Development Team Project: Project Report

Submitted by: Group (B)

Table of Contents

System Requirements and Design Decisions

System Requirements

Designing an academic research agent needs a precise list of technical tools. This ensures the system is dependable, can grow, and is simple to use. Python is the chosen language. It is easy to read and has many helpful libraries. The Python developer group is also very active (Van Rossum & Drake, 2009). Python offers the adaptability needed for web task automation. It also handles organised data effectively.

The following requirements form the foundation of the system:

- **Programming Environment**:

Python 3.10 or newer is required. This version supports the necessary libraries.

- **Web Scraping Libraries**:

The Requests library handles HTTP tasks. BeautifulSoup parses HTML files. Selenium manages dynamic content driven by JavaScript. (Mitchell, 2018).

- **Data Handling Libraries**:

Pandas is used for data cleaning. It structures data and exports it to CSV or Excel.

- **Agent Behaviour Framework**:

Spade enables the simulation of goal-driven agent actions. (Bellifemine, Caire & Greenwood, 2007).

- **Development Tools**:

Jupyter Notebook aids early testing. PyCharm or Visual Studio Code serve as integrated development environments.

- **System Environment**:

The system runs on Windows 11 or Ubuntu Linux. It needs 8 GB of RAM. A steady internet connection is vital for live data. These needs make the agent light and flexible. Each part was chosen to cut the excess. This also allows adding new features for research.

Design Decisions

Functionality and simplicity guided the design choices. Key decisions shaped the project:

1. **Language Selection (Python)**

Python was selected for development over Java or C++. This choice stemmed from its clear code and quick building times. Python also offers extensive libraries for data analysis and web data collection. Academic sources confirm Python's widespread use in AI and studies. This is due to its adoption in university settings. (Lutz, 2013).

2. **Scraping Approach (Requests + BeautifulSoup + Selenium)**

A mixed scraping method was used. Requests and BeautifulSoup work well for static pages. Selenium is used for dynamic sites like Google Scholar. These sites load content slowly. This approach allows handling many different academic sources.

3. **Data Representation (pandas with CSV/Excel Export)**

Pandas was selected for clean data storage. This organised data is ready for future checks. Storing data in CSV or Excel formats aids transferability. It also ensures compatibility with other data analysis programs. (McKinney, 2012).

4. **Agent-Like Behaviour (spade)**

Spade was chosen to model goal-driven actions. This choice helps mimic autonomy. It supports a belief-desire-intention model. This aligns the system with agent design. (Wooldridge, 2009).

5. **System Architecture (Modular Design)**

The system has three parts. One part finds data. Another gets and processes data. The third stores data. This modular setup eases finding errors. It helps teamwork. It also fits the need to simulate agent behaviour.

6. **Development Methodology (Agile with Iterative Prototyping)**

An Agile approach guided development. The team tests the function often. They work in short cycles. This matches the team contract. It calls for regular updates. Reviews are key to progress.

Critical Reflection on Design Choices

Each decision considered potential sacrifices. Selenium, for example, grants greater adaptability. However, its browser automation can reduce speed. Spade enhances agent realism. This, though, adds intricacy beyond simple scripts. These limitations were acknowledged. They supported adherence to agent design principles.

Employing open-source Python libraries lowers expenses. It also supports consistent outcomes. Yet, future maintenance causes concern. Libraries may become obsolete. To address this, the system is designed modularly. This permits component replacement. The entire agent will not require reconstruction.

Graphical Design & Report Presentation

Visualisation of the elements of the system is aimed at communicating the structure of the academic research agent in a clear and non-ambiguous way. Graphic models assist in rendering the architecture self-explanatory, logically organised, and understandable for both technical and non-technical users. Consequently, two UML diagrams have been generated: a class diagram and an activity diagram.

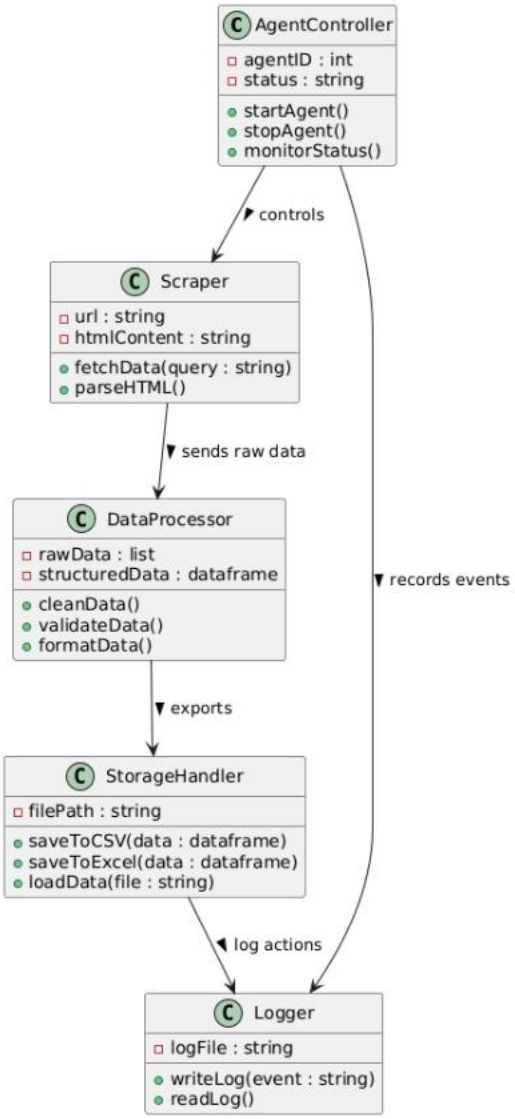
The Class Diagram gives a picture of the structural design of the agent system. It indicates the key classes, including AgentController, Scraper, DataProcessor, and StorageHandler. Their relationships depict the control of the behaviour of the agents, scraping of online academic content, processing the obtained results, and exporting the results into a valuable format. This kind of design is conducive to modularity and easy future modification or addition of features, which is in line with agile development principles (Pressman & Maxim, 2020).

The Activity Diagram is used to describe the workflow of the agent. It shows the sequence of steps: requesting a query, scraping content through the application of scraping tools, processing the data obtained, and storing the output. The diagram functions best under the situation of describing users interacting with automated processes, thus illustrating how the system takes actions autonomously to achieve project objectives (Booch, Rumbaugh & Jacobson, 2005).

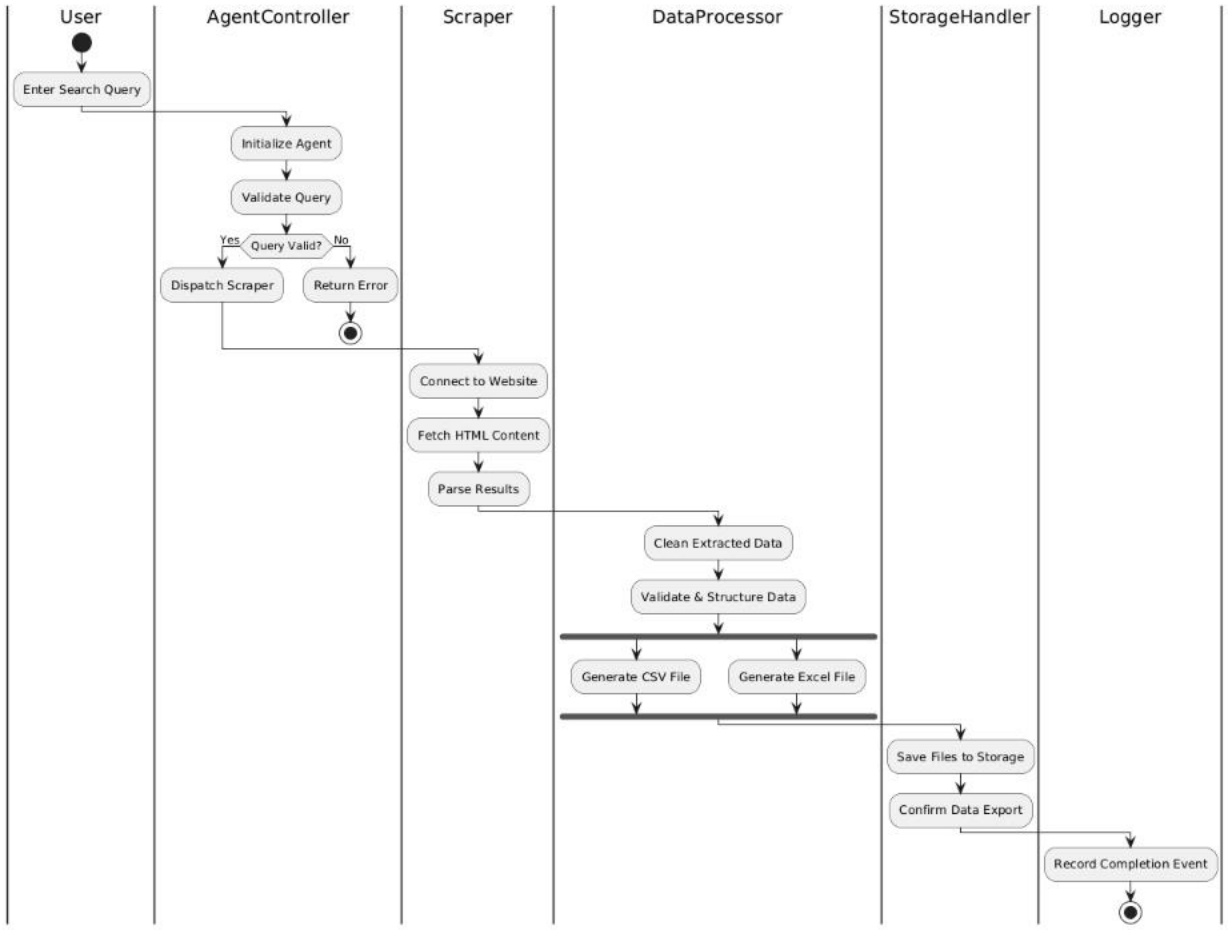
Finally, the report presentation should not only be technically correct but also professionally straightforward. All the diagrams are labelled and integrated uniformly, formatting adheres to academic standards, and proofreading enables easy reading. Overall, all these factors guarantee that the final submission is in accordance with both structural and academic standards.

UML Diagrams

1. Class Diagram



2. Activity Diagram



Critical Analysis of Agent Design and Methods

The academic research agent design exhibits significant capabilities in modularity, scalability and compliance with agent-oriented principles. The division of the duties between the AgentController, Scraper, DataProcessor, StorageHandler and Logger makes sure that every single element is focused and testable. Such a modular design is inspired by the idea of Pressman and Maxim (2020), focusing on maintainability, and it is conducive to the agile process. It can also be extended to future improvements, like the insertion of machine learning modules to perform semantic filtering without having to re-architect the basic system.

One of the major strengths is the hybrid scraping strategy of Requests, BeautifulSoup, and Selenium. This also provides flexibility of both static and dynamic content, which is in line with the robust web scraping guideline by Mitchell (2018). Selenium, however, creates some overhead, performance and consumption, which restricts the efficiency of large-scale deployments. This trade-off illustrates the accuracy/speed balance in the retrieval of data.

The simulation's conceptual authenticity and the simulation of autonomy of the multi-agent system theory by Wooldridge (2009) are integrated with Spade to simulate belief-desire-intention (BDI) behaviour can give the project conceptual authenticity. However, spade adds and brings in complexity to basic automation, and this makes the issue of necessity to be compared with demonstration at the academic level.

Pandas data processing provides structured, transferable results in CSV/Excel, which contributes to McKinney's (2012) focus on interoperability. However, the use of third-party libraries is associated with longer-term maintenance risks because updates or deprecations of the library can break functionality.

Lastly, the design recognises the ethical and legal issues in scraping, which is a matter brought up very infrequently. With a built-in Logger to monitor system actions, the agent provides accountability and reproducibility but continues to be a limitation on compliance with platform policies.

References

Bellifemine, F., Caire, G. & Greenwood, D., 2007. *Developing multi-agent systems with JADE*. Chichester: John Wiley & Sons.

Booch, G., Rumbaugh, J., & Jacobson, I., 2005. *The Unified Modelling Language user guide*. 2nd ed. Boston: Addison-Wesley.

Lutz, M., 2013. *Learning Python*. 5th ed. Sebastopol: O'Reilly Media.

McKinney, W., 2012. *Python for data analysis: Data wrangling with pandas, NumPy, and IPython*. Sebastopol: O'Reilly Media.

Mitchell, R., 2018. *Web scraping with Python: Collecting data from the modern web*. 2nd ed. Sebastopol: O'Reilly Media.

Pressman, R. & Maxim, B., 2020. *Software engineering: A practitioner's approach*. 9th ed. New York: McGraw-Hill.

Van Rossum, G. & Drake, F., 2009. *Python 3 reference manual*. Scotts Valley: CreateSpace.

Wooldridge, M., 2009. *An introduction to multi-agent systems*. 2nd ed. Chichester: John Wiley & Sons.