Movie recommendation App: Software Developer

Ludovica Caiola,
University of Trieste, ITALY
ludovica.caiola@studenti.units.it

1. INTRODUCTION & PROBLEM STATEMENT

Choosing a movie in the evening could be a waste of time if people have a limited one. It could happen that the platform offering a movie streaming service doesn't do a specific recommendation. Even though there is the possibility of choosing between film and series and selected it depending on their genre, people have other limits that may influence their choice. To resolve this, a software was created in order to help people not to waste their time for the selection of the movie.

The aim of this project was to build a software based on a machine learning system. The general idea was an interactive web application in which users provide some inputs and the application will suggest the ten most recommended movies, based on their preferences.

All of this could lead to an optimization of time when it occurs to choose a movie and it may also suggest the streaming platform to provide some tool of movie recommendation like this one.

2. BACKGROUND

The development of the software followed an incremental approach. This approach gave the possibility to produce an initial sketch of the structure and then implemented it. The good practices of python were taken into consideration. So, the code was refactored, the variables' names were as short and comprehensible as possible and they respected the common rules, depending on their type. Moreover, comments were added to clarify the following code.

In addition, three (out of four) principles of good software were considered most:

- maintainability: the software's structure had easily faced evolutions and changes over the time;
- efficiency: the system's responses were quite rapid, and the structure of the code presented if-then-else and for-loop structures that were easily manageable;
- acceptability: since it was a software for both children and adult, it's easy to understand what the system requested from the user.

The software was entirely programmed on an Integrated Development Environment (IDE) called Spyder [1].

This software was entirely related to Streamlit [2], a free and open-source framework, that is a platform that helps to create web application by providing customized components and lots of other tools. It was then uploaded on a code hosting platform for version control, called Github [3]. By creating a repository on Github, it was possible to use Streamlit Cloud, an open and free platform where developers can upload and deploy their application. In the end, the software was schematized using the Unified Modeling Language (UML) [4].

3. METHODOLOGY

It's important to notice that in order to generate predictions, it was used a machine learning system. Here it's not analyzed, but more details are shown in the respective files, presented in the Github repository[1].

3.1 TOOLS

The application followed exactly the path described in the previous section, in the interest of having a functional application and not just a local Streamlit page. The code was created on *Spyder*, then it was uploaded on *Github*. At this point an application[2] was deployed using the *Streamlit Cloud* service.

The programming languages used for the project were both python for the main structure and CSS/HTML for styling and modifying the existing graphical components already provided by *Streamlit*.

The UML diagrams were made with a software called *Visual Paradigm* [5].

3.2 SOFTWARE

This section aims to clarify all the software development processes that lead to the final version of the software itself.

At first, the main requirements of the software were outlined, then it was created a deadline for all the things to do.

The software should provide:

- an **introduction page**, which explained the aim of the application and what the user has to do;
- an **input's user page**, where the user interacted with the application and expressed his/her preferences;
- an **output page**, that displayed recommendations based on the inputs that the user gave in the previous page.

The first structure of the software was intended to provide 3 pages, just like the description above. But after a first attempt, it was necessary to split the program in more pages than that. This option improved the principle of maintainability in the application because the split, in more than three pages, provided a greater ability to adapt the software to change.

These changes were also realized for the type of inputs. The user had to declare if he/she was an adult or a child. This choice influenced the outcome page:

- if the user was an adult, the following page required 3 inputs: mood, movie and time;
- if the user was a child, the following page required just 2 inputs: movie and time.

The following and final design consisted in 6 pages. The first page contained a brief description of the web application. Then, by clicking on the button, the user moved forward the page in which

[1] https://github.com/ldovu/Movie-recommendation

[2] https://ldovu-movie-recommendation-page0-awd4vf.streamlit.app/

he/she selected between two options: if the user is a child or an adult. This choice determined the resulting following page where the adult user can choose: the emotion that the movie raised, the title of the movie similar to the one the user wanted to watch (as an optional input) and the time available to the user (if infinite or limited, and in the last case the user specifies the minutes). The child user is redirected to an input page that doesn't provide the mood selection but do provide the two other inputs.

In the final page the ten recommended movies were displayed, and the user can select which overview to read among the movies selected by the ML system.

For realizing all of this, the links of the pages were necessary. So, the first page was uploaded on Streamlit Cloud and by then all the pages were uploaded and so the links were created.

After refactoring the code, the web application was personalized with the use of HTML and CSS: the size and color of headers were modified. The code concerning the buttons were also written in these languages and contained the links for the redirection to the other pages. The user is redirected to the following page by clicking to the button if, given that the user has inserted the movie title, this input is correctly written. Otherwise, the button is put in a disabled condition and the suggestions of a correct title are displayed. The user at this point can copy and paste the suggestion or provide a new one.

3.3 UML Diagrams

Before starting to code and during the development of the application, the diagrams were created in order to give a clearer idea of the structure and behaviour of the application itself. Here diagrams are reported.

3.3.1 Structure diagrams

3.3.1.1 Class diagram

The creation of classes was misleading instead of a linear path: writing a simple and clearer code. In fact, the nature of the project didn't directly suggest an object-oriented structure: the web application needed to collect some inputs in order to generate an output. For these reasons the code didn't contain classes and so the class diagram is not provided.

3.3.1.2 Component diagram

The component diagram is, together with class diagram, a type of structure diagram, so it describes how the software is divided into different components and how these components interact with each other. The diagram was structured regarding the pages: each big colored component represented a **page** of the web application. Instead, the components inside each page (with the same, but darker color) represented other elements like:

- a function inside the page,
- a Streamlit component (in the case of Person, Mood, MovieTarget and Time),
- type of elements such as a list or a text (in the case of recommended and overview).

One page was related to the other like this: the page that was connected to a colored circle was a "Provided interface", instead the one connected with an arc was a "Required interface".

The port was used when an internal component provided an interface for the required input of the external and general component.

The realization of the diagram was done in this way: every page, having a button, provides an interface and is connected to a required interface, for the fact that the user, by just clicking to the button, is redirected to a page that is indirectly required.

Starting from the beginning, "page0" provided an interface and by just clicking at the button "start", there is a redirection to the "chooseChildAdult" page. This contained the component "Person" that provides an interface for the fact that let the users insert their choice. After selecting the age and clicking to the button, the user is redirected to the "inputAdult" or "inputChild" page, according to the choice previously selected. Analyzing the case of the adult page, the three inputs were, in turn, components and they provided an interface to the inputs that the user is required to insert. These are involved in two types of relationship: the dashed line with empty arrow indicated a "Uses a" relationship which means that a component depended on other components; the dashed line with full colored arrow indicates an "Has a" relationship which means that a component contained a reference to another function in other files.

The pseudo-code clarifies the diagram: whenever "movieTarget" is inserted, the title is checked. The system verifies if the input is correctly typed, if not, it recalls a function "correctTitle" contained inside another file, called DataInspection. So, the function checks if the user had correctly typed the movie target and if not, it provides three suggestions of movies, as close as possible to the one that was wrongly typed.

```
function checkInput(movieTarget) {
    if isCorrect(movieTarget) then {
        return movieTarget
    } else {
        corrections ← DataInspection.correctTitle(movieTarget)
        return corrections
    }
}
```

Figure 1. Pseudo-code

Furthermore, the "goToPage" function had a reference to all the inputs given. Note that the same processes happened inside the "inputChild" component, for this reason it's not described again. Once more, the button inside the input pages provided an interface, so the output pages were the respective required interface. Inside these components there were two other main components: "recommended" and "overview". They were both involved in a "Uses a" relationship with other function, contained in other python's files. Recommendations and overviews were generated from the relationship they were involved into the other component and that justified the direction of the arrows.

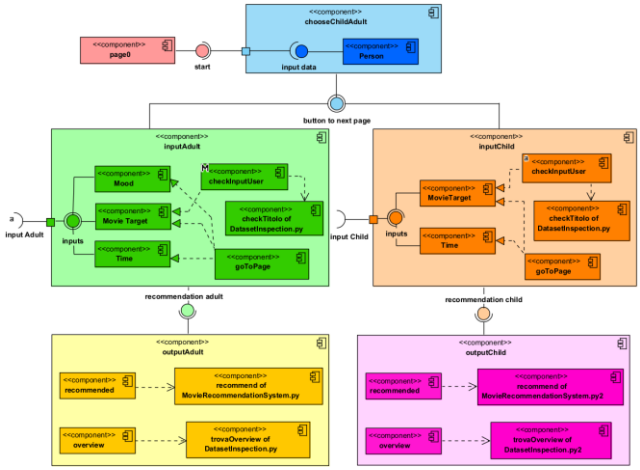

Figure 2. Component diagram

3.3.2 Behaviour diagram

3.3.2.1 Activity diagram.

The activity diagram is a type of behaviour diagram that describes the flow charts of the application. The diagram was divided in three columns: the first one represented the user's choices and actions, the second one provided the responses of the system to the user inputs and the last one represented the contribution of the machine learning system.

The user can start the session by clicking to the link. The application drives the user to select the child/adult option. If the user is an adult, the user can start with the selection of the mood, otherwise the user can begin with the time. Then, he/she can optionally write the movie, so if the title is not provided, the user can directly enter the previous data. Instead, if the user provides a title the system checks it: if it is correctly typed the user can enters the data, otherwise the system displays the error and suggests three correct titles, at this point the user can either correct it or not provide it. After entering the data, the machine learning system outputs the recommended movies and the user has the possibility to read the overview of the recommended movies. At this point, the session is finished.

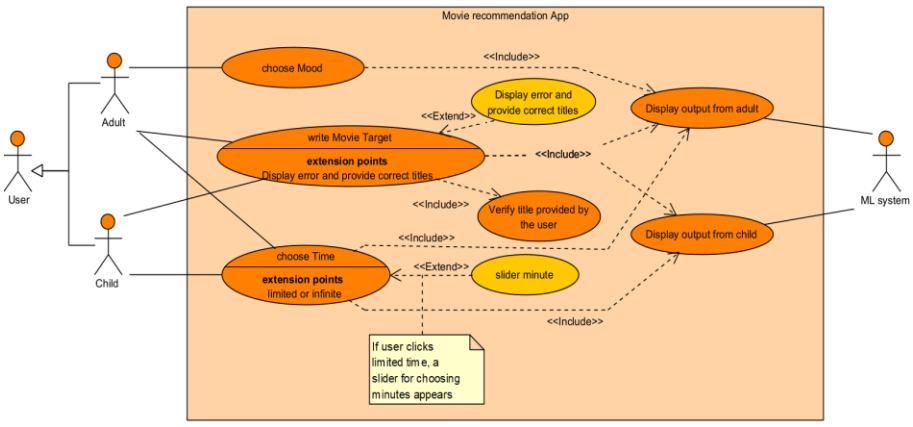

Figure 3. Use-Case diagram

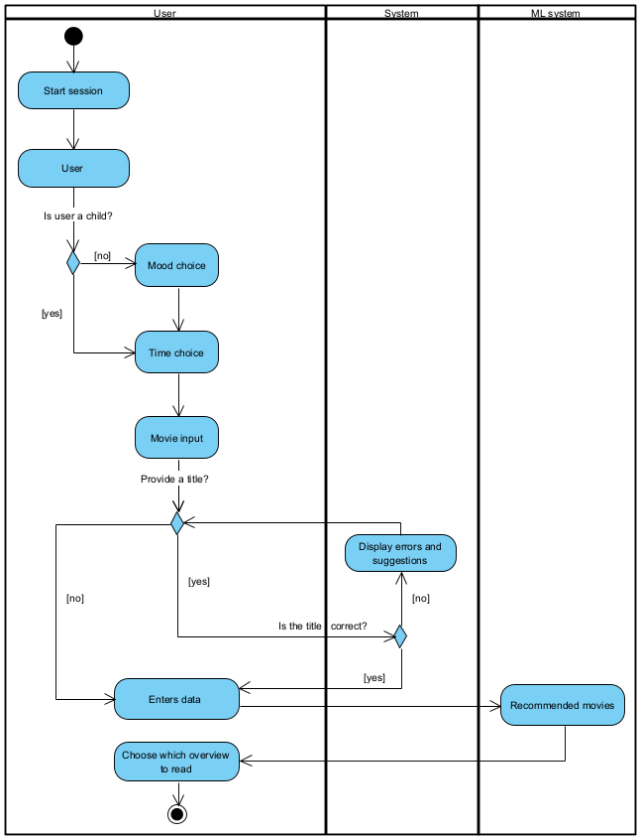

Figure 4. Activity diagram

3.3.2.2 Use-case diagram.

The use-case diagram is the second behaviour diagram analyzed in this report. It described the actions that a system can implement and the actors that can perform them. The actions were represented by use-cases and were incapsulated inside the big box, instead the actors were drawn outside the overall system.

In the use-case diagram the primary actor was the user of the application. The user can be though a generalization of two type of user: an adult user and a child user. They had two actions in common out of three, the adult in addition can choose the mood. All the inputs were involved in an <<Include>> relationship with the output. The "write Movie Target" use-case had an additional <<Include>> relationship that suggested that every time the user write the name of the movie (base use-case) the title is verified (included use-case). Instead, the <<Extend>> relationship showed that the error and suggestions of correct titles (extended use-case) are not displayed every time the user writes the movie target (base use-case), but only if the input was wrongly typed. For the "choose Time" use-case, if the option "limited" was selected then it is displayed a Streamlit slider for selecting the exact minutes the user had, as the additional notes describes. After collecting all the inputs, the output is displayed as a result of the machine learning system, considered as a secondary actor in the overall system.

4. RESULTS

The overall planning and techniques explained later in the document were successful. The incremental approach was the best solution adopted: all the processes were initially sketched and later implemented. The refactoring permitted to give a smoother and clearer structure to the code, so it can be easily read. Even though the code is repeated, the split of the code gave more flexibility and possibility to add new features in the future. This choice also made the UML diagrams better structured.

As a result, a functional application was obtained. The application can be used all over the world and may also satisfy the needs of some people who intend to watch a movie, having at the same time some limitations on time and on the type of film they want to watch.

5. DISCUSSION & CONCLUSION

Along the project, it happened that the upload of pages on Streamlit cloud gave some problems. This was solved by simply delete the existing page on Streamlit cloud and by uploading a new one from the same file. The error might have been caused by a temporary disconnection to the server.

Even though this problem occurred, there were no other problems, and the application was successfully launched, as explained in the previous section.

6. REFERENCES

[1] Raybaut, P. 2009. *Home — Spyder IDE*. DOI= https://www.spyder-ide.org.

[2] Treuille, A., Kelly, A. and Teixeira, T. 2018. *Streamlit • The fastest way to build and share data apps*. San Francisco, CA. DOI= https://streamlit.io/.

[3] Preston-Werner, T., Wanstrath, C., Hyett, P.J., Chacon, S. 2008. *GitHub: Let's build from here · GitHub.* San Francisco, CA. DOI= https://github.com/.

[4] Booch, G., Rumbaugh, J. and Jacobson, I., 1996. *Unified Modeling Language (UML).* DOI= https://www.uml-diagrams.org/.

[5] Visual Paradigm International Ltd. 2020. *Visual Paradigm.* DOI= https://www.visual-paradigm.com/.