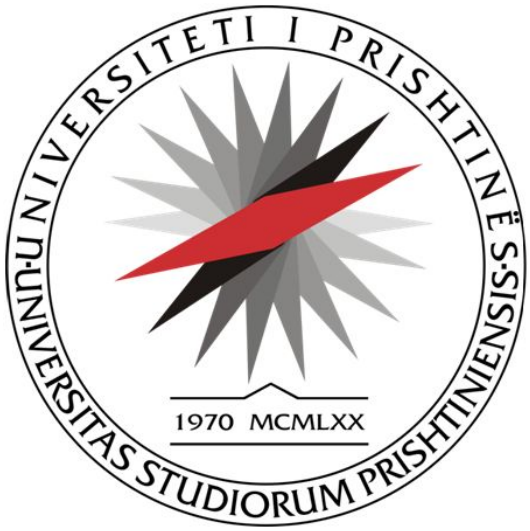

“PROGRAMIMI ME SOKETA”

Punoi: ***Ylber Gashi***

Fakulteti i Inxhinierise Elektrike dhe Kompjuterike

Lënda: ***Rrjeta Kompjuterike***

Ligjerata: ***Prof. Dr. Blerim Rexha***

Ushtrime: ***MSc. Haxhi Lajqi Phd cand.***

Data: **08/04/2020**

Përmbajtja:

1. Hyrje

1.1. Tools

Për realizimin e këtij projekti janë përdorur këto vegla:

- Sistemi operativ: **Windows 10**
- Gjuha programuese: **Python 3.6.6**
- IDE: **PyCharm Community Edition by JetBrains 2019.3.3**

1.2. Pershkrim i shkurtër i projektit

Ky është projekti i parë në lëndën Rrjeta Kompjuterike. Qëllimi i këtij projekti ishte që të arrinim të programonim Socket-at në gjuhën programuese Python, duke i shtuar funksionalitete shtesë për dallim nga shembujt e zhvilluar gjatë ushtrimeve.

Në kuadër të këtij projekti përfshihen dy palë Socket-a, ku njëra palë realizohet me anë të protokolit TCP, ndërsa tjetra duke e shfrytëzuar protokolin UDP.

1.3. TCP Protokoli

TCP (Transmission Control Protocol) është një protokol komunikimi i cili e përcakton se si të krijohet e të menaxhohet një lidhje në rrjet, me anë të së cilës, programe të ndryshme mund të shpërndajmë të dhëna ndërmjet vete.

Karakteristikë kryesore e TCP protokolit është se ky e siguron lidhjen para se të fillojë bartja e të dhënave, karakteristikë kjo e cila është thelbesore per shumicën e programeve qe përdoren sot jashtëzakonisht shumë.

TCP konsiderohet si connection-oriented dhe te dhenat i dërgon të renditura sipas një rendi të specifikuar, pra pa e humbur integritetin e të dhënave.

TCP përdoret nga protokolet tjera si HTTP, HTTPs, FTP, SMTP dhe Telnet. Shpejtësia e dërgimit të të dhënave nëpërmjet TCP është më e vogel se ajo e UDP, për shkak të kontrollimeve shtesë që bëhen nga TCP.

TCP protokoli e krijon lidhjen sipas “three way handshake” (SYN, SYN-ACK, ACK).

1.4. UDP Protokoli

UDP (User Datagram Protocol) protokoli në anën tjetër është një protokol komunikimi i cili përdoret në raste të caktuara, si per shembull kur kemi të bëjmë me sherbimet “real-time”, ku nevojitet transmetim i shpejtë i të dhënave, siç janë Streaming Services, Multiplayer Gaming, etj. UDP konsiderohet si “connectionless” për shkak se një lidhje mund të përfundojë menjëherë pas dërgimit të të dhënave nga njëra palë, duke mos u brengosur kështu se a kanë mbërritur të dhënat.

UDP përdoret nga protokole tjera si: DNS, DHCP, TFTP, SNMP, RIP, VOIP.

UDP, për dallim nga TCP protokoli, nuk brengoset për renditjen e paketave të të dhënave. Kjo përgjegjesi i mbetet nivelit të aplikacionit.

UDP është më i shpejtë sesa TCP, pikërisht nga fakti që nuk ka kontrollime shtesë për gabime eventuale ne komunikim. Po ashtu, vlen të përmendet që lidhjet në UDP nuk mundësohen nga “handshakes” sikur tek TCP protokoli.

2. Metodat

2.1. Metodat e kërkuara

Metodat e kërkuara në projekt janë:

- IPADDRESS
- PORT
- COUNT
- REVERSE
- PALINDROME
- TIME
- GAME
- CONVERT
- GCF

Së pari kam filluar të krijoj logjikën e metodave, para fillimit te krijimit te socketave. Për secilen metodë do ta përshkruaj kerkesën e metodës dhe realizimin e tyre në Python.

2.1.1. IPADDRESS

-Funksioni:

Kthen IP Adresen e klientit përkatës, e cila merret nga tuple value **address**, e cila permban në vete IP adresen dhe portin e klientit të lidhur.

-Kodi në Python:

```
def PORTI(address):
    return address[1]
```

Fig. 1

2.1.2. PORT

-Funksioni:

Kthen portin e klinetit përkatës, në të njejtën mënyrë sikur tek IPADDRESS, përveç se ketu indeksi 1 tek variabla address e përmban portin.

-Kodi në Python:

```
def PORTI(address):
    return address[1]
```

Fig. 2

2.1.3. COUNT

-Funksioni:

Numëron zanoret dhe bashkëtingëlloret e një teksti të dhënë nga klienti. Teksti jepet si parameter. Së pari me **.lower()** e kthejme ate tekst në shkronja të vogla për ta pasur më lehte tek kontrollimi i zanoreve dhe bashkëtingëlloreve, duke e pasur parasysh që shkronjat e mëdhaja kanë numër të ndryshëm nga ato të voglat në **ASCII code**. Ecuria e kontrollimit pra është bërë pra duke e kontrolluar për çdo shkronjë për zanore, ndërsa për bashkëtingëllore, nëpërmjet vlerave në ASCII code.

-Kodi në Python:

```
def COUNT(text):
    text = text.lower()
    nrz = 0
    nrb = 0
    for x in text:
        if x == 'a' or x == 'e' or x == 'i' or x == 'u' or x == 'o':
            nrz += 1
        elif x >= 'a' and x <= 'z':
            nrb += 1
    final = "Teksti i pranuar permban " + str(nrz) + " zanore dhe " + str(nrb) + " bashketingellore."
    return final
```

Fig. 3

2.1.4. REVERSE

-Funksioni:

E kthen një tekst në renditje të kundërt ndaj asaj të dhënë. Realizimi i kesaj metode është bërë duke iteruar një for loop me një rang sa gjatësia e tekstit hyrës. Për çdo iterim, ne i'a shtojmë shkronjën përkatëse të tekstit hyrës tek indeksi [gjatësia e tekstit - 1] një stringu bosh të krijuar më lartë. Në fund, e kthejme si rezultat atë string, të cilit i largohen hapesirat në skaje.

-Kodi në Python:

```
def REVERSE(text):
    backw = ""
    gjatesia = len(text)
    for x in range(gjatesia):
        backw += text[gjatesia - 1]
        gjatesia -= 1
    return backw.strip()  # e kthen tekstin reverse me hapesirat e fillimit dhe te fundit te larguara
```

Fig. 4

2.1.5. PALINDROME

-Funksioni:

Kjo metodë na e kthen TRUE nëse teksti i dhënë si parametër është palindrome, ndërsa FALSE, për rastin e kundërt. Procesi shkon pothuaj njejtë sikur tek REVERSE, përveç se, ketu kontrollohet në fund se a janë të barabarta teksti i fituar pas loop-es me tekstin hyrës, me ç'rast nënkupton që teksti hyrës është një palindrome.

-Kodi në Python:

```
def PALINDROME(text):
    backw = ""
    gjatesia = len(text)
    for x in range(gjatesia):
        backw += text[gjatesia - 1]
        gjatesia -= 1
    if text == backw:
        return str(True)
    else:
        return str(False)
```

Fig. 5

2.1.6. TIME

-Funksioni:

Kjo metodë na e kthen kohën aktuale në momentin që thirret. Për këtë metode kam shfrytezuar funskionet **localtime, strftime** nga moduli **time** në Python. Pra në return kam kthyer një tekst i cili rezulton nga formatimi i bërë si më poshtë.

-Kodi në Python:

```
def TIME():
    return strftime("%Y-%m-%d %H:%M:%S PM", localtime())
```

Fig. 6

2.1.7. GAME

-Funksioni:

Metoda GAME na e kthen një listë të kthyer në string, e cila i përmban 5 numra të gjeneruar në mënyrë të rëndomtë, të renditur sipas madhesisë dhe unikë nga njëri-tjetri. Metoda është realizuar nëpërmjet një while loop-e, e cila përsëritet përderisa nuk bëhet 5 numri i anëtarëve në listë. Për gjenerimin e numrave në mënyrë të rëndomtë është përdorur **moduli random**, përkatësisht funksioni **randint()**, ku si parametër ka rangun 1-36(nr. i fundit nuk përfshihet), në të cilin duam të gjenerojmë numra. Pastaj, për mos të pasur duplikate në listë, e kam kontrolluar me if se a ekziston tashmë në listë numri i gjeneruar.

-Kodi në Python:

```
def GAME():
    lista = []

    while len(lista) != 5:
        y = random.randint(1, 36)
        if y not in lista:
            lista.append(y)

    listToStr = ', '.join([str(elem) for elem in lista])
    return listToStr
```

Fig. 7

2.1.8. CONVERT

-Funksioni:

Metoda CONVERT pranon dy parametra, numrin që do të konvertohet dhe opsionin në të cilin do të konvertohet, e cila në fund na kthen rezultatin, varësisht nga opsioni i kërkuar.

-Kodi në Python:

```
def CONVERT(number, option):
    if option == "CMTOFEET":
        return str(round((number * 0.0328084), 2)) + "ft"
    elif option == "FEETTOCM":
        return str(round((number / 0.0328084), 2)) + "cm"
    elif option == "KMTOMILES":
        return str(round((number * 0.621371), 2)) + "miles"
    elif option == "MILESTOKM":
        return str(round((number / 0.621371), 2)) + "km"
    else:
        return "Invalid option choosen."
```

Fig. 8

2.1.9. GCF

Funksioni:

Kjo metodë na kthen faktorin më të madh të përbashkët ndërmjet dy numrave të dhënë si parametra. Ecuri shkon kështu: x-it i jepet vlera e numrit të dytë, ndërsa y-it i jepet vlera e x%y, pra mbetja e x/y. Kjo vazhdon përderisa mbetja nuk bëhet 0 dhe në fund kthehet x, në të cilin mbetet vlera e fundit nga loop-a, e cila e paraqet edhe faktorin më të madh të përbashkët të këtyre dy numrave.

Kodi në Python:

```
def GCF(x, y):
    while y != 0:
        (x, y) = (y, x % y)
    return str(x)
```

Fig. 9

2.2. Metodat e shtuara nga studenti

2.2.1. CALCULATE

Funksioni:

Metoda CALCULATE paraqet një kalkulator të thjeshtë i cili mund të kryejë veprime matematikore si: **mbledhje, zbritje, shumëzim, pjestim, fuqi, rrënjë katrore dhe përqindje.** Realizimi i kësaj metodë është bërë si në vijim: së pari, si parametra janë numri i parë, operatori dhe numri i dytë. Numri i dytë është një parameter opsional, kjo sepse nëse përdoruesi e zgjedh operatorin sqrt (rrënjën katrore), nuk ka nevojë për ndonjë numër tjetër. Duke e pasur këtë parasysh, njëherë e kthejmë në float numrin e parë, x, dhe pastaj, e kontrollojmë se sa është gjatësia e anëtarëve të parametrit opsional (parametri *n trajtohet si një array në python). Nëse gjatësia e anëtarëve në parametrin *n është më e madhe se një, dalim nga metoda duke i treguar përdoruesit se nuk duhet të jepen më shume se tre parametra gjithsej. Në të kundërtën, vazhdojmë në një for loop. Në këtë for loop, iterohet sipas anëtarëve të parametrit opsional *n, ku në rast se ekziston një i tillë, ai anëtar shndërrohet në float.

Pas kësaj, në vartësi të operatorit të zgjedhur, kthehet rezultati përkatës.

Kodi në Python:

```
def CALCULATE(x, op,*n):
    x = float(x)
    if len(n) > 1:
        return ("CALCULATE pranon vetem tre argumente.")
        pass
    y = 0
    for nr in n:
        y = float(nr)
    if op == "SQRT":
        return round((x ** (1 / 2)), 2)
    elif op == "%":
        return (x * (0.01) * y)
    elif op == "+":
        return x + y
    elif op == "-":
        return x - y
    elif op == "*":
        return x * y
    elif op == "/":
        return x / y
    elif op == "^":
        return x ** y
```

Fig. 10

2.2.2. PASSWORD

Funksioni:

Metoda PASSWORD gjeneron një password me një gjatësi që specifikohet nga përdoruesi, e cila jepet si parametër hyrës. Për këtë metodë e kam përdorur **modulin random** dhe **modulin string**, me anë të së cilit i kam ruajtur në variablën chars të gjitha shkronjat e alfabetit latin, numrat prej 0 në 9 dhe simbolet që i ofron ky modul. Pastaj, duke iteruar në një for loop në rang të gjatësisë së dhënë, për aq herë, me anë të funksionit **random.choice()** e zgjedh në mënyrë të rëndomtë një karkater në stringun e krijuar **chars,** të cilat i bashkangjiten listës së krijuar më lart. Në fund kthehet rezultati duke i bërë join elementet në listën e fituar, duke e nxjerrë kështu një string qe paraqet një password mjaft të sigurt për shkak të diversitetit te karaktereve qe gjenerohen në këtë mënyrë.

Kodi në Python:

```
def password(gjatesia):
    gjatesia = int(gjatesia)
    chars = string.ascii_letters + string.digits + string.punctuation
    lista = []
    for x in range(gjatesia):
        lista.append(random.choice(chars))
    return ''.join(lista)
```

Fig. 11

2.2.3. IS_NUM

Funksioni:

Metoda IS_NUM, është metodë e krijuar enkas për shkak të nevojës së parashtruar gjatë validimit të kërkesave që i jep klienti (pra është metodë e implementuar vetëm në anën e klientit), të cilat do të shtjellohen më hollësisht më poshtë. Në këtë metodë e kontrollojmë se a është numër inputi i dhënë nga klienti. Metoda built in të Python siç është p.sh isdigit() nuk m'i ka plotësuar nevojat, pasi që **isdigit()** kthen False nëse ka ndonjë shenjë tjetër inputi, përveç numrave. Pra duke e pasur parasysh që inputi i përdoruesit mund të ketë edhe numra me presje dhjetore, atëherë më lindur ideja që ky validim të bëhet duke tentuar ta kthejmë inputin në **float**. E dijmë edhe nëse kthejme integer në float ne nuk humbim ndonjë gjë, prandaj, nëse inputi është një numër, qoftë ai i plotë apo me presje dhjetore, ky konvertim do të jetë i suksesshëm dhe do ta kthejë **True**. Në rast se inputi përmban ndonjë shkronjë apo diçka tjetër që nuk mund të konvertohet në float, atëherë na shfaqet një **ValueError**, të cilin e kam trajtuar në atë mënyrë që të kthejë **False**.

Kodi në Python:

```
def is_num(string):
    try:
        float(string)
        return True
    except ValueError:
        return False
```

Fig. 12

3. Fiek TCP

3.1. Serveri

3.1.1. Përshkrim i shkurtër:

Serveri FIEK TCP, është serveri i cili është krijuar sipas protokolit TCP. Ky server përmban në vete të gjitha metodat e paraqitura më lartë, përveç metodës IS_NUM, e cila ndodhet në anën e klientit.

Së pari janë shkruar metodat, pastaj më poshtë është krijuar socket-i përkatës i këtij serveri, i cili do të na mundësojë lidhjen me klientët. Krijimi i socket-it që përkrah protokolin TCP është bërë në këtë mënyrë (**SOCK_STREAM** tregon që kemi të bëjmë me protokolin **TCP**, ndërsa **AF_INET** tregon që kemi të bëjmë me tipin e IP adresave **IPv4**):

```
TCPserver = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Pastaj, pas krijimit të socket-it, ne duhet ta bëjmë të gatshëm të pranojë nga jashtë me anë të funksionit **bind()**, sipas Hostit dhe Portit përkatës:

```
TCPserver.bind((HOST, PORT))
```

Pasi e kemi shfrytëzuar **bind()**, vazhdojmë me metodën **listen()**, e cila i jep mundësinë serverit që të dëgjojë kërkesa nga klientët, ku si parameter vendoset numri maksimal i klientëve që i trajton serveri në të njejtën kohë.

Më pas, me anë të funksionit accept(), ne i pranojmë kërkesat e dërguara nga klienti.

```
conn, address = TCPserver.accept()
```

Conn e paraqet lidhjen e krijuar me klientin si një objekt, dhe trajtohet si i tillë më tutje në kod, ndërsa address, është një variabël dyvlerëshe në të cilën ruhen IP adresa dhe Porti i klientit që ka dërguar.

Me anë të **`conn.recv()`** ne i marrim të dhënat që dërgohen, ku si parametër vendoset buffersize i lidhjes. Të dhënat e pranuara më tutje dekodohen dhe përdoren për validimin e kërkesës.

Trajtimi i disa klientëve në të njejtën kohë na mundësohet nëpërmjet **modulit threading**, i cili secilin klient të lidhur në server e ndan si një thread në vete, duke e bërë këtë proces shumë më të shpejtë dhe shumë më të çëndrueshëm, sesa nëse do i trajtonim në mënyrë të zakontë sekuenciale. Funksionimi i threading në këtë server shtjellohet më hollësisht më poshtë.

3.1.2. Kodi për FIEK TCP Serverin:

Vini re se kodi i metodave nuk është paraqitur këtu, pasi gjendet më lartë.

Krijimi i socket-it:

Procesi i krijimit të socketit është futur brenda një blloku try except, për t'i trajtuar gabimet eventuale qe mund të ndodhin gjatë krijimit të socket-it. Host by default i është lënë vlera 'localhost', ndërsa Port by default është 13000, sipas kërkesës së projektit.

Ecuria e krijimit bëhet ashtu siç u përshkrua më lart.

```
try:
    HOST = 'localhost'
    PORT = 13000
    print("\n\t\t\t\t\t\tFIEK TCP Server\n"
          "------------------------------------------------------------\n")
    TCPserver = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print("\nSocket is being created...")
except socket.error as err:
    print("Error while creating socket" + str(err))
```

Fig. 13

Socket Binding:

Për ta bërë bind socket-in, e kam krijuar një funksion të posaçëm, ku përveç ecurise bazë të shpjeguar më lartë, e kam futur kodin në një bllok **try except**, me anë të së cilit trajothen gabimet eventuale në socket ose **TypeError** (i cili mund të shfaqet nëse dështimi i bind() ka qenë për arsye se porti nuk është një numër i plotë), ku në këtë rast, kërkohen të jepen edhe një herë Hosti dhe Porti, në mënyrë që të sigurohemi që bind() nuk ka dështuar si pasojë e këtyre parametrave të dhënë gabim. Në fund, pas trajtimit të errorit funksioni e thërret vetëveten (rekurzion), duke tentuar që të bëhet bind përsëri. Po ashtu, variablat **HOST** dhe **PORT** deklarohen si **globale,** pasi që ato po merren nga jashtë funksionit.

```
# SocketBinding()
def socketBinding():
    try:
        global HOST
        global PORT
        TCPserver.bind((HOST, PORT))
        print("\nServeri eshte startuar ne HOST: " + HOST + ", me PORT: ", PORT)
        TCPserver.listen(8)
        print("Serveri eshte duke pritur per ndonje kerkese nga klientet")
    except (socket.error, TypeError) as err:
        print("Bind failed. Error: " + str(err))
        print("\nKontrolloni IP adresen dhe PORTIN qe e keni dhene.\n")
        HOST = input("Jepeni IP adresen perseri: ")
        try:
            PORT = int(input("Jepni PORTin perseri: "))
        except (ValueError, OverflowError):
            PORT = int(input("Ju lutem sigurohuni qe PORT te jete nje numer: "))
        socketBinding()
```

Fig. 14

Pranimi i kërkesave

Figura në vijim paraqet kodin përgjegjës për pranimin e kërkesave dhe ndarjen e klientëve në thread-e të veçanta. Në while loop, së pari pranohet kërkesa me .accept(), pastaj duke e shfrytëzuar **modulin threading**, e dërgojmë lidhjen me klientin që ka dërguar kërkesën, së bashku me IP adresen dhe portin e tij, metodës së krijuar **ThreadedServer**, e cila e krijon një thread për klientin në fjalë. Threadi niset me anë të funksionit **start()**, që vie nga moduli threading. Në fund të while loop-ës printohet në server numri i klintëve aktiv, i cili mundësohet nëpërmjet funksionit **activeCount(),** të cilës i zbritet 1, pasi që në të llogaritet edhe main threadi,

i cili nuk paraqet ndonjë klient. Në fund, pas daljes nga while loop, lidhja mbyllet me anë të funksionit close().

```
socketBinding()

# Accepting client requests
while True:
    conn, address = TCPserver.accept()
    print('----------------------------------------')
    print('Klienti u lidh me %s me portin %s' % address)
    th = threading.Thread(target=ThreadedServer, args=(conn, address))
    th.start()                                          # e cila na n
    print("\nKliente aktiv: ",
          threading.activeCount() - 1)  # Na jep numrin e klienteve akt

conn.close()
```

Fig. 15

Multithreading

Në funksionin e mëposhtëm **ThreadedServer()** është bërë pranimi, shqyrtimi e validimi i kërkesave të ardhura nga klienti. Ky funksion thirret për çdo thread të ri që krijohet për secilin klient unik, i cili u pështkrua pak më lartë. E gjithë while loop-a është e futur brenda një try except për ti trajtuar gabimet e mundshme që mund të ndodhin në këtë kod, të cilat kanë të bëjnë me lidhjen që ekziston me klientin përkatës nga i cili vie kërkesa (**ConnectionError, ConnectionResetError, ConnectionAbortedError, ConnectionRefusedError)**. Logjika e kodit të mëposhtëm është si në vijim:

Pranohet kërkesa, shndërrohet në karaktere të mëdha, dekodohet, pastaj ndahet dhe vendoset në një listë stringjesh me anë të funksionit **split()**. Më poshtë shikohet se a është kërkesa e njejte me një nga opsionet që ofrohen nga serveri, ku në rast se perputhen, atëhere thirret metoda përkatëse

e krijuar më lart dhe rezultati i dërgohet klientit. E kam pasur parasysh që enkodimi i rezultatit që dërgohet kushtëzon që ky rezultat duhet të jetë si string.

```
# Funksioni me ane te cilit e mundesojme casjen disa klienteve njekohesisht ne server
# Ky funksion thirret nga funksioni run() i modulit threading kur ne e bejme start() thread-in
def ThreadedServer(conn, address):
    try:
        while True:

            data = (conn.recv(128)).upper()  # te dhenat qe i ka derguar klienti pranohen dhe cdo shkronje behet
            data = data.decode()
            print("\nKerkesa: " + data)
            args = data.split()
            gjatesia = len(args)  # ketu e ruajme numrin e argumenteve qe i ka derguar klienti, me ane te se cile
            kerkesa = args[0]

            if kerkesa == "IPADDRESS":
                pergjigjja = "IP Adresa e klientit eshte: " + str(IPADDRESS(address))
                conn.send(str.encode(pergjigjja))
            elif kerkesa == "PORT":
                pergjigjja = "Klienti eshte duke perdorur portin: " + str(PORTI(address))
                conn.send(str.encode(pergjigjja))
            elif kerkesa == "TIME":
                conn.send(str.encode(str(TIME())))
            elif kerkesa == "GAME":
                conn.send(str.encode(GAME()))
            elif kerkesa == "EXIT":
                print("\nLidhja me klientin" + " IP: ", address[0], " PORT: ", address[1], " eshte shkeputur.")
                break


            elif kerkesa == "COUNT":
                text = data[len(kerkesa):]
                conn.send(str.encode(COUNT(text)))
            elif kerkesa == "REVERSE":
                text = data[len(kerkesa):]
                conn.send(str.encode(REVERSE(text)))
            elif kerkesa == "PALINDROME":
                text = args[1]
                conn.send(str.encode(PALINDROME(text)))
            elif kerkesa == "CONVERT":
                number = int(args[1])
                option = args[2]
                conn.send(str.encode(CONVERT(number, option)))
            elif kerkesa == "GCF":
                x = (int)(args[1])
                y = (int)(args[2])
                conn.send(str.encode(GCF(x, y)))
```

```
        elif kerkesa == "GCF":
            x = (int)(args[1])
            y = (int)(args[2])
            conn.send(str.encode(GCF(x, y)))
        elif kerkesa == "CALCULATE":
            x = args[1]
            op = args[2]
            if (gjatesia > 3):  # kjo eshte bere per shkak se sqrt kerkon vetem nje numer dhe operatorin
                y = args[3]
                conn.send(str.encode(str(CALCULATE(x, op, y))))
            elif (gjatesia == 3):
                conn.send(str.encode(str(CALCULATE(x, op))))
        elif kerkesa == "PASSWORD":
            size = args[1]
            conn.send(str.encode(str(password(size))))
except (ConnectionError, ConnectionRefusedError, ConnectionAbortedError, ConnectionResetError) as msg:
    print("Connection error: ", msg)
```

Fig. 16

3.2. Klienti

3.2.1. Kodi për FIEK TCP Klientin

Krijimi i socket-it dhe lidhja me server

Së pari është krijuar funksioni **socketCreate()**, me anë të cilit e mundësojmë krijimin e socket-it në anën e klientit dhe lidhjen e tij me serverin përkatës. Për këtë është përdorur **moduli socket** nga i cili janë shfrytëzuar funksionet **socket()** dhe **connect()**. **Host** dhe **Port** jipen si input nga klienti, të cilat janë të deklaruara si **global**, pasi që të njejtat çasen edhe jashtë këtij funksioni më poshtë. **Try except** është shtuar për ti parandaluar gabimet eventuale gjatë lidhjes së klientit me server, ku përdoren Host dhe Port, që nëse janë dhënë gabim nga klienti, dështon procesi i krijimit të socketit, me ç'rast thirret edhe një herë funksioni në vetëvete(rekurzion).

```
import socket


def socketCreate():
    global HOST
    global PORT
    global TCPclient
    try:
        HOST = input("Jepni IP Adresen (Default - localhost): \n")
        PORT = int(input("Jepni portin (Default - 13000): \n"))
        TCPclient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        TCPclient.connect((HOST, PORT))
    except (socket.error, OverflowError, ValueError) as err:
        print("Error while creating socket: " + str(err))
        socketCreate()


socketCreate()
```

Fig. 17

Udhëzimet

Në fig. 18 më poshtë janë informatat e nevojshme që i printohen klientit për ta udhëzuar në shkrimin e kërkesave. Po ashtu, janë futur në lista metodat e mundshme, operacionet për metoden CALCULATE, dhe të dhëna tjera që e kanë ndihmuar procesin e validimit të bërë më poshtë.

```
print("\t\t\t\t\t\tFIEK TCP client\n"
      "----------------------------------------------------------------\n"
      "1. IPADDRESS\n"
      "2. PORT\n"
      "3. COUNT text\n"
      "4. REVERSE text\n"
      "5. PALINDROME text\n"
      "6. TIME\n"
      "7. GAME\n"
      "8. GCF number1 number2\n"
      "9. CONVERT number cmtofeet/feettocm/kmtomile/milestokm\n"
      "10. CALCULATE number1 operator number2   --> operator: +, -, *, /, ^, sqrt, %\n"
      "11. PASSWORD gjatesia\n"
      "\n"
      "Shtyp 0 nese deshironi ta nderroni IP adresen dhe Portin\n")

methods = ["ipaddress", "port", "count", "reverse", "palindrome", "time", "game", "gcf", "convert", "calculate",
           "password", "exit", "0"]
soloMethods = ["ipaddress", "port", "time", "game"]
convertOptions = ["cmtofeet", "feettocm", "kmtomile", "milestokm"]
operators = ["+", "-", "*", "/", "^", "%"]
```

Fig. 18

Metoda **IS_NUM** e cila shfrytëzohet për validimin e numrave, funksionaliteti i së cilës është treguar hollësisht më lartë tek **Metodat**

```
def is_num(string):
    try:
        float(string)
        return True
    except ValueError:
        return False
```

Fig. 19

Validmi, dërgimi dhe pranimi

Brenda këtij while loop-it gjendet validimi për të gjitha metodat, pra marrja e kërkesës së përdoruesit nëpërmjet inputit dhe shqyrtimi i saj. Së pari, nëse përdoruesi nuk ka shtypur asgjë, injorohet i tërë kodi i mëposhtëm dhe kalon në iterimin e radhës në loop. Pastaj, nëse pjesa e parë e kërkesës së dhënë nga përdoruesi, nuk paraqet ndonjë nga metodat që ekzistojnë në server, përsëri kalohet menjëherë në iterimin e radhës. Nëse kërkesa e dhënë është në listën **soloMethods** (Fig. 19), kjo është valide dhe i dërgohet serverit.

Nëse pjesa e parë e kërkesës është një nga metodat **COUNT, PALINDROME** ose **REVERSE**, dhe nëse gjatësia e kërkesës, të shndërruar në listë, është më e madhe se 1, atëherë dihet se ka edhe argumente tjera që janë të nevojshme për këto metoda, e cila e bën kërkesën valide dhe i dërgohet serverit.

Edhe validimi për metodat tjera bëhet pothuaj me logjikë të njejtë, përveç se, te metodat ku priten argumente numra, është përdorur edhe metoda e krijuar posaçërisht për këtë (**IS_NUM**). Për çdo rast tjetër, përdoruesit i tregohet se kërkesa e dhënë nuk është valide.

Të dhënat e dërguara nga serveri pranohen me **recv()**, dekodohen dhe i paraqiten klientit në ekran.

Po ashtu, i gjithë procesi i validimit dhe i dërgimit të kërkesave në server gjendet brenda bllokut **try except**, me anë të cilit i trajtojmë dhe i tregojmë klientit për gabimet që mund të kenë ndodhur pas një mosfunksionimi të lidhjes së tij me serverin. Kur dalim nga while loop, thirret metoda për mbylljen e lidhjes së klientit me server: **close()**.

```
while True:
    kerkesa = (input("Type request: ").lower()).strip()
    listed = kerkesa.split()

    try:
        if len(kerkesa) == 0:
            continue

        elif listed[0] not in methods:
            print("Invalid request...\n")
            continue

        elif kerkesa in soloMethods:
            TCPclient.send(str.encode(kerkesa))

        elif listed[0] == "count" or listed[0] == "palindrome" or listed[0] == "reverse":
            if len(listed) > 1:
                TCPclient.send(str.encode(kerkesa))
            else:
                continue

        elif len(listed) > 1 and listed[0] == "calculate" and is_num(listed[1]):
            if len(listed) == 4 and is_num(listed[3]) and listed[2] in operators:
                TCPclient.send(str.encode(kerkesa))
            elif len(listed) == 3 and listed[2] == "sqrt":
                TCPclient.send(str.encode(kerkesa))
            else:
                print("Invalid request...\n")
                continue

        elif listed[0] == "gcf" and is_num(listed[1]) and is_num(listed[2]):
            TCPclient.send(str.encode(kerkesa))

        elif len(listed) > 1 and listed[0] == "convert" and is_num(listed[1]) and listed[2] in convertOptions:
            TCPclient.send(str.encode(kerkesa))

        elif listed[0] == "password" and listed[1].isdigit():
            TCPclient.send(str.encode(kerkesa))

        elif listed[0] == "0":
            HOST = input("Jepni IP adresen: ")
            PORT = int(input("Jepni PORTIN: "))
            TCPclient.send(str.encode("exit"))
            TCPclient.close()
            TCPclient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            TCPclient.connect((HOST, PORT))
            continue
```

```
    elif listed[0] == "exit":
        TCPclient.send(str.encode(kerkesa))
        print("Jeni shkeputur nga serveri.")
        break

    else:
        print("Invalid request...\n")
        continue

    data = TCPclient.recv(128)
    data = data.decode()
    data = str(data)
    print("\nPergjigjja nga serveri: " + data + "\n\n")
except (ConnectionError, ConnectionResetError, ConnectionAbortedError, ConnectionRefusedError) as err:
    print("Connection error: ", err)
    break

TCPclient.close()
```

Fig. 20

4. Fiek UDP

4.1. Serveri

4.2. Përshkrim i shkurtër:

Për dallim nga TCP, tek UDP nuk është përdorur multithreading për t'i trajtar secilin klient me thread të veçantë, por, pasi që UDP konsiderohet si "connectionless", nuk ka rëndësi se sa lidhje ekzistojnë apo se cili klient po shërbehet. Prandaj, kodi i UDP serverit është paksa më i thjeshtë se ai tek TCP.

4.3. Kodi për FIEK UDP Serverin:

Vini re se kodi i metodave nuk është paraqitur këtu, pasi gjendet më lartë.

Për UDP serverin janë përdorur modulet si socket, random, string dhe time (Fig. 21).

```
import socket
import random
import string
from time import localtime, strftime
```

Fig. 21

Krijimi i socket-it:

Socket-i krijohet sipas portit **13000** dhe host-it '**localhost**', duke shfrytëzuar modulin socket. Këtu, në vend të **SOCK_STREAM** përdoret **SOCK_DGRAM.** Kodi është i futur brenda një blloku try except, për t'i kapur gabimet e mundshme.

```
# SOCKET

try:
    HOST = 'localhost'
    PORT = 13000
    UDPserver = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    print("Socket is being created...")
except socket.error as err:
    print("Error while creating socket", err)
```

Fig. 22

Socket Binding:

Socket binding është pothuar i njejtë me atë të TCP, përveç se këtu tek UDP nuk përdoret funksioni **listen().** Kodi është i futur brenda nje blloku për trajtimin e gabimeve të mundshme, me anë të cilit mund t'a tejkalojme çdo gabim që na paraqitet gjatë binding. Vërehet se edhe këtu kemi rekurzion, ashtu si në TCP socket binding.

```
def socketBinding():
    try:
        global HOST
        global PORT
        global UDPserver
        UDPserver.bind((HOST, PORT))
        print("\nServeri eshte startuar ne localhost me portin: " + str(PORT))
        print("Serveri eshte duke pritur per ndonje kerkese\n"
              "-----------------------------------------")
    except socket.error as err:
        print("Bind failed. Error: ", err)
        print("\nKontrolloni IP adresen dhe PORTIN qe e keni dhene.\n")
        HOST = input("Jepni IP adresen perseri: ")
        try:
            PORT = int(input("Jepni PORTin perseri: "))
        except (ValueError, OverflowError):
            PORT = int(input("Ju lutem sigurohuni qe PORT te jete nje numer (1024 - 65535): "))
        socketBinding()


socketBinding()
```

Fig. 23

Pranimi i kërkesave:

Për dallim nga TCP, këtu pranimi dhe kontrollimi i kërkesës së pranuar nga klienti nuk trajtohet me multithreading. Pra, brenda një while loop-e gjendet i tërë kodi i cili i pranon (recvfrom), dekodim dhe kontrollimin e kërkesës së pranuar. Kontrollimi i kërkesës bëhet plotësisht njejtë sikur tek funksioni ThreadedServer tek TCP serveri, por këtu, në vend të **break** tek kërkesa **EXIT**, bëhet **continue** për të kaluar në iterimin tjetër. Dallim tjetër që vlen të theksohet është se kur pranohet kërkesa me recvfrom(), variabla e pare këtu e përmban atë qe e ka dërguar klienti, ndërsa e dyta e përmban adresën e klientit. Kjo pasi që tek UDP nuk kemi **accept()**.

```
while True:
    try:
        dataRecieved, address = UDPserver.recvfrom(128)
        data = dataRecieved.decode()
        data = data.upper()
        print("\nKerkesa nga klienti me IP: '" + str(address[0]) + "', dhe Port: " + str(address[1]) + "\n" + data)

        args = data.split()
        gjatesia = len(args)
        kerkesa = args[0]

        if kerkesa == "TEST":
            continue
        elif kerkesa == "IPADDRESS":
            pergjigjja = "IP Adresa e klientit eshte: " + str(IPADDRESS(address))
            UDPserver.sendto(pergjigjja.encode(), address)
        elif kerkesa == "PORT":
            pergjigjja = "Klienti eshte duke perdorur portin: " + str(PORTI(address))
            UDPserver.sendto(pergjigjja.encode(), address)
        elif kerkesa == "TIME":
            UDPserver.sendto(str(TIME()).encode(), address)
        elif kerkesa == "GAME":
            UDPserver.sendto(GAME().encode(), address)
        elif kerkesa == "EXIT":
            print("Lidhja me klientin eshte shkeputur.")
            continue
```

```
        elif kerkesa == "COUNT":
            text = data[len(kerkesa):]
            UDPserver.sendto(COUNT(text).encode(), address)
        elif kerkesa == "REVERSE":
            text = data[len(kerkesa):]
            UDPserver.sendto(REVERSE(text).encode(), address)
        elif kerkesa == "PALINDROME":
            text = args[1]
            UDPserver.sendto(PALINDROME(text).encode(), address)
        elif kerkesa == "CONVERT":
            number = int(args[1])
            option = args[2]
            UDPserver.sendto(CONVERT(number, option).encode(), address)
        elif kerkesa == "GCF":
            x = (int)(args[1])
            y = (int)(args[2])
            UDPserver.sendto(GCF(x, y).encode(), address)
```

```
        elif kerkesa == "CALCULATE":
            x = args[1]
            op = args[2]
            if gjatesia > 3:  # kjo eshte bere per shkak se sqrt kerkon vetem nje numer dhe operatorin
                y = args[3]
                UDPserver.sendto(str(CALCULATE(x, op, y)).encode(), address)
            elif gjatesia == 3:
                UDPserver.sendto(str(CALCULATE(x, op)).encode(), address)
        elif kerkesa == "PASSWORD":
            gjatesia = args[1]
            UDPserver.sendto(str(password(gjatesia)).encode(), address)
    except (ConnectionError, ConnectionRefusedError, ConnectionAbortedError, ConnectionResetError) as err:
        print("Server side error... ", err)
```

Fig. 24

4.4. Klienti

4.4.1. Kodi për FIEK UDP Klientin

Krijimi i socket-it dhe lidhja me server:

Krijimi i socket-it bëhet nëpërmjet një funksioni të krijuar, njejtë sikur tek TCP klienti. Mirëpo, në mënyrë që të testohet lidhja dhe gabimet eventuale gjatë krijimit të lidhjes me server, serverit me **host** dhe **port** që jepet nga klienti i dërgohet kërkesa "**test**", në mënyrë që nëse lidhja është e suksesshme, serveri na kthen diçka dhe konsiderohet si lidhje e realizuar me sukses. Përndryshe, nëse nuk mund të bëhet **sendto** kërkesa, gabimi trajtohet duke e thirrur funksioni prapë vetëveten(rekurzion).

```
import socket


def socketCreate():
    try:
        global HOST
        global PORT
        global UDPclient
        HOST = input("Jepni IP Adresen (Default - localhost): \n")
        PORT = int(input("Jepni portin (Default - 13000): \n"))
        UDPclient = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        UDPclient.sendto("test".encode(), (HOST, PORT))
    except (socket.error, ValueError, OverflowError) as err:
        print("Error while creating socket: ", err)
        socketCreate()


socketCreate()
```

Fig. 25

Vini re se udhëzimet për shkrimin e kërkesave janë plotësisht të njejta me atë të TCP klientit, prandaj nuk është këtu. Ato mund t'i gjeni tek Fig. 18.

Validmi, dërgimi dhe pranimi:

Brenda këtij while loop-it gjendet validimi për të gjitha metodat, pra marrja e kërkesës së përdoruesit nëpërmjet inputit dhe shqyrtimi i saj. Procesi i validimit është pothuaj tërësisht i njejtë si tek TCP klienti. Për dallim nga ky i fundit, tek UDP kërkesat i dërgohen klientit me anë të funksionit **sendto()**.

```
while True:
    kerkesa = (input("Type request: ").lower()).strip()

    listed = kerkesa.split()  # Argumentet e dhena si input i ruajme ne nje liste si stringje

    if len(kerkesa) == 0 or len(kerkesa.encode()) > 128:
        continue

    elif listed[0] not in methods:
        print("Invalid request...\n")
        continue

    elif kerkesa in soloMethods:
        UDPclient.sendto(kerkesa.encode(), (HOST, PORT))

    elif listed[0] == "count" or listed[0] == "palindrome" or listed[0] == "reverse":
        if len(listed) == 2:
            UDPclient.sendto(kerkesa.encode(), (HOST, PORT))
        else:
            continue


    elif len(listed) > 1 and listed[0] == "calculate" and is_num(listed[1]):
        if len(listed) == 4 and is_num(listed[3]) and listed[2] in operators:
            UDPclient.sendto(kerkesa.encode(), (HOST, PORT))
        elif len(listed) == 3 and listed[2] == "sqrt":
            UDPclient.sendto(kerkesa.encode(), (HOST, PORT))
        else:
            print("Invalid request...\n")
            continue

    elif len(listed) < 4 and listed[0] == "gcf" and is_num(listed[1]) and is_num(listed[2]):
        UDPclient.sendto(kerkesa.encode(), (HOST, PORT))

    elif len(listed) > 1 and listed[0] == "convert" and is_num(listed[1]) and listed[2] in convertOptions:
        UDPclient.sendto(kerkesa.encode(), (HOST, PORT))

    elif listed[0] == "password" and listed[1].isdigit():
        UDPclient.sendto(kerkesa.encode(), (HOST, PORT))

    elif listed[0] == "0":
        HOST = input("Jepni IP adresen: ")
        PORT = int(input("Jepni PORTIN: "))
        UDPclient.sendto(kerkesa.encode(), (HOST, PORT))
        UDPclient.close()
        UDPclient = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        continue
```

```
    elif listed[0] == "exit":
        UDPclient.sendto(kerkesa.encode(), (HOST, PORT))
        print("Jeni shkeputur nga serveri.")
        break

    else:
        print("Invalid request...\n")
        continue
    dataRecieved, address = UDPclient.recvfrom(128)
    data = dataRecieved.decode()
    data = str(data)

    print("\nPergjigjja nga serveri: " + data + "\n\n")
```

Fig. 26

5. Testimi

5.1. FIEK TCP Server

```
                        FIEK TCP Server
---------------------------------------------------------------


Socket is being created...

Serveri eshte startuar ne HOST: localhost, me PORT:  13000
Serveri eshte duke pritur per ndonje kerkese nga klientet
---------------------------------------

---------------------------------------
Eshte krijuar lidhja me klientin me IP: 127.0.0.1 dhe Port 56440

Kliente aktiv:  1

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56440
IPADDRESS

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56440
PORT

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56440
COUNT AAA BBB

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56440
REVERSE YLBER GASHI

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56440
PALINDROME KEK
```

```
Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56440
TIME

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56440
GAME

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56440
GCF 9 12


Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56403
CONVERT 3.2 CMTOFEET

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56403
CALCULATE 45.2 * 18

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56403
PASSWORD 8

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56403
EXIT


----------------------------------------
Eshte krijuar lidhja me klientin me IP: 127.0.0.1 dhe Port 56444

Kliente aktiv:  2

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56442
GAME

Kerkesa nga klienti me IP: 127.0.0.1 dhe Port: 56444
PORT
```

5.2. FIEK TCP Klienti

```
Jepni IP Adresen (Default - localhost):
localhost
Jepni portin (Default - 13000):
13000
                        FIEK TCP client
---------------------------------------------------------------
1. IPADDRESS
2. PORT
3. COUNT text
4. REVERSE text
5. PALINDROME text
6. TIME
7. GAME
8. GCF number1 number2
9. CONVERT number cmtofeet/feettocm/kmtomile/milestokm
10. CALCULATE number1 operator number2   --> operator: +, -, *, /, ^, sqrt, %
11. PASSWORD gjatesia

Type request: ipaddress

Pergjigjja nga serveri: IP Adresa e klientit eshte: 127.0.0.1


Type request: port

Pergjigjja nga serveri: Klienti eshte duke perdorur portin: 56440


Type request: count aaa bbb

Pergjigjja nga serveri: Teksti i pranuar permban 3 zanore dhe 3 bashketingellore.


Type request: reverse Ylber Gashi

Pergjigjja nga serveri: IHSAG REBLY
```

```
Type request: palindrome kek

Pergjigjja nga serveri: True

Type request: time

Pergjigjja nga serveri: 2020-04-08 19:26:35 PM


Type request: game

Pergjigjja nga serveri: 2, 11, 20, 21, 31


Type request: gcf 9 12

Pergjigjja nga serveri: 3


Type request: convert 3.2 cmtofeet

Pergjigjja nga serveri: 0.1ft


Type request: calculate 45.2 * 18

Pergjigjja nga serveri: 813.6


Type request: password 8

Pergjigjja nga serveri: `/wN4E.U


Type request: exit
Jeni shkeputur nga serveri.

Process finished with exit code 0
```

5.3. FIEK UDP Serveri

```
Socket is being created...

Serveri eshte startuar ne localhost me portin: 13000
Serveri eshte duke pritur per ndonje kerkese
---------------------------------------------

Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 52987
TEST


Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 52843
IPADDRESS

Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 52843
PORT

Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 52843
COUNT AAAA BBBB

Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 52843
REVERSE FIEK UDP

Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 52843
PALINDROME ALFA

Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 52843
TIME

Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 52843
GAME


Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 59018
EXIT
Lidhja me klientin eshte shkeputur.
```

Dy klientë:

```
Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 54654
TEST

Kerkesa nga klienti me IP: '127.0.0.1', dhe Port: 54654
TIME
```

5.4. FIEK UDP Klienti

```
Jepni IP Adresen (Default - localhost):
localhost
Jepni portin (Default - 13000):
13000
                        FIEK UDP
---------------------------------------------------------------
1. IPADDRESS
2. PORT
3. COUNT text
4. REVERSE text
5. PALINDROME text
6. TIME
7. GAME
8. GCF number1 number2
9. CONVERT number cmtofeet/feettocm/kmtomile/milestokm
10. CALCULATE number1 operator number2   --> operator: +, -, *, /, ^, sqrt, %
11. PASSWORD gjatesia

Shtyp 0 nese deshironi ta nderroni IP adresen dhe Portin
```

```
Type request: ipaddress

Pergjigjja nga serveri: IP Adresa e klientit eshte: 127.0.0.1


Type request: port

Pergjigjja nga serveri: Klienti eshte duke perdorur portin: 52843


Type request: count aaaa bbbb

Pergjigjja nga serveri: Teksti i pranuar permban 4 zanore dhe 4 bashketingellore.
```

.

.

.

6. Conclusion

Si përfundim, duhet të theksoj se ky projekt në tërësi është realizuar mjaft mirë, duke i zvogëluar margjinat e gabimeve sa më shumë që ka qenë e mundur.

Programet e serverëve, e po ashtu edhe programet e klientëve, për të dyja palët (TCP dhe UDP), punojnë pa kurrfarë problemi. Çdo kërkesë që nuk i përputhet rregullave të përcaktuara nëpërmjet validimit injorohen dhe i tregohet klientit se kërkesa nuk është valide. Kërkesat jovalide nuk dërgohen në server, kjo pasi që kam ndjekur logjikën e të mbajturit serverin sa më pak të ngarkuar, e sa më të gatshëm për të gjithë klientët aktiv. Kjo e ka rritur kompleksitetin e kodit në anën e klientit, por pasi që programi i klientit nuk kërkon fuqi kompjuterike jashtë normales, e kam parë të udhës që programit të klientit t'i shtohet ky kompleksitet, i cili nuk duhet të paraqet ndonjë problem sa i përket ngarkimit të pajisjes së klientit, një pikë kjo e cila do të duhej të mirrej parasysh nëse do të kishim të bënim me ndonjë program tejet të gjerë dhe kompleks.

Anës së serverit i mbetet vetëm ta kontrollojë kërkesën e pranuar nga klienti dhe të thërrasë metodën përkatëse në përputhje me të. Po ashtu, serveri është siguruar me shumë blloqe try except (ashtu edhe klienti), të cilat i parandalojnë, ose thënë më mirë, i trajtojnë dhe shumëkund edhe lajmërojnë, për gabimet (errors) që mund të na shfaqen eventualisht në program.

Të dyja serveret, TCP dhe UDP, janë testuar me shumë klientë (me mbi 5 klientë për server), dhe trajtimi i tyre bëhet në mënyrë të përkryer nga të dy serverët, pa asnjë problem. Dallimi kryesor është se TCP për këtë shfrytëzon modulin threading, për t'i trajtuar të gjithë klientët njëkohësisht, ndërsa për ta arritur këtë gjë tek UDP nuk është shfrytëzuar moduli threading, kjo

pasi që UDP, duke e marrë parasysh që konsiderohet si “connectionless”, nuk ka nevojë për një gjë të tillë, pasi që të gjithë klientët e kyçur në të trajtohen pa problem njëkohësisht.

Po ashtu, vlen të përmendet se secila metodë funksionon pa kurrëfarë problemi, si në TCP, po ashtu edhe në UDP.

7. Referencat

1. https://www.guru99.com/tcp-vs-udp-understanding-the-difference.html
2. https://docs.python.org/3/library/socket.html
3. https://www.geeksforgeeks.org/socket-programming-python/
4. https://docs.python.org/3/library/threading.html
5. https://www.python-course.eu/threads.php