**DOCUMETATION**

PROJECT’S TITLE: SYSTEMY KOLEJKOWE

SUBJECT: PODSTAWY INFORMATYKI I PROGRAMOWANIA

Content:

1. Project assumptions
2. Implementation description
3. Project work summary
4. Imported libraries

PROJECT ASSUMPTIONS

The aim of this project was to neatly present data from the site (link: [https://api.um.warszawa.pl/#](https://api.um.warszawa.pl/)) in a consistent way. In order to achieve that the user interface, which allows User to trace the current state of the queue in chosen institutions, was created.

IMPLEMETATION DESCRIPTION

Program consists of five files. Each of the file is responsible for separate action while program is running. Such a division is clear to read and allows to find errors in a small period of time.

Program should be started from *gui.py* file. It creates a user interface, on which Users choose an institution and right after that by pressing the button “**Zatwierdź**”; the program switches to *queue\_system.py* file and sends requests to API to get a list of all the official matters. Then returns to running *gui.py* file and waits for User to choose a matter. After User’s decision on choosing the matter the program switches to *queue\_system.py* file in order to get information about current queue. After selecting a case, the user can display a plot showing the number of people waiting in the queue, monitored on the interface of the case. If the user decides to display it, the program goes to the *queue\_plot.py* file and calls the plot creation method. This operation is being passed again if the User’s choice is being changed.

In the *authorities\_dict.py* file has a dictionary located, which every institution has assigned an id in. With id added to base URL address the program is receiving data through API.

File *queue\_system.py* contains a *Queue* class, which has methods such as: *get\_id(),* *list\_of\_institutions()*, *actual\_queue()*. By calling these methods program is able to formulate accurate URL address in order to get a request from API. Moreover, *Queue* class contains methods, which saves data that will be printed on a graphic user interface in the form of a list. This file imports *requests* library and dictionary from *authorities\_dict.py*.

File *queue\_plot.py* contains a *QueuePlot* class, which prints out a plot demonstrating a current number of people in a queue for a chosen case in chosen institutions. This class possesses a couple of attributes, for instance: *institution*, *case* and *number* that are uploaded from the user’s interface, as well as *number\_of\_people* and *x\_data* that are list types and store data to make a plot. Every call of a *get\_chosen\_institution* method causes an action, which appends to attribute *number\_of\_people* a current number of people in a queue and to attribute *x\_data* current time, checking weather parameters of a *get\_chosen\_institution* function remain unchanged. Otherwise attributes *number\_of\_people* and *x\_data* are set clear. *QueuePlot* class contains also a method that allows a plot to be shown and labeled. This file imports *matplotlib.pyplot* and *datetime* library.

File *gui.py* is responsible for creating a graphic user’s interface. It contains an *Application* class, which inherits from *tkinter* *Frame* class. This file connects other python files, hence has multiple methods, for instance: *create\_widgets* method, which creates and designs user’s interface. What is more, the *Application* class has some events’ methods, which allows widgets to interact with others. This file imports dictionary from *aututhorities\_dict.py*, *Queue* class from *queue\_system.py* and *QueuePlot* class from *queue\_plot.py*.

The last file is *test\_queue­\_system.py*. It contains unit tests of several functions from *queue\_system.py*, *queue\_plot.py*. It imports *tkinter* library as well as *authorities* from *authorities\_dict.py*, *Queue* from *queue\_system.py*, *QueuePlot* from *queue\_plot.py* and *Application* from *gui.py*.

PROJECT WORK SUMMARY

While creating the code matching project’s requirements I faced up some difficulties, at first the lack of a list gathering all institutions’ id-s. To solve this problem I created a nested dictionary in an *authorities\_dict.py* file. Such a solution allows to connect the API base for certain data.

In *queue­\_system.py* I created a class; which methods allow selecting data needed to run a program data from API *json* dictionary.

I realized that some of the institutions do not have any data, their API is empty. This problem was solved with a help of a construction *try: … catch TypeError: …*, which prints out “**Brak danych**” in entry label whenever User chooses that type of an institution.

I have chosen a graphic user’s interface to present data and serve the User, as it is intuitive to use. After a couple of attempts of placing all widgets in right place with certain comments: *pack*, *grid* or *place*. I decided to use a *grid* to place all widgets. Then I combined some widgets to others to make my user interface more active and more user-friendly.

After designing a graphic user’s interface I had a difficulty in updating data written on it. It turned out that library, which creates user’s interface, has a built in function *after()* that may solve my problem. Function *after* from *tkinter* library located in a certain function can repeat certain function every amount of miliseconds. Right after the usage of a function *after()* in a right place, my program started updating data printed on interface.

What I did not accomplished is showing a plot that allows a User to track a current amount of people still waiting in a queue. By pressing the "**Pokaż wykres**" button, the user can display a chart showing how many people are waiting for a chosen case at the moment, provided that the chart is not dynamic and does not update in real time. I cannot overcome a problem with updating it.

IMPORTED LIBRARIES

In order to run the program there should be libraries such as *requests*, *tkinter*, *datetime* and *matplotlib.pyplot* installed.