Institute of Mathematics and Informatics

Web Scraping and Monte Carlo Simulations for Analytical Forecasting

Author
Lorand Heidrich
Computer Science BSc

Supervisor
Adam Kovacs
Teaching Assistant

Eger, 2024

Acknowledgments

I would like to express my gratitude to Dr. Eric Grono and Mr. Garrett Weinzierl for their mentorship and support throughout my journey into the domain of Monte Carlo simulation and analytical forecasting. Their expertise, patience, and guidance have not only paved the way for new avenues of exploration but have also instilled within me an appreciation for the intersection of computer science and predictive modeling.

Their influence has played an instrumental role in shaping my academic and professional trajectory. I am indebted to them for their impact on my endeavors.

Thank you both for your generosity in sharing your time and knowledge, and your patience in addressing my myriad of questions throughout my studies.

Contents

Chapter 1

Introduction

In today's world, data has become of paramount importance, profoundly influencing our lives and shaping decision making processes. The acquisition, processing, and interpretation of data is fundamental across multiple domains. [1] Recognized as the cornerstone of contemporary insights, data serves as the basis of deriving valuable insights, and making informed projections, thereby guiding strategic planning and allowing for suitable preparation in the face of uncertainty. However, utilizing the full potential of acquired information effectively in a complex, multi-variable dynamic environment can be a challenging task [2].

This thesis approaches data collection and forecasting from a sports analytical perspective, aiming to derive statistical insights and formulate projections regarding future performance. It endeavors to utilize a combination of web scarping techniques [3] and Monte Carlo simulation [4] for analytical forecasting. Through the integration of these techniques, this research aims to explore a comprehensive methodology for data acquisition and predictive modeling.

1.1 Contextual Background

The National Basketball Association (NBA) [5] is well known for its worldwide prominence and dedicated fan base. Its enduring popularity has resulted in a multitude of analytical data relating to historic games. This abundance of statistical data, along with a widespread general awareness of the sport and my personal enthusiasm for it, positions historic NBA games an ideal domain for exploring predictive modeling based on data obtained through web scraping.

1.2 Motivation

The incentive for this research is derived from a keen interest in the technical intricacies of web scraping and probabilistic elegance of Monte Carlo simulations. The application of these techniques transcends the domain of sports analytics, with uses in finance [6], physics [4], and beyond [7].

1.3 Objectives

The primary objective of this thesis is two-fold. Initially, to employ web scarping techniques to gather comprehensive historical NBA game data from the early 1990s. Subsequently, to utilize said data to simulate a general probabilistic outcome for selected historic NBA games.

Specifically, the research aims to:

- Develop a multi-approach web scraping pipeline to gather comprehensive historical data for a given NBA season and team.
- Manage and store the acquired data.
- Implement a multi-epoch Monte Carlo simulation to model potential game outcomes based on the attained data through modeling offensive possessions.
- Evaluate the predictive accuracy and reliability of the proposed methodology through empirical testing and validation against actual historic game results.

Through these objectives, this thesis undertakes to promote a deeper understanding of web scraping and predictive modeling within sports analytical forecasting.

Chapter 2

Methodology

2.1 Web Scraping Techniques

2.2 Monte Carlo Simulation

Chapter 3

Requirements

3.1 Requirements List

The system shall be constructed to uniquely fulfill both the requirements of a computer science thesis, and the domain of data acquisition and simulation based projections. It should therefore result in an intuitive end-user experience, leveraging the web scraping and Monte Carlo simulation methodologies explored in this thesis.

The client interface should allow users to interact with the business logic[1], thereby accessing the database through built-in functions. It should also allow for the utilization of web scraping and Monte Carlo methodologies. The Use Case diagram depicted below outlines the basic functionality described by the Functional - (see table 3.1), Non-Functional - (see table 3.2), and Platform Requirements (see table 3.3) outlined in this chapter.

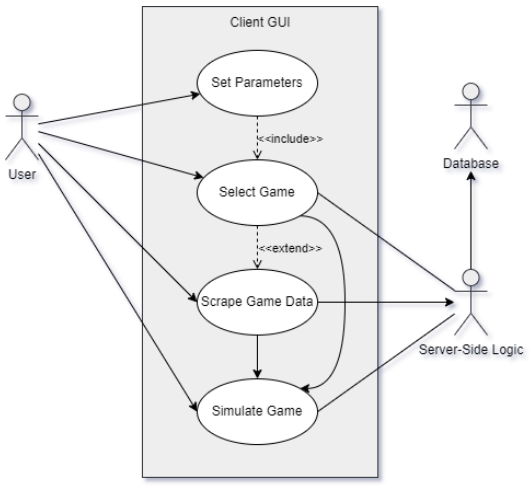

Figure 3.1: Use Case Diagram

[1]The term refers to the collection of algorithms responsible for allocating and processing data through communication with the database in order to serve the user interface, while maintaining its independence from both. For further information, please see [8].

3.1.1 Functional Requirements

ID	Name	Description
R1	Database	The system must allow users to select either the default database or utilize their own, based on a URI connection string.
R2	Game Parameters	It should provide users with a method to set the season, home- and away team.
R3	Epochs	The system must enable users to specify the number of epochs for the Monte Carlo simulation.
R4	Game Data	Historic game data should be displayed based on these settings for user review.
R5	Select Game	Users must be able to select an exact game to simulate from the displayed list of historic games.
R6	Missing Game	The system must recognize if the selected historic game is not in the database.
R7	Scrape Method	It should provide users with options for scraping the missing data trough different web scraping methods.
R8	Proxies	Scraping options should include the ability to use proxies.
R9	Proxy List	Users should have the ability to utilize their own proxy lists.
R10	Forced Scrape	The system must allow users the option to scrape game data even when it is deemed unnecessary by the algorithm.
R11	Validation	The system must ensure that data is not duplicated in the database.
R12	Simulation	The system must execute Monte Carlo simulations based on the selected game parameters.
R13	Graphs	It should visualize simulation results with graphs, including a probability density graph and a violin graph.
R14	Metrics	The system must return basic metrics such as the number of wins for each team and the mode of scores.
R15	Comparison	Users should be able to compare simulation results with original game data.

Table 3.1: List of Functional Requirements

3.1.2 Non-Functional Requirements

ID	Name	Description
NR1	Anonymity	The system must take steps to attempt anonymity throughout the web scraping process.
NR2	Validation	It should validate user input parameters, throwing errors when incorrectly set.
NR3	Errors	Users should be notified of errors during the application's operation.
NR4	Logging	It must utilize a logging system to allow for easier debugging.
NR5	Intuitive	The system must have an easy-to-use and intuitive interface.
NR6	Requests	The client side of the system must communicate with the server-side logic using HTTP to attain services as a responses.
NR7	SQL	The server-side logic should interact with the database using SQL queries.
NR8	Database	The system must be able to utilize separate MySQL database servers.
NR9	Testing	It should undergo thorough testing and validation to ensure accuracy, reliability, and robustness.
NR10	Regulation	The system must comply with relevant legal and regulatory requirements.

Table 3.2: List of Non-Functional Requirements

3.1.3 Platform Requirements

ID	Component	Requirement
PR1	Client	The application should be compatible with Windows 10 (or later) operating systems.
PR2	Client	The operating system is required to have .Net Framework 4.0.3 (or later).
PR3	Host	Server environment must be capable of running a Python application with a Flask framework.
PR4	Requirements	Back-end application requirements are available at: `https://github.com/lesheidrich/WebScraping_and_MCSim/blob/master/requirements.txt`.
PR5	Database	The database server must be compatible with either XAMPP or MySQL.

Table 3.3: List of Platform Requirements

3.1.4 Use Cases

- **Game Selection**: The user selects parameters such as the desired season, home- and away team, to initialize game selection, then chooses the desired match from the returned table.
- **Validation**: After accidentally setting a team to play against themselves, the user receives an error message alerting them of the mistake.
- **Scraping**: Following the game selection process, the system determines the game data is not in the database, then proceeds to utilize web scraping techniques to gather the data from online sources.
- **Simulation**: A user parameterizes the number of epochs for the Monte Carlo simulation and initializes game selection. The system utilizes the acquired historical data to run simulations, generating probabilistic outcomes for the historic NBA game.
- **Comparison**: Users compare the results of the Monte Carlo simulation with the original game data, assessing the accuracy of the model. The system further provides probability density- and violin graphs to further facilitate result analysis.
- **Error Handling**: When the user tries to initialize game selection, the database is down. The host service returns an error, notifying the user of the access issue. The user escalates the error, and upon its resolution normal system operations resume.

Chapter 4

Architecture

4.1 Design Concepts

At its core, the application relies heavily on a three principal layer [9, p. 19] concept, commonly found in systems utilizing a database and presentation layer. Fowler refers to these as presentation logic, domain logic, and data source. The presentation logic facilitates user interaction with the system, the data source handles data transactions and houses application information, while the domain logic's algorithms are responsible for data modification and layer interaction.

This is in line with the Gang-of-Four's [1] Model-View-Controller (MVC) [10, p. 529] design pattern. The View receives user-initiated interactions along with their parameters, and presents the application's data. It is capable of connecting directly to the model, while operating in conjunction with the Controller. The Controller interacts with both components as it processes their data and coordinates operations. The Model houses and manages the application data.

As discussed by Ahlan, A. R., Ahrnud, M. B., and Arshad, Y. [12], there have been several uses and variations of thin client applications since the 1970s. As a generalization, the *view* in its capacity as the client receives application data and logic based services from a host system. This application adheres to the this concept quite strictly, with the client acting as an intermediary between the user and the host, taking use parameters and displaying host response results. The host service, operating as the back-end, encompasses the previously discussed Model, Controller and all other components of the application.

[1] The Gang of Four [11] (GOF) are a group of four writers, all computer science professionals and entrepreneurs. Their literature and courses focus on professional development in the domain of computer science.

4.2 Components

Following the MVC design pattern's component structure, the application's presentation logic is allocated to the *view* package. The database and simple business logic allowing for record management is stored in the *model*. Acting as an intermediary between the two, the *controller* package orchestrates the flow of information along with its processing. The *webscraper* and *simulator* packages also tie into the *controller*, offering web scraping, and Monte Carlo simulation logic respectively, while further decoupling the application's components and allowing for better organization and maintainability through separation of concerns [2].

The application is organized in a manner, that all components embody system packages allowing improved readability and usability. The following subsections discuss each package's purpose and functionality.

4.2.1 *controller*

Purpose: The package is responsible for processing component interaction, and serves as the core logic of the system.

Functionality: Serving as the system's host, the *controller* is responsible for handling user prompts sent by the client in order to formulate an appropriate response. In order to fulfill this function it utilizes its connection to each component of the application, accessing their functionality as needed.

One of its responsibilities is accumulating data through the application of various web scraping methods, which are stored for future use. It's web scraper control functionality enables it to utilize the *webscraper* package to appropriate preset website data into memory, then reallocate it to the database using a combination of its own control processes along with the business logic of the *model* package. The web scraping sequence diagram illustrates the process (see 4.4).

When running a simulation for a parameterized game, the *controller* restructures relevant data for the selected historic games from the *model* into player, roster, and team objects usable by the *simulator*. The simulator returns the results to the host, which are forwarded back to the client. The Monte Carlo simulation sequence diagram shows further details (see 4.3).

Interaction: In its capacity as the main communications hub of the application, the *controller* interacts with every component in the system. The package's component

[2]Separation of concerns (SoC) is a software development design principle promoting segregation of source code elements by functionality in order to improve readability, organization and modification [13].

diagram provides a high level overview of basic component interaction (see4.1).

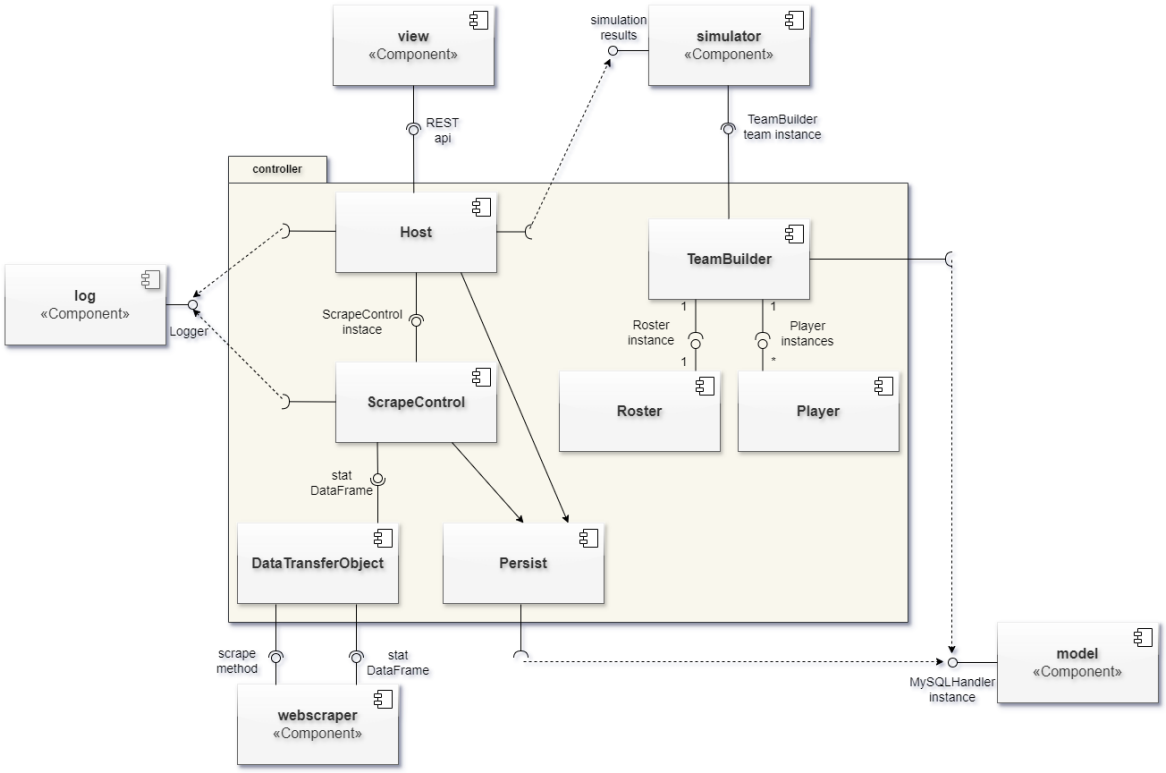

Figure 4.1: *controller* UML Component Diagram

4.2.2 *log*

Purpose: The component provides logging functionality for back-end operations.

Functionality: The *log* package contains logic allowing components to access system logs to document runtime errors and operations, useful for debugging potential issues. Log management functionality is also provided by the package, ensuring proper settings, size limitations and functionality.

Interaction: The *log* interacts with the *controller* and *webscraper* packages.

4.2.3 *model*

Purpose: The package ensures successful data management services within the application, through the storage and manipulation of data records and structures.

Functionality: The business logic enables system interaction with the database, encapsulating data access and administration services. Data security is enforced by minimizing vulnerabilities and prevents data corruption trough validation logic. The *model* services the system through the *controller* package's data manipulation and retrieval components.

The package also contains the NBA team enums [3] employed by the back-end logic. These ensure data validation and encapsulate all occurrences of team names in the system and its dataset, streamlining their application throughout processes.

Interaction: The primary business logic elements of the *model* communicate exclusively with the *controller*, enhancing system modularity (see 4.1). Due to their earlier described functionality, NBA team enums are also widely employed throughout the *simulation* package.

4.2.4 *simulation*

Purpose: The package's primary objective encompasses the repetitive simulation of a basketball game between selected teams at a specific point in time, with the end goal of returning the outcomes as graphical representations, comparable with the original historic game's outcome.

Functionality: The core functionality of the application revolves around implementing Monte Carlo simulations over a preset range of epochs to determine the probabilistic outcome of a historic NBA game.

The *simulator* package receives the relevant compiled data from the *controller*, initializing the creation of the simulation. Upon successful completion of the simulation, probability density and violin graphs are returned to the *controller* along with minimal game statistic like total win percentage and mode of scores reached per team. A detailed description of the process is illustrated in the Monte Carlo simulation's sequence diagram (see 4.3).

Interaction: The *simulator* relies on the *model*'s NBA team enum during operation, along with the *controller* for providing the necessary historic game data for simulation's successful operations. It further communicates with the *controller* in its capacity as the host. The *simulation* component diagram offers an overview of component interaction (see 4.2).

[3] In their capacity as a distinct object type, enumerations (enums) offer value binding across a group of encapsulated constants tied together through enumeration. Each value can act as a key identifier during instantiation, making enums a powerful structure for housing validated data [15].

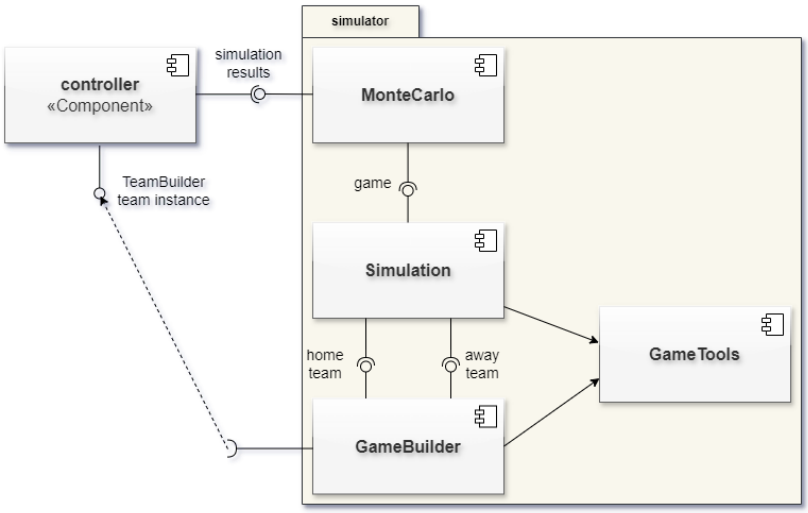

Figure 4.2: *simulation* UML Component Diagram

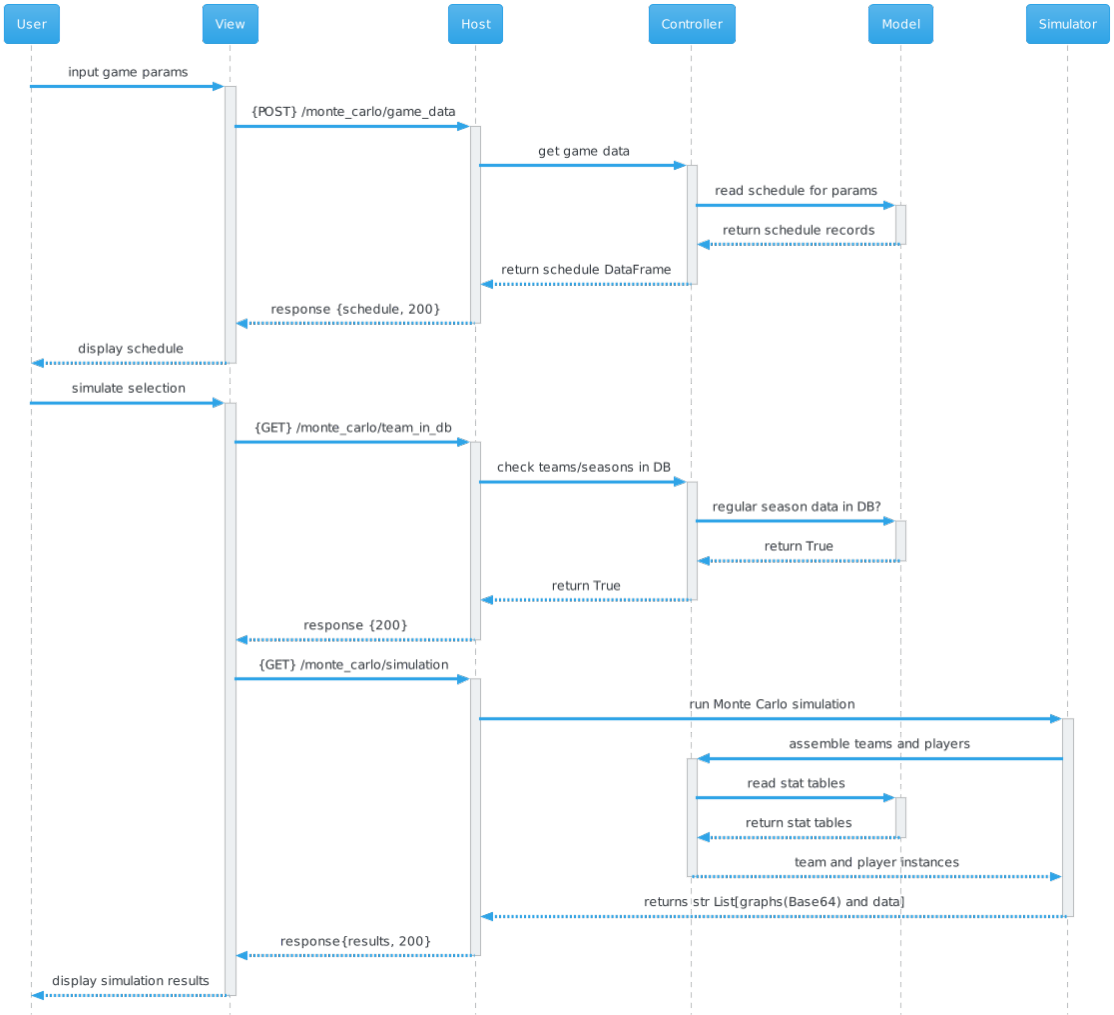

Figure 4.3: Monte Carlo Simulation UML Sequence Diagram

4.2.5 *test*

Purpose: The component is responsible for providing comprehensive back-end testing services.

Functionality: The test package constitutes a wide spectrum of tests focusing on the back-end of the system. It provides full-scale uni tests organized by module. Integration testing and linting is also included. The Testing and Validation chapter (see 6) contains a detailed breakdown of the package's functionality and implementation.

Interaction: Each unit test interacts with their respective components on the back-end. Integration tests interact with multiple packages following their functionality, in their attempt to check full system compatibility.

4.2.6 *view*

Purpose: To allow for user interaction with the system through communication with the host along with its application logic and data.

Functionality: The view package operates as a thin client, taking user input through a graphic user interface (GUI) [14]. User data is cached on the client side for the sake of user convenience, decreasing the input required to achieve functionality. Host responses, along with operational errors are displayed for user viewing. Errors are not logged on the client side of the application.

Interaction: The component interacts with the user and the host module of the *controller* package.

4.2.7 *webscraper*

Purpose: The package completes data gathering services from the amalgamation of preset websites and parameters representing selected NBA games, in order to supply historic statistical game data for the application.

Functionality: The acquisition process extracts data from a combination of preset URLs, set to match parameterized game data originating from the user. It includes functionality to interact with each URL in a manner defined by the user's chosen scraping method. Collected information is preprocessed and transformed before being committed to memory in a preset reusable format, easily accessed by the *controller* as it looks to the *webscraper* to acquire new information. Web scraping requests are generally not repeated, as the system is built to house already accessed data, thereby

minimizing dependency on online sources to bare necessity. The component's sequence diagram (see 4.4) illustrates the process during runtime.

Interaction: The *webscraper* package primarily communicates with the *controller* in order to receive data acquisition requests and parameters. Its preprocessed results are also returned to the *controller*. The package's component diagram (see 4.5) illustrates the *webscraper*'s interactions. Logging is utilized throughout the process.

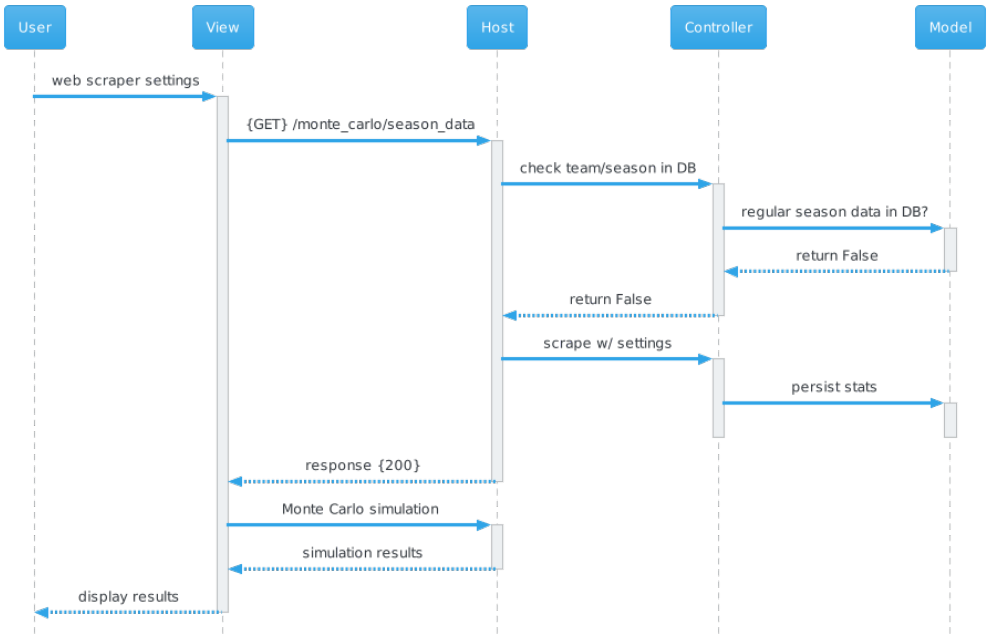

Figure 4.4: Web Scraping UML Sequence Diagram

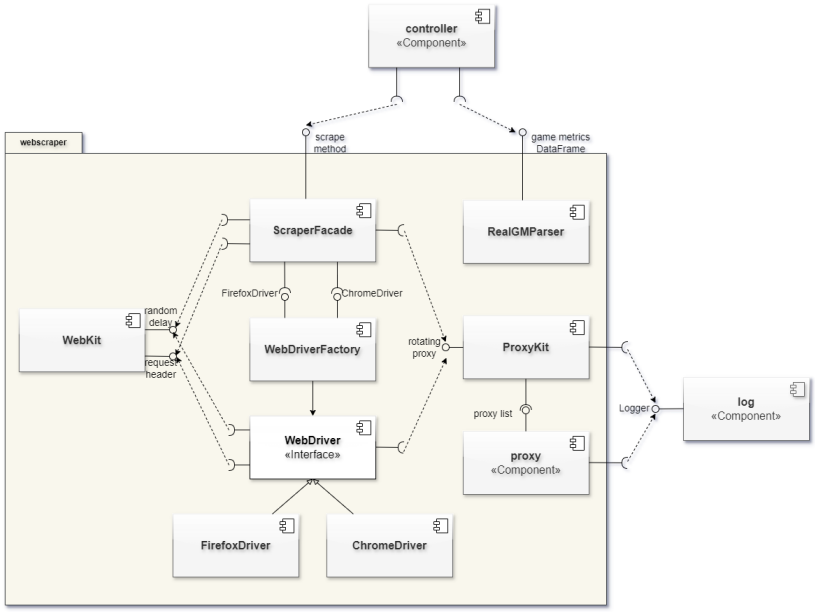

Figure 4.5: *webscraper* UML Component Diagram

4.3 Technologies and Frameworks

4.3.1 Beautiful Soup 4

Beautiful Soup [17] is a popular python package, designed to extracts data from HTML [4] and XML [5] documents sourced from the web. By leveraging attributes unique to the selected markup language, it facilitates tag and text content based parsing in order to precisely identify and extract the desired data. Developed for web scarping purposes, it is often used alongside packages responsible for making content requests to websites.

4.3.2 Fake User Agent

Throughout the extraction of data from websites it is crucial to consider the exchange of information between the client making the request, and the website's host server responding to the request. During this process, the host receives client information in the header, allowing it to discern information about the requesting client, like its operating system and browser type (see subsection 4.3.4 for information on HTTP requests and their headers).

Operations requiring repeated content requests to a single host, such as web scraping, can therefore be identified as outside the scope of regular usage requirements provided by the website. This in turn may lead to limitations of service, disrupting web scraping-based system operations.

Fake user agent applications are designed to solve this problem. Python's *fake-useragent* [20] library allows for the creation of random user agent headers, which can be assigned to requests. Users have the option to select from a wide range of preset options, which can also be filtered to only mimic specific browser technologies and operating systems. While this method only changes a portion of the request data, when used alongside other methodologies promoting anonymity, it can prove to be a powerful tool.

4.3.3 Flask

4.3.4 HTTP

Hypertext Transfer Protocol (HTTP) [19, 23] is a method of communication employed to facilitate data exchange between servers hosting web-based resources, and clients such as web browsers or web-applications.

[4] HyperText Markup Language (HTML) is a markup language used to house online content and set its layout, thereby creating the basic structure of web pages [16].

[5] Extensible Markup Language (XML) is a markup language for data transfer and storage. Rather than offer preset tags, users can organize content subjectively through the application of a key value pair data structure [18].

HTTP is unidirectional, operating on a request-response basis. When the client, such as a web browser requests services in the form of online content from a server, it does so through an HTTP request. The server processes the request, and sends an appropriate HTTP response. Data can be transmitted in multiple formats. Common examples include JSON [6] , HTML, and XML.

The client's request contains information such as the request line, message body, and headers [21]. The request line contains the destination resource, the message body stores parameters used by request methods, and the headers contain information about the client's choice of operating system and web browser. The response is comprised of a status code, headers and message body, which houses the requested data.

HTTP offers methods which can apply parameterized actions to the selected resource. Some examples include:

- The GET method presents the specified resource.
- POST submits data parameters to the server which are required for further functionality.
- PUT is used to update a selected resource on the server.
- The DELETE method removes the parameterized resource from the server.

Status codes are utilized to indicate the outcome of processes initiated by the client's HTTP request. They are categorized into groups, covering categories such as informational responses of success as well as errors. Some common examples of success codes include:

- 200: The requested transaction was completed successfully and the requested content is returned.
- 204: The server processed the request, and is not returning any content.
- 400: A client error obstructed the server's attempt to complete the request.
- 404: The server does not contain the requested resource.
- 500: Indicates the occurrence of an internal server error, obstructing the fulfillment of the client request.

Due to security concerns, a more secure version of HTTP Secure (HTTPS) [24] was created. It employs encryption mechanisms to ensure transaction security, thereby thwarting potential eavesdropping and tampering attempts.

[6] JavaScript Object Notation (JSON) is a popular lightweigth data structure which utilizes key-value pair functionality. Due to it's simplicity and compatibility with many programming languages it is a common choice for online information exchange [22].

4.3.5 Matplotlib

4.3.6 MySQL

4.3.7 Numpy

4.3.8 Pandas

4.3.9 Pylint

4.3.10 Requests

4.3.11 RestSharp

4.3.12 REST api

4.3.13 Selenium

discuss firefox, chrome and undetectedchrome drivers

4.3.14 Python

4.3.15 C sharp

4.3.16 .Net

4.3.17 XAMPP

4.3.18 White

4.3.19 WinForm

Chapter 5

Implementation

Chapter 6

Testing and Validation

6.1 Unit Testing

6.2 Integration Testing

6.3 System Testing

6.4 Performance Evaluation

Chapter 7

Results and Discussion

7.1 Analysis of Web Scraping Results

7.2 Evaluation of Monte Carlo Simulations

7.3 Comparison with Existing Methods

Chapter 8

Conclusion

8.1 Summary of Findings

8.2 Contributions to Knowledge

8.3 Limitations and Future Work

Theorem 8.1. *Text.*

Proof. Text. □

Definition 8.2. "Antinomies"

Remark 8.3. Text.

Chapter 9

Appendices

9.1 Code Samples

9.2 GUI Mockups

9.3 Test Cases

Bibliography

[1] Medium, *The Power of Data: Understanding Its Impact and Applications Across Various Domains*, Jonathan Mondaut, 2023, https://medium.com/@jonathanmondaut/the-power-of-data-understanding-its-impact-and-applications-across-various-doma [Retrieved: 2 March 2024]

[2] Northeastern University, College of Science, *Why it's so hard to make accurate predictions*, Jason Kornwitz, 2017, https://cos.northeastern.edu/news/hard-make-accurate-predictions/, [Retrieved: 27 February 2024]

[3] Moaiad Ahmad Khder, *Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application*, International Journal of Advance Soft Computing and Applications, Vol. 13, No. 3, 2021, Print ISSN: 2710-1274, Online ISSN: 2074-8523, Al-Zaytoonah University of Jordan

[4] Adekitan Aderibigbe, *A Term Paper on Monte Carlo Analysis/Simulation*, Department of Electrical and Electronic Engineering, Faculty of Technology, University of Ibadan, 2014.

[5] Wikipedia, *National Basketball Association*, 2024, https://en.wikipedia.org/wiki/National_Basketball_Association, [Retrieved: 27 February 2024]

[6] Don L. McLeish: *Monte Carlo Simulation and Finance*, Hoboken, New Jersey, USA, John Wiley & Sons, Inc., 2005.

[7] Paul Steffen: *Statistical Modeling of Event Probabilities Subject to Sports Bets: Theory and Applications to Soccer, Tennis, and Basketball*, Statistics [math.ST], Université de Bordeaux, 2022. English. NNT: 2022BORD0210. tel-03891393.

[8] Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston, *Object-Oriented Analysis and Design with Applications*, Massachusetts, USA, Addison-Wesley, 2007.

[9] Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford, *Patterns of Enterprise Application Architecture*, USA, Addison-Wesley Professional, 2002.

[10] Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra, *Head First Design Patterns*, O'Reilly, 2004.

[11] Spring Framework Guru, *National Basketball Association*, 2024, `https://springframework.guru/gang-of-four-design-patterns/`, [Retrieved: 5 March 2024]

[12] Abdul Rahman bin Ahlan, Murni bt Mahmud, Yusri bin Arshad, *Conceptual Architecture Design and Configuration of Thin Client System For Schools in Malaysia: A Pilot Project*, Department of Information System, Kulliyyah of Information and Communication Technology, Kuala Lumpur, Malaysia, 2010.

[13] Chris Reade, *Elements of Functional Programming*, Boston, USA, Addison-Wesley Longman, 1989.

[14] Wikipedia, *Graphical user interface*, 2024, `https://en.wikipedia.org/wiki/Graphical_user_interface`, [Retreived: 5 March 2024]

[15] Microsoft Learn, *Enumeration types (C# reference)*, Bill Wagner, 2023, `https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum` [Retreived: 6 March 2024]

[16] MDN Web Docs, *HTML: HyperText Markup Language*, 2024, `https://developer.mozilla.org/en-US/docs/Web/HTML`, [Retrieved: 6 March 2024]

[17] Beautiful Soup 4.12.0 documentation, *Beautiful Soup Documentation*, 2004-2023 Leonard Richardson, `https://www.crummy.com/software/BeautifulSoup/bs4/doc/#`, [Retrieved: 6 March 2024]

[18] MDN Web Docs, *XML: Extensible Markup Language*, 2024, `https://developer.mozilla.org/en-US/docs/Web/XML`, [Retrieved: 6 March 2024]

[19] MDN Web Docs, *HTTP*, 2024 `https://developer.mozilla.org/en-US/docs/Web/HTTP`, [Retrieved: 6 March 2024]

[20] PyPI Python Package Index, *fake-useragent*, 2023, `https://pypi.org/project/fake-useragent/#description`, [Retrieved: 6 March 2024]

[21] MDN Web Docs, *HTTP headers*, 2024, `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers`, [Retrieved: 6 March 2024]

[22] MDN Web Docs, *JSON*, 2024, https://developer.mozilla.org/en-US/docs/Glossary/JSON, [Retrieved: 7 March 2024]

[23] Wikipedia, *HTTP*, 2024, https://en.wikipedia.org/wiki/HTTP, [Retrieved: 7 March 2024]

[24] Wikipedia, *HTTPS*, 2024, https://en.wikipedia.org/wiki/HTTPS, [Retrieved: 7 March 2024]

[25] , , , [Retrieved:]

[26] , , , [Retrieved:]

[27] , , , [Retrieved:]

[28] , , , [Retrieved:]

[29] , , , [Retrieved:]

[30] , , , [Retrieved:]

[31] , , , [Retrieved:]

[32] , , , [Retrieved:]

[33] , , , [Retrieved:]

[34] Donald Ervin Knuth: *Deformation modelling tracking animation and applications*, Berlin, Heidelberg, Springer, 2001.

[35] Christopher Manning, Prabhakar Raghavan, Hinrich Schütze: *Introduction to Information Retrieval*, New York, USA, Cambridge University Press, 2008.

tools: - draw.io - plantuml.com - chat.openai.com - stackoverflow.com - google scholar