Syrian Arab Republic
Lattakia - Tishreen University
Department of Communication and electrical engineering
5th , Network Programming : Homework No1

الجمهورية العربية السورية
اللاذقية -جامعة تشرين
كلية الهندسة الكهربائية والميكانيكية
قسم هندسة الاتصالات والالكترونيات
السنة الخامسة: وظيفة1 برمجة شبكات

Name: Haneen Moalla , Number: 2844 , Submitted To GitHub: https://hneen-moalla.github.io/Haneenm/

Question 1: Bank ATM Application with TCP Server/Client and Multi-threading

Project Description:

Build a TCP server and client Bank ATM application using Python. The server should handle

multiple client connections simultaneously using multi-threading. The application should

allow clients to connect, perform banking operations (such as check balance, deposit, and

withdraw), and receive their updated account status upon completion.

Requirements:

A. The server should be able to handle multiple client connections concurrently.

B. The server should maintain a set of pre-defined bank accounts with balances.

C. Each client should connect to the server and authenticate with their account details.

D. Clients should be able to perform banking operations: check balance, deposit money, and withdraw money.

E. The server should keep track of the account balances for each client.

F. At the end of the session, the server should send the final account balance to each client

قمنا بالبحث في المواقع المرفقة ومرجع المادة وتم التوصل الى عدة نقاط أساسية مفيدة في تطوير البرنامج:

-نحتاج لبرمجة المقابس باستخدام socket حيث واجهنا صعوبة في المزامنة بشكل صحيح بالرسائل بين المخدم والزبون.

-نحتاج لبرمجة الى التفرع باستخدام multithreading حيث عانينا من الحاجة الى ضمان تعديل البيانات بشكل صحيح وعدم جمود النظام.

-نحتاج الى قاعدة بيانات لتخزين المعلومات حيث تم استخدام مخدم mongodb.

الحاجة الى بناء واجهة مستخدم رسومية حيث تم استخدام PyQt5.

كود المخدم الرئيسي:

```
from Lib.server import MultiThreadedServer
from bank_server_protocol import BankServerProtocol
import configparser
import os

CONFIG_FILENAME = 'config.ini'

def read_config_file():
    '''
    Reads config file, which contains client and database name\n
    Config filename set as constant (config.ini)
    '''
    try:
        config = configparser.ConfigParser()
        curr_dir = os.path.dirname(os.path.abspath(__file__))
        initfile = os.path.join(curr_dir, CONFIG_FILENAME)
        config.read(initfile)
        client = config.get('DB_SETTINGS', 'CLIENT')
        database_name = config.get('DB_SETTINGS', 'DATABASE_NAME')
        host = config.get('SERVER_SETTINGS', 'HOST')
        port = config.get('SERVER_SETTINGS', 'PORT')
        return  host, int(port), client, database_name
```

```
    except Exception as e:
        print(e)

if __name__ == '__main__':
    host, port, client, database = read_config_file()
    protocol = BankServerProtocol(client, database)
    bank_server = MultiThreadedServer(ip=host, port=port,
protocol=protocol)    # default ip is localhost (ip=''), port 1234. To
change ip, give as parameter ip=xxx.xxx.xxx.xxx
    bank_server.listen()
```

كود عمل الوظائف الأساسية للبنك في المخدم:

```
from datetime import datetime
import random
import pymongo
from return_messages import *
from constants import *
from Lib.server import ServerProtocol

class BankServerProtocol(ServerProtocol):

    def __init__(self, client, database):
        self.client = pymongo.MongoClient(client)
        self.db = self.client.get_database(database)

    def process_request(self, input_msg):
        print('Received message from client: ', input_msg)
        arr = input_msg.split()
        result = self.perform_action(arr)
        return str(result)

    def perform_action(self, arr):

        username = arr[0]   # String we get according to protocol:
<USERNAME> <PIN> <ACTION> ...
        pin = int(arr[1])
        action = arr[2]

        customer = self.authenticate(username, pin)
        if customer is not None:

            if action == 'WITHDRAW':
                amount = arr[3]
                return self.withdraw(customer['cid'], float(amount))

            elif action == 'DEPOSIT':
```

```
                amount = arr[3]
                return self.deposit(customer['cid'], float(amount))

            elif action == 'CHANGE_PIN':
                new_pin = int(arr[3])
                return self.change_customer_pin(customer['cid'],
new_pin)

            elif action == 'GET_BALANCE':
                return self.get_customer_balance(customer['cid'])

            else:
                return ACTION_NOT_FOUND

        return WRONG_CREDENTIALS

    def authenticate(self, username, pin):
        '''
        Checks customer's credentials\n
        Takes customer's entered username, pin as arguments\n
        If username & pin match, returns True\n
        If username & pin do not match, returns False
        '''
        customer = self.find_customer(username)
        if customer is not None:
            r_pin = customer['pin']    # Get pin from database
            if r_pin == pin:
                return customer
        return None

    def find_customer(self, username):
        '''
        Checks if customer's username exists in customers' collection\n
        Takes username as argument\n
        Returns cid (customer's id) if customer exists\n
        Returns None if he doesn't
        '''
        customer = self.db.customer.find_one({'username': username})
        if customer is not None:
            return customer
        return None

    def get_customer_balance(self, cid):
        '''
        Informs customer about his balance\n
        Takes cid (customer id) as argument\n
        Returns balance if found\n
        Returns None if not found
```

```
        '''
        try:
            self.charge(cid, BALANCE_INFO_CHARGES,
BALANCE_INFO_CHARGES_DESCR)
            balance = self.db.balance.find_one({'cid': cid},
{'balance': 1})['balance']
            return format(balance, '.3f')
        except:
            return BALANCE_NOT_FOUND_ERR

        return None

    def charge(self, cid, amount, descr):
        '''
        Charges customer when is asking about balance information\n
        Takes cid (customer id), charge amount and charge description
as parameters
        '''
        try:
            balance_doc = {'$inc': {'balance': -amount}, '$set':
{'last_updated': datetime.today().strftime('%Y-%m-%d-%H:%M:%S')}}
            chid = self.db.charge.count_documents({})+1
            charge_doc = {'chid': chid, 'cid': cid, 'amount': amount,
'descr': descr, 'date': datetime.today().strftime('%Y-%m-%d-%H:%M:%S')}
            self.db.balance.update_one({'cid': cid}, balance_doc)
            self.db.charge.insert_one(charge_doc)
            print('Customer charged')
        except Exception as e:
            print(e)

    def insert_customer(self, username, full_name):
        '''
        Adds a customer to customers' collection\n
        Takes username as argument, generates id and pin\n
        Returns True if addition succeeds\n
        Returns False if addition fails
        '''
        try:
            cus = self.db.customer.find_one({'username': username})
            if cus is not None:
                return USERNAME_TAKEN_ERR
        except:
            print('Application crashed')
        try:
            cid = self.db.customer.count_documents({})+1
            customer = {'cid': cid, 'username': username, 'full_name':
full_name, 'pin': self.generate_pin()}
            self.db.customer.insert_one(customer)
```

```
            self.init_balance(cid)
            print(CUSTOMER_ADDITION_SUCCESS_MSG + ' ' + username)
            return CUSTOMER_ADDITION_SUCCESS_MSG
        except:
            return CUSTOMER_NOT_ADDED_ERR

    def delete_customer(self, cid=None, username=None):
        '''
        Removes a customer from customers' collection\n
        Takes id and/or username as arguments\n
        Returns True if removal succeeds\n
        Returns False if removal fails
        '''
        if username is None:    # if we track the document by id
            try:
                self.db.customer.delete_one({'cid': cid})
                print(CUSTOMER_REMOVAL_SUCCESS_MSG + ' with id '+cid)
                return CUSTOMER_REMOVAL_SUCCESS_MSG
            except:
                print(CUSTOMER_NOT_REMOVED_ERR)
                return CUSTOMER_NOT_REMOVED_ERR
        # else we track the document by username
        try:
            self.db.customer.delete_one({'username': username})
            print(CUSTOMER_REMOVAL_SUCCESS_MSG +' with username '
+username)
            return CUSTOMER_REMOVAL_SUCCESS_MSG
        except:
            return CUSTOMER_NOT_REMOVED_ERR

    def change_customer_pin(self, cid, new_pin):
        '''
        Changes customer's pin (based on id)\n
        with another random one
        '''
        try:
            self.db.customer.update_one({'cid': cid}, {'$set': {'pin':
new_pin}})
            return CUSTOMER_PIN_CHANGE_SUCCESS_MSG
        except:
            return CUSTOMER_PIN_CHANGE_FAILURE_ERR

    def generate_pin(self):
        '''
        Generates random 4-digit pin and returns it
        '''
        return random.randint(999, 9999)
```

```
    def init_balance(self, cid):
        '''
        Initializes new customer's balance
        '''
        try:
            bid = self.db.balance.count_documents({})+1
            balance_doc = {'bid': bid, 'cid': cid, 'balance': 0.0,
'last_updated': datetime.today().strftime('%Y-%m-%d-%H:%M:%S')}
            self.db.balance.insert_one(balance_doc)
            return True
        except:
            print(BALANCE_NOT_INIT_ERR)
            return False

    def withdraw(self, cid, amount):
        '''
        Customer withdraws from his account\n
        Takes cid and amount as arguements\n
        Returns True if withdrawal successful\n
        Returns False if withdrawal unsuccessful
        '''

        if amount <= 0:
            return AMOUNT_NOT_VALID_ERR

        if self.db.balance.find_one({'cid': cid}, {'balance':
1})['balance'] < amount:
            return BALANCE_NOT_ENOUGH_ERR

        if not self.check_banknotes(amount):
            return BANKNOTES_NOT_VALID_ERR

        if self.daily_withdrawal_limit_reached(cid, amount):
            return DAILY_WITHDRAWAL_LIMIT_ERR

        try:
            withdrawal_doc = {'wid':
self.db.withdraw.count_documents({})+1, 'amount': amount, 'cid': cid,
'time': datetime.today().strftime('%Y-%m-%d-%H:%M:%S')}
            balance_doc = {'$inc': {'balance': -float(amount)}, '$set':
{'last_updated': datetime.today().strftime('%Y-%m-%d-%H:%M:%S')}}
            self.db.withdraw.insert_one(withdrawal_doc)
            self.db.balance.update_one({'cid': cid}, balance_doc)
            print(WITHDRAWAL_SUCCESS_MSG)
            return WITHDRAWAL_SUCCESS_MSG
        except:
            print(WITHDRAWAL_FAILURE_ERR)
            return WITHDRAWAL_FAILURE_ERR
```

```
    def daily_withdrawal_limit_reached(self, cid, amount):
        '''
        Checks if customer reached his daily withdrawal limit (Set as
constant 850)\n
        Takes cid as parameter\n
        Returns True if limit reached\n
        Returns False if limit is not reached
        '''
        if amount > DAILY_WITHDRAWL_LIMIT:
            return False
        try:
            curr_date = datetime.today().strftime('%Y-%m-%d')
        except Exception as e:
            print(e)
        pipe = [{ "$match": { 'cid': { "$eq": cid } } }, { "$match": {
'time': { "$regex": '.*'+curr_date+'.*' } } }, {'$group': {'_id':
"$cid", 'total_amount': {'$sum': '$amount'}}}]
        results = list(self.db.withdraw.aggregate(pipeline=pipe)) # we
get a list with one dict inside (cid and amount that was withdrawn
today)
        if results:
            total_amount_withdrawn = results[0]['total_amount']
            if total_amount_withdrawn > DAILY_WITHDRAWL_LIMIT:
                return True
        return False

    def check_banknotes(self, amount):
        '''
        Checks if amount is divided by 20 or 50 or 70\n
        Returns True if it is\n
        Returns False if not
        '''
        return (amount % 20 == 0 or amount % 50 == 0 or amount % 70 ==
0)

    def deposit(self, cid, amount):
        '''
        Customer deposits into his account\n
        Takes cid and amount as arguements\n
        Returns True if deposition successful\n
        Returns False if deposition unsuccessful
        '''

        if amount <= 0:
            return AMOUNT_NOT_VALID_ERR

        try:
```

```
            deposition_doc = {'did':
self.db.deposit.count_documents({})+1, 'amount': amount, 'cid': cid,
'time': datetime.today().strftime('%Y-%m-%d-%H:%M:%S')}
            balance_doc = {'$inc': {'balance': float(amount)}, '$set':
{'last_updated': datetime.today().strftime('%Y-%m-%d-%H:%M:%S')}}
            self.db.deposit.insert_one(deposition_doc)
            self.db.balance.update_one({'cid': cid}, balance_doc)
            return DEPOSITION_SUCCESS_MSG
        except:
            return DEPOSITION_FAILURE_ERR
```

كود الزبون الرئيسي مع واجهة:

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtGui import QKeySequence
from PyQt5.QtCore import QCoreApplication
from PyQt5.QtMultimedia import QSound
from Lib.client import Client
from bank_client_protocol import BankClientProtocol
import os
from return_messages import errors

KEYS_STYLESHEET = 'background-color: rgb(206, 206, 206);'

BTN_STYLESHEET = 'QPushButton {background-color: qlineargradient(x1: 0,
y1: 0, x2: 0, y2: 1, stop: 0 rgb(120,120,120), stop: 1 rgb(80,80,80));
border: 1px solid rgb(20,20,20); color: rgb(230,230,230); padding: 4px
8px;} QPushButton:hover {background-color: rgb(70,110,130);}
QPushButton:pressed {border-color: rgb(90,200,255); padding: 1px -1px -
1px 1px; } QPushButton:checked {background-color: qlineargradient(x1:
0, y1: 0, x2: 0, y2: 1, stop: 0 rgb(40,150,200), stop: 1
rgb(90,200,255)); color: rgb(20,20,20);} QPushButton:checked:hover {
background-color: rgb(70,110,130);} QPushButton:disabled {background-
color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0
rgb(160,160,160), stop: 1 rgb(120,120,120)); border-color:
rgb(60,60,60); color: rgb(40,40,40);}'

ACTIONS = {'WITHDRAWAL': 'WITHDRAW', 'DEPOSITION': 'DEPOSIT', 'CHANGE
PIN': 'CHANGE_PIN', 'BALANCE INQUIRY':'GET_BALANCE'}

def beep():
    '''
    Creates a beep sound
    '''
    if os.name == 'nt':
```

```
        import winsound
        winsound.Beep(400, 250)
    else:
        sys.stdout.write("\a")

class NumButton(QtWidgets.QPushButton):

    '''
    Custom class for num keys
    We created this class to add num_signal
    '''

    num_signal = QtCore.pyqtSignal(QtWidgets.QPushButton)
    def __init__(self, parent=None):
        super(NumButton, self).__init__(parent)

    def mousePressEvent(self, event):
        beep()
        self.clicked.emit(True)
        self.num_signal.emit(self)

class CustomQLabel(QtWidgets.QLabel):
    '''
    Custom QLabel class with mouse press event
    '''
    clicked=QtCore.pyqtSignal()
    def __init__(self, parent=None):
        QtWidgets.QLabel.__init__(self, parent)

    def mousePressEvent(self, ev):
        self.clicked.emit()

class Ui_MainWindow(object):
    def __init__(self):
        self.username = None
        self.pin = None
        self.action = None  # we want to know what action the user is
going to perform
        self.new_pin = None
        self.amount = None

    def setupUi(self, MainWindow):
        self.MainWindow = MainWindow
        self.MainWindow.setObjectName("MainWindow")
        self.MainWindow.resize(958, 669)

        self.curr_screen = 0    # initial screen is 0
```

```
        self.font = QtGui.QFont()
        self.font.setFamily("Calibri")
        self.font.setPointSize(12)
        self.centralwidget = QtWidgets.QWidget(self.MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.screen_panel = QtWidgets.QFrame(self.centralwidget)
        self.screen_panel.setGeometry(QtCore.QRect(250, 10, 400, 300))
        self.screen_panel.setAutoFillBackground(False)
        self.screen_panel.setStyleSheet("background-color: rgb(0, 177,
51);")
        self.screen_panel.setFrameShape(QtWidgets.QFrame.Panel)
        self.screen_panel.setFrameShadow(QtWidgets.QFrame.Raised)
        self.screen_panel.setObjectName("screen_panel")
        self.gridLayoutWidget = QtWidgets.QWidget(self.screen_panel)
        self.gridLayoutWidget.setGeometry(QtCore.QRect(0, 0, 401, 301))
        self.gridLayoutWidget.setObjectName("gridLayoutWidget")
        self.gridLayout = QtWidgets.QGridLayout(self.gridLayoutWidget)
        self.gridLayout.setContentsMargins(0, 0, 0, 0)
        self.gridLayout.setObjectName("gridLayout")
        self.change_pin_btn =
QtWidgets.QPushButton(self.gridLayoutWidget)
        self.change_pin_btn.setObjectName("change_pin_btn")
        self.change_pin_btn.setStyleSheet(BTN_STYLESHEET)
        self.gridLayout.addWidget(self.change_pin_btn, 1, 2, 1, 1)
        self.withdraw_btn =
QtWidgets.QPushButton(self.gridLayoutWidget)
        self.withdraw_btn.setObjectName("withdraw_btn")
        self.withdraw_btn.setStyleSheet(BTN_STYLESHEET)
        self.gridLayout.addWidget(self.withdraw_btn, 0, 0, 1, 1)
        self.deposit_btn = QtWidgets.QPushButton(self.gridLayoutWidget)
        self.deposit_btn.setObjectName("deposit_btn")
        self.deposit_btn.setStyleSheet(BTN_STYLESHEET)
        self.gridLayout.addWidget(self.deposit_btn, 1, 0, 1, 1)
        self.balance_btn = QtWidgets.QPushButton(self.gridLayoutWidget)
        self.balance_btn.setObjectName("balance_btn")
        self.balance_btn.setStyleSheet(BTN_STYLESHEET)
        self.change_pin_btn.clicked.connect(self.create_username_entry_
screen)
        self.deposit_btn.clicked.connect(self.create_username_entry_scr
een)
        self.balance_btn.clicked.connect(self.create_username_entry_scr
een)
        self.withdraw_btn.clicked.connect(self.create_username_entry_sc
reen)
        self.change_pin_btn.clicked.connect(beep)
        self.deposit_btn.clicked.connect(beep)
        self.balance_btn.clicked.connect(beep)
        self.withdraw_btn.clicked.connect(beep)
```

```
        self.gridLayout.addWidget(self.balance_btn, 0, 2, 1, 1)
        spacerItem = QtWidgets.QSpacerItem(100, 10,
QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Minimum)
        self.gridLayout.addItem(spacerItem, 0, 1, 1, 1)
        self.movie =
QtGui.QMovie(os.path.dirname(os.path.realpath(__file__))+'\images\\atm.
gif')
        self.movie.setCacheMode(QtGui.QMovie.CacheAll)
        self.movie_lbl = CustomQLabel(self.centralwidget)
        self.movie_lbl.setGeometry(self.screen_panel.geometry())
        self.movie_lbl.setMinimumSize(self.screen_panel.size())
        self.movie_lbl.setMovie(self.movie)
        self.movie_lbl.setSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Expanding)
        self.movie_lbl.clicked.connect(self.on_click_gif)
        self.movie.start()
        self.keys_panel = QtWidgets.QFrame(self.centralwidget)
        self.keys_panel.setGeometry(QtCore.QRect(270, 330, 361, 291))
        self.keys_panel.setStyleSheet("background-color: rgb(154, 154,
154);")
        self.keys_panel.setFrameShape(QtWidgets.QFrame.Panel)
        self.keys_panel.setFrameShadow(QtWidgets.QFrame.Raised)
        self.keys_panel.setObjectName("keys_panel")

        self.init_btns()
        self.set_num_key_slots()

        self.clear_btn = QtWidgets.QPushButton(self.keys_panel)
        self.clear_btn.setGeometry(QtCore.QRect(280, 10, 61, 61))
        self.clear_btn.setStyleSheet("background-color: rgb(207, 207,
0);")
        self.clear_btn.setObjectName("clear_btn")
        self.clear_btn.clicked.connect(beep)
        self.clear_btn.clicked.connect(self.clear_operation)
        self.ok_btn = QtWidgets.QPushButton(self.keys_panel)
        self.ok_btn.setGeometry(QtCore.QRect(280, 150, 61, 61))
        self.ok_btn.setStyleSheet("background-color: rgb(0, 170, 0);")
        self.ok_btn.setObjectName("ok_btn")
        self.ok_btn.clicked.connect(beep)
        self.ok_btn.clicked.connect(self.okay_pressed)
        self.actionPressOk =
QtWidgets.QShortcut(QKeySequence("Return"), MainWindow)
        self.actionPressOk.activated.connect(self.okay_pressed)
        self.cancel_btn = QtWidgets.QPushButton(self.keys_panel)
        self.cancel_btn.setGeometry(QtCore.QRect(280, 80, 61, 61))
        self.cancel_btn.setStyleSheet("background-color: rgb(170, 0,
0);")
        self.cancel_btn.setObjectName("cancel_btn")
```

```
        self.cancel_btn.clicked.connect(beep)
        self.cancel_btn.clicked.connect(self.cancel_operation)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 958, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def init_btns(self):
        self.num1_btn = NumButton(self.keys_panel)
        self.num1_btn.setGeometry(QtCore.QRect(10, 10, 61, 61))
        self.num1_btn.setFont(self.font)
        self.num1_btn.setAutoFillBackground(False)
        self.num1_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num1_btn.setObjectName("num1_btn")
        self.num4_btn = NumButton(self.keys_panel)
        self.num4_btn.setGeometry(QtCore.QRect(10, 80, 61, 61))
        self.num4_btn.setFont(self.font)
        self.num4_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num4_btn.setObjectName("num4_btn")
        self.num7_btn = NumButton(self.keys_panel)
        self.num7_btn.setGeometry(QtCore.QRect(10, 150, 61, 61))
        self.num7_btn.setFont(self.font)
        self.num7_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num7_btn.setObjectName("num7_btn")
        self.num2_btn = NumButton(self.keys_panel)
        self.num2_btn.setGeometry(QtCore.QRect(100, 10, 61, 61))
        self.num2_btn.setFont(self.font)
        self.num2_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num2_btn.setObjectName("num2_btn")
        self.num8_btn = NumButton(self.keys_panel)
        self.num8_btn.setGeometry(QtCore.QRect(100, 150, 61, 61))
        self.num8_btn.setFont(self.font)
        self.num8_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num8_btn.setObjectName("num8_btn")
        self.num5_btn = NumButton(self.keys_panel)
        self.num5_btn.setGeometry(QtCore.QRect(100, 80, 61, 61))
        self.num5_btn.setFont(self.font)
        self.num5_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num5_btn.setObjectName("num5_btn")
        self.num3_btn = NumButton(self.keys_panel)
        self.num3_btn.setGeometry(QtCore.QRect(190, 10, 61, 61))
```

```
        self.num3_btn.setFont(self.font)
        self.num3_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num3_btn.setObjectName("num3_btn")
        self.num9_btn = NumButton(self.keys_panel)
        self.num9_btn.setGeometry(QtCore.QRect(190, 150, 61, 61))
        self.num9_btn.setFont(self.font)
        self.num9_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num9_btn.setObjectName("num9_btn")
        self.num6_btn = NumButton(self.keys_panel)
        self.num6_btn.setGeometry(QtCore.QRect(190, 80, 61, 61))
        self.num6_btn.setFont(self.font)
        self.num6_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num6_btn.setObjectName("num6_btn")
        self.num0_btn = NumButton(self.keys_panel)
        self.num0_btn.setGeometry(QtCore.QRect(100, 220, 61, 61))
        self.num0_btn.setFont(self.font)
        self.num0_btn.setStyleSheet(KEYS_STYLESHEET)
        self.num0_btn.setObjectName("num0_btn")

    def set_num_key_slots(self):
        '''
        We set the slots in order to know
        what to type when a specific num key is pressed
        '''
        self.num0_btn.num_signal.connect(self.type_num)
        self.num1_btn.num_signal.connect(self.type_num)
        self.num2_btn.num_signal.connect(self.type_num)
        self.num3_btn.num_signal.connect(self.type_num)
        self.num4_btn.num_signal.connect(self.type_num)
        self.num5_btn.num_signal.connect(self.type_num)
        self.num6_btn.num_signal.connect(self.type_num)
        self.num7_btn.num_signal.connect(self.type_num)
        self.num8_btn.num_signal.connect(self.type_num)
        self.num9_btn.num_signal.connect(self.type_num)

    def on_click_gif(self):
        '''
        When the animation is clicked, show main menu
        '''
        self.movie_lbl.setParent(None)
        self.curr_screen = 1    # menu screen is 1

    def create_username_entry_screen(self):
        '''
        When user chooses to proceed with an operation,
        create & show the pin enter screen (screen 2)
        '''
```

```
        self.curr_screen = 2    # pin entry screen is 2

        self.action = ACTIONS[self.MainWindow.sender().text().upper()]

        self.balance_btn.setParent(None)
        self.withdraw_btn.setParent(None)
        self.deposit_btn.setParent(None)
        self.change_pin_btn.setParent(None)
        self.username_lineedit =
QtWidgets.QLineEdit(self.gridLayoutWidget)
        self.username_lineedit.setMaximumSize(QtCore.QSize(210, 20))
        self.username_lineedit.setStyleSheet("background-color:
rgb(255, 255, 255);")
        self.username_lineedit.setAlignment(QtCore.Qt.AlignCenter)
        self.username_lineedit.setObjectName("username_lineedit")
        self.gridLayout.addWidget(self.username_lineedit, 1, 1, 1, 1)
        self.enter_username_lbl =
QtWidgets.QLabel(self.gridLayoutWidget)
        self.enter_username_lbl.setMaximumSize(QtCore.QSize(210, 30))
        self.enter_username_lbl.setText("Please enter your username")
        self.gridLayout.addWidget(self.enter_username_lbl, 0, 1, 1, 1)
        font = QtGui.QFont()
        font.setPointSize(16)
        self.enter_username_lbl.setFont(font)
        self.enter_username_lbl.setStyleSheet("color: rgb(255, 255,
255);")
        self.enter_username_lbl.setObjectName("enter_username_lbl")

    def create_pin_entry_screen(self):
        '''
        When user chooses to proceed with an operation,
        create & show the pin enter screen (screen 2)
        '''

        self.curr_screen = 3    # pin entry screen is 3

        self.gridLayout.removeWidget(self.username_lineedit)
        self.pin_lineedit = self.username_lineedit
        self.pin_lineedit.clear()
        self.pin_lineedit.setReadOnly(True)
        self.pin_lineedit.setInputMethodHints(QtCore.Qt.ImhMultiLine)
        self.pin_lineedit.setEchoMode(QtWidgets.QLineEdit.Password)
        self.pin_lineedit.textChanged.connect(self.on_pin_text_changed)
        self.gridLayout.addWidget(self.pin_lineedit, 1, 1, 1, 1)
        self.enter_pin_lbl = self.enter_username_lbl
        self.enter_pin_lbl.setMaximumSize(QtCore.QSize(210, 30))
        self.enter_pin_lbl.setText("Please enter your pin")
        self.gridLayout.addWidget(self.enter_pin_lbl, 0, 1, 1, 1)
```

```
        self.enter_pin_lbl.setObjectName("enter_pin_lbl")

    def create_new_pin_enter_screen(self):
        '''
        If user chooses to change pin,
        create & show the new pin enter screen
        '''

        self.curr_screen = 5    # create new pin screen is 5

        try:
            self.gridLayout.removeWidget(self.pin_lineedit) # deleting
previous screen's widgets
            self.gridLayout.removeWidget(self.enter_pin_lbl)

            self.enter_new_pin_lbl = self.enter_pin_lbl
            self.new_pin_lineedit = self.pin_lineedit

            self.enter_new_pin_lbl.setMaximumSize(QtCore.QSize(250,
30))
            self.new_pin_lineedit.setMaximumSize(QtCore.QSize(250, 20))
            self.enter_new_pin_lbl.setText("Please enter your new pin")
            self.new_pin_lineedit.clear()

            self.gridLayout.addWidget(self.enter_new_pin_lbl, 0, 1, 1,
1)
            self.gridLayout.addWidget(self.new_pin_lineedit, 1, 1, 1,
1)
        except Exception as e:
            print(e)

    def create_amount_entry_screen(self):
        '''
        If the action is deposition or withdrawal,
        the enter amount screen is showed to user
        '''
        self.curr_screen = 4    # amount entry screen is 4
        try:
            self.gridLayout.removeWidget(self.pin_lineedit) # deleting
previous screen's widgets
            self.gridLayout.removeWidget(self.enter_pin_lbl)

            self.enter_amount_lbl = self.enter_pin_lbl
            self.amount_lineedit = self.pin_lineedit

            self.gridLayout.addWidget(self.enter_amount_lbl, 0, 1, 1,
1)
            self.gridLayout.addWidget(self.amount_lineedit, 1, 1, 1, 1)
```

```
            self.enter_amount_lbl.setMaximumSize(QtCore.QSize(240, 30))
            self.enter_amount_lbl.setText("Please enter an amount")
            self.amount_lineedit.disconnect()
            self.amount_lineedit.setMaximumSize(QtCore.QSize(220, 20))
            self.amount_lineedit.setEchoMode(QtWidgets.QLineEdit.Normal
)
            self.amount_lineedit.clear()

        except Exception as e:
            print(e)

    def create_response_screen(self, input_message):
        '''
        After all information is entered by user,
        we need to create the screen, in which we will
        show him the response of his request
        '''
        prev_screen = self.curr_screen  # keep previous screen number
(we need to know which items to remove dynamically from panel)
        self.curr_screen = 6    # response screen is 5
        try:
            if prev_screen == 4:    # if before response we were in
enter amount screen (deposition or withdrawal)
                self.gridLayout.removeWidget(self.enter_amount_lbl) #
deleting previous screen's widgets
                self.gridLayout.removeWidget(self.amount_lineedit)
                self.amount_lineedit.setParent(None)

                self.response_lbl = self.enter_amount_lbl

            else:   # if before response we were in enter pin screen
                self.gridLayout.removeWidget(self.enter_pin_lbl) #
deleting previous screen's widgets
                self.gridLayout.removeWidget(self.pin_lineedit)
                self.pin_lineedit.setParent(None)

                self.response_lbl = self.enter_pin_lbl

            if input_message in errors:
                self.response_lbl.setText(input_message)

            else: self.response_lbl.setText('Your Balance is
'+input_message) if self.action == 'GET_BALANCE' else
self.response_lbl.setText(input_message)

            self.gridLayout.addWidget(self.response_lbl, 0, 1, 1, 1)
```

```
            self.response_lbl.setMaximumSize(QtCore.QSize(240, 30))
        except Exception as e:
            print(e)

    def on_pin_text_changed(self):
        '''
        We ensure that the user wont enter a pin whose
        length is more that 4 digits
        '''
        try:
            if len(self.pin_lineedit.text()) > 4:
                self.pin_lineedit.setText(self.pin_lineedit.text()[:-
1])
        except Exception as e:
            print(e)

    def okay_pressed(self):
        '''
        After ok pressed
        '''
        if self.curr_screen == 2:   # if ok pressed while we are in
enter username screen
            self.username = self.username_lineedit.text()
            self.create_pin_entry_screen()
        elif self.curr_screen == 3: # if ok pressed while we are in
enter pin screen
            self.pin = self.pin_lineedit.text()
            if self.action == 'DEPOSIT' or self.action == 'WITHDRAW':
                self.create_amount_entry_screen()
            elif self.action == 'CHANGE_PIN':
                self.create_new_pin_enter_screen()
            elif self.action == 'GET_BALANCE':
                self.establish_connection()
        elif self.curr_screen == 4:    # if ok pressed while we are in
enter amount screen or enter new pin screen
            self.amount = self.amount_lineedit.text()
            self.establish_connection()
        elif self.curr_screen == 5:
            self.new_pin = self.new_pin_lineedit.text()
            self.establish_connection()

    def type_num(self, key):
        '''
        When a key is pressed, fill qlineEdit with appropriate num
        '''
        try:
            tmp_str = self.pin_lineedit.text()
            tmp_str += ''.join(i for i in key.text() if i.isdigit())
```

```
            self.pin_lineedit.setText(tmp_str)
        except Exception as e:
            print(e)

    def clear_operation(self):
        '''
        When clear btn is clicked, clear the qlineEdit
        '''
        try:
            if self.curr_screen == 2:
                self.username_lineedit.clear()
            elif self.curr_screen == 3:
                self.pin_lineedit.clear()
            elif self.curr_screen == 4:
                self.amount_lineedit.clear()
            elif self.curr_screen == 5:
                self.new_pin_lineedit.clear()
        except Exception as e:
            print(e)

    def cancel_operation(self):
        '''
        If cancel btn is clicked, initialize again the ui
        '''
        try:
            self.setupUi(self.MainWindow)
        except Exception as e:
            print(e)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        self.MainWindow.setWindowTitle(_translate("MainWindow", "ATM
Client"))
        self.num1_btn.setText(_translate("MainWindow", "1"))
        self.num4_btn.setText(_translate("MainWindow", "4"))
        self.num7_btn.setText(_translate("MainWindow", "7"))
        self.num2_btn.setText(_translate("MainWindow", "2"))
        self.num8_btn.setText(_translate("MainWindow", "8"))
        self.num5_btn.setText(_translate("MainWindow", "5\n""-"))
        self.num3_btn.setText(_translate("MainWindow", "3"))
        self.num9_btn.setText(_translate("MainWindow", "9"))
        self.num6_btn.setText(_translate("MainWindow", "6"))
        self.balance_btn.setText(_translate("MainWindow", "Balance
Inquiry"))
        self.withdraw_btn.setText(_translate("MainWindow",
"Withdrawal"))
        self.deposit_btn.setText(_translate("MainWindow",
"Deposition"))
```

```
        self.change_pin_btn.setText(_translate("MainWindow", "Change
Pin"))
        self.clear_btn.setText(_translate("MainWindow", "CLEAR"))
        self.ok_btn.setText(_translate("MainWindow", "OK"))
        self.cancel_btn.setText(_translate("MainWindow", "CANCEL"))
        self.num0_btn.setText(_translate("MainWindow", "0"))

    def establish_connection(self):
        '''
        Establishes connection with the server
        when user has entered all the info needed for
        a request
        '''
        try:
            protocol = BankClientProtocol(self.username, self.pin,
self.action, self.amount, self.new_pin)
            bank_client = Client(protocol=protocol)
            response_txt = bank_client.open()
        except Exception as e:
            print(e)

        self.create_response_screen(response_txt)

if __name__ == '__main__':

    import sys

    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Question 2: Simple Website Project with Python Flask Framework (you have choice to use Django or any Other Deferent Useful Python Project “from provide Project Links”)

Create a simple website with multiple pages using Flask, HTML, CSS, and Bootstrap. The website should demonstrate your understanding of web design principles.

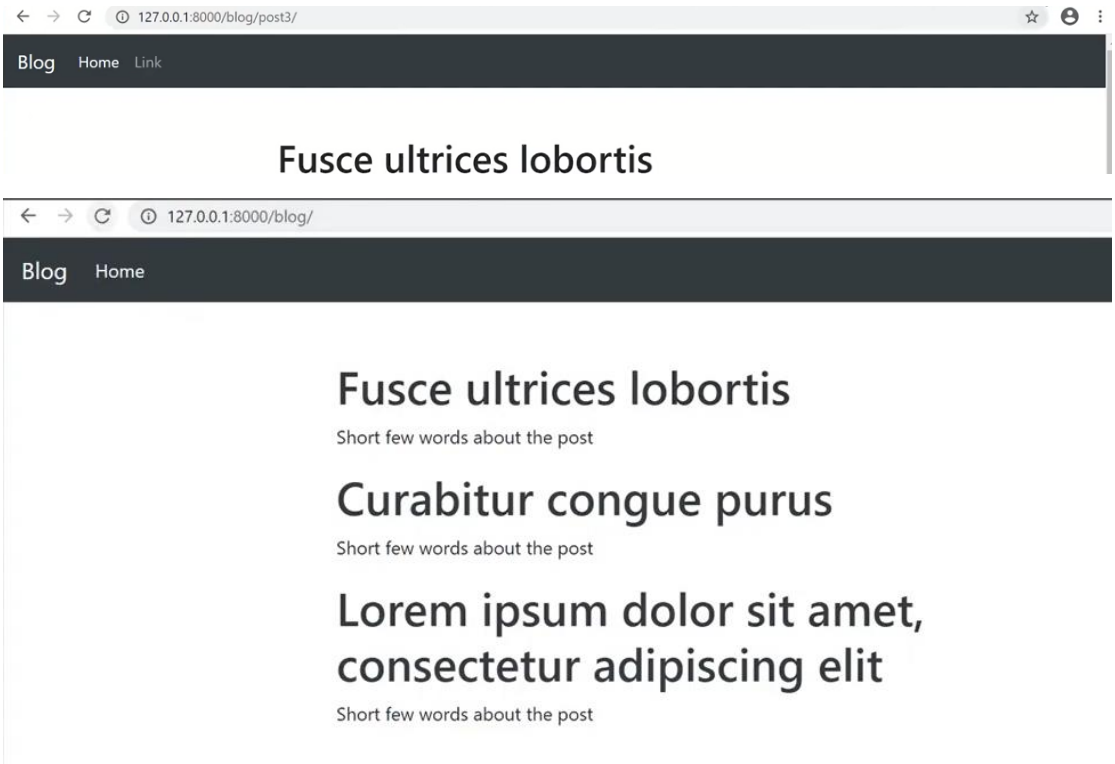
تم استخدام منصة Django في تطوير الموقع حيث قمنا بتطوير مدونة بسيطة.

كود تضمين لغة bootstrap في الموقع:

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.
min.css" integrity="sha384-
Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">

    <title>CMS Primer</title>
  </head>
  <body>
    {% include 'core/navbar.html' %}

    {% block content %}

    {% endblock %}

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
integrity="sha384-
J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.
js" integrity="sha384-
Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
crossorigin="anonymous"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.mi
n.js" integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
crossorigin="anonymous"></script>

  {% include 'core/footer.html' %}
  </body>
</html>
```

كود بايثون لخدمات الموقع الأساسية:

```
from .models import Core
from django.views.generic import ListView, DetailView, UpdateView,
DeleteView, CreateView
from django.urls import reverse_lazy


class IndexView(ListView):
    model = Core
    template_name = 'core/index.html'
    context_object_name = 'index'


class SingleView(DetailView):
    model = Core
    template_name = 'core/single.html'
    context_object_name = 'post'


class PostsView(ListView):
    model = Core
    template_name = 'core/posts.html'
    context_object_name = 'post_list'


class AddView(CreateView):
    model = Core
    template_name = 'core/add.html'
    fields = '__all__'
    success_url = reverse_lazy('core:posts')


class EditView(UpdateView):
    model = Core
    template_name = 'core/edit.html'
    fields = '__all__'
    pk_url_kwarg = 'pk'
    success_url = reverse_lazy('core:posts')


class Delete(DeleteView):
    model = Core
    pk_url_kwarg = 'pk'
    success_url = reverse_lazy('core:posts')
    template_name = 'core/confirm-delete.html'
```


127.0.0.1:8000/blog/post3/
Blog
Home
Link
Fusce ultrices lobortis
127.0.0.1:8000/blog/
Blog
Home
Fusce ultrices lobortis
Short few words about the post
Curabitur congue purus
Short few words about the post
Lorem ipsum dolor sit amet, consectetur adipiscing elit
Short few words about the post