Databases with Python

Documentation

Author: Michał Deć

Nowy Sącz, 2024

List of contents

1. Project Scope

We will code simple application in Python. This application will map car dealer. All the data will be stored in database. Application will allow for creating new clients, adding new cars, and order fulfillment.

Using Virtual Box, we will create new Linux Debian virtual machine. We will install MariaDB database server. In Python application we will write scripts to database, which will allow for registering new clients, adding new cars, and processing orders. All the registered data will be stored in database.

In Python application we will implement basic defense mechanisms. Passwords will be hashed before stored in database. Client will have to login to application before he can add new order. Only administrator will have rights to viewing all orders from database. We will also implement basic logic in application. For example, if price of car is higher than client's budget, he cannot buy that car.

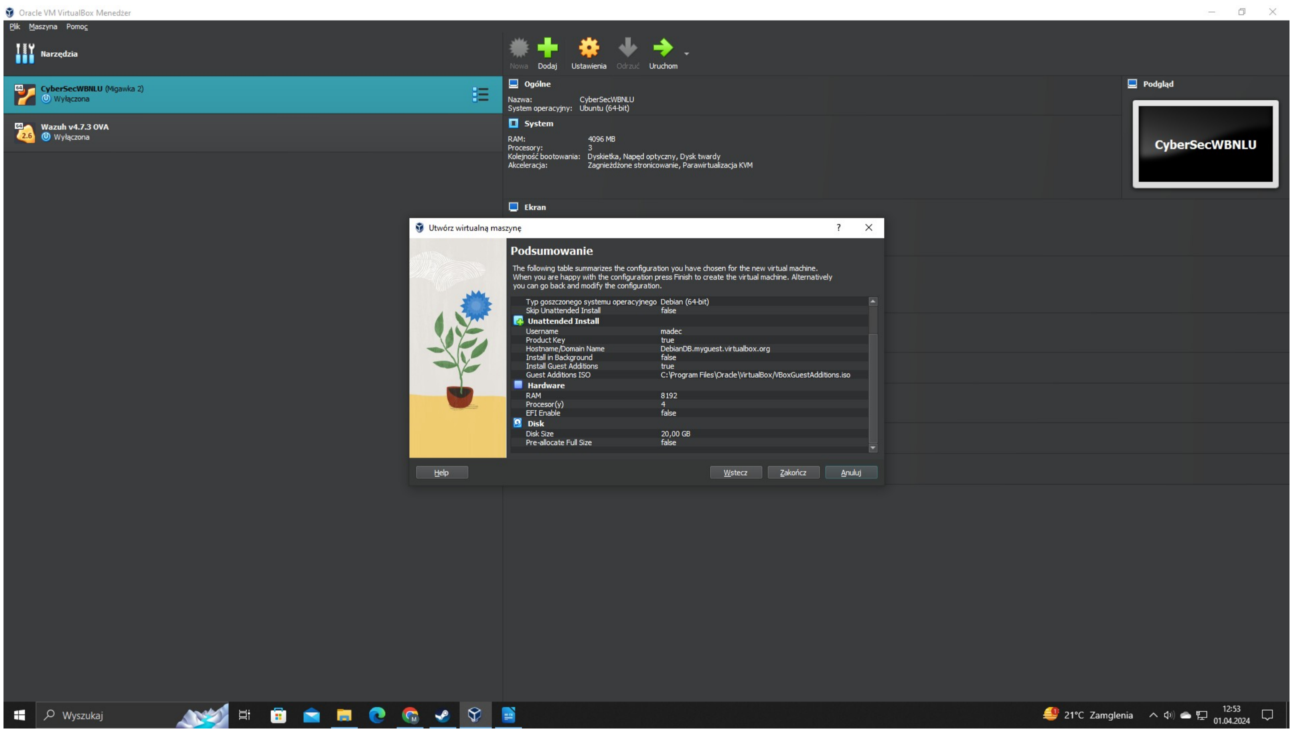
2. Installing Linux Debian Virtual Machine

2.1 Downloading image

We can downlad image of Linux Debian from site: https://www.debian.org/download

2.2 Creating new Virtual Machine

Once we downloaded image, we can start creating new virtual machine. In Virtual Box, we have to import downloaded image and create new virtual machine. We have to specify of CPU and RAM allocation.



Nazwa: CyberSecWBNLU
System operacyjny: Ubuntu (64-bit)

System
RAM: 4096 MB
Procesory: 3
Kolejność bootowania: Dyskietka, Napęd optyczny, Dysk twardy
Akceleracja: Zagnieżdżone stronicowanie, Parawirtualizacja KVM

Utwórz wirtualną maszynę

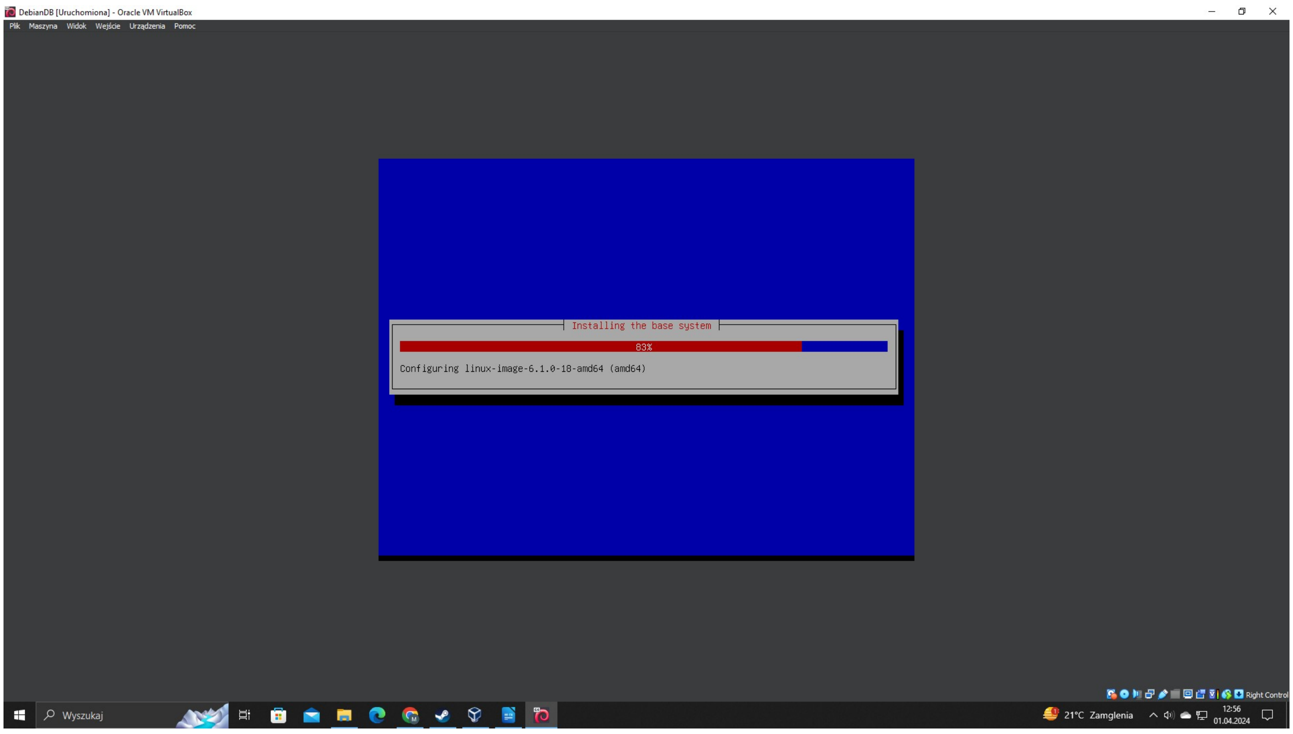
Podsumowanie

The following table summarizes the configuration you have chosen for the new virtual machine. When you are happy with the configuration press Finish to create the virtual machine. Alternatively you can go back and modify the configuration.

Typ goszczonego systemu operacyjnego	Debian (64-bit)
Skip Unattended Install	false
Unattended Install	
Username	madec
Product Key	true
Hostname/Domain Name	DebianDB.myguest.virtualbox.org
Install in Background	false
Install Guest Additions	true
Guest Additions ISO	C:\Program Files\Oracle\VirtualBox/VBoxGuestAdditions.iso
Hardware	
RAM	8192
Procesor(y)	4
EFI Enable	false
Disk	
Disk Size	20,00 GB
Pre-allocate Full Size	false

2.3 Installing Debian

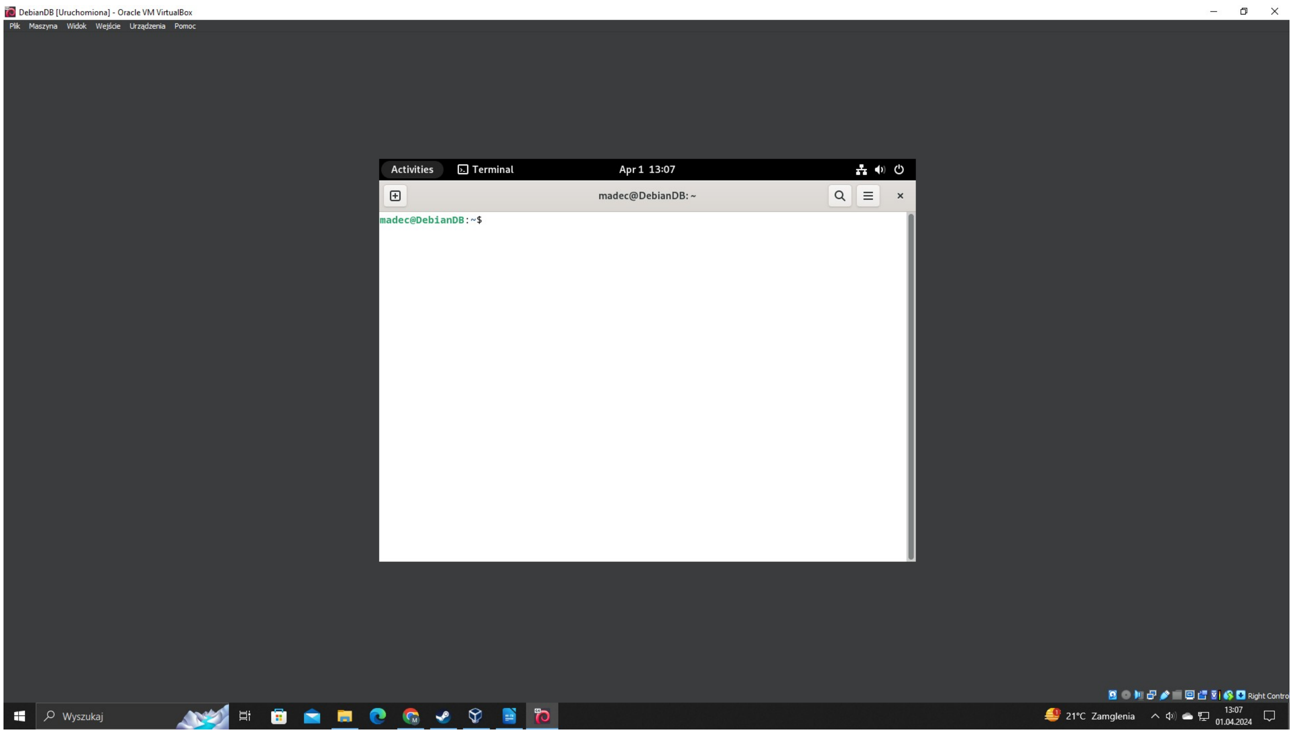
Once we created new virtual machine, the installation process will automatically boot up. It will automatically install all packages and folders needed to run Linux.


DebianDB [Uruchomiona] - Oracle VM VirtualBox
Plik Maszyna Widok Wejście Urządzenia Pomoc
Right Control
Wyszukaj
21°C Zamglenia
12:56
01.04.2024

```
Installing the base system
83%
Configuring linux-image-6.1.0-18-amd64 (amd64)
```

2.4 Viewing Debian

After successfull installation, we can login to newly created virtual machine. We can login with credentails defined during creating new virtual machine.

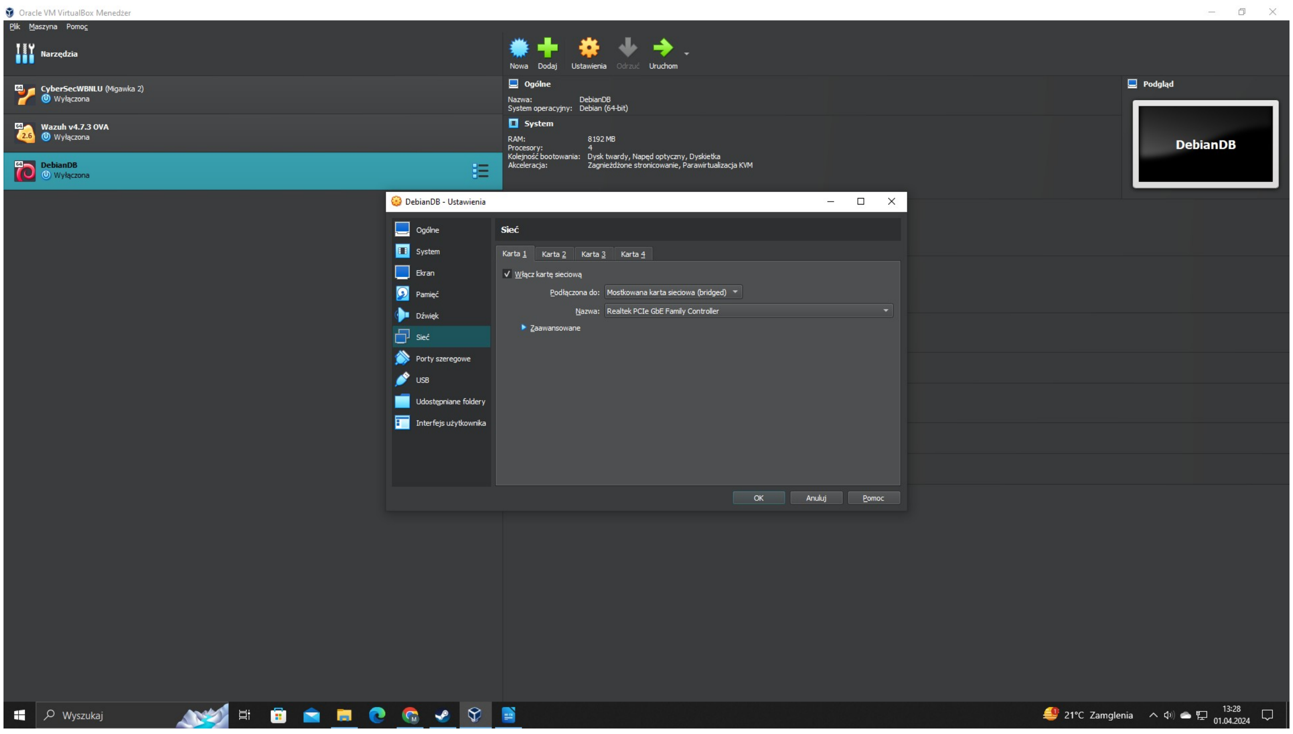


```
madec@DebianDB:~$
```

3. Setting up Debian environment

3.1 Network interface configuration

We will configure nework interface in Debian. We will implement static IP address and default gateway, which will be our local router in this case. Before we can do it, we have to make sure that network card of virtual machine is set to bridged mode.



DebianDB - Ustawienia

Sieć

Karta 1

✓ Włącz kartę sieciową

Podłączona do: Mostkowana karta sieciowa (bridged)

Nazwa: Realtek PCIe GbE Family Controller

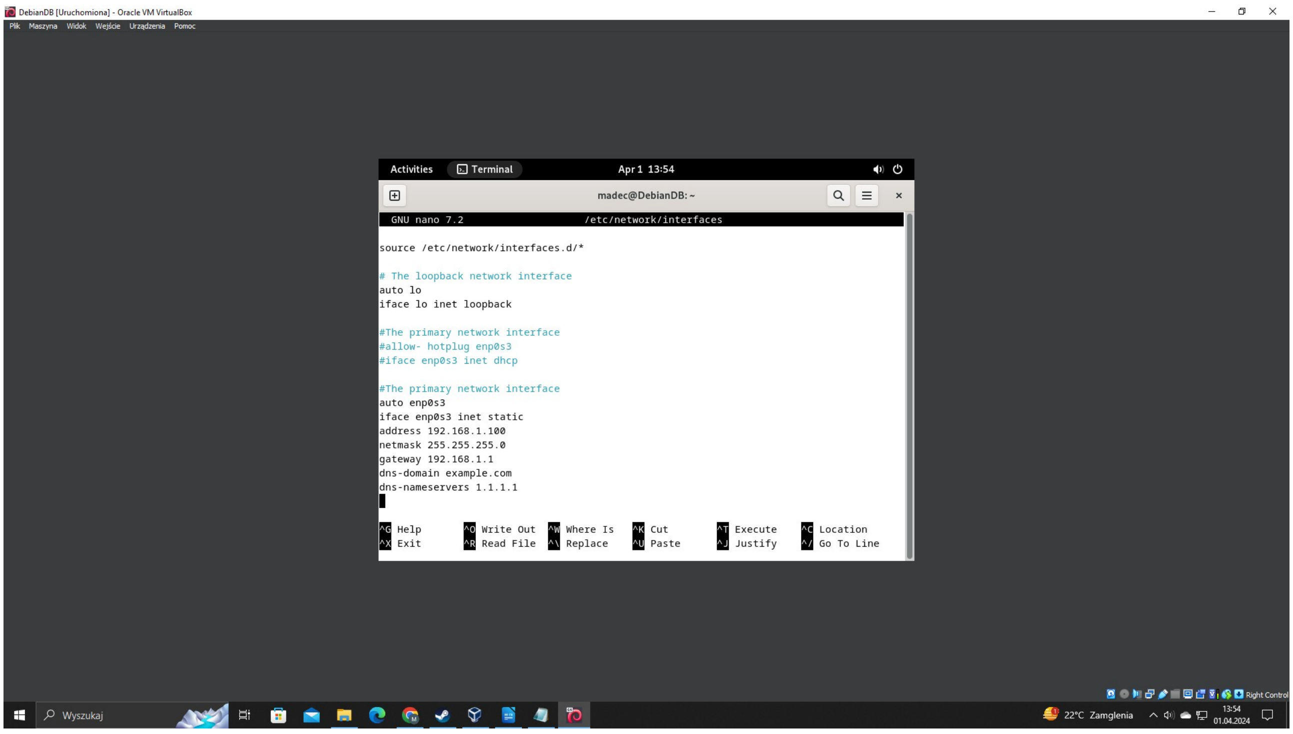
Ogólne

Nazwa: DebianDB
System operacyjny: Debian (64-bit)

System

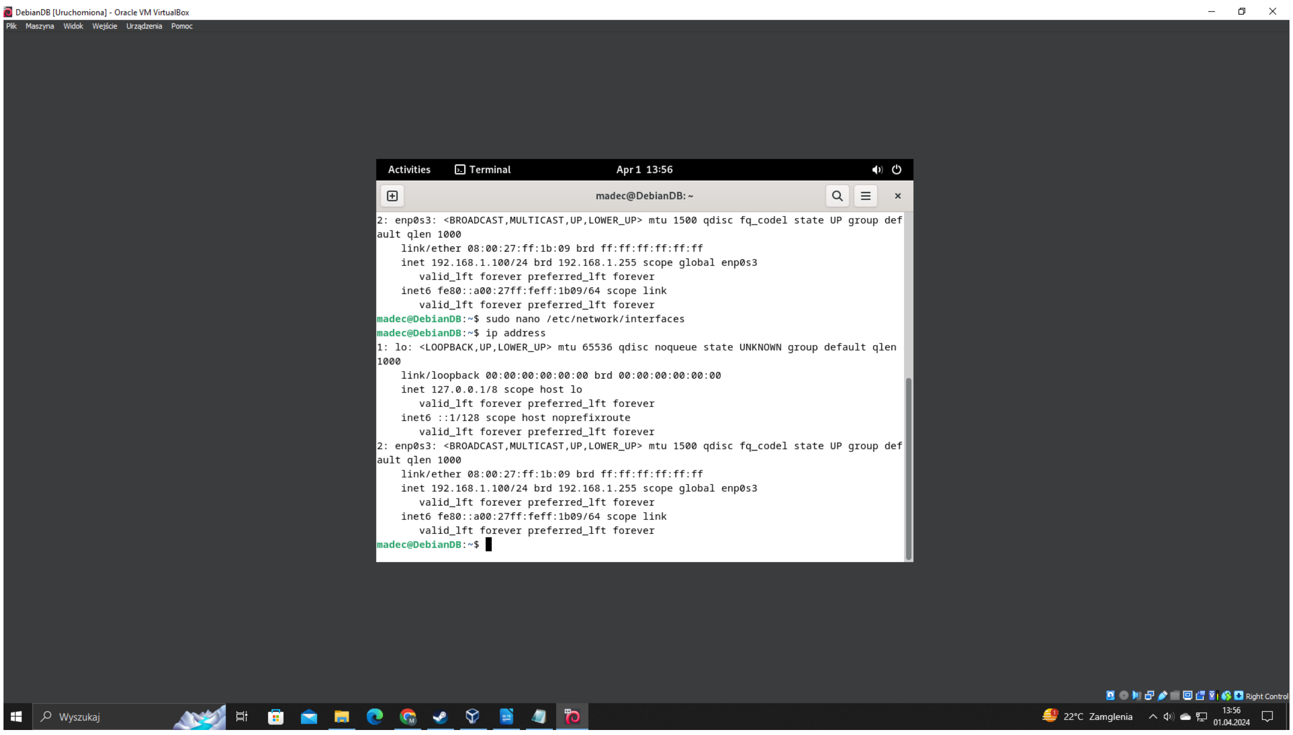
RAM: 8192 MB
Procesory: 4
Kolejność bootowania: Dysk twardy, Napęd optyczny, Dyskietka
Akceleracja: Zagnieżdżone stronicowanie, Parawirtualizacja KVM

Once we made sure, we can start configuring network interface. To do that we have to switch to /etc/network/interfaces and edit it with nano. We can do it by issuing command in terminal: sudo nano /etc/network/interfaces



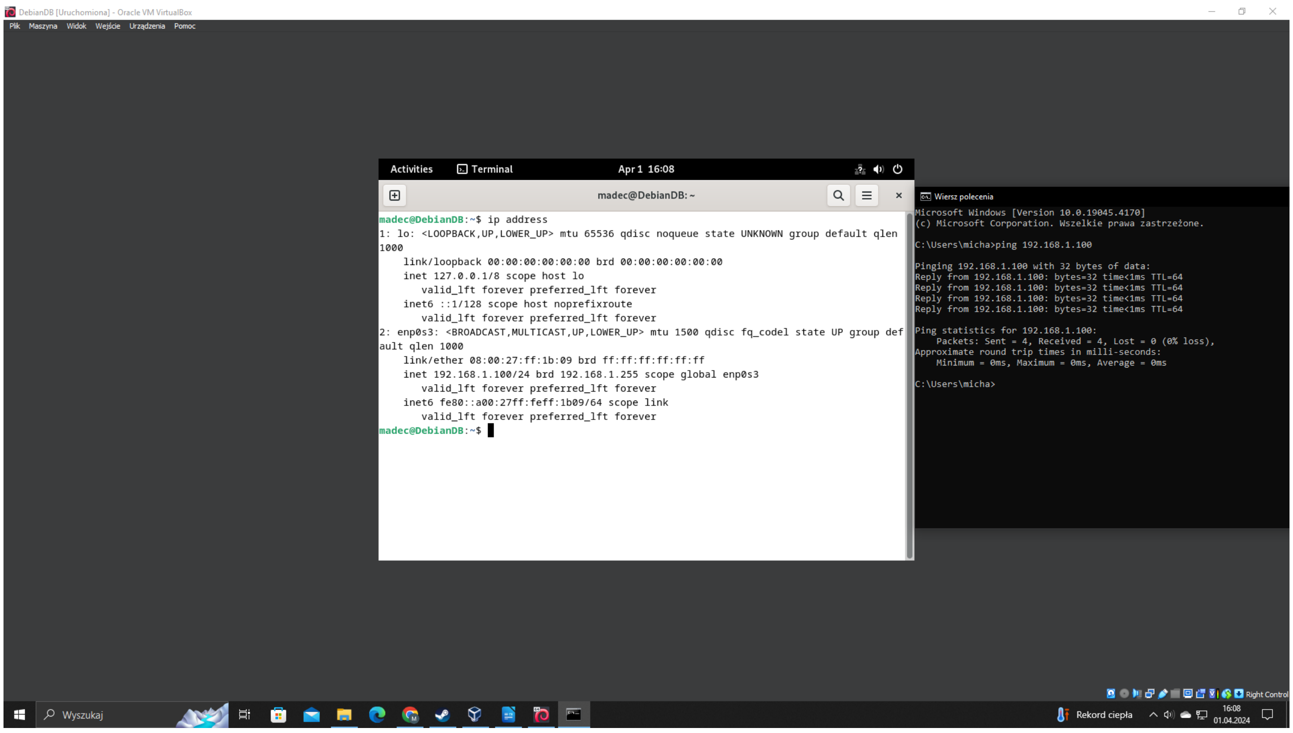
```
  GNU nano 7.2                    /etc/network/interfaces

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

#The primary network interface
#allow- hotplug enp0s3
#iface enp0s3 inet dhcp

#The primary network interface
auto enp0s3
iface enp0s3 inet static
address 192.168.1.100
netmask 255.255.255.0
gateway 192.168.1.1
dns-domain example.com
dns-nameservers 1.1.1.1


^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

Once we configured interfaces file, we have to restart networking. We can do it by issuing command: sudo systemctl restart networking. After that we can check if new address is loaded, by issuing command: ip address.



```
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group def
ault qlen 1000
    link/ether 08:00:27:ff:1b:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/24 brd 192.168.1.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feff:1b09/64 scope link
       valid_lft forever preferred_lft forever
madec@DebianDB:~$ sudo nano /etc/network/interfaces
madec@DebianDB:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group def
ault qlen 1000
    link/ether 08:00:27:ff:1b:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/24 brd 192.168.1.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feff:1b09/64 scope link
       valid_lft forever preferred_lft forever
madec@DebianDB:~$
```

Last step in this section is to verify connection. We can test commenction by pinging virtual machine from our local Windows machine. To do it, we have to enter to command prompt in Windows and issue: ping 192.168.1.100, which is local IP address of our Debian server.



```
madec@DebianDB:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group def
ault qlen 1000
    link/ether 08:00:27:ff:1b:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/24 brd 192.168.1.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feff:1b09/64 scope link
       valid_lft forever preferred_lft forever
madec@DebianDB:~$
```

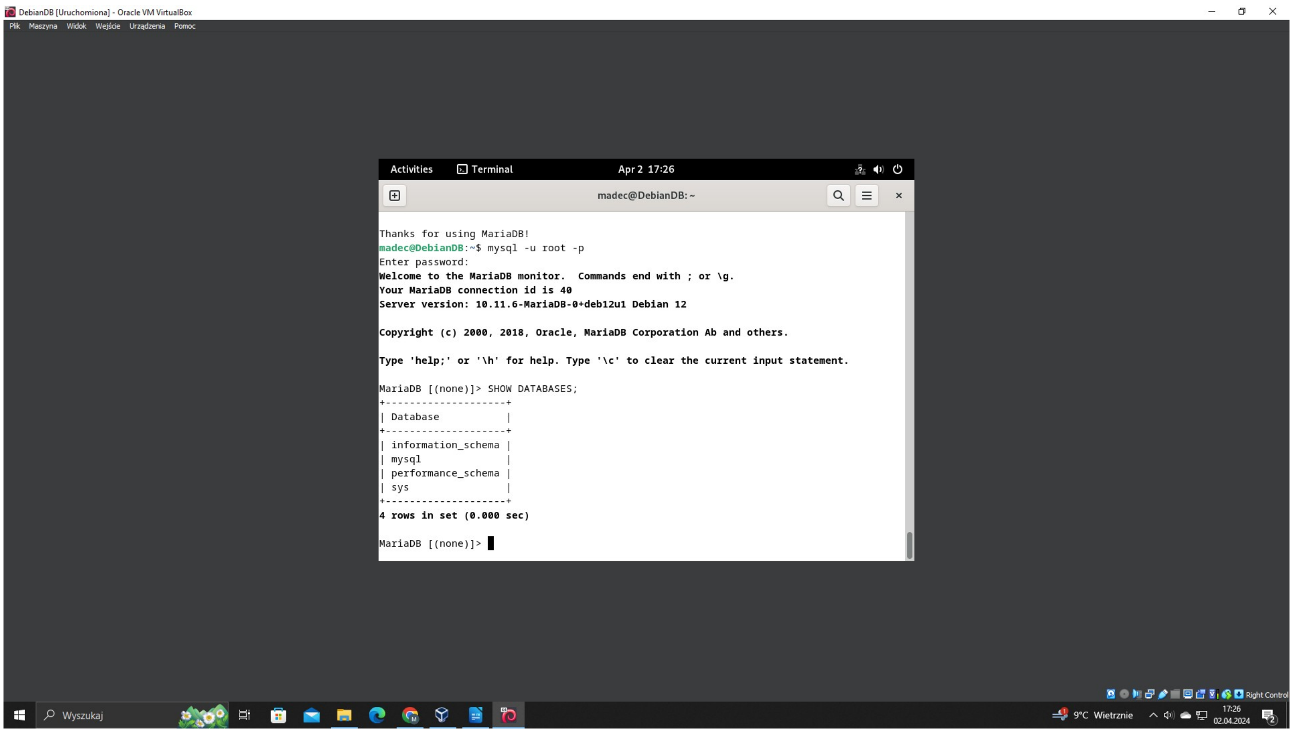
```
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\micha>ping 192.168.1.100

Pinging 192.168.1.100 with 32 bytes of data:
Reply from 192.168.1.100: bytes=32 time<1ms TTL=64
Reply from 192.168.1.100: bytes=32 time<1ms TTL=64
Reply from 192.168.1.100: bytes=32 time<1ms TTL=64
Reply from 192.168.1.100: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.1.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\micha>
```

3.2 Installing Database Server

In this section we will install MariaDB database server on Debian machine. Before we do it, we have to update all packages by issuing command: sudo apt-get update.

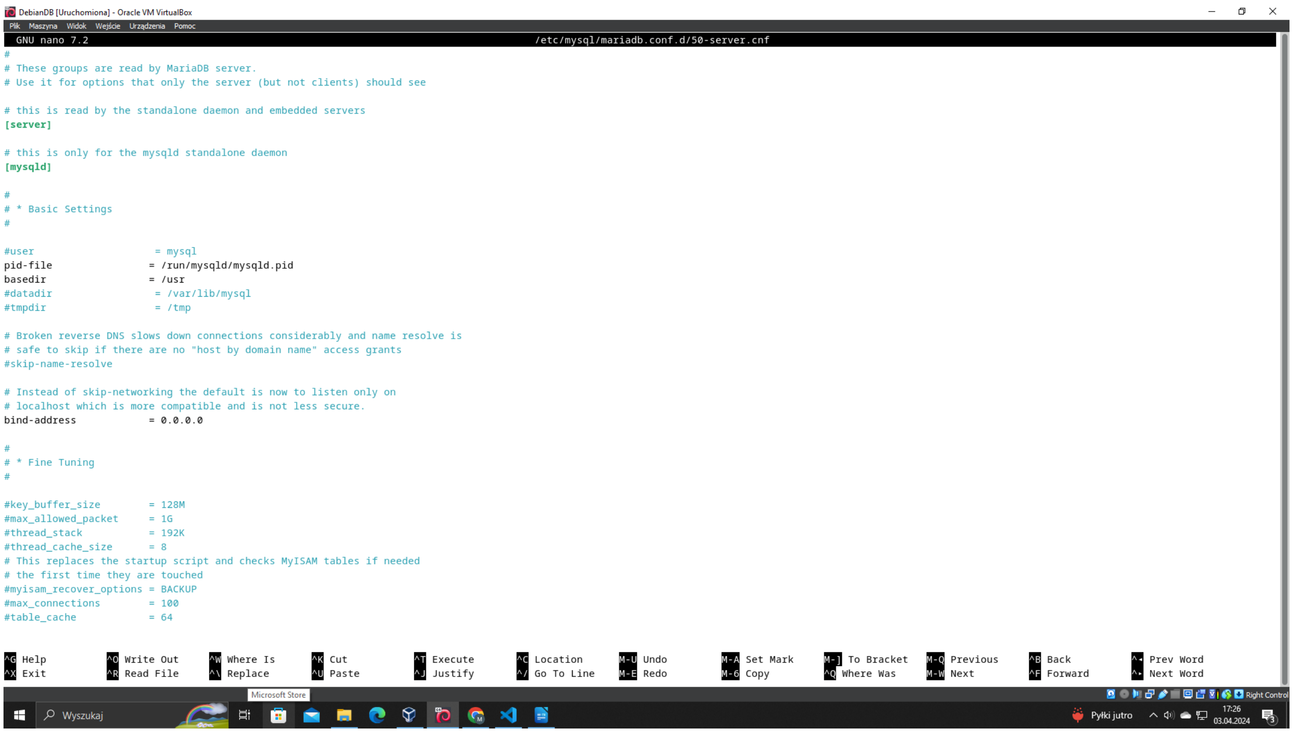
Next we have to install MariaDB. We can do it by command: sudo apt install mariadb-server. Next we have to configure access to database server. We issue command in terminal: sudo mysql_secure_installation. In this step we configure password for root access and we can disable remote access if needed.

Once done we can connect to MariaDB by command: mysql -u root -p. After successfull login, we can view all available databases and remove them if not needed.



```
Thanks for using MariaDB!
madec@DebianDB:~$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 40
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.000 sec)

MariaDB [(none)]> 
```

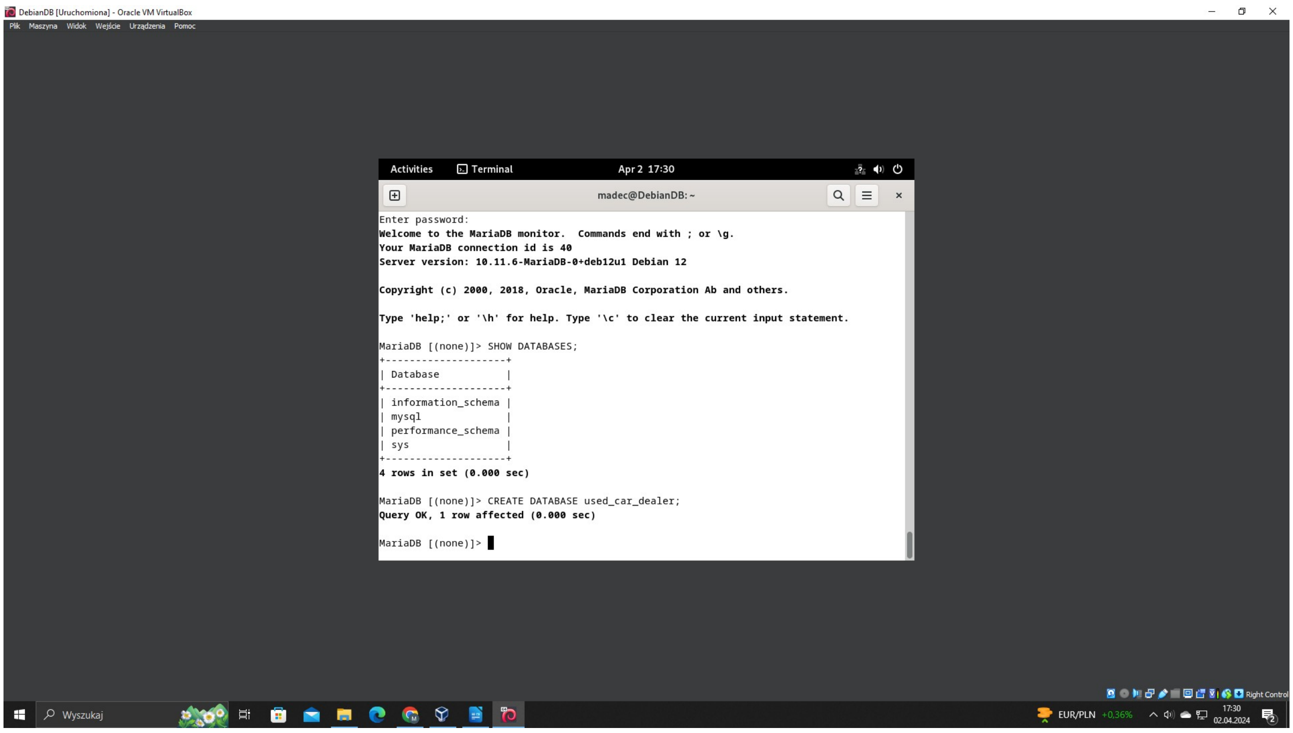
One last thing we need to do is to configure bind address. By default in file /etc/mysql/mariadb.conf.d/50-server.cnf bind address is set to 127.0.0.1, which is localhost. We need to change it to 0.0.0.0, so our database server could listen on all addresses. We can perform this operation by opening this file in nano with sudo command:


DebianDB [Uruchomiona] - Oracle VM VirtualBox
Plik Maszyna Widok Wejście Urządzenia Pomoc

```
  GNU nano 7.2                                   /etc/mysql/mariadb.conf.d/50-server.cnf
#
# These groups are read by MariaDB server.
# Use it for options that only the server (but not clients) should see

# this is read by the standalone daemon and embedded servers
[server]

# this is only for the mysqld standalone daemon
[mysqld]

#
# * Basic Settings
#

#user                    = mysql
pid-file                = /run/mysqld/mysqld.pid
basedir                 = /usr
#datadir                 = /var/lib/mysql
#tmpdir                  = /tmp

# Broken reverse DNS slows down connections considerably and name resolve is
# safe to skip if there are no "host by domain name" access grants
#skip-name-resolve

# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0

#
# * Fine Tuning
#

#key_buffer_size        = 128M
#max_allowed_packet     = 1G
#thread_stack           = 192K
#thread_cache_size      = 8
# This replaces the startup script and checks MyISAM tables if needed
# the first time they are touched
#myisam_recover_options = BACKUP
#max_connections        = 100
#table_cache            = 64

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark   M-] To Bracket M-Q Previous   ^B Back        ^◂ Prev Word
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy       ^Q Where Was   M-W Next       ^F Forward     ^▸ Next Word
```

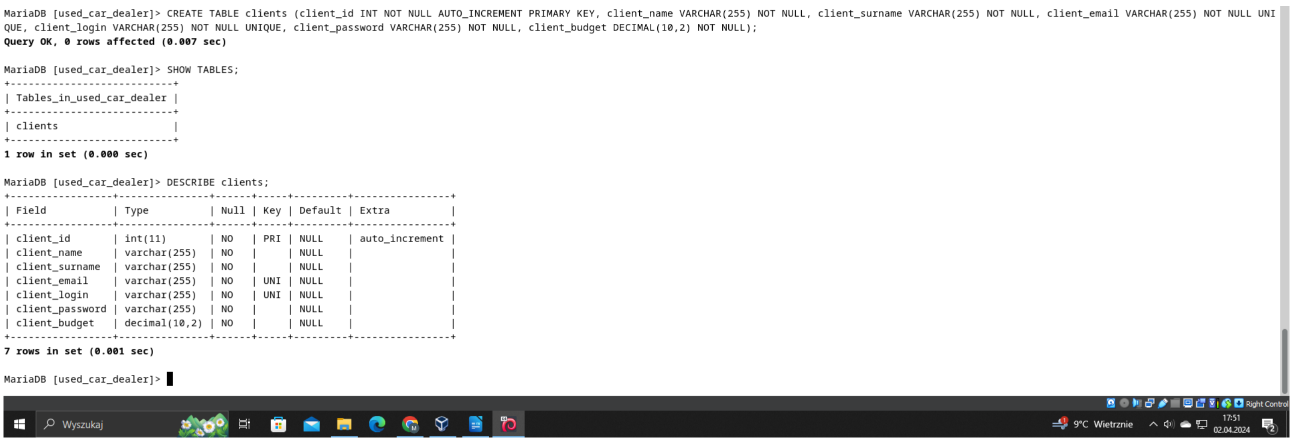
3.3 Confiuring MariaDB

Before we can store information, we have to create new database. We will name it used_car_dealer. To do it we can issue command in MariaDB: CREATE DATABASE used_car_dealer;.



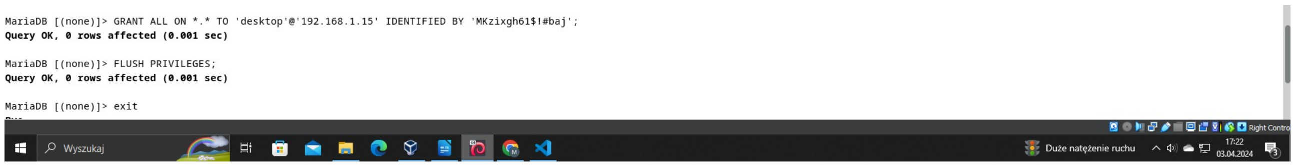
```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 40
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.000 sec)

MariaDB [(none)]> CREATE DATABASE used_car_dealer;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> 
```

Now we can switch to newly created database by command: USE used_car_dealer;. Once done, we will create first table for storing clients information. The table will contain following fields: client id, client name, client surname, client login, client password, client e-mail, and client budget. Field client_id will be primary key. Fields email and login will be unique. We can perform this operation by command:



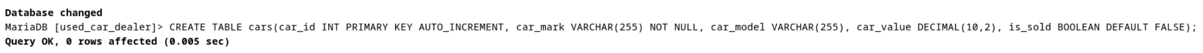
```
MariaDB [used_car_dealer]> CREATE TABLE clients (client_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, client_name VARCHAR(255) NOT NULL, client_surname VARCHAR(255) NOT NULL, client_email VARCHAR(255) NOT NULL UNI
QUE, client_login VARCHAR(255) NOT NULL UNIQUE, client_password VARCHAR(255) NOT NULL, client_budget DECIMAL(10,2) NOT NULL);
Query OK, 0 rows affected (0.007 sec)

MariaDB [used_car_dealer]> SHOW TABLES;
+---------------------------+
| Tables_in_used_car_dealer |
+---------------------------+
| clients                   |
+---------------------------+
1 row in set (0.000 sec)

MariaDB [used_car_dealer]> DESCRIBE clients;
+-----------------+---------------+------+-----+---------+----------------+
| Field           | Type          | Null | Key | Default | Extra          |
+-----------------+---------------+------+-----+---------+----------------+
| client_id       | int(11)       | NO   | PRI | NULL    | auto_increment |
| client_name     | varchar(255)  | NO   |     | NULL    |                |
| client_surname  | varchar(255)  | NO   |     | NULL    |                |
| client_email    | varchar(255)  | NO   | UNI | NULL    |                |
| client_login    | varchar(255)  | NO   | UNI | NULL    |                |
| client_password | varchar(255)  | NO   |     | NULL    |                |
| client_budget   | decimal(10,2) | NO   |     | NULL    |                |
+-----------------+---------------+------+-----+---------+----------------+
7 rows in set (0.001 sec)

MariaDB [used_car_dealer]>
```

Now we will create user, who we will use to connect remotely to our database. We will define him on local IP address: 192.168.1.15, which is local IP address of our Windows machine. On this machine we will write scripts in Python. We can do it by:



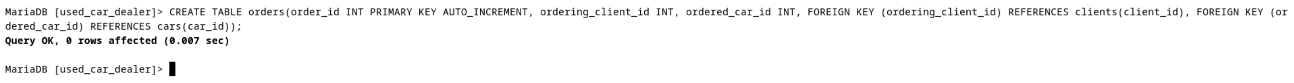
```
MariaDB [(none)]> GRANT ALL ON *.* TO 'desktop'@'192.168.1.15' IDENTIFIED BY 'MKzixgh61$!#baj';
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> exit
```

Now we will create new table in database. We will name it „cars”. This table will store fields like: car_id, car_mark, car_model, car_vin, car_value, and is_sold. Field is_sold will be bool value. We will need this to list all available cars for client. When car is sold, this field will change to yes. We can achive that by issuing command in MariaDB:



```
Database changed
MariaDB [used_car_dealer]> CREATE TABLE cars(car_id INT PRIMARY KEY AUTO_INCREMENT, car_mark VARCHAR(255) NOT NULL, car_model VARCHAR(255), car_value DECIMAL(10,2), is_sold BOOLEAN DEFAULT FALSE);
Query OK, 0 rows affected (0.005 sec)
```

We will create last table named „orders”. There will be field like: order_id, ordering_client_id, and ordered_car_id. Field order_id will be primary key. Fields ordering_client_id and ordered_car_id will be foreigns keys from tables „clients” and „cars”. We can make that by command:



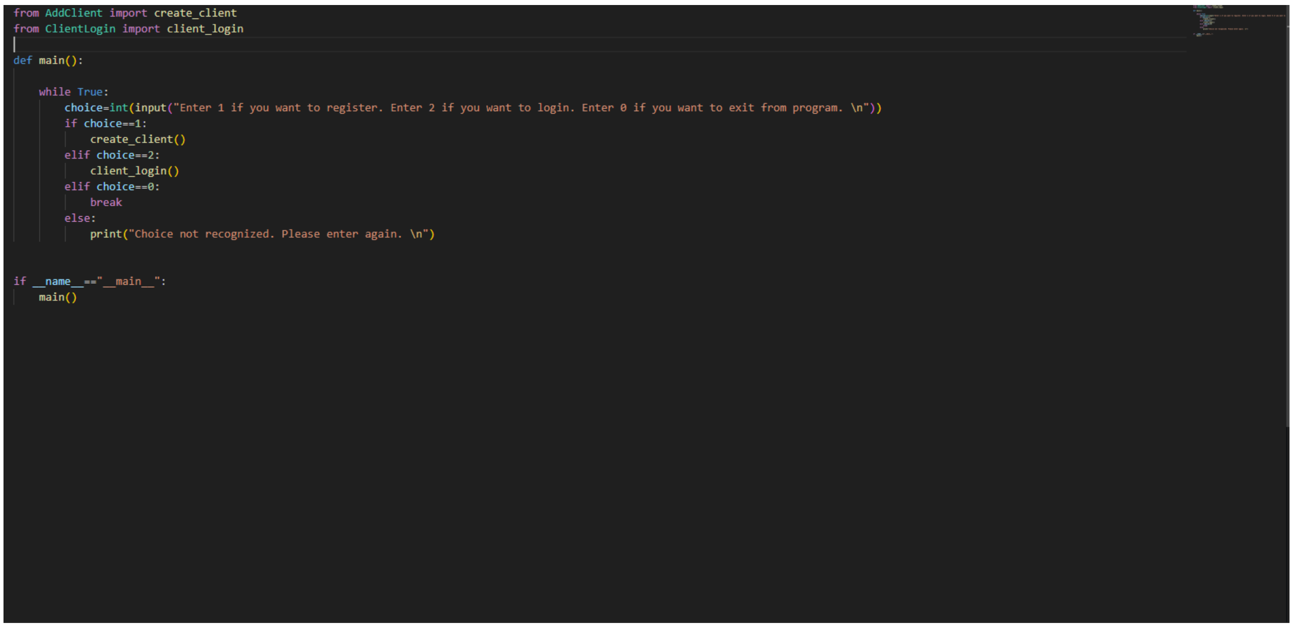
```
MariaDB [used_car_dealer]> CREATE TABLE orders(order_id INT PRIMARY KEY AUTO_INCREMENT, ordering_client_id INT, ordered_car_id INT, FOREIGN KEY (ordering_client_id) REFERENCES clients(client_id), FOREIGN KEY (or
dered_car_id) REFERENCES cars(car_id));
Query OK, 0 rows affected (0.007 sec)

MariaDB [used_car_dealer]> █
```

4. Python application

4.1 Client control panel

4.1.1 main() function

In this module we have defined main() function. It is control panel for client. Based on choice it allows either to register or to login. If there is not matched choice it outpust that choice is not recognized and asks us to enter the choice again. If user enters 0, the program stops.

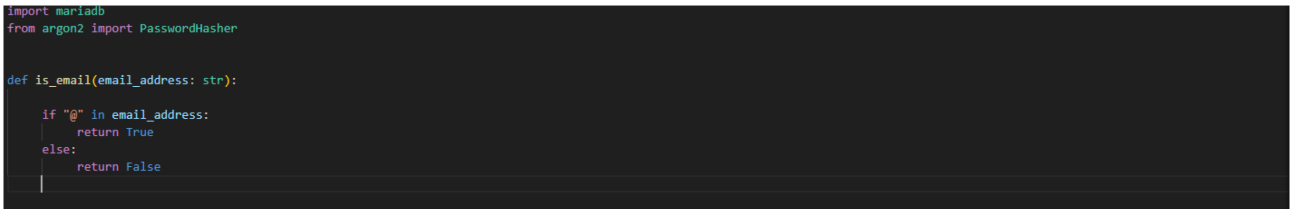


```python
from AddClient import create_client
from ClientLogin import client_login

def main():

    while True:
        choice=int(input("Enter 1 if you want to register. Enter 2 if you want to login. Enter 0 if you want to exit from program. \n"))
        if choice==1:
            create_client()
        elif choice==2:
            client_login()
        elif choice==0:
            break
        else:
            print("Choice not recognized. Please enter again. \n")


if __name__=="__main__":
    main()
```

4.2 Adding client

4.2.1 is_email() function

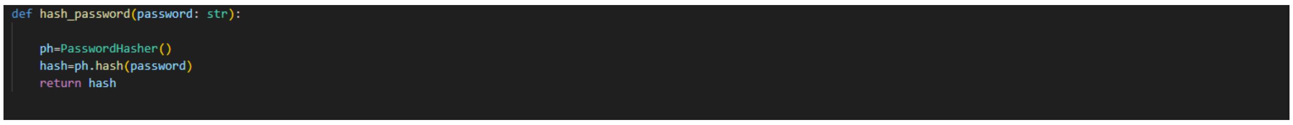
In module AddClient we have defined function is_email(). This function checks if symbol @ is included in e-mail entered by client. If yes, returns true, otherwise false.



```
import mariadb
from argon2 import PasswordHasher


def is_email(email_address: str):

    if "@" in email_address:
        return True
    else:
        return False
```

4.2.2 hash_password() function

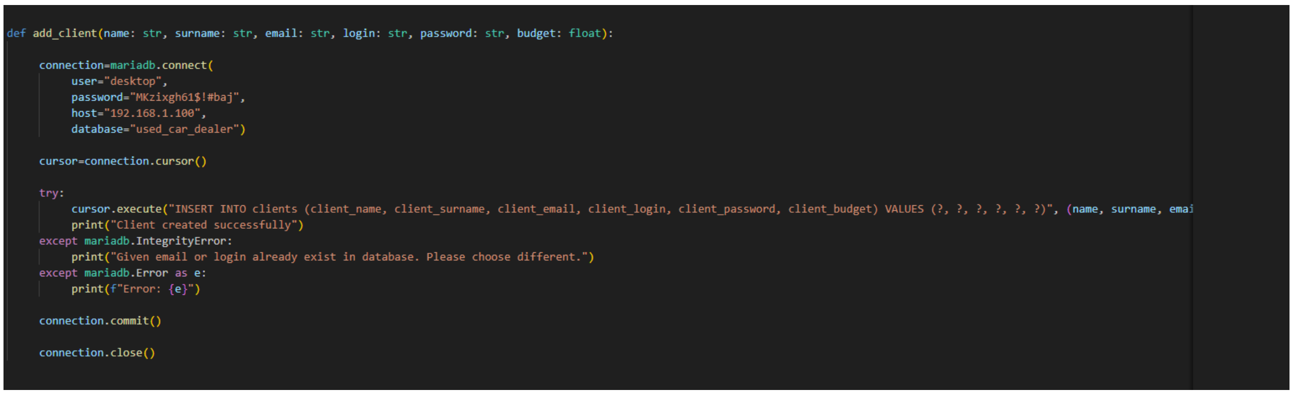
This function hashes entered password by user with Argon2. It returns hashed password. Argon2 is a key derivation function that was selected as the winner of the 2015 Password Hashing Competition.



```
def hash_password(password: str):

    ph=PasswordHasher()
    hash=ph.hash(password)
    return hash
```

4.2.3 add_client function()

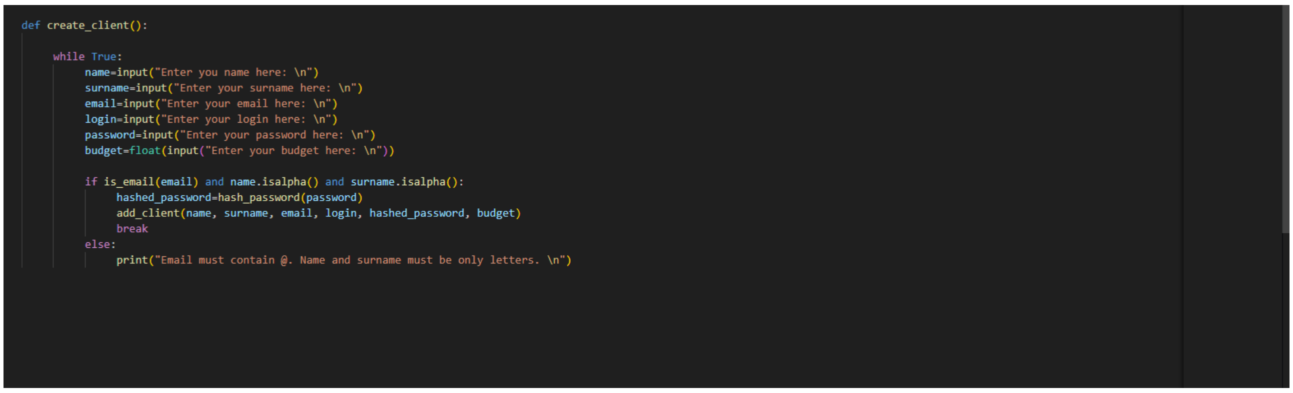
This function inserts entered client data to the table „clients” in „used_car_dealer” database. We have defined exceptions that might raise. Fileds e-mail and login are unique in our table. If the client enters login or e-mail that is already stored in database, an exception is catched and says that user has to use different e-mail or login.



```
def add_client(name: str, surname: str, email: str, login: str, password: str, budget: float):

    connection=mariadb.connect(
        user="desktop",
        password="MKzixgh61$!#baj",
        host="192.168.1.100",
        database="used_car_dealer")

    cursor=connection.cursor()

    try:
        cursor.execute("INSERT INTO clients (client_name, client_surname, client_email, client_login, client_password, client_budget) VALUES (?, ?, ?, ?, ?, ?)", (name, surname, emai
        print("Client created successfully")
    except mariadb.IntegrityError:
        print("Given email or login already exist in database. Please choose different.")
    except mariadb.Error as e:
        print(f"Error: {e}")

    connection.commit()

    connection.close()
```

4.2.4 create_client() function

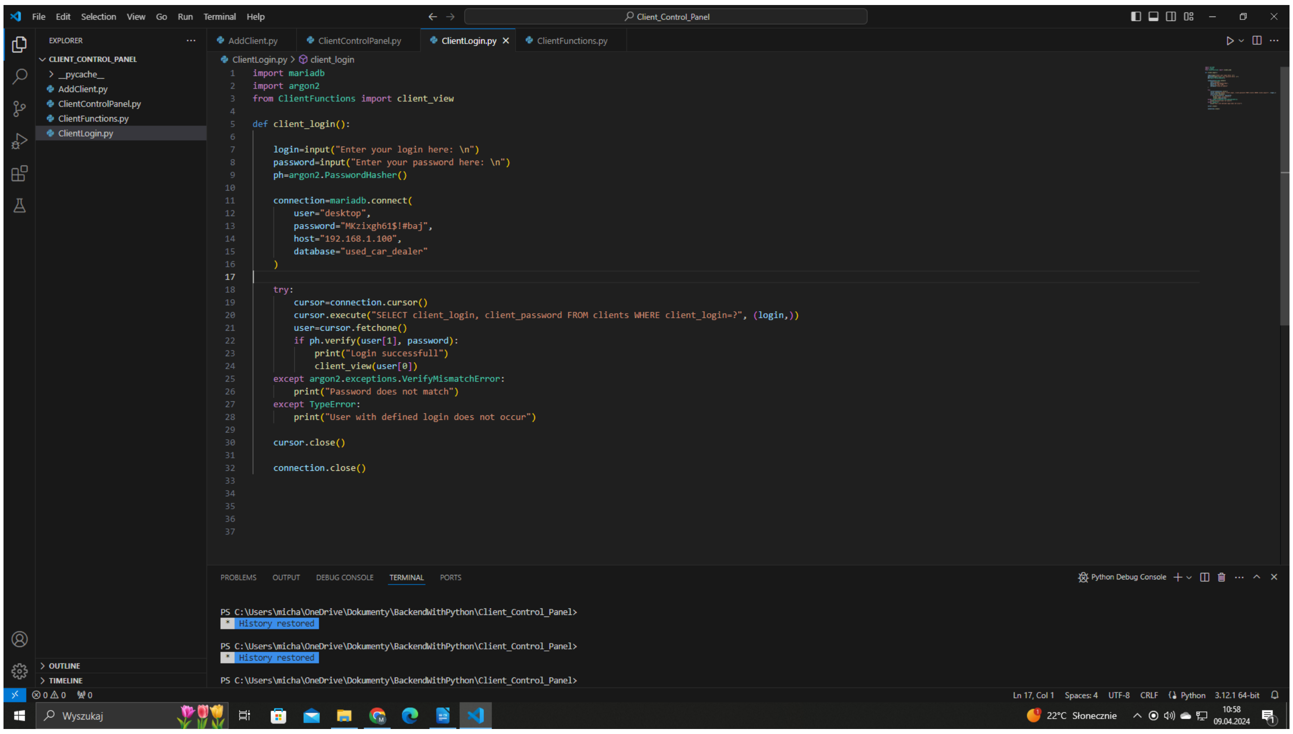
This function loads data that was entered by client. There is a logic condition that e-mail address has to contaion @ symbol. Name and surname have to contain only letters. If the logic condition is passed, a password is hashed with the use of hash_password() function and then all the data is passed to function add_client().



```python
def create_client():

    while True:
        name=input("Enter you name here: \n")
        surname=input("Enter your surname here: \n")
        email=input("Enter your email here: \n")
        login=input("Enter your login here: \n")
        password=input("Enter your password here: \n")
        budget=float(input("Enter your budget here: \n"))

        if is_email(email) and name.isalpha() and surname.isalpha():
            hashed_password=hash_password(password)
            add_client(name, surname, email, login, hashed_password, budget)
            break
        else:
            print("Email must contain @. Name and surname must be only letters. \n")
```

4.3 Client login

In module ClientLogin we have function client_login(). In this function client passes his login and password. Based on login we have SELECT query and this query downloads client's password stored in database. Client's login is unique field in our database. Next, the downloaded client's password is compared with passed client's password in function. As we described previously, client's password is hashed with the use of Argon2. If the password, which client passed in function matches with stored password, then client can have access to control panel, which allows for order processing. All this operation is written in try-except block. We can get two exceptions here. First – passed client's password does not match with stored. Second – login, which client passed in function does not occur in database.



```python
import mariadb
import argon2
from ClientFunctions import client_view

def client_login():

    login=input("Enter your login here: \n")
    password=input("Enter your password here: \n")
    ph=argon2.PasswordHasher()

    connection=mariadb.connect(
        user="desktop",
        password="MKzixgh61$!#baj",
        host="192.168.1.100",
        database="used_car_dealer"
    )

    try:
        cursor=connection.cursor()
        cursor.execute("SELECT client_login, client_password FROM clients WHERE client_login=?", (login,))
        user=cursor.fetchone()
        if ph.verify(user[1], password):
            print("Login successfull")
            client_view(user[0])
    except argon2.exceptions.VerifyMismatchError:
        print("Password does not match")
    except TypeError:
        print("User with defined login does not occur")

    cursor.close()

    connection.close()
```

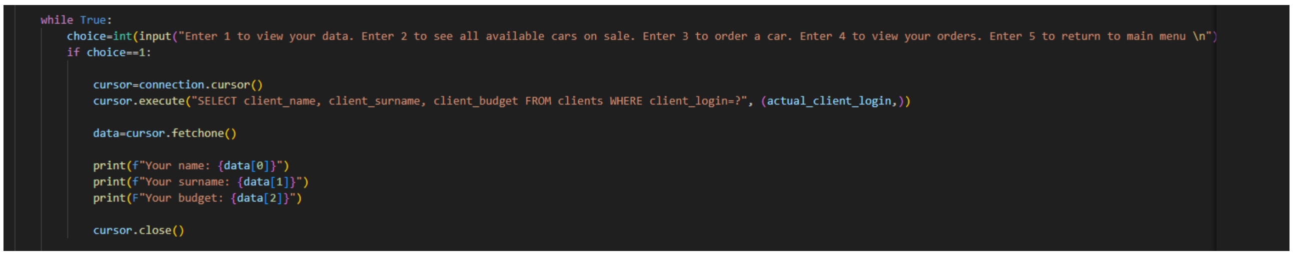
```
PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Client_Control_Panel>
 *  History restored

PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Client_Control_Panel>
 *  History restored

PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Client_Control_Panel>
```

4.4 Client Functions

In module ClientFunctions we have defined function client_view(). We pass client’s login to this function as an argument. Based on this login we can write SQL queries, beacuse this field is unique.

4.4.1 Choice 1

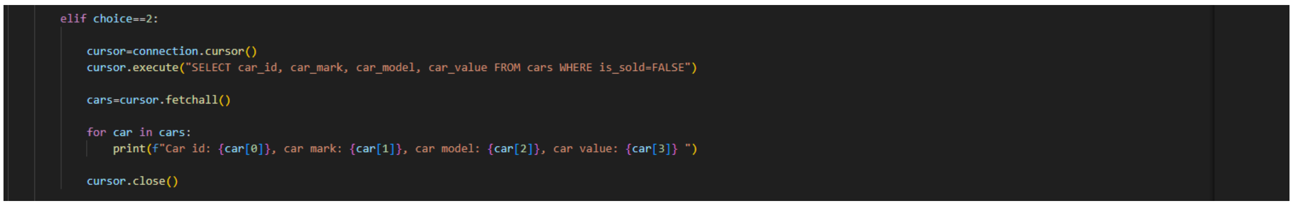
In control panel we have defined infinite while loop. What program will do depends on client’s choice. In choice 1 we have cursor that downloads client’s data from database from table clients.



```
while True:
    choice=int(input("Enter 1 to view your data. Enter 2 to see all available cars on sale. Enter 3 to order a car. Enter 4 to view your orders. Enter 5 to return to main menu \n")
    if choice==1:

        cursor=connection.cursor()
        cursor.execute("SELECT client_name, client_surname, client_budget FROM clients WHERE client_login=?", (actual_client_login,))

        data=cursor.fetchone()

        print(f"Your name: {data[0]}")
        print(f"Your surname: {data[1]}")
        print(F"Your budget: {data[2]}")

        cursor.close()
```

4.4.2 Choice 2

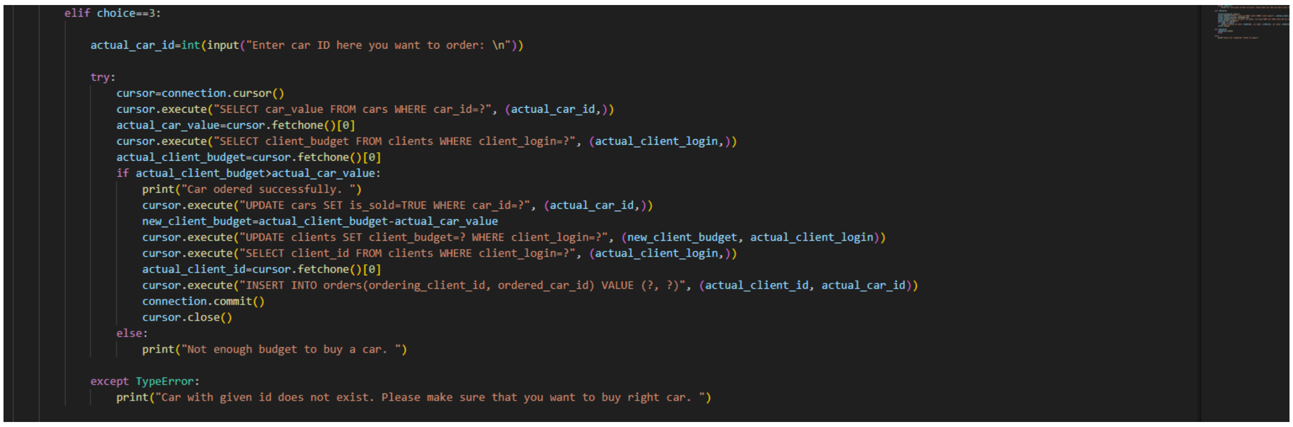
In choice 2 we have written query for our database. This query show us all available cars on sale. We can do that, beacuse table cars has boolean field is_sold. This SQL query shows us all available cars with WHERE condition based on False.



```
    elif choice==2:

        cursor=connection.cursor()
        cursor.execute("SELECT car_id, car_mark, car_model, car_value FROM cars WHERE is_sold=FALSE")

        cars=cursor.fetchall()

        for car in cars:
            print(f"Car id: {car[0]}, car mark: {car[1]}, car model: {car[2]}, car value: {car[3]} ")

        cursor.close()
```

4.4.3 Choice 3

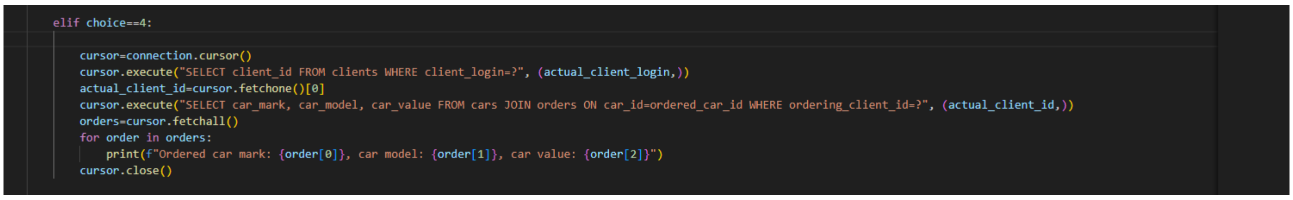
In choice 3 we have defined order processing. Client can buy a car based on ID. All the operation is defined in try-except statement. We can get an exception here – client can pass ID, which does not occur in database. Client can buy a car only when his budget is greater than car’s value. When the order is processed, client’s budget is updated and field is_sold from cars table is changed to True.



```python
elif choice==3:

    actual_car_id=int(input("Enter car ID here you want to order: \n"))

    try:
        cursor=connection.cursor()
        cursor.execute("SELECT car_value FROM cars WHERE car_id=?", (actual_car_id,))
        actual_car_value=cursor.fetchone()[0]
        cursor.execute("SELECT client_budget FROM clients WHERE client_login=?", (actual_client_login,))
        actual_client_budget=cursor.fetchone()[0]
        if actual_client_budget>actual_car_value:
            print("Car odered successfully. ")
            cursor.execute("UPDATE cars SET is_sold=TRUE WHERE car_id=?", (actual_car_id,))
            new_client_budget=actual_client_budget-actual_car_value
            cursor.execute("UPDATE clients SET client_budget=? WHERE client_login=?", (new_client_budget, actual_client_login))
            cursor.execute("SELECT client_id FROM clients WHERE client_login=?", (actual_client_login,))
            actual_client_id=cursor.fetchone()[0]
            cursor.execute("INSERT INTO orders(ordering_client_id, ordered_car_id) VALUE (?, ?)", (actual_client_id, actual_car_id))
            connection.commit()
            cursor.close()
        else:
            print("Not enough budget to buy a car. ")

    except TypeError:
        print("Car with given id does not exist. Please make sure that you want to buy right car. ")
```

4.4.4 Choice 4

In choice 4, a client can view his all orders. In table orders we have two foreign keys – ordering_client_id as client_id from clients table and ordered_car_id as car_id from cars table. In this SQL query defined inside this choice we use JOIN operator. We JOIN cars table. Every order id has its reference to the cars and clients table.

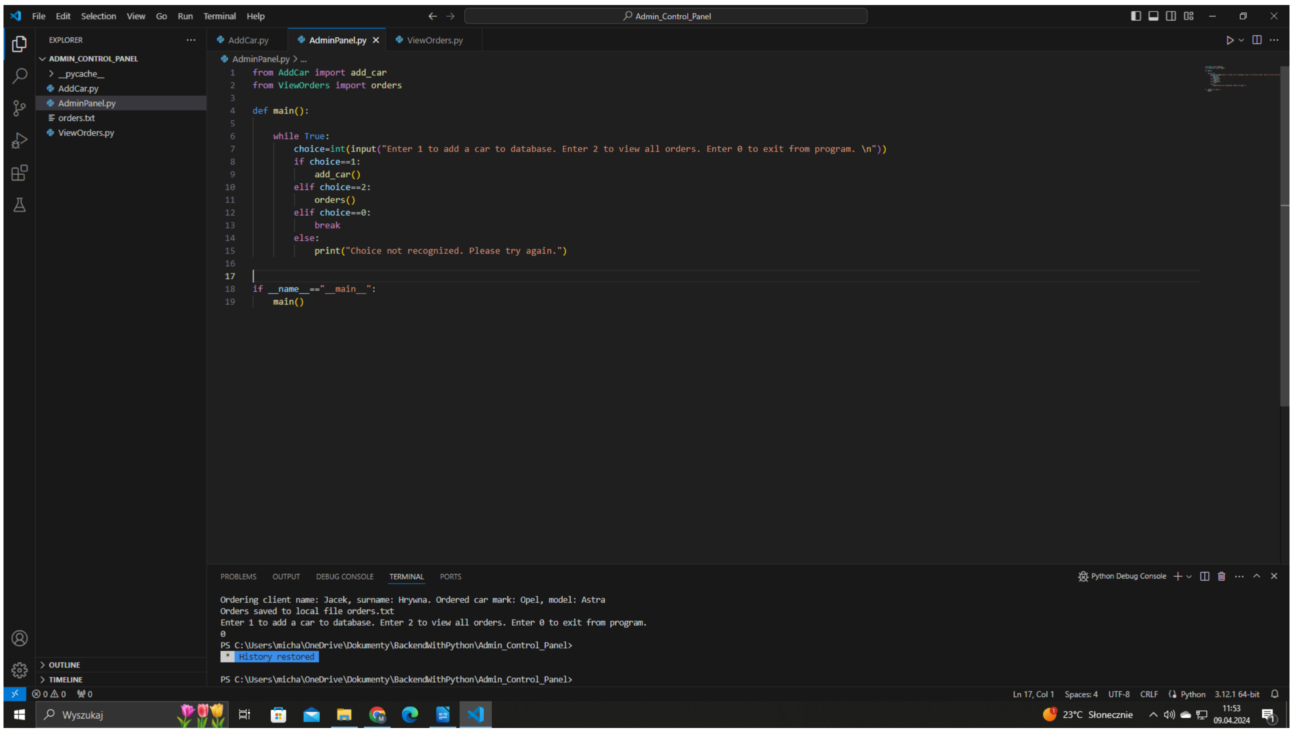


```python
elif choice==4:

    cursor=connection.cursor()
    cursor.execute("SELECT client_id FROM clients WHERE client_login=?", (actual_client_login,))
    actual_client_id=cursor.fetchone()[0]
    cursor.execute("SELECT car_mark, car_model, car_value FROM cars JOIN orders ON car_id=ordered_car_id WHERE ordering_client_id=?", (actual_client_id,))
    orders=cursor.fetchall()
    for order in orders:
        print(f"Ordered car mark: {order[0]}, car model: {order[1]}, car value: {order[2]}")
    cursor.close()
```

In choice 5 we have simple break statement. Inside this choice we also close connection to the database.

4.5 Admin Panel

In other directory named Admin_Control_Panel we have defined modules in Python for administrator. Only administrator can add a car to the database and list all orders from clients. Inside AdminPanel module we have defined main() function. Based on admin's choice application can either add a car to the database or list all available orders from database.

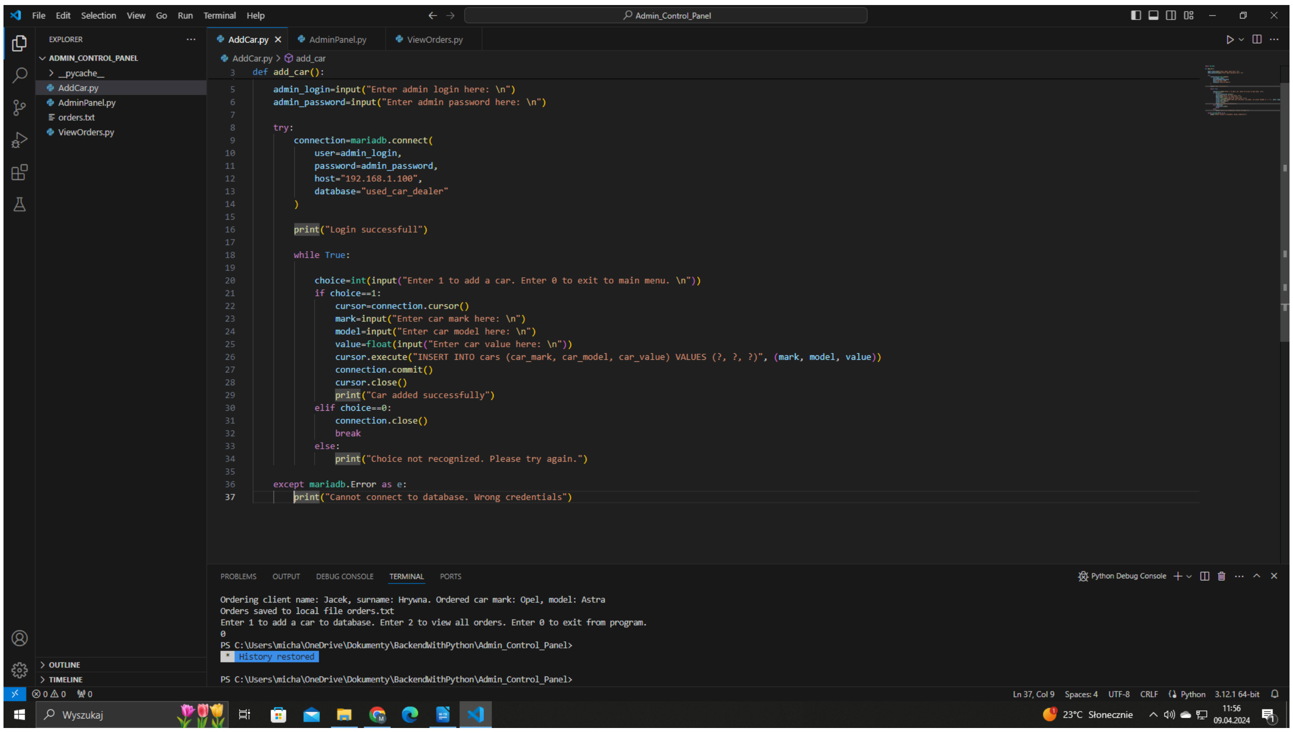


```python
from AddCar import add_car
from ViewOrders import orders

def main():

    while True:
        choice=int(input("Enter 1 to add a car to database. Enter 2 to view all orders. Enter 0 to exit from program. \n"))
        if choice==1:
            add_car()
        elif choice==2:
            orders()
        elif choice==0:
            break
        else:
            print("Choice not recognized. Please try again.")


if __name__=="__main__":
    main()
```

```
Ordering client name: Jacek, surname: Hrywna. Ordered car mark: Opel, model: Astra
Orders saved to local file orders.txt
Enter 1 to add a car to database. Enter 2 to view all orders. Enter 0 to exit from program.
0
PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Admin_Control_Panel>
 *  History restored

PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Admin_Control_Panel>
```

4.6 Add Car

Inside AddCar module we have defined add_car() function. Admin can add a car after when he logs in. Inside this function admin passes his login and password. We connect to the database based on these credentials. If they are wrong, we cannot connect to the database. If the creadentials are true, admin can add a car to the database.

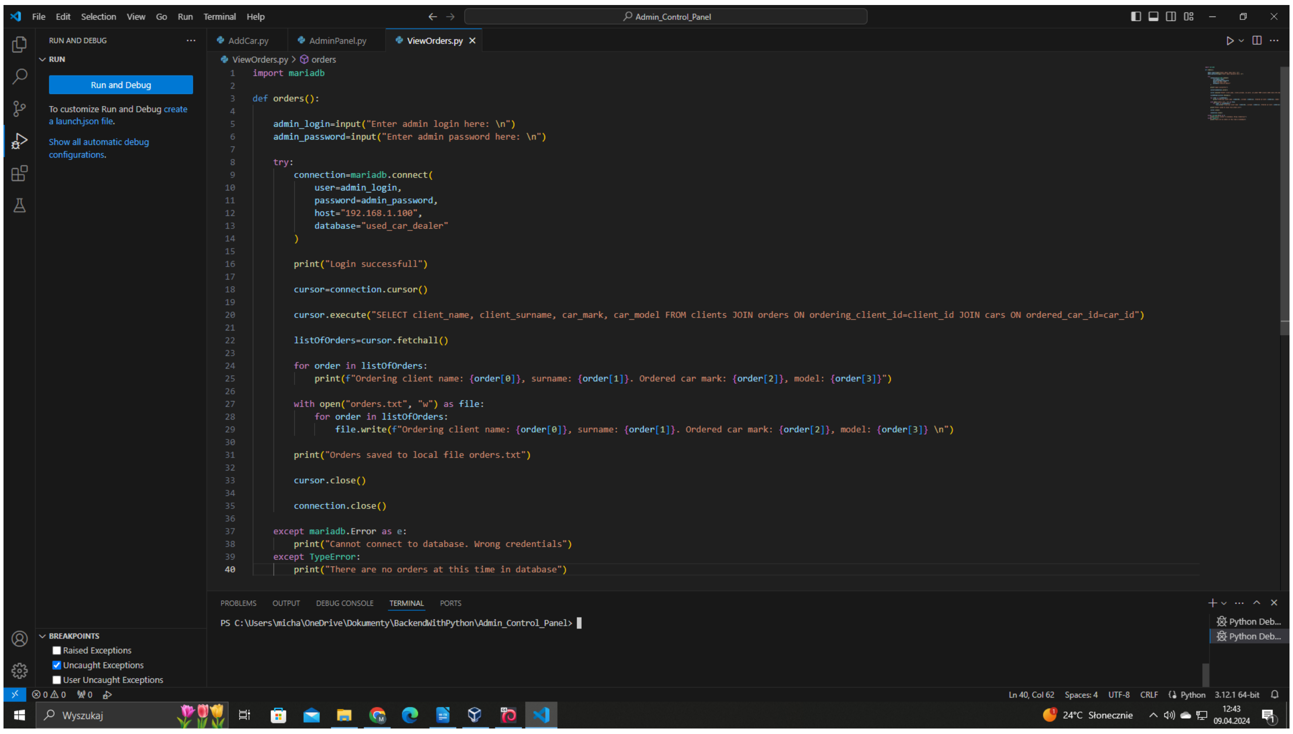


```python
def add_car():

    admin_login=input("Enter admin login here: \n")
    admin_password=input("Enter admin password here: \n")

    try:
        connection=mariadb.connect(
            user=admin_login,
            password=admin_password,
            host="192.168.1.100",
            database="used_car_dealer"
        )

        print("Login successfull")

        while True:

            choice=int(input("Enter 1 to add a car. Enter 0 to exit to main menu. \n"))
            if choice==1:
                cursor=connection.cursor()
                mark=input("Enter car mark here: \n")
                model=input("Enter car model here: \n")
                value=float(input("Enter car value here: \n"))
                cursor.execute("INSERT INTO cars (car_mark, car_model, car_value) VALUES (?, ?, ?)", (mark, model, value))
                connection.commit()
                cursor.close()
                print("Car added successfully")
            elif choice==0:
                connection.close()
                break
            else:
                print("Choice not recognized. Please try again.")

    except mariadb.Error as e:
        print("Cannot connect to database. Wrong credentials")
```

```
Ordering client name: Jacek, surname: Hrywna. Ordered car mark: Opel, model: Astra
Orders saved to local file orders.txt
Enter 1 to add a car to database. Enter 2 to view all orders. Enter 0 to exit from program.
0
PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Admin_Control_Panel>
 *  History restored

PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Admin_Control_Panel>
```

4.7 View Orders

Inside VieOrders module we have defined orders() function. In this function admin have to again pass his credentials. If they are correct, admin can list all available orders from database and save them to the local .txt file named orders.txt



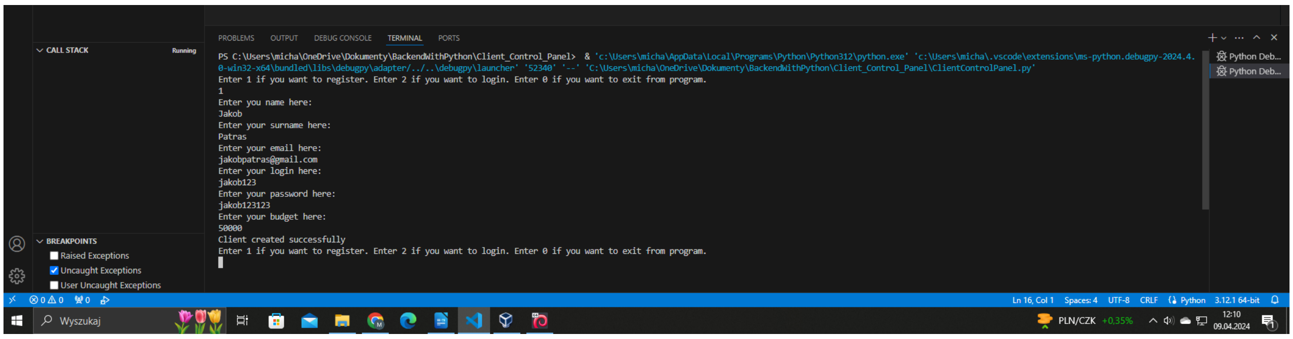
```python
import mariadb

def orders():

    admin_login=input("Enter admin login here: \n")
    admin_password=input("Enter admin password here: \n")

    try:
        connection=mariadb.connect(
            user=admin_login,
            password=admin_password,
            host="192.168.1.100",
            database="used_car_dealer"
        )

        print("Login successfull")

        cursor=connection.cursor()

        cursor.execute("SELECT client_name, client_surname, car_mark, car_model FROM clients JOIN orders ON ordering_client_id=client_id JOIN cars ON ordered_car_id=car_id")

        listOfOrders=cursor.fetchall()

        for order in listOfOrders:
            print(f"Ordering client name: {order[0]}, surname: {order[1]}. Ordered car mark: {order[2]}, model: {order[3]}")

        with open("orders.txt", "w") as file:
            for order in listOfOrders:
                file.write(f"Ordering client name: {order[0]}, surname: {order[1]}. Ordered car mark: {order[2]}, model: {order[3]} \n")

        print("Orders saved to local file orders.txt")

        cursor.close()

        connection.close()

    except mariadb.Error as e:
        print("Cannot connect to database. Wrong credentials")
    except TypeError:
        print("There are no orders at this time in database")
```

```
PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Admin_Control_Panel>
```

5. Testing

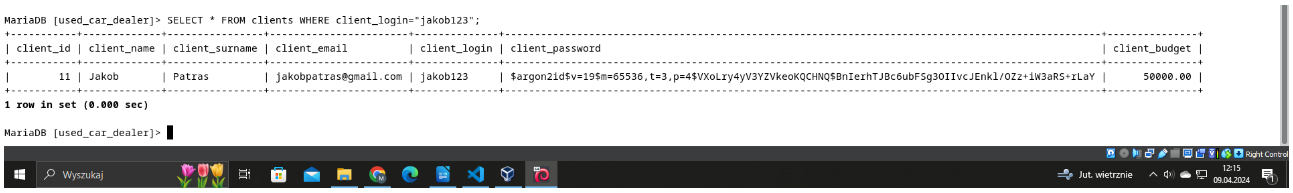
5.1 Adding a client

We will add client with defined credentials – login: jakob123 and password: jakob123123. We will set his budget to 50000 dollars.



```
PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Client_Control_Panel>  & 'c:\Users\micha\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\micha\.vscode\extensions\ms-python.debugpy-2024.4.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '52340' '--' 'C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Client_Control_Panel\ClientControlPanel.py'
Enter 1 if you want to register. Enter 2 if you want to login. Enter 0 if you want to exit from program.
1
Enter you name here:
Jakob
Enter your surname here:
Patras
Enter your email here:
jakobpatras@gmail.com
Enter your login here:
jakob123
Enter your password here:
jakob123123
Enter your budget here:
50000
Client created successfully
Enter 1 if you want to register. Enter 2 if you want to login. Enter 0 if you want to exit from program.
```

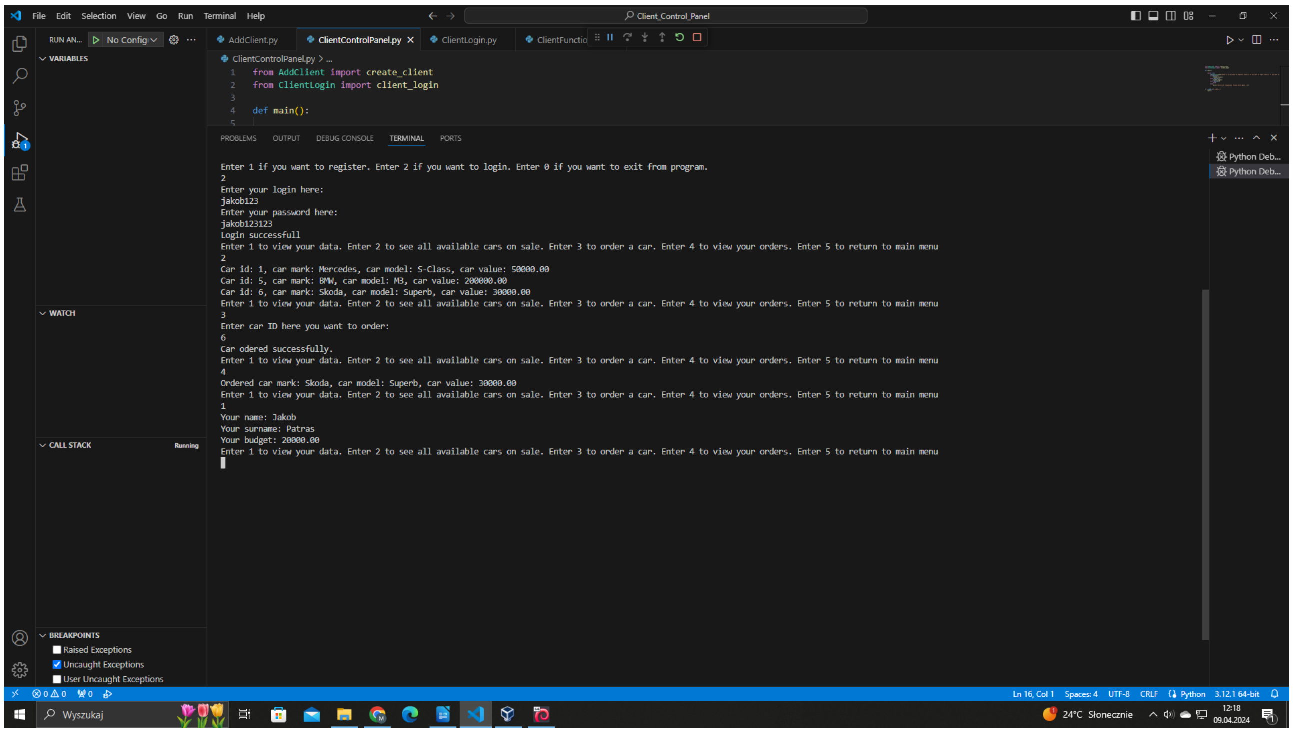
We can look to the database if client's credentials were added successfully. We can do that by issuing: SELECT * FROM clients WHERE client_login=”jakob123”;.



```
MariaDB [used_car_dealer]> SELECT * FROM clients WHERE client_login="jakob123";
+-----------+-------------+----------------+-----------------------+--------------+----------------------------------------------------------------------------------------------------+---------------+
| client_id | client_name | client_surname | client_email          | client_login | client_password                                                                                    | client_budget |
+-----------+-------------+----------------+-----------------------+--------------+----------------------------------------------------------------------------------------------------+---------------+
|        11 | Jakob       | Patras         | jakobpatras@gmail.com | jakob123     | $argon2id$v=19$m=65536,t=3,p=4$VXoLry4yV3YZVkeoKQCHNQ$BnIerhTJBc6ubFSg3OIIvcJEnkl/OZz+iW3aRS+rLaY |      50000.00 |
+-----------+-------------+----------------+-----------------------+--------------+----------------------------------------------------------------------------------------------------+---------------+
1 row in set (0.000 sec)

MariaDB [used_car_dealer]>
```

5.2 Logging in as a client

Now we will log in as a new created client and we will try to order a car. We will pass defined credentials jakob123 as a login and jakob123123 as a password.



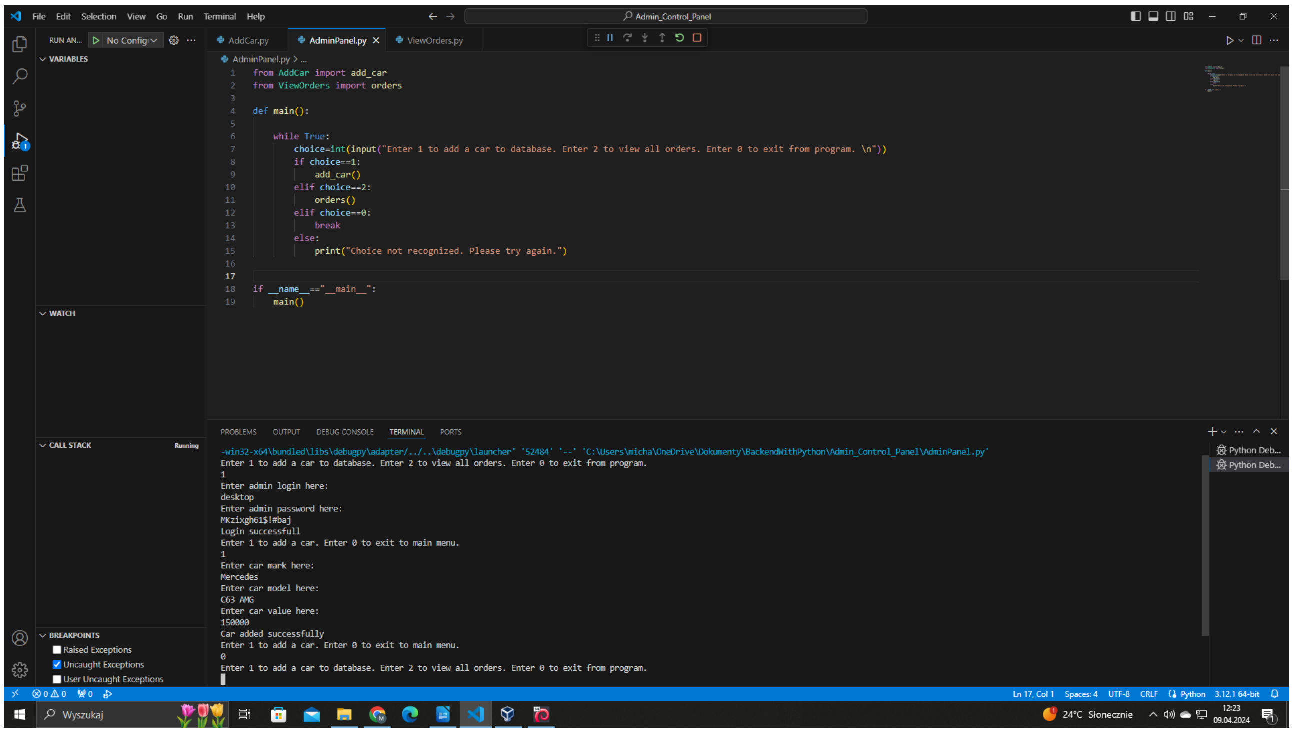
```python
from AddClient import create_client
from ClientLogin import client_login

def main():
```

```
Enter 1 if you want to register. Enter 2 if you want to login. Enter 0 if you want to exit from program.
2
Enter your login here:
jakob123
Enter your password here:
jakob123123
Login successfull
Enter 1 to view your data. Enter 2 to see all available cars on sale. Enter 3 to order a car. Enter 4 to view your orders. Enter 5 to return to main menu
2
Car id: 1, car mark: Mercedes, car model: S-Class, car value: 50000.00
Car id: 5, car mark: BMW, car model: M3, car value: 200000.00
Car id: 6, car mark: Skoda, car model: Superb, car value: 30000.00
Enter 1 to view your data. Enter 2 to see all available cars on sale. Enter 3 to order a car. Enter 4 to view your orders. Enter 5 to return to main menu
3
Enter car ID here you want to order:
6
Car odered successfully.
Enter 1 to view your data. Enter 2 to see all available cars on sale. Enter 3 to order a car. Enter 4 to view your orders. Enter 5 to return to main menu
4
Ordered car mark: Skoda, car model: Superb, car value: 30000.00
Enter 1 to view your data. Enter 2 to see all available cars on sale. Enter 3 to order a car. Enter 4 to view your orders. Enter 5 to return to main menu
1
Your name: Jakob
Your surname: Patras
Your budget: 20000.00
Enter 1 to view your data. Enter 2 to see all available cars on sale. Enter 3 to order a car. Enter 4 to view your orders. Enter 5 to return to main menu
```

As we can see we logged in as a jakob123. We bought a Skoda car. Our budget has changed from 50000 dollars to 20000 dollars, beacuse car value was 30000 dollars. Also we can list our orders. In this case we have only one order, which is Skoda Superb car.

5.3 Adding a new car

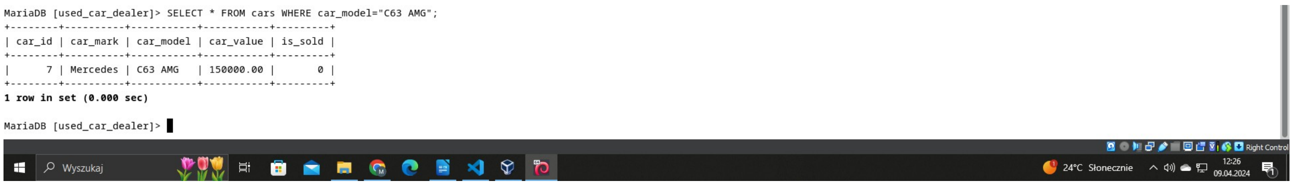
We will add a new car from admin control panel. We will define new car as a Mercedes C63 AMG with car value set to 150000 dollars.



```python
from AddCar import add_car
from ViewOrders import orders

def main():

    while True:
        choice=int(input("Enter 1 to add a car to database. Enter 2 to view all orders. Enter 0 to exit from program. \n"))
        if choice==1:
            add_car()
        elif choice==2:
            orders()
        elif choice==0:
            break
        else:
            print("Choice not recognized. Please try again.")


if __name__=="__main__":
    main()
```

```
-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '52484' '--' 'C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Admin_Control_Panel\AdminPanel.py'
Enter 1 to add a car to database. Enter 2 to view all orders. Enter 0 to exit from program.
1
Enter admin login here:
desktop
Enter admin password here:
MKzixgh61$!#baj
Login successfull
Enter 1 to add a car. Enter 0 to exit to main menu.
1
Enter car mark here:
Mercedes
Enter car model here:
C63 AMG
Enter car value here:
150000
Car added successfully
Enter 1 to add a car. Enter 0 to exit to main menu.
0
Enter 1 to add a car to database. Enter 2 to view all orders. Enter 0 to exit from program.
```

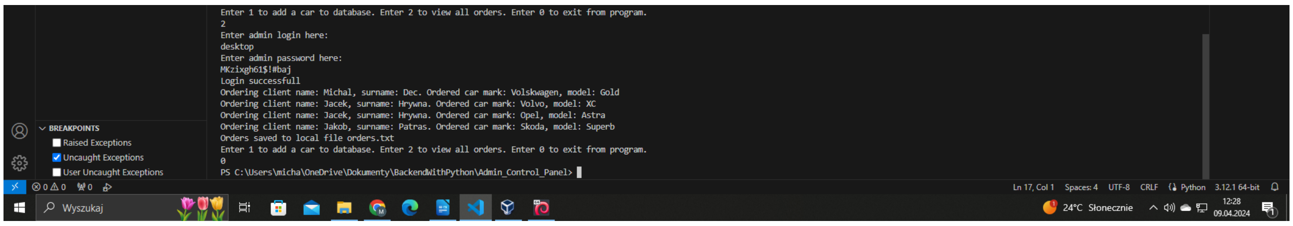
We can check if new car is added successfully to our database. We can achive that by SQL query: SELECT * FROM cars WHERE car_model=”C63 AMG”;.



```
MariaDB [used_car_dealer]> SELECT * FROM cars WHERE car_model="C63 AMG";
+--------+----------+-----------+-----------+---------+
| car_id | car_mark | car_model | car_value | is_sold |
+--------+----------+-----------+-----------+---------+
|      7 | Mercedes | C63 AMG   | 150000.00 |       0 |
+--------+----------+-----------+-----------+---------+
1 row in set (0.000 sec)

MariaDB [used_car_dealer]>
```

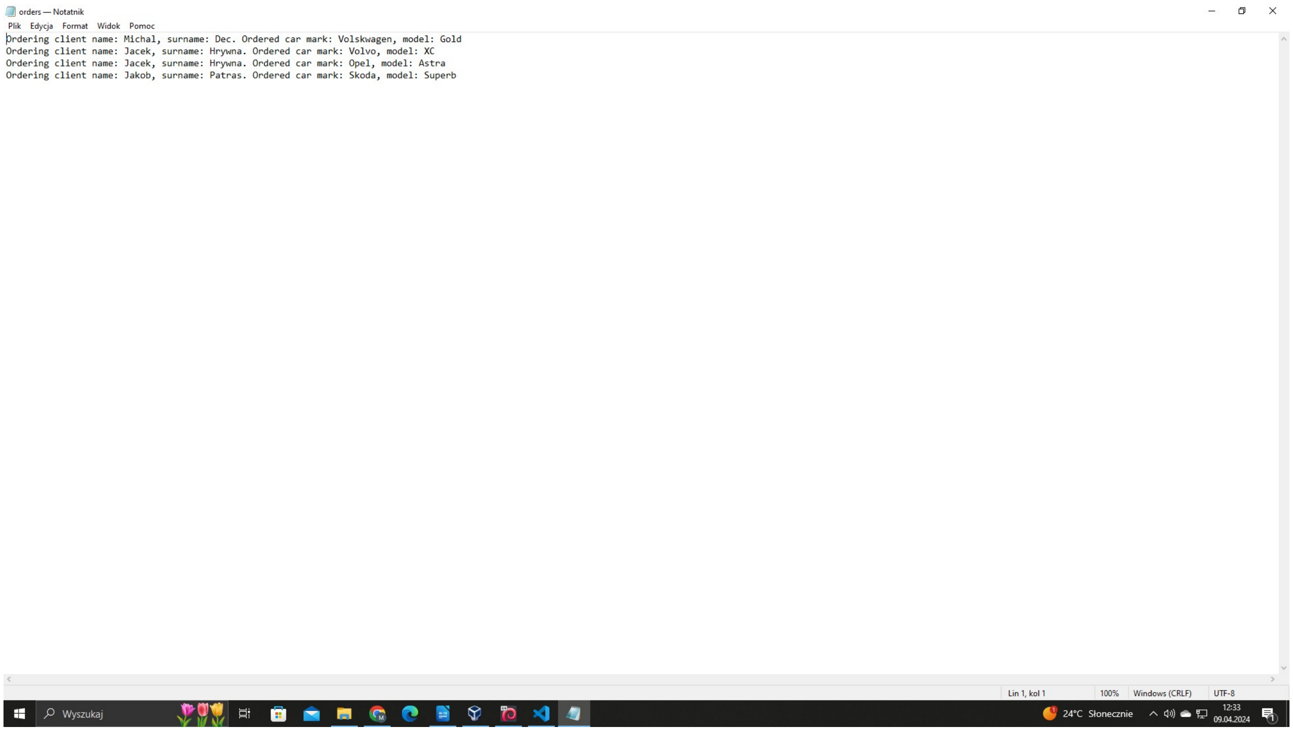
5.4 Viewing all clients orders

Now we will use orders() function from admin control panel. We will list all orders processed in database and save them to the local file named orders.txt.



```
Enter 1 to add a car to database. Enter 2 to view all orders. Enter 0 to exit from program.
2
Enter admin login here:
desktop
Enter admin password here:
MKzixgh61$!#baj
Login successfull
Ordering client name: Michal, surname: Dec. Ordered car mark: Volskwagen, model: Gold
Ordering client name: Jacek, surname: Hrywna. Ordered car mark: Volvo, model: XC
Ordering client name: Jacek, surname: Hrywna. Ordered car mark: Opel, model: Astra
Ordering client name: Jakob, surname: Patras. Ordered car mark: Skoda, model: Superb
Orders saved to local file orders.txt
Enter 1 to add a car to database. Enter 2 to view all orders. Enter 0 to exit from program.
0
PS C:\Users\micha\OneDrive\Dokumenty\BackendWithPython\Admin_Control_Panel>
```

We can also check if the orders were saved successfully to the local file. We have to open directory in Windows with admin control panel and look for text file named orders.txt.



```
Ordering client name: Michal, surname: Dec. Ordered car mark: Volskwagen, model: Gold
Ordering client name: Jacek, surname: Hrywna. Ordered car mark: Volvo, model: XC
Ordering client name: Jacek, surname: Hrywna. Ordered car mark: Opel, model: Astra
Ordering client name: Jakob, surname: Patras. Ordered car mark: Skoda, model: Superb
```

6. Materials and resources used for the need of this report

1) IDE: Visual Studio Code - https://code.visualstudio.com/
2) Python: Version 3.12.1 - https://www.python.org/downloads/
3) MariaDB: How to connect Python programs to MariaDB - https://mariadb.com/resources/blog/how-to-connect-python-programs-to-mariadb/
4) PyPi: Argon2 for Python - https://pypi.org/project/argon2-cffi/