NBA Salary Predictor: Software Developer

Pellizzaro Federico

University of Trieste

Software Developer

FEDERICO.PELLIZZARO@studenti.units.it

1. PROBLEM STATEMENT

Deciding the annual salary of a professional player has become, throughout the years, a very important aspect in the world of sports. Nowadays, lots of people are involved in this process since the annual salary of an athlete has become a very expensive thing. Very often, there are teams composed by data analysts that look and study the data of a player and then make up a possible price for his annual salary. This entire process most of the times can require time and lots money, because people must be paid to do this work. In this project our goal is to predict a salary of a player's contract based on given data from the user that contains the basic statistics of a player plus his name, surname and the salary asked from the player. The data used to help the application predicting the salary are data from the contracts stipulated in the NBA in the last 10 years.

2. BACKGROUND

To understand the content of this project, first it is necessary to mention some concepts that have been used throughout the development. This project, for the most parts, has been developed with the Programming Language **Python** and some key elements that have been used to develop the project are:

- **Python classes**: that provides a way of wrapping data and functions together
- **Code Refactoring (good practice)**: which means rewriting code for clarity, not bug fixing
- **Code Commenting (good practice)**: which means explaining what the code does in the files using the comment format
- **Inheritance concept**: which allows to define a class that inherits all the methods, properties and functions of another class
- **Modules**: which are collections of functions, classes, variables, constants usable in the program
- **Libraries/Packages**: which are collections of modules

For developing the real the web application, it has been used an **Integrated Development Environment** (IDE), which is a software application that provides comprehensive facilities to computer programmers for software development. At the beginning the software has been tested locally and then it has been transferred in a **Web Hosting Service** which is a type of Internet hosting service that hosts websites for clients. Since Python was the main programming language used, in order to manage libraries, packages and application deployment, a Python Distribution has been used. In addition to that, to keep track of the changes in files, a **Version Control System** (VSC) has been used.

3. METHODOLOGY

3.1 Technologies Used

Most of the code was written in Python, but thanks to the Streamlit Library, also HTML and CSS languages were used to incorporate more elements and styles in the web application. To develop the application these technologies were used:

- **IDE**: Spyder
- **Python Distribution**: Anaconda
- **VCS**: GitHub
- **Web Hosting Service**: Streamlit Cloud.

3.2 Problem Solving Approach

For this type of project, as Software Developer, the hardest part was to structure the entire system in separated blocks. These blocks (python classes) then would have been connected and/or used together to create the main Components of the web application. Two approaches have been attempted:

- **No classes**: in this was the code was very quick to write and tests were fast, since I could immediately see components and how they were behaving. But this approach would have not been sustainable for long period of time
- **Classes**: by using classes there would have been a much cleaner and reusable code inside of python files and due to that, a more solid code structure would have been created (to keep code/functions in the right place)

At the beginning, the first approach was used and then thanks to the code refactor the second approach was implemented. By structuring the web application with classes, it has been decided to create a *Container* class (the super class) and three subclasses:

- **MainPageContainer**: containing the core items of the web application (class used in the *MainPage.py* file)
- **ContactContainer**: containing the contacts of people that worked on this project (class used in the *Contacts.py* file)
- **DescriptionContainer**: used to describe what the project was (class used in the *Description.py* file)

After the creation of these four classes, other classes have been created to fill the content of the containers (more on the Software Organization section). It is important to note that each class created is a component in the UML component Diagram.

3.3 Software Organization

3.3.1 UML Component Diagram

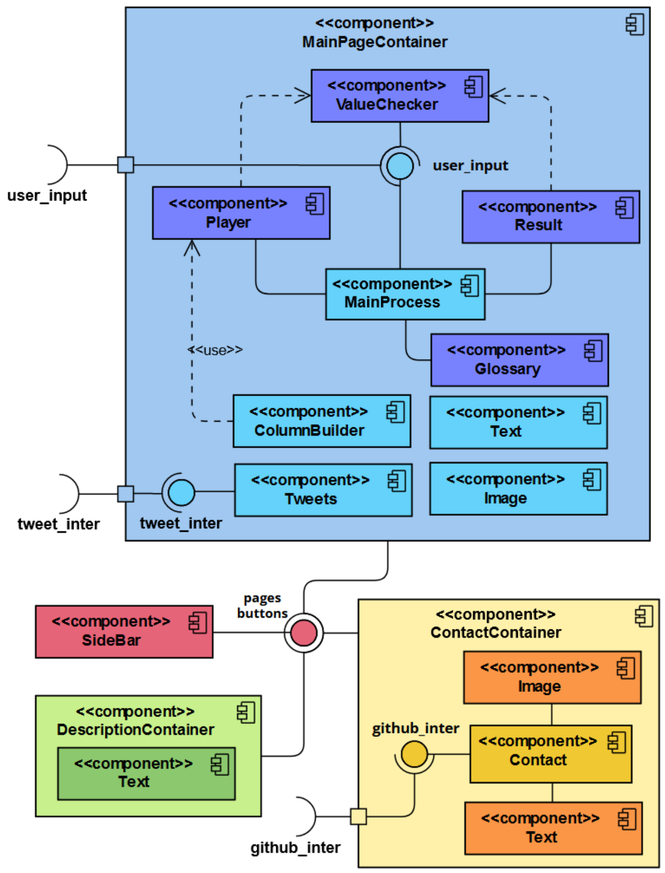

Figure 1. UML Component Diagram

This web application is composed by 4 mayor components:

MainPageContainer Component

This is the component in which the user should spend most of its time because inside of this container there is the core of this web application. This component uses **1) MainProcess Component** that includes the main entities that define the purpose of this project. The component itself provides an **interface** (form) for the user to input data (like Name, Surname, Salary asked, statistics of the Player) that will be used to predict a salary of a player. After inserting the data and clicking the "*Submit*" button the component *ValuesChecker* will go in action, checking if the data in input are correct and valid. If there are no errors in the input data, those data will be put in a list and passed as parameter in a **Machine Learning function** which, at the end, will give us the predicted Salary of a Player (saved in the *Result* component). The result of the executed function and some other variables (Name, Surname, Salary asked of the Player) then will be saved in the *Player* component. It is important to note that the Player is also saved in a list inside the **session_state** with which we will be able to create another component *ColumnBuilder*, used to create a table in the MainPageContainer, which shows the recent searches of the user. The *MainProcess* component uses also the *Glossary* component which helps the user to understand the meaning of the values required for the input data. **2) ColumnBuilder Component** which is used to create table's columns in which there will be shown the recent searches of the user. **3) Tweets Component** which updates the user about some recent news about NBA or Player's Statistics. The component also offers an **interface** to interact with tweets. And finally, the last components, **4) Image and Text Components** which are used in order to provide some styling for the web page (like custom titles or custom images).

ContactContainer Component

The purpose of this component is to acknowledge the people who have worked for this project. This component uses the **Contact Component** to store the actual contacts we want to display. Each Contact uses the **Text** (which can include links to other pages) and **Image Components** to display the information of a person of the Team.

DescriptionContainer Component

In this component there is a brief description of what the project is and how it has been developed. As shown in the diagram, this component uses only the **Text Component**.

SideBar Component

Even though it's not an actual real component inside of the web application, the Streamlit Library provides a way to create a multi-page application without writing python code to manage pages [1]. By using this technique, in the web application, it will be displayed a sidebar with buttons, therefore an **interface**, to navigate from one page to another.

3.3.2 UML Class Diagram

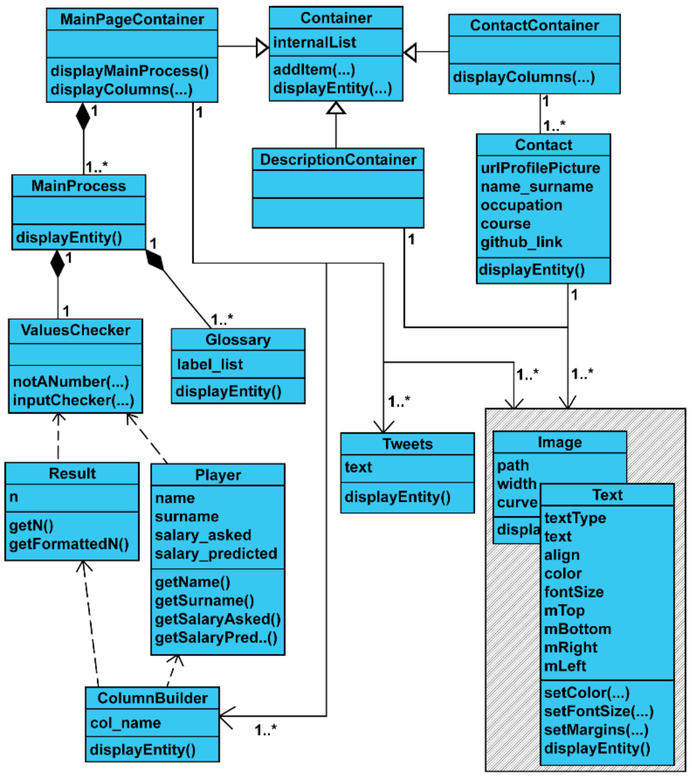

Figure 2. UML Class Diagram

For this project classes have been implemented, as said in the sections above, in order to make the Python code more readable and reusable. All these classes are stored in a separate file called *classes.py*. Each class of the diagram has one or more relationship with other classes. The classes are:

Container Class

This class is used by apply the inheritance concept between the Containers classes. This specific class is the "*super class*", the *internalList* variable is used to store objects (like Images and Texts). The function *addItem*(…) is used to append objects in the *internalList* and the function *displayEntity*() is used to actual display the objects inside of the *internalList* variable.

MainPageContainer Class

This class is a "*subclass*" of the Container class, so it inherits his variables and functions. In addition to that, in this class there are also 2 other functions. The *displayMainProcess*() function is used to display the MainProcess object (the core of the web application), the *displayColumns*(…) on the other hand is used to display columns of a table.

ContactContainer Class

This class is a "*subclass*" of the Container class, with the inherited functions and variables it also has another function, *displayColumns*(…) used to display columns of a table.

DescriptionContainer Class

This class is still a "*subclass*" of the Container class, but it does not have personal variables or functions. This class was created to clear the program structure.

It is important to note that both MainPageContainer and ContactContainer classes implements the same function *displayColumns*(…) but the DescriptionContainer class doesn't. That's why I decided not to include that function inside of the "*super class*".

MainProcess Class

This is the class in which all the core functions and tasks are executed. Inside of this class there are these elements:

- **User Input Interface**: interface displayed for the user in order to input data (form)
- **Glossary**: that helps for the user to understand what the input data are
- **ValuesChecker**: used to check that all the input data are correct and valid
- **Result**: to store the results after the execution of the Machine Learning function
- **Player**: used to store important data and results of the entire process in order to use those data in other sections of the web application

In order to display all the things mentioned above this class provides a function *displayEntity*() used to display the entire core process.

ValuesChecker Class

Class created to check the input data given from the user. This class contains no variables but there are two functions. The *notANumber*(…) checks if an input data is a number or something else, and the *inputChecker*(…) function checks if the entire list of input data is valid or not.

Glossary Class

This class takes care of the glossary section, it has one variable "*label_list*" that contains the labels of each item in the glossary and a function *displayEntity*() used to display the entity in the web application.

Result class

The Result class is used to store a number "*n*". The function *getFormattedN*() provides a different way to display the number (example: instead of 3712834 it is 3.712.824 $), and the function *getN*() simply return the number "*n*". The Result class is used if the function *inputChecker*() returns True.

Player class

The Player class was created to store valuable information about the player's contract salaries and to be able to create the "Recent Searches" section inside the web application. For each player searched, Name, Surname, Salary asked, and Salary predicted are saved in the Player object, this object then is saved in a list stored in the **session_state** and with that list the program will be able to build the section "Recent Searches". The functions of this class are "getters" to get each variable of the class. Like the Result class, this class is used if the function *inputChecker*() returns True.

Tweets class

The Tweets class provides a custom component with which, given a URL of a twitter page ("text"), the user can see the most recent tweets about the given twitter page. The *displayEntity*() function used to display the entity in the web application.

Text class

The Text class is used to show text in a HTML format. Parameters can be set, to make the text look different, thanks to the "setters" functions.

Image class

The Image class, given a path of an image, is used to display images. The image border can be curved with the parameter "curve".

Both *Text* and *Image* class have the *displayEntity*() function which is used to display the entity in the web application.

Contact class

The Contact class is used to store the information of a contact.

3.4 Software Functionalities

3.4.1 UML Activity Diagram

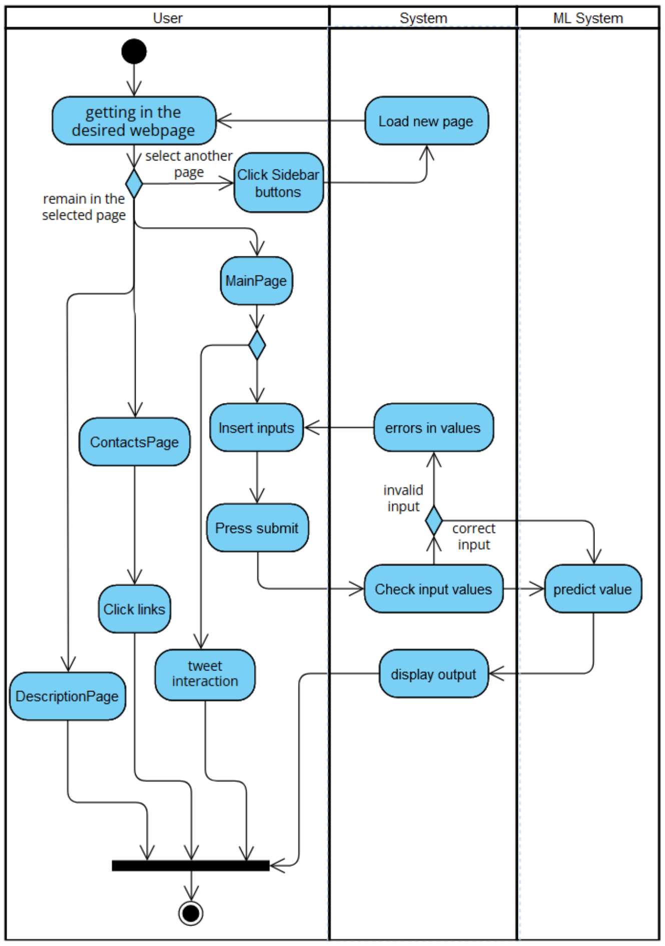

Figure 3. UML Activity Diagram

In the UML Activity Diagram there are shown the possible activities that a user can do inside of each of the web pages. Everything starts from getting inside of the desired webpage with the Sidebar buttons, then we can have 3 different possibilities:

MainPage Activities

For this webpage the user can decide to *input values* to make get a Predicted Salary or *interact with the tweet's* interfaces. In the first case, the user will insert some data and then press the "*Submit*" button. The values will be checked by the **System** to see if there are any incorrect values and, if they are present, the webpage will show an error message and will ask the user to insert the values again. If the values are valid the **Machine Learning System** will make a prediction of the salary and the **System** will display the result in the webpage. For the second case, the user can interact with the tweets scrolling them or going in the twitter webpage.

ContactPage Activities

In this webpage the user can click some links present in the contacts displayed on the webpage.

Description Activities

In the DescriptionPage there are no activities that a user can do.

3.4.2 UML Use-Case Diagram

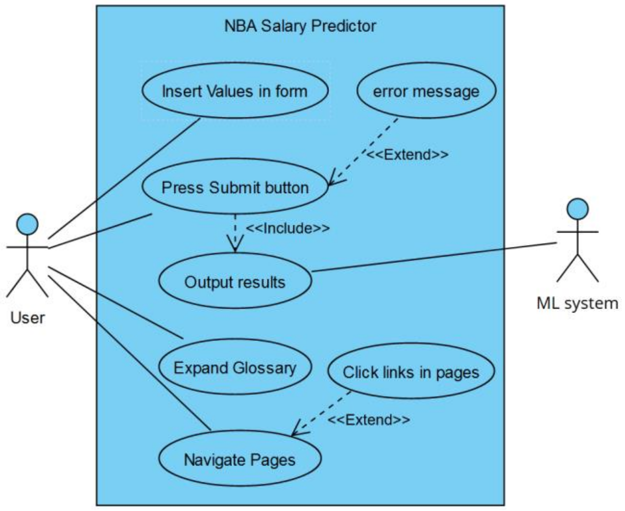

Figure 4. UML Use-Case Diagram

In this web application there are two actors involved, the **User** who is the entity/person that is going to use the web application, and the **Machine Learning System** which is the system with which the web application gives the user a result (the Predicted Salary of a Player).

As shown in the diagram the user can do various things like:

- Inserting values in the form. Values that will be used to predict the Salary of a Player
- Pressing the "*Submit*" button to activate the Machine Learning function which will return a predicted value
- Expand the *Glossary* component to understand the values that the user must input
- Navigating through pages with the *SideBar* component.

The use-case action of "*pressing the submit button*" **includes** the "*output results*" use-case action, because without pressing the "*Submit*" button the user wouldn't see the output. In addition to that, it **extends** the "*error message*" use-case action since if the user input is incorrect an error will be returned. The "*navigate pages*" use-case action **extends** the "*click links in pages*" since by navigating in certain webpages the user can click links inside of them. On the other hand, the Machine Learning System can:

- Predict the output result after the user has pressed the "Submit" button.

4. RESULTS

The outcome of this whole project ended up being an online functioning web application. The *use of class* approach resulted in a **more effective** way to organize the project even tough it required more lines of code to work. The other approach would have been **more efficient** (less code) but **less effective** (messy code) and **less sustainable**. Also, code editing would have been much harder since the code would have been all scattered, with a not cohesive structure. The use of both Python, HTML and CSS also led up to a more *diverse* web application, both integrating Python elements and HTML/CSS structures.

5. DISCUSSION AND CONCLUSION

Overall, all the components added in the web application are working correctly, so in a general prospective this entire project was a success. This project could be potentially used by some user, getting profit from it. Even tough the project is successful, because it works, there have been some issues during the development.

5.1 Issues discovered

Issues were discovered when the web application had been transferred from the Local Machine to the Web Hosting Service. One issue concerned the local paths used in the code, which had to be changed to other paths/URL to get the wanted resources.

Another issue concerned libraries used in the Python files, some of them were not installed yet in the Streamlit Cloud Environment. To fix this problem I had to make a custom file "*requirements.txt*" in which I had to include what were the packages that should have been installed in the environment (example: scikit-learn==1.0.2).

Another issue had to do with the tweet user interaction. Until 20/1/2023 the tweet component was working well, and a user could interact with it. But after that day the component stopped working, giving an error in the browser console. After asking in the Streamlit forums if the problem was related to them (it was not), I went on the Twitter Developer forums in which, me and other users (which had the same problem) asked why and what was the problem behind this thing. At the end after going back and forth with tests and tries, the Twitter Developer that was following this problem fixed it and the component started working again.

6. REFERENCES

[1] https://docs.streamlit.io/library/get-started/multipage-apps/create-a-multipage-app