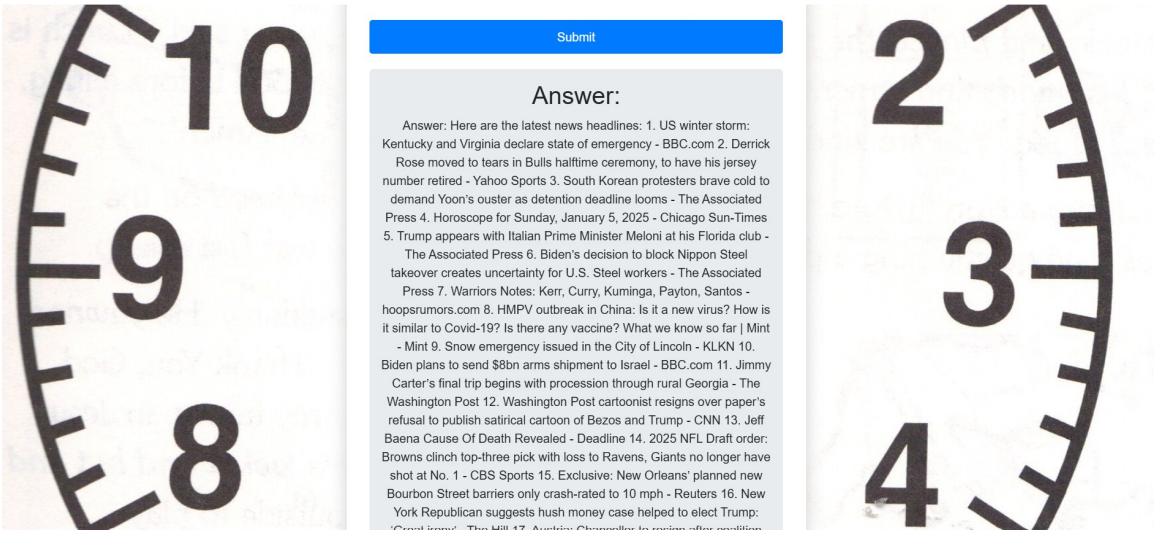

Asking about weather:



Asking about Calculations




Asking about news




# 8 Future works

It is possible to extend this agent by further improving upon the functionalities and abilities that it can perform and answer