# Project Description

**Artem Romanenko**

In this project, my task was to create 2 programs, first, the (client-side) program will collect information about the system and send it to the server, and the second program (server-side) should collect the data and store it in the database.

Here I will explain the structure of my program using screenshots and plain text. Also, you can view the source code that I will attach to the report.

I will start my report with a description of the client program.

```
import psutil
import time
import multiprocessing
import socket
from pip._vendor import requests
```

In my project, I am using libraries: “psutil”, “time”, “multiprocessing”, “socket”, “requests”.

Program is organized using functions that make the program comprehensive and extendable.

```
def get_memory_info():
    mem = psutil.virtual_memory()

    def bytes_to_gb(amount):
        return round(amount / 1e+9, 2)

    data = {
        "total": str(bytes_to_gb(mem.total)),
        "inUse": str(bytes_to_gb(mem.used))
    }

    return str(data).replace("\'", "\"")
```

get_memory_info() function obtains information about the RAM using “psutil” library and forms the body for the post request that will send the data server.
Before returning the data we should replace all ‘ characters with “characters to form valid JSON.

```
def get_cpu_info():
    data = {
        "cpuNumber":
str(multiprocessing.cpu_count()),
        "usedInPercents":
str(int(psutil.cpu_percent()))
    }
    return str(data).replace("\'", "\"")
```

get_cpu_info function collects the data about CPU and CPU usage and forms the body for the post request that will send the data to the server.

```
def get_disks_info():
    par = psutil.disk_partitions()
    # getting all of the disk partitions
    disks = []
    for x in par:
        dsk = psutil.disk_usage(x.mountpoint)
        disk = {
            "name": str(x.device),
            "total": str(dsk.total),
            "used": str(dsk.used),
            "usedInPercents": str(int(dsk.percent))
        }
        disks.append(disk)
    return "{\"disks\":" + str(disks).replace("\'",
"\"") + "}"
```

get_discs_info() function using the “psutil” library to get the information about all existing disks inside the system, and store them inside the array (usually the system contains more than 1 disk).

In the return statement, I am transforming the obtained array to the valid JSON format that can be sent to the server.

```
def get_ports():
   ports = []
   lc = psutil.net_connections('inet')
   for c in lc:
       (ip, port) = c.laddr
       if ip == '0.0.0.0' or ip == '::':
           if c.type == socket.SOCK_STREAM and
c.status == psutil.CONN_LISTEN:
               proto_s = 'tcp'
           elif c.type == socket.SOCK_DGRAM:
               proto_s = 'udp'
           else:
               continue
           pid_s = str(c.pid) if c.pid else
'(unknown)'
           ports.append(port)

   return "{ \"ports\": \"" + str(ports).replace("
", "").replace("[", "").replace("]", "") + "\"}"
```

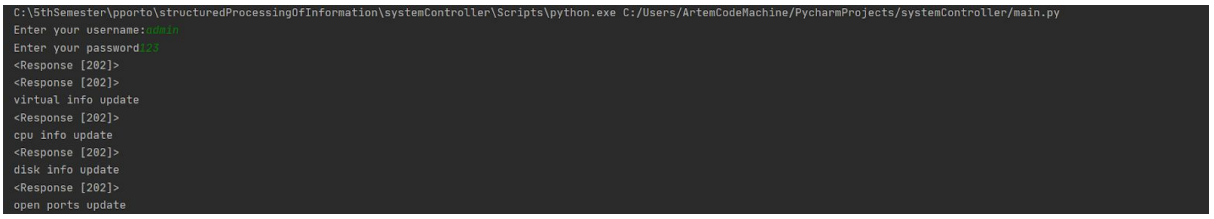
get_posrts() gets all open ports by trying to connect using socket object to each port registered in the system obtained using “psutils” library.
In the return statement, I am forming data to a valid JSON format that can be sent to the server using post request.

```
def authenticate():
   username = input('Enter your username:')
   password = input('Enter your password')
   url =
"http://localhost:8081/checkUserCredentials/"+user
name+"/"+password
   response = requests.get(url)  # check user
credentials
   print(response)
   if response.status_code == 202:
       execute()
   else:
       authenticate()
```

Authenticate function is used to start the program execution. It requires from user to input his username and password(provided by the manager), then credentials are validated on the server if a user with such username exists and the password is right then the execution of the main logic starts by calling the execute function. If the login was not successful program will ask to input credentials again by recursively calling the authentication function.

```
def execute():
       while 1 == 1:
       # update indo about virtual memory
           url =
"http://localhost:8081/setVirtualMemory/100"
           response = requests.post(url,
data=get_memory_info(), headers=headers)
  # update virtual memory info on server
           print(response)
           print("virtual info update")
           # update info about CPU
           url =
"http://localhost:8081/setCPUInfoMemory/100"
           response = requests.post(url,
data=get_cpu_info(), headers=headers)
  # update virtual memory info on server
           print(response)
           print("cpu info update")
           url =
"http://localhost:8081/setDiskInfo/100"
           response = requests.post(url,
data=get_disks_info(), headers=headers)
  # update disk information
           print(response)
           print("disk info update")
           url =
"http://localhost:8081/setOpenPorts/100"
           response = requests.post(url,
data=get_ports(), headers=headers)
  # update open ports
           print(response)
           print("open ports update")
           time.sleep(10)
```

execute function performs the main logic of the program, mostly by calling a function declared previously. It sends requests formed by functions declared above and prints status codes of the server responses (to show the user that the program is working correctly).



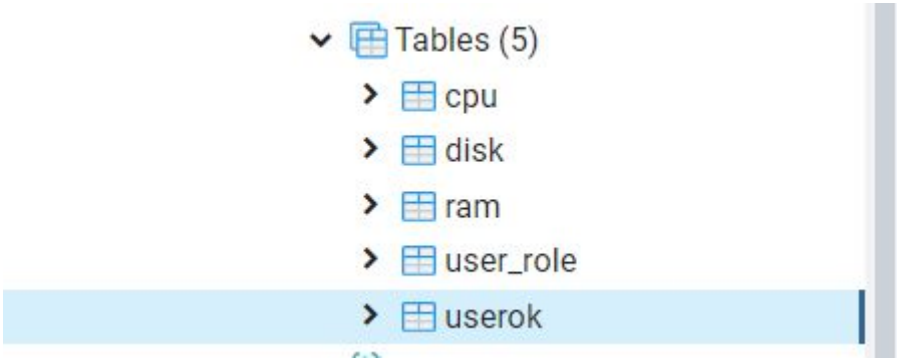

Here you can see the output of the program, all requests were successfully accepted, and the authentication function works correctly.

Next I will describe the back-end part of my app.

The back-end server is written in java language, I will skip the precise description of the server’s code as it is not a topic of the subject, but you can see the source code attached to the report.

Back-end program stores data obtained from the client to the Postgres database.

I will quickly show that all data is stored correctly in the database.




Here you can see my database structure.

| id [PK] bigint | activation_code character varying (255) | email character varying (255) | password character varying (255) | ports character varying (1000) | username character varying (255) | cpu_id bigint | ram_id bigint |
|---|---|---|---|---|---|---|---|
| 1 | 100 | fadsfasd | sdfsadf | 123 | 49667,49666,808,49666,49669,... | admin | 434 | 433 |

the user table stores all data about users (username, password, email), open ports, and references to other complex objects.

| id [PK] bigint | in_use character varying (255) | total character varying (255) |
|---|---|---|
| 1 | 433 | 6.94 | 8.47 |

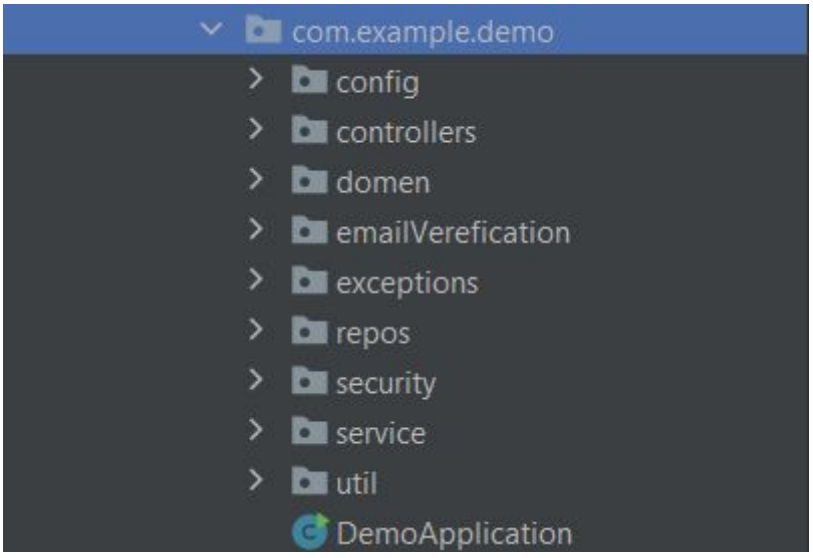
ram table stores info about the virtual memory of the user, each row corresponds to one user. Each row is referenced to the user record through foreign_key() stored in a user record (see the user table above)

| id [PK] bigint | total character varying (255) | used character varying (255) | used_in_percents integer | user_id bigint | name character varying (255) |
|---|---|---|---|---|---|
| 1 | 7443 | 255382777856 | 243783114752 | 95 | 100 | C:\ |

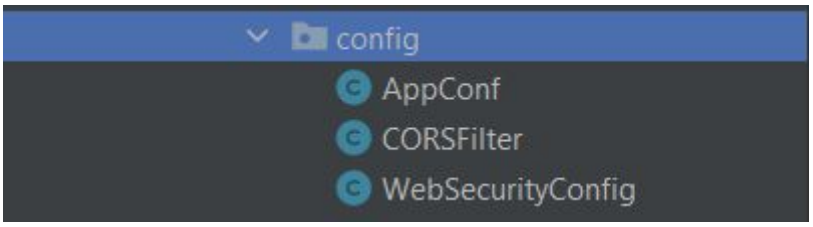
disk table stores information about disk on user's machines. (I have only one disk on my machine). Each row is referenced to the user record through foreign_key() stored in a disk record (see the picture above). One_to_many relations is used (1 user can have many disks at the same time)

| | id [PK] bigint | cpu_number integer | used_in_percents integer |
|---|---|---|---|
| 1 | 434 | 8 | 10 |
| | | | |

CPU table stores information about users' CPU. Each row is referenced to the user record through foreign_key() stored in a user record (see the user table above). One_to_one relation is used.




The program is divided into directories, a list of directories you can see above. Names of the directories are self-explain.




AppConf and CORSFilter class are used to disable CORS policy, it allows two programs (client and server) to communicate.

WebSecurityConfig sets the rules of accessing provided API.

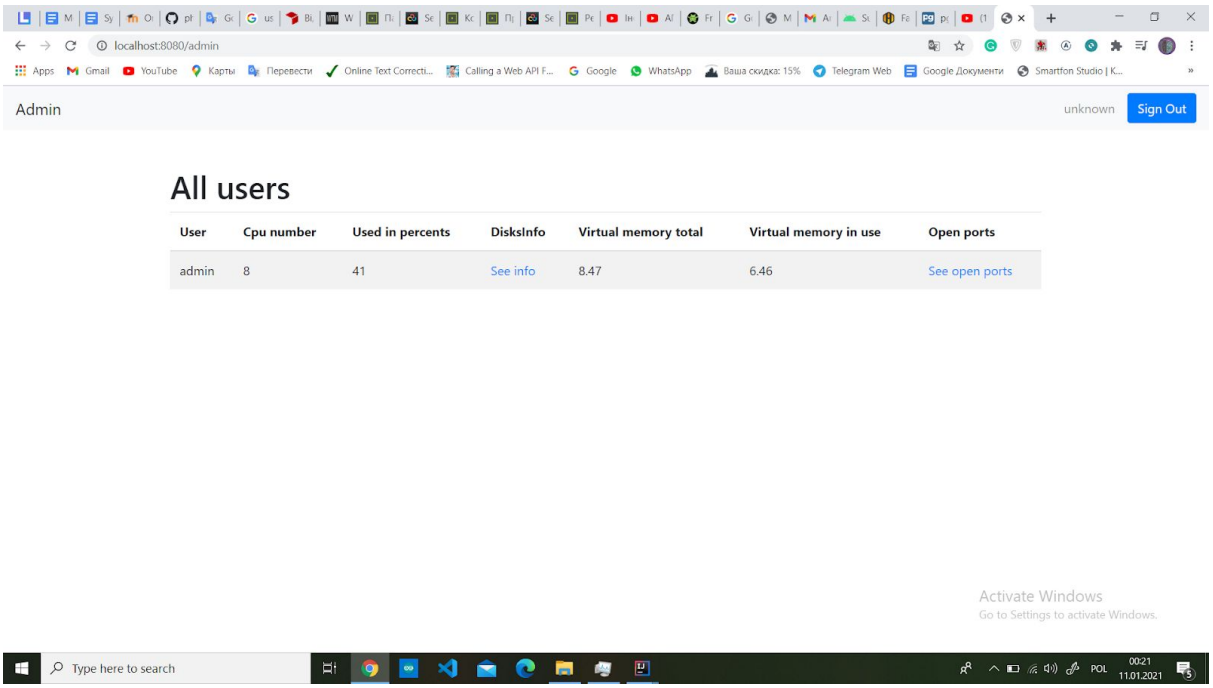
Lastly, I've decided to create a simple interface to display collected data.

To display the data I've created another web-based application that uses the same database as the previously defined back-end application(to ease the communication between them).

It is written using the combination of java and HTML/CSS files (source code will be also attached to this report).

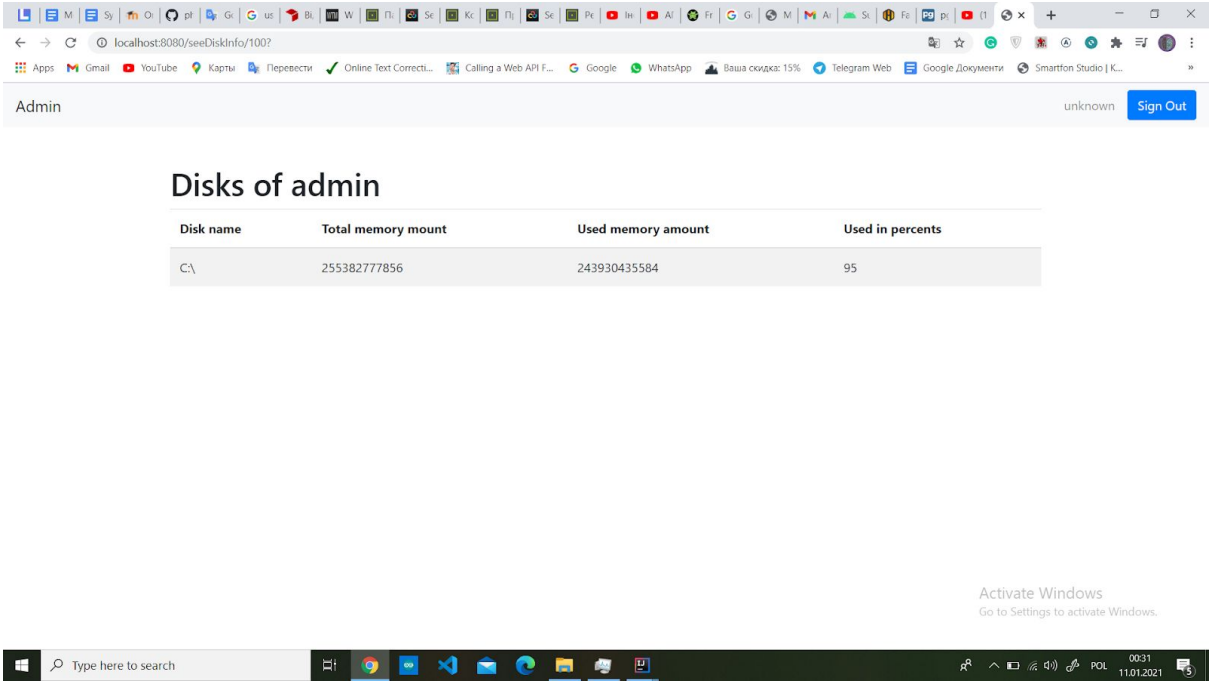
The app consists of 3 programmatically generated pages.

First shows the list of all users and info about their devices.

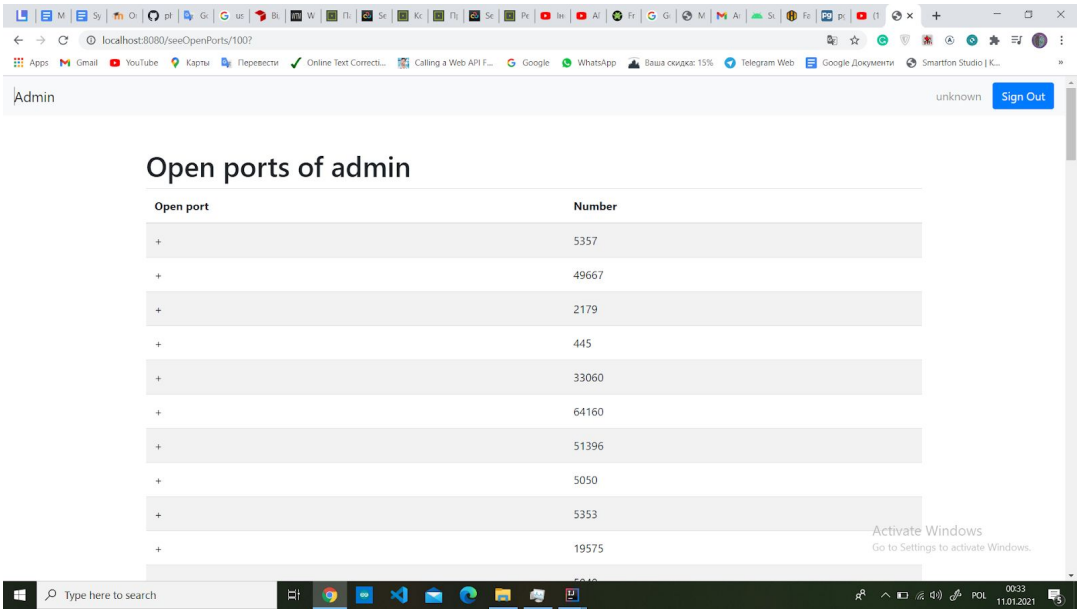



Information about disks and ports is available on separate pages because each device can have more than 1 disk and more than 1 port.

Disks page shows information about disks that are related to the specific user.



“ports” page shows all open ports of the specific users.




Thank you for reading this report.