Warsaw University of Technology

group project

General Design Document

Authors:
Kołodziejczyk Filip, Świstak Jakub

Supervisor:
prof. dr hab. inż. Przemysław Biecek

Version: 1.3

November 14, 2023

Contents

1 History of Changes

Table 1 History of Changes

Date	Author	Description	Version
12.10.2023	Filip Kołodziejczyk	First version (Abstract, Table of Contents, System Architecture, Modules Design, Communication)	1.0
12.10.2023	Jakub Świstak	Added User Interface Vision, External Interfaces and Technology Selection sections	1.1
13.10.2023	Filip Kołodziejczyk	Added Main Components Description section	1.2
13.10.2023	Jakub Świstak	Small fixes	1.3

2 Abstract

This comprehensive document provides a detailed blueprint for the development of a system, covering aspects crucial to its successful implementation. The document is organized into seven main sections, each addressing a critical aspect of the design process.

1. **System Architecture** - Overview of logical modules of our system, concerning the responsibilities, as well as the physical location of those modules, considering the computational unit on which the module is run.
2. **Modules Design** - One more look at the system architecture, this time touching on their relationships, and interfaces between them.
3. **Communication** - Final reference to system architecture, describing protocols and libraries used for inter-module communication. Network configuration of the system is discussed as well.
4. **Main Components Description** - Visual representations of the system from different perspectives, using UML diagrams.
5. **User Interface Vision** - Representation of UI vision via wireframes. All of the necessary functionalities of the frontend will be described too.
6. **External Interfaces** - Note all the standards used in this project, except the communication protocols,
7. **Technology selection** - Rationalized choices for languages, libraries, platforms, and OS.

This document serves as a foundational reference, streamlining the understanding of the system's design for effective collaboration and development.

3 System Architecture

The system architecture is simple. It consists of two main components: frontend application, providing a graphical user interface for interactions with our backend application, and backend, which handles the whole process of semi-automatic analysis of a given data set. For the production-ready version, it is assumed system will support multiple users. This will impose the need for a database with users' data, and some storage, to persist imported data sets and analysis results. To have the whole picture of our solution, supporting components must be included too. It is a runtime, where code used for analysis will be executed. Runtime is responsible for the report of data analysis summary generation. Another key component is an LLM. To be exact, two separate agents are used, one for providing the ideas for analysis process steps (theoretical one) and code generation agent, converting those concepts to executable code, which will realize those concepts. In Figure 1. this architecture is visualized.

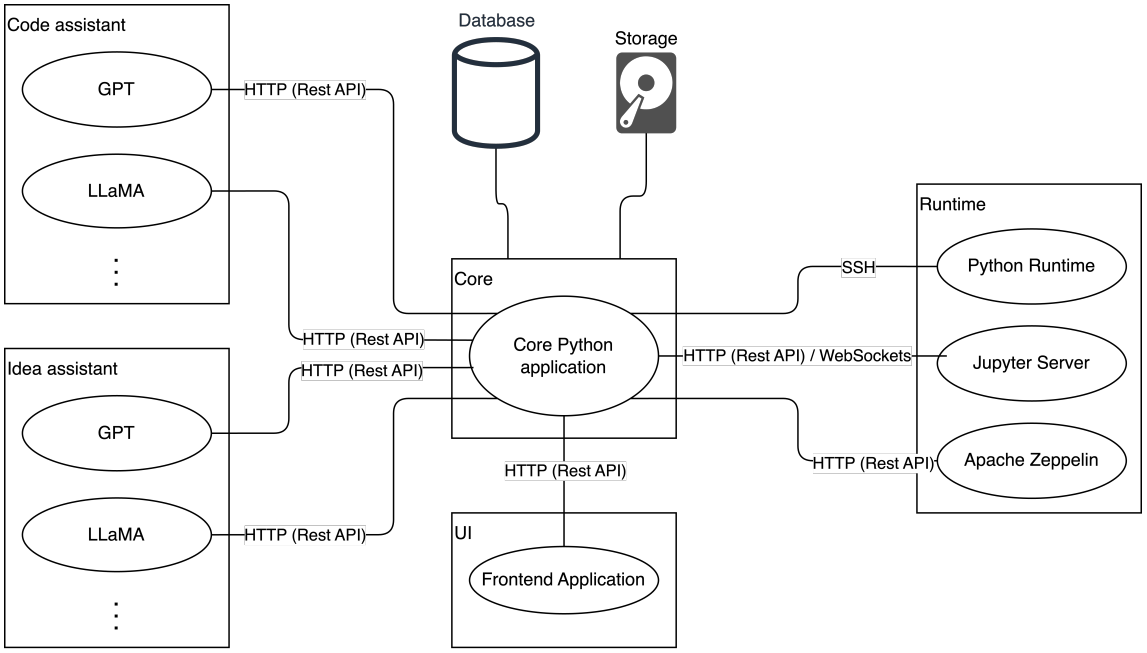

Figure 1: System architecture overview

In this figure, each rectangle represents one module of our system, which, at the same time, represents one processing unit (workstation, virtual machine, or container). Ellipses, in turn, are concrete instances of the modules. Thus, we may notice that our backend, so the whole system, is modular since it supports different LLMs and runtimes.

Here is the detailed list of all modules next to their responsibilities:

- **Core (backend)** - prompt management, conversation (with LLMs) management, communication with LLMs and runtime, handling of frontend requests
- **UI (fronted)** - graphical user interface allowing the user to run the analysis, configure parameters of this analysis (like used assistant and runtime), and display the results of this analysis
- **Runtime** - dataset reading, analysis code execution, report creation and edition
- **Idea assistant** - generation of next step of analysis (concept), based on the history of analysis done so far, including code execution outputs
- **Code assistant** - generation of code intended to realize the step of analysis suggested by idea assistant

- **Database** - user's data storage
- **Storage** - the persistence of datasets imported by users to the system and history of analysis (reports, chat history)

4 Modules Design

The system's backend is the module, which connects another module together. It is in the center of the system. For frontend connection, REST API is provided. For communication with runtime and assistants, interfaces are prepared, so those modules may be easily switched. To be able to function properly, the system requires all the modules to be available, except the frontend (the backend can be operated in terminal mode, however it is meant only for development).

Although, each module is independent, in terms of the startup. If the system is in idle state, no dependencies are enforced on the modules, so it makes it easy to swap some modules, and there is no strict way of system initialization.

5 Communication

In Figure. 1 all of the protocols used for communication between modules are shown. Most of the connections are handled via HTTP (RestAPI). For some runtimes, additionally, WebSockets and SSH are utilized. It is assumed that runtime will be isolated from the Internet, so the risk of malicious code is reduced. It should be able to be reached only from the backend, The same goes for database and storage, to lower the attack possibilities. Assistants may be either hosted either in an internal network or externally (OpenAI ChatGPT is hosted by 3rd party).

6 Main Components Description

In this section high-level overview of implementation will be shown. Since frontend is only an UI interface, without real effect on how the processing is done, it won't be discussed here. Database and storage will also be excluded, because they are just tools for storing the data present in application for longer periods of time.

6.1 Classes

We may start with the class diagram, shown in Figure 2.

The core class is ConversationManager, which is responsible for the whole processing, and it interconnects (utilizes) all other components. After the program is initialized, control is just handed to this class (in reality on top of this there will be an API framework). ConversationManager class uses 3 interfaces:

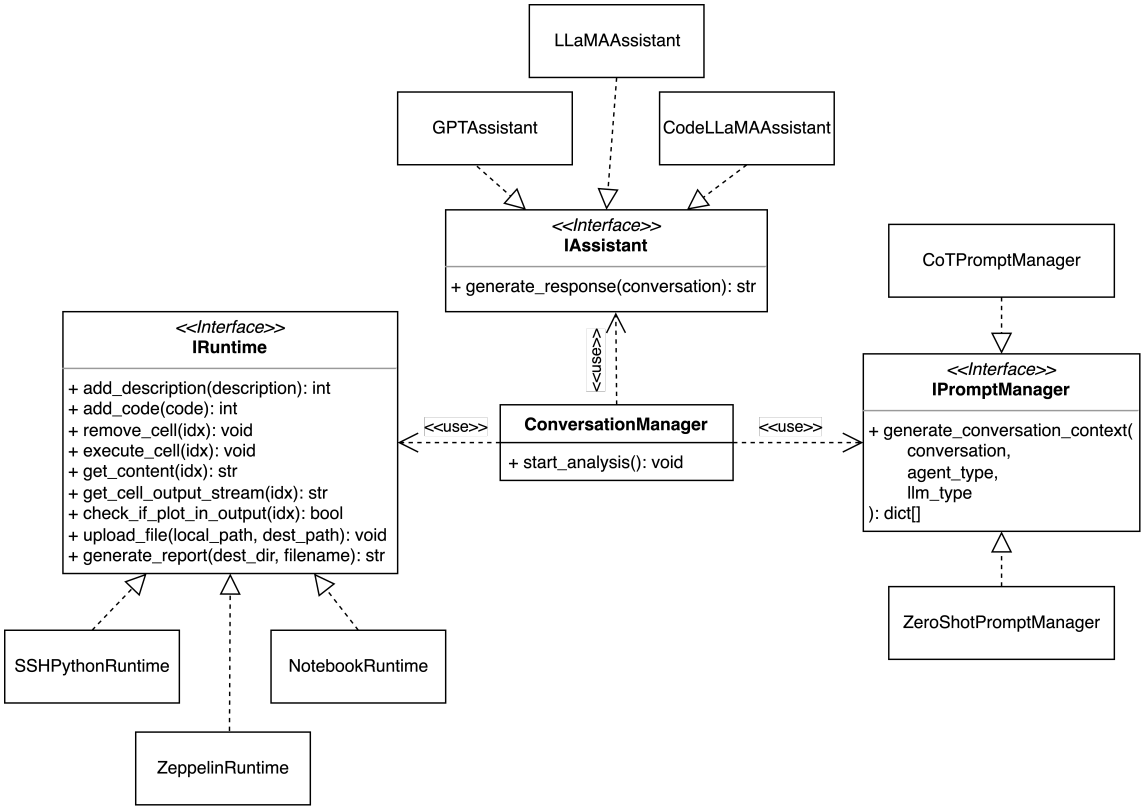

Figure 2: Class diagram

- IRuntime - wrapper for external code execution environment, and tools for report generation
- IAssistant - wrapper for communication with LLM assistant
- IPromptManager - unified API for generating prompts using different methods.

The following classes implementing those interfaces are planned in base system:

- SSHPythonRuntime (IRuntime) - handling pure Python runtime via SSH and report in markdown
- NotebookRuntime (IRuntime) - handling Jupyter Server as a runtime via RestAPI and WebSockets and report in JupuyterNotebook
- ZeppelinRuntime (IRuntime) - handling Apache Zeppeling as a runtime via RestAPI and report in Zeppelin Notebook
- GPTAssistant (IAssistant) - handling communication with OpenAI's GPT
- LLaMAAssistant (IAssistant) - handling communication with Meta's LLaMA 2
- CodeLLaMAAssiatant (IAssistant) - handling communication with Meta's LLaMA Code

There is still no agreement on what concrete classes will be implemented for IPrompt-Manager, but for sure they will cover popular prompt engineering techniques.

6.2 States

Most of the components is (almost) stateless, like assistant or runtime from the system's perspective. Runtime obsiously keeps the variables and other object defined during given analysis. One could say that runtime may be assigned two 2 states, executing and idle, but since it is synchronized with our backend, it does not make sense to emphasise that either. Application is either idle, waiting for new analysis or it is in such process, passing the data and prompts between runtime and assistants, until the result is satisfactory and report from analysis can be genereted and provided to the user.

6.3 Activities

Finally, using activity diagram the control flow of system from user's perspective will be presented in Figure 3.

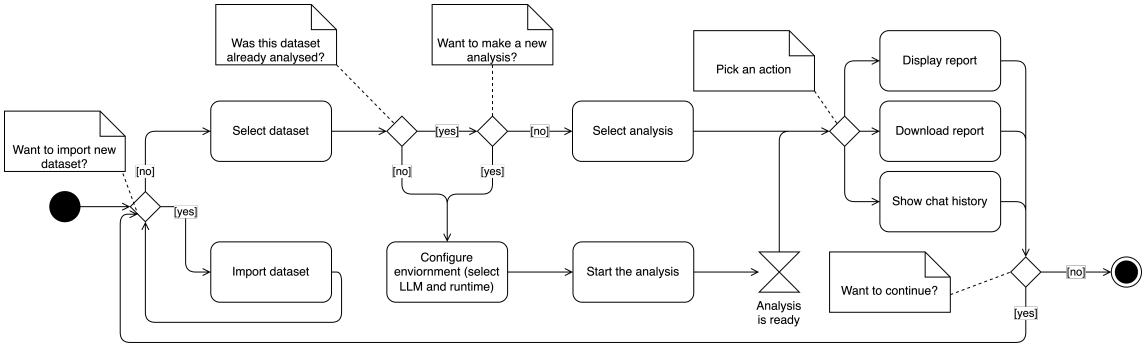

Figure 3: Activity diagram

7 User Interface Vision

Web application is expected to be the main way to access the system by end users. It shall provide the following functionalities:

- Import and export CSV datasets
- Select the dataset for analysis
- Run the analysis
- Display the report generated based on analysis
- Download the report generated based on analysis
- View the chat history behind the analysis process

- Display and download the report for analysi run in the past
- Configure the analysis (runtime and LLMs selections, and extra settings required by them)

Moreover, user interface should be simple, thus easy to use. Vision of such interface is depicted on Figure 4. and Figure 5.

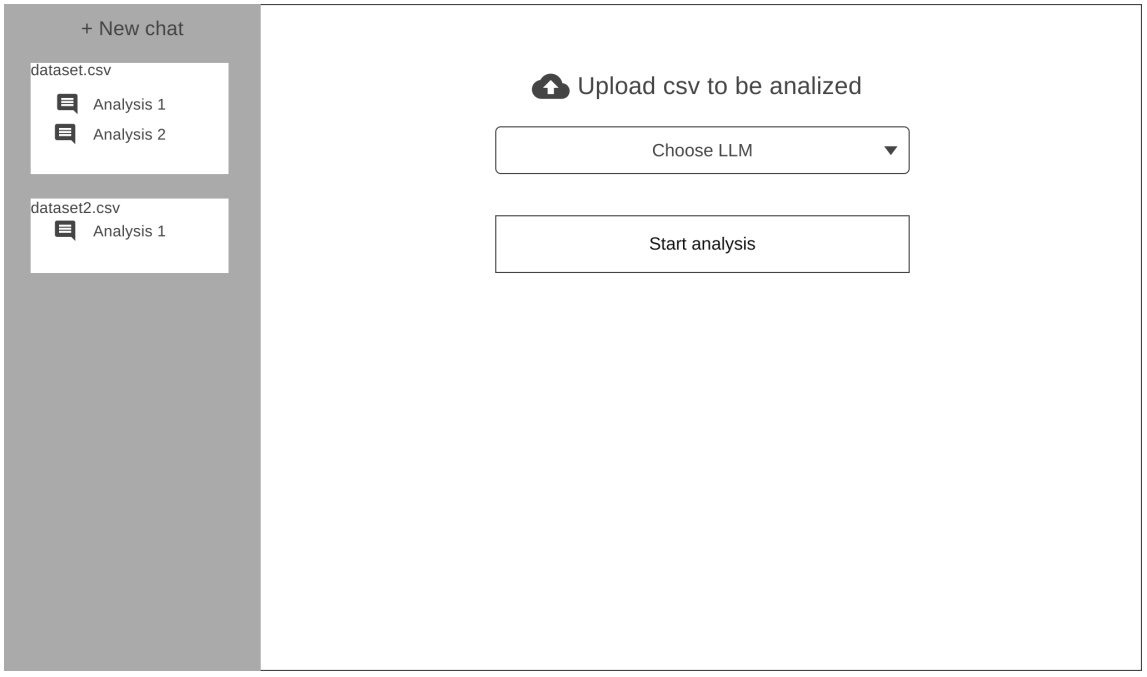

Figure 4: Initial page

8 External Interfaces

There are a few assumptions with respect to standards used in project. Those are:

- Datasets are expected to be in a CSV format.
- Final report will be generated in one format - PDF file. However, it will be possible to export intermediate files used for report creation. Format of this intermediate file depends on used runtime. For Jupyter Server it is a Jupyter Notebook, for Python container, it is a markdown file.
- Code to be used for analysis must be Python 3.10 compliant, and it cannot use any of the 'magic commands' of IPython kernel. Code cannot use libraries other than predefined (in section 8.).

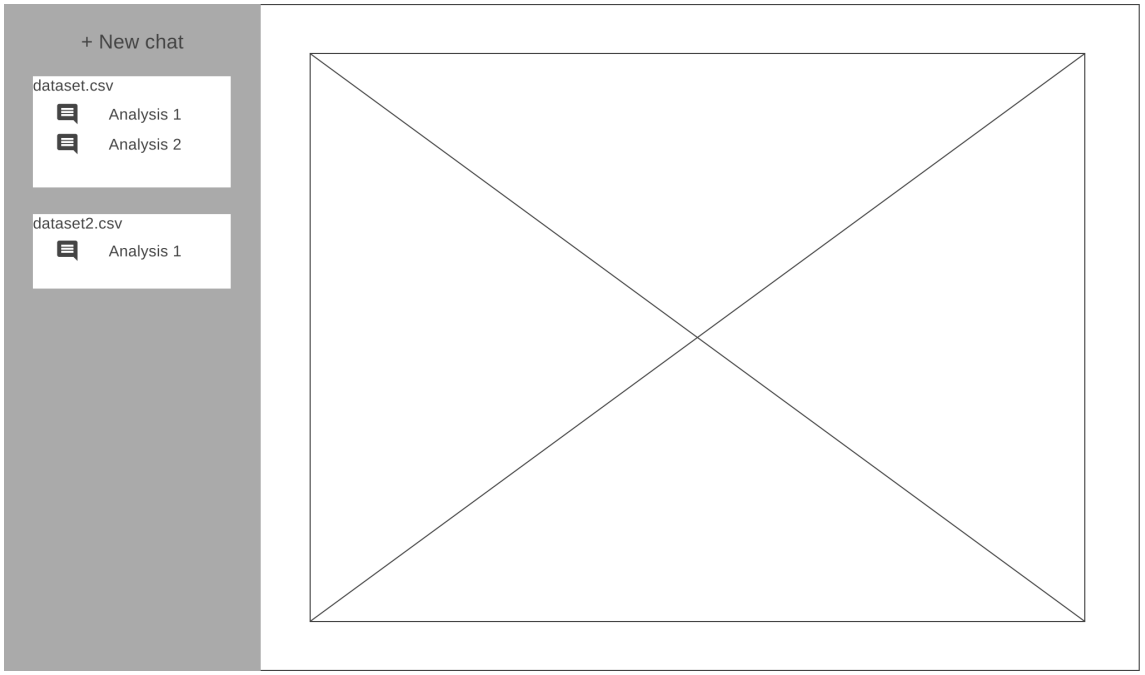

Figure 5: Report view page

9 Technology Selection

Our system will use modern and popular technology for good hardware support, ease of use, maintainability, and long-term support. Therefore, our core backend application will be written in Python language. It will be based on FastAPI framework, to provide reliable and stable RestAPI for frontend. Frontend will be written in JS using React framework. React provides consistent and seamless performance from user's perspective. Similarly to Python, it is a well-known, hence good support. On top of that, the vast amount of extensions are ready to use with React.

We will try to utilize as many libraries as possible to avoid necessary work, as long as given library suits our needs. Hence, among the most important libraries, for backend we will use:

- Pydantic - for data validation
- nbformat and nbconvert - for programmatic creation and edition of Jupyter Notebook, as well as its conversion to other format documents (like PDF)
- paramiko, websocket-client, requests - for communication with runtimes and LLMs using ssh, websockets and HTTP

We handpicked the set of most popular data analysis libraries, which are guaranteed to be provided in runtime, and thus may be utilized by generated code. Those are:

- pandas

- scikit-learn
- numpy
- matplotlib
- seaborn

Each of the core modules of software (UI + backend) is multiplatform, hence it can be run in most of the popular OSes and on servers with popular CPU architectures. It can be run in containers too. The target infrastructure for the whole system (except third-party hosted modules like ChatGPT) is a cloud environemtn based on Docker containers run under Linux.