DISTRIBUTED COMPUTING PROJECT

NAME: **Debadrita Roy**

CLASS: BCSE-IV

ROLL NUMBER: 001910501025

PROBLEM DESCRIPTION

A System for dynamically updating changes to class timetable and manage class or exam clashes for faculty and the students
This system can be used in a college or university where classes are usually subject to a lot more changes and rescheduling than in a school, like depending on the availability of teachers and students. Also, this will help students and teachers keep track of their weekly schedule. Classes and exams scheduled by teachers for a particular course will be visible to the other teachers so that they can take it into account while planning their own. The administration will be able to declare holidays and will upload the original timetable at the beginning of the semester to which changes might be made by the respective teachers later.

WHY IS DISTRIBUTED COMPUTING REQUIRED HERE

A distributed computing approach has been adopted to solve the above problem. The reasons for this are:

1. **By the nature of the application**, it has to cater to a large number of users, including students (across a department for now), teachers in that department as well as the administration. So, **multiple client support** is needed.
2. **To increase scalability of the system** as growth is likely to occur in the dimensions of size, geography and across administrative domains.
3. **To help in load sharing** by having different functions performed by the different servers.
4. **To provide extensibility to the system**. Due to changing requirements over time or administrative or policy changes, one might need to add or replace system components. A distributed paradigm will enable one to easily add another service to the existing system or take down or modify a part which is no longer required.
5. **To increase modularity** by separating the functionality into several smaller services. This will make it easier to debug or repair when the system is down.
6. **To make it easier to update a single feature** (or list of actions for a particular client type when new services are developed) without having to rebuild and retest the entire system.
7. **To decrease the down-time of the entire system**. Suppose some code needs to be added or modified for a single server, then that server can be taken down while the other services are still running.

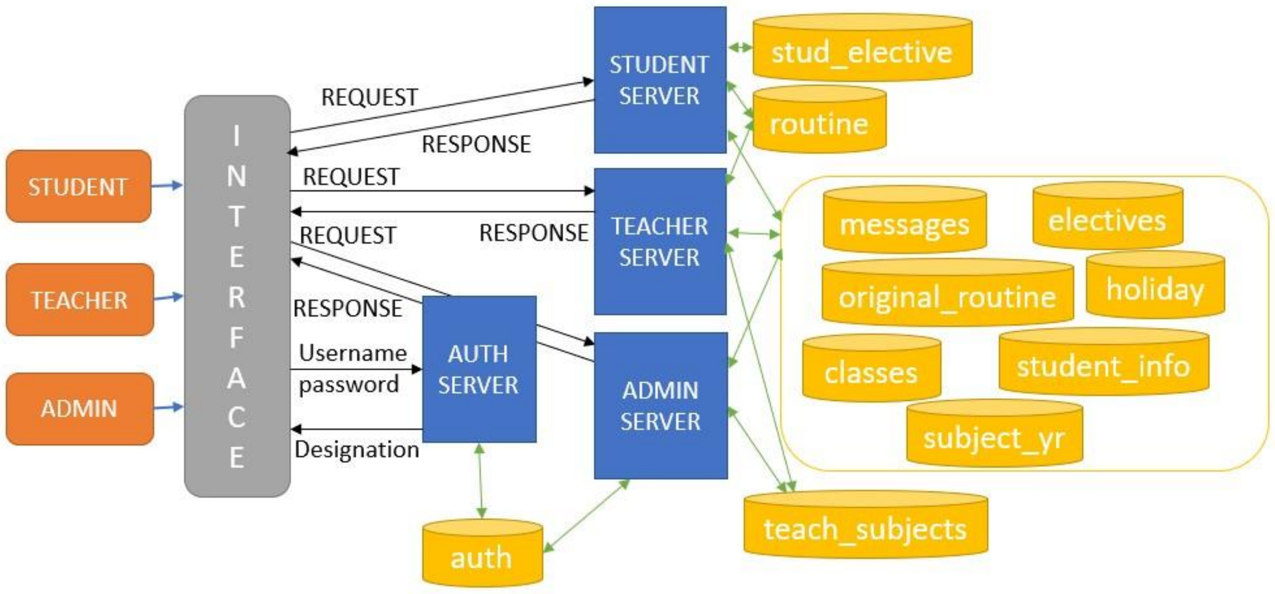
8. **For security of the database and to prevent unauthorised access**. The servers which need access to a part of the database can be given access to only that part of the database and not others.

DESIGN

OVERVIEW

There are 4 servers set up — an authorization server (authenticates username and password before allowing users to connect to domain specific servers), and 3 domain specific servers, one each for students, teachers and administration. The servers are connected to a database which stores the relevant data. A client interface is set up which connects to the specific servers based on the response from the authorization server.

SCHEMATIC DIAGRAM



MESSAGE EXCHANGE

The client first connects with the authorization server and sends the username and password. The server starts a thread for each client and is ready for serving requests. It checks the database and if a match is found, sends the designation (student / teacher / admin) to the client interface. Then based on it the interface connects with the relevant server and sends requests and receives responses until the client logs out.

It is implemented with the help of **TCP Socket**. TCP is a reliable, lossless, connection-oriented protocol which provides a good flow control mechanism. Thus, TCP allows for reliable, sequential transfer and the order of bits in the file is not changed or lost. TCP provides error-checking and guarantees delivery of data. TCP transfers data as a stream of bytes.

IMPLEMENTATION

The solution is written in Python. Socket programming is used to establish connections between the client(s) and the server(s). There are 5 programs — **login.py** (client interface), **authorization.py** (authorization server), **student_server.py** (student server), **admin_server.py** (administrative server) and **teacher_server.py** (teacher server). When all the servers (or the servers needed — authorization server is needed for every client) are running, an user can run login.py for the interface and connect to the respective server(s).

STORAGE OF DATA

The data is stored in a **mySQL database** running on port **3306** of the machine. There are ten tables: auth(username, password, designation), classes(cls, yr), electives(cls, subject, elec_no, yr), holiday(date, name), messages(username, timestamp, msg, st), original_routine(subject, class, year, day, changed, start_time, end_time), routine(subject, class, year, date, type, istemp, start_time, end_time), stud_elective(username, elective, no), student_info(username, class, year), subject_yr(subject, class, year) and teach_subjects(username, subject, class, year, com). **The original routine uploaded by the administration at the start of the semester is saved for future reference**.

MULTIPLE CLIENT SUPPORT

All servers support multiple clients: a **new thread is created for every client** logged in.

Code Snippet (main() of authorization.py)

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('127.0.0.1', 2010))
while True:
  s.listen(5)
  (conn, (ip, port)) = s.accept()  # accept connection from a client
  conn.send('Connected to authorization server'.encode('utf-8'))
  u = conn.recv(1024).decode('utf-8')
  p=conn.recv(1024).decode('utf-8')
  newthread = ThreadForClient(ip, port, u,p, conn)  # create a new thread for this client
  newthread.start()  # start the thread
```

CLIENT INTERFACE

It is responsible for sending requests to the server based on the inputs from the user and displaying the response to the user.

Code Snippet (main() of login.py):

```
host = '127.0.0.1'
ints = {'A': admin, 'T': teacher, 'S': student}
BUFFER_SIZE = 1024
m = ''
```

```
print('Dynamic TimeTable System Interface')
while True:
   u = input('Enter your username: ')
   p = input('Enter your password: ')
   r = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   r.connect((host, 2010))
   print(r.recv(1024).decode('utf-8'))
   r.send(u.encode('utf-8'))
   r.send(p.encode('utf-8'))
   m = r.recv(1024).decode('utf-8')
   r.close()
   if len(m) > 1:
       print(m)
       ch = input('Do you want to try again or quit? \nEnter 1 for first option '
                  'and any other number for the second\n')
       if ch == '1':
           continue
       else:
           break
   else:
       break
if m != '':
   ints[m](u)
```

SERVERS

The servers are running on different ports (**2010** for authorization, **2012** for admin, **2014** for teacher and **2016** for student).

AUTHORIZATION SERVER

It authenticates the username and password of the client and returns their designation so that they can then connect with the respective servers.

Code Snippet (Action performed for each client thread):

```
def run(self):
   mydb = mysql.connector.connect(host="localhost", user="user", password="1234",
database="mydb")
   mycursor = mydb.cursor()
   mycursor.execute("SELECT * FROM auth where username='" + self.username + "'")
   u, p, t = mycursor.fetchone()
   if p == self.password:
       msg=t
   else:
       msg="Incorrect username or password"
   self.conn.send(msg.encode('utf-8'))
```

ADMINISTRATIVE SERVER

It is responsible for taking care of the administrative actions to be performed. These actions include registering new users, uploading the timetable for the different classes at the beginning of the semester, declaring holidays, publishing list of electives for students, updating the subject-teacher combinations, etc.

Code Snippet for implementation of a few actions:

```
if data == '1':     # registration of new user
   pkt=self.conn.recv(1024).decode('utf-8')
   uu=pkt[:pkt.index(';')]
   pp=pkt[pkt.index(';')+1:pkt.rindex(';')]
   tt = pkt[pkt.rindex(';')+1:]
   cc=''
   yy=''
   # print(uu,' ',pp,' ',tt)
   if tt=='Student':
       # print('here')
       cc=self.conn.recv(1024).decode('utf-8')
       yy=self.conn.recv(1024).decode('utf-8')
       # print(cc,' ',yy)
   c=mydb.cursor()
   c.execute("SELECT * FROM auth where username='" + uu + "'")
   rr = c.fetchone()
   # print(rr)
   if rr is None:
       sql='INSERT INTO auth VALUES(%s,%s,%s)'
       val=(uu,pp,tt[0])
       c.execute(sql,val)
       if tt=='Student':
           sql='INSERT INTO student_info VALUES(%s,%s,%s)'
           val=(uu,cc,int(yy))
           c.execute(sql, val)
       mydb.commit()
       message='New user successfully registered'
   else:
       message='Username already exists..try again with another username'
elif data=='3': # declare holidays
   self.conn.send('Ready'.encode('utf-8'))
   hh=pickle.loads(self.conn.recv(1024))
   print(str(hh))
   c=mydb.cursor()
   sql='INSERT INTO holiday VALUES(%s,%s)'
   for h in hh:
       print(str(h),' declared')
```

```
        c.execute(sql,h)
        mydb.commit()
    message='Holidays Successfully Declared'
elif data=='4': # upload electives
    c = mydb.cursor()
    c.execute("SELECT * FROM classes")
    all_clss = c.fetchall()
    c.execute("SELECT cls,yr FROM electives")
    done_classes = c.fetchall()
    to_do_classes = [x for x in all_clss if x not in done_classes]
    self.conn.send(pickle.dumps(to_do_classes))
    ind = int(self.conn.recv(1024).decode('utf-8'))
    eles = pickle.loads(self.conn.recv(1024))
    sql = 'INSERT INTO electives VALUES(%s,%s,%s,%s)'
    for i,op in enumerate(eles):
        for ss in op:
            c.execute(sql, (to_do_classes[ind][0],to_do_classes[ind][1],ss,i+1))
        mydb.commit()
    message = 'List Successfully Uploaded'
```

TEACHER SERVER

It is responsible for taking care of the actions to be performed by a teacher. These actions include viewing one's own timetable, viewing timetables of the students whom the teacher teaches, scheduling or cancelling classes and exams,etc.

Code Snippet for implementation of action - scheduling a class or exam:

```
c=mydb.cursor()
c.execute("SELECT class,year FROM teach_subjects where username='" + self.username +
"'")
classes = c.fetchall()
clsses=[str(x[0])+'-'+str(x[1]) for x in classes]
self.conn.send(pickle.dumps(clsses))
cl=self.conn.recv(1024).decode('utf-8')
cls=classes[clsses.index(cl)]
c.execute("SELECT subject FROM teach_subjects where class='" + cls[0] + "' and year='" +
str(cls[1]) + "'")
subjects = c.fetchall()
times = []
today = datetime.datetime.strptime(date, '%d-%m-%Y').weekday()
week_start = datetime.datetime.strptime(date, '%d-%m-%Y') -
datetime.timedelta(days=today)
week_start_date = week_start.strftime("%d-%m-%Y")
week_end = week_start + datetime.timedelta(days=7)
week_end_date = week_end.strftime("%d-%m-%Y")
for su in subjects:
  c.execute("SELECT subject,day,start_time,end_time FROM original_routine where 
subject='" + su[
      0] + "' and class='" + cls[0] + "' and year='" + str(cls[1]) + "' and changed='N'")
  t = c.fetchall()
  for x in t:
      times.append(x)
  c.execute("SELECT subject,date,start_time,end_time,type FROM routine where subject='"
+ su[
      0] + "' and class='" + cls[0] + "' and year='" + str(cls[1]) + "' and istemp='N'")
  t = c.fetchall()
  for x in t:
      times.append(x)
  c.execute("SELECT subject,date,start_time,end_time,type FROM routine where subject='"
+ su[
      0] + "' and class='" + cls[0] + "' and year='" + str(cls[
              1]) + "' and istemp='Y' and date>='" + week_start_date + "' and date<='" +
week_end_date + "'")
  t = c.fetchall()
  for x in t:
      times.append(x)
week = dict()

for tin in range(len(times)):
  ti = times[tin]
  if len(ti) == 5:
      day = datetime.datetime.strptime(ti[1], '%d-%m-%Y').weekday()
      if week_days[day] in week:
          week[week_days[day]].append((ti[0], ti[2], ti[3], ti[4]))
      else:
          week[week_days[day]] = [(ti[0], ti[2], ti[3], ti[4])]
  else:
      if ti[1] in week:
          week[ti[1]].append((ti[0], ti[2], ti[3], 'c'))
      else:
          week[ti[1]] = [(ti[0], ti[2], ti[3], 'c')]
c.execute(
  "SELECT * FROM holiday where date>='" + week_start_date + "' and date<='" +
week_end_date + "'")
hhs = c.fetchall()
hs, bc = [], []
for h in hhs:
  d, n = h[0], h[1]
  day = datetime.datetime.strptime(d, '%d-%m-%Y').weekday()
  hs.append(week_days[day])
  bc.append(n)
tt = ''
for w in week_days:
  if w in hs:
      tt += w + ": HOLIDAY- " + bc[hs.index(w)] + "\n"
      continue
  if w not in week:
      tt+=w+": No Classes\n"
      continue
  clss = week[w]
  clss.sort(key=lambda x: x[1])
  tt += w + ":\n"
  for cls in clss:
      tt += cls[1] + "-" + cls[2] + " " + cls[0] + "(" + cls[3] + ")\n"
message = tt
```

STUDENT SERVER

It is responsible for taking care of the actions to be performed by a student. These include viewing one’s own timetable, signing up for the electives, etc.

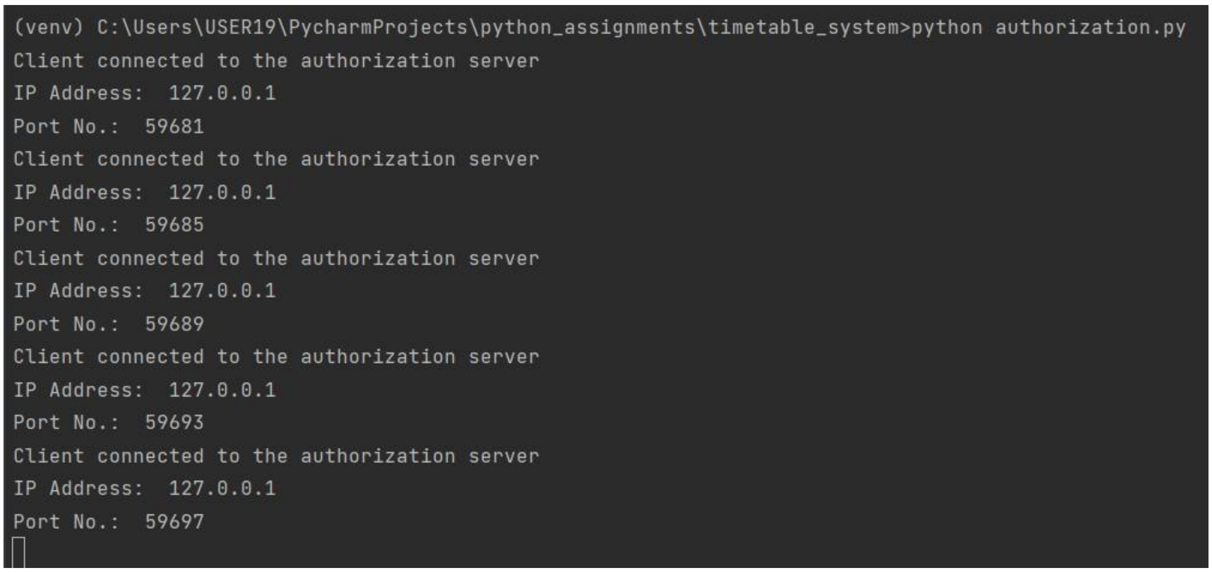
Code Snippet for implementation of action - viewing own timetable:

```
if data == '1':
    c=mydb.cursor()
    c.execute("SELECT class,year FROM student_info where username='" + self.username +
"'")
    res=c.fetchone()
    c.execute("SELECT subject FROM teach_subjects where class='"+res[0]+"' and
year='"+str(res[1])+"' and com='Y'")
    subjects=c.fetchall()
    c.execute("SELECT elective FROM stud_elective where username='" + self.username +
"'")
    electives=c.fetchall()
    times=[]
    today=datetime.datetime.strptime(date,'%d-%m-%Y').weekday()
    week_start= datetime.datetime.strptime(date,'%d-%m-%Y') -
datetime.timedelta(days=today)
    week_start_date=week_start.strftime("%d-%m-%Y")
    week_end= week_start + datetime.timedelta(days=7)
    week_end_date = week_end.strftime("%d-%m-%Y")
    for su in subjects:
        c.execute("SELECT subject,day,start_time,end_time FROM original_routine where
subject='"+su[0]+"' and class='"+res[0]+"' and year='"+str(res[1])+"' and changed='N'")
        t=c.fetchall()
        for x in t:
            times.append(x)
        c.execute("SELECT subject,date,start_time,end_time,type FROM routine where
subject='" + su[0] + "' and class='" + res[0] +"' and year='"+str(res[1])+"' and istemp='N'")
        t=c.fetchall()
        for x in t:
            times.append(x)
        c.execute("SELECT subject,date,start_time,end_time,type FROM routine where
subject='" + su[
            0] + "' and class='" + res[0] + "' and year='" + str(res[1]) + "' and istemp='Y' and
date>='"+week_start_date+"' and date<='"+week_end_date+"'")
        t = c.fetchall()
        for x in t:
            times.append(x)
    for el in electives:
        c.execute("SELECT subject,day,start_time,end_time FROM original_routine where
subject='" + el[
            0] + "' and class='" + res[0] + "' and year='" + str(res[1]) + "' and changed='N'")
        t = c.fetchall()
        for x in t:
            times.append(x)
        c.execute("SELECT subject,date,start_time,end_time,type FROM routine where
subject='" + el[
            0] + "' and class='" + res[0] + "' and year='" + str(res[1]) + "' and istemp='N'")
        t = c.fetchall()
        for x in t:
            times.append(x)
        c.execute("SELECT subject,date,start_time,end_time,type FROM routine where
subject='" + el[
            0] + "' and class='" + res[0] + "' and year='" + str(res[
                1]) + "' and date>='" + week_start_date + "' and istemp='Y' and date<='" +
week_end_date + "'")
        t = c.fetchall()
        for x in t:
            times.append(x)
    week=dict()
    week_days=['MON','TUE','WED','THU','FRI','SAT','SUN']
    for tin in range(len(times)):
        ti=times[tin]
        if len(ti)==5:
            day=datetime.datetime.strptime(ti[1],'%d-%m-%Y').weekday()
            if week_days[day] in week:
                week[week_days[day]].append((ti[0],ti[2],ti[3],ti[4]))
            else:
                week[week_days[day]]=[(ti[0], ti[2], ti[3], ti[4])]
        else:
            if ti[1] in week:
                week[ti[1]].append((ti[0],ti[2],ti[3],'c'))
            else:
                week[ti[1]]=[(ti[0], ti[2], ti[3], 'c')]
    c.execute("SELECT * FROM holiday where date>='"+week_start_date + "' and date<='" +
week_end_date + "'")
    hhs=c.fetchall()
    hs,bc=[],[]
```

```
for h in hhs:
    d,n=h[0],h[1]
    day = datetime.datetime.strptime(d, '%d-%m-%Y').weekday()
    hs.append(week_days[day])
    bc.append(n)
tt=''
for w in week_days:
    if w in hs:
        tt+=w+": HOLIDAY- "+bc[hs.index(w)]+"\n"
        continue
    if w not in week:
        tt+=w+": No Classes\n"
        continue
    clss=week[w]
    clss.sort(key=lambda x: x[1])
    tt+=w+":\n"
    for cls in clss:
        tt+=cls[1]+"-"+cls[2]+" "+cls[0]+"("+cls[3]+")\n"
message=tt
```

RESULTS

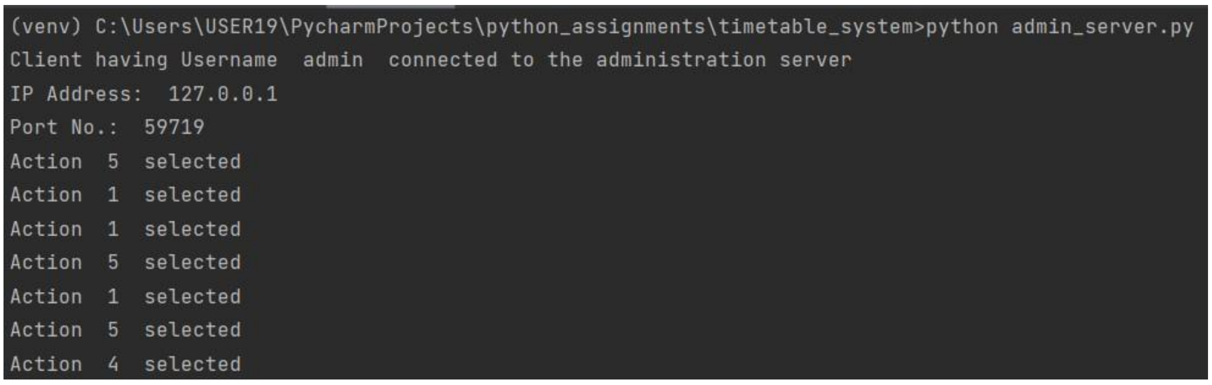
Sample 1: To check whether authorization server is running



```
(venv) C:\Users\USER19\PycharmProjects\python_assignments\timetable_system>python authorization.py
Client connected to the authorization server
IP Address:  127.0.0.1
Port No.:  59681
Client connected to the authorization server
IP Address:  127.0.0.1
Port No.:  59685
Client connected to the authorization server
IP Address:  127.0.0.1
Port No.:  59689
Client connected to the authorization server
IP Address:  127.0.0.1
Port No.:  59693
Client connected to the authorization server
IP Address:  127.0.0.1
Port No.:  59697
```

The above screenshot shows that the different clients are getting connected to the server.

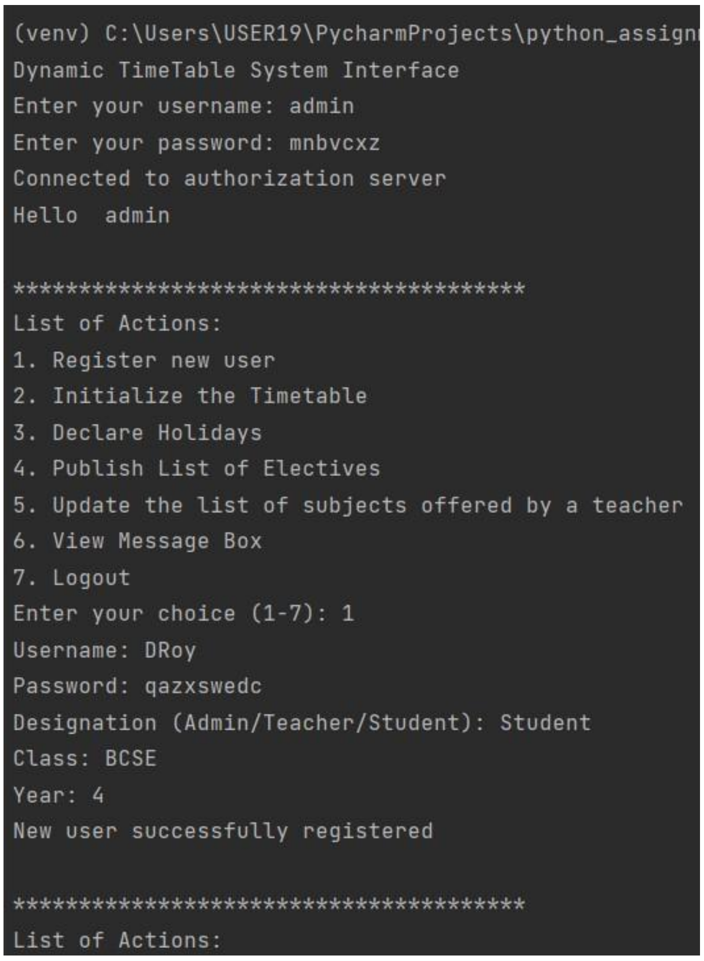
Sample 2: To check whether the actions to be performed are being communicated properly to the server (s)



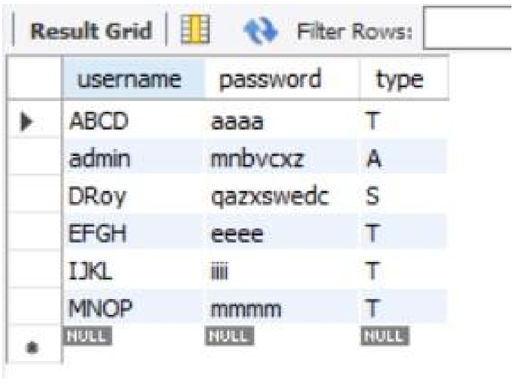
```
(venv) C:\Users\USER19\PycharmProjects\python_assignments\timetable_system>python admin_server.py
Client having Username  admin  connected to the administration server
IP Address:  127.0.0.1
Port No.:  59719
Action  5  selected
Action  1  selected
Action  1  selected
Action  5  selected
Action  1  selected
Action  5  selected
Action  4  selected
```

The above screenshot displays the action numbers being communicated to the server which matches the ones selected.

Sample 3: To check whether a new user is being added by the admin server

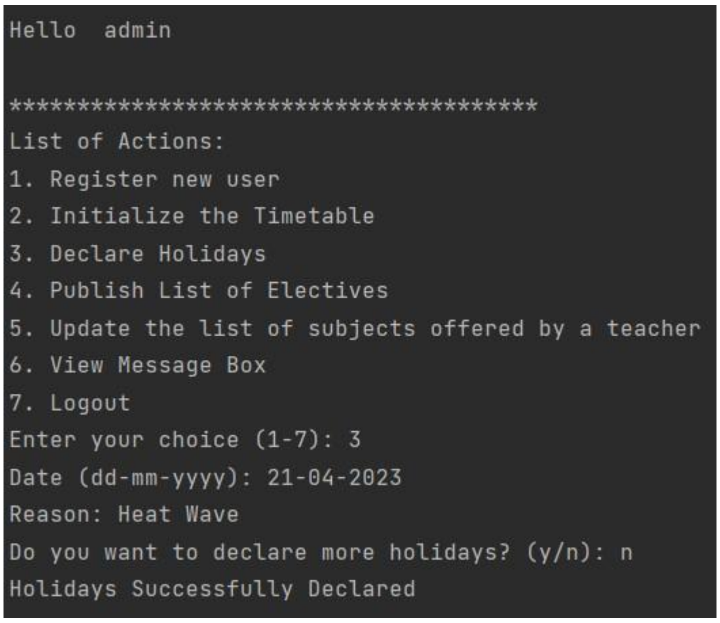


```
(venv) C:\Users\USER19\PycharmProjects\python_assign
Dynamic TimeTable System Interface
Enter your username: admin
Enter your password: mnbvcxz
Connected to authorization server
Hello  admin

****************************************
List of Actions:
1. Register new user
2. Initialize the Timetable
3. Declare Holidays
4. Publish List of Electives
5. Update the list of subjects offered by a teacher
6. View Message Box
7. Logout
Enter your choice (1-7): 1
Username: DRoy
Password: qazxswedc
Designation (Admin/Teacher/Student): Student
Class: BCSE
Year: 4
New user successfully registered

****************************************
List of Actions:
```

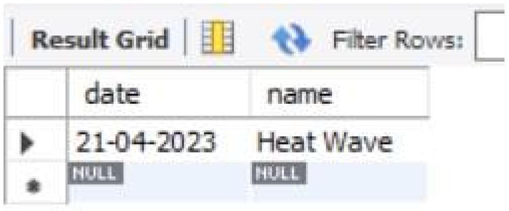


username	password	type
ABCD	aaaa	T
admin	mnbvcxz	A
DRoy	qazxswedc	S
EFGH	eeee	T
IJKL	iiii	T
MNOP	mmmm	T
NULL	NULL	NULL

Sample 4: To check whether holidays are being declared by the admin server

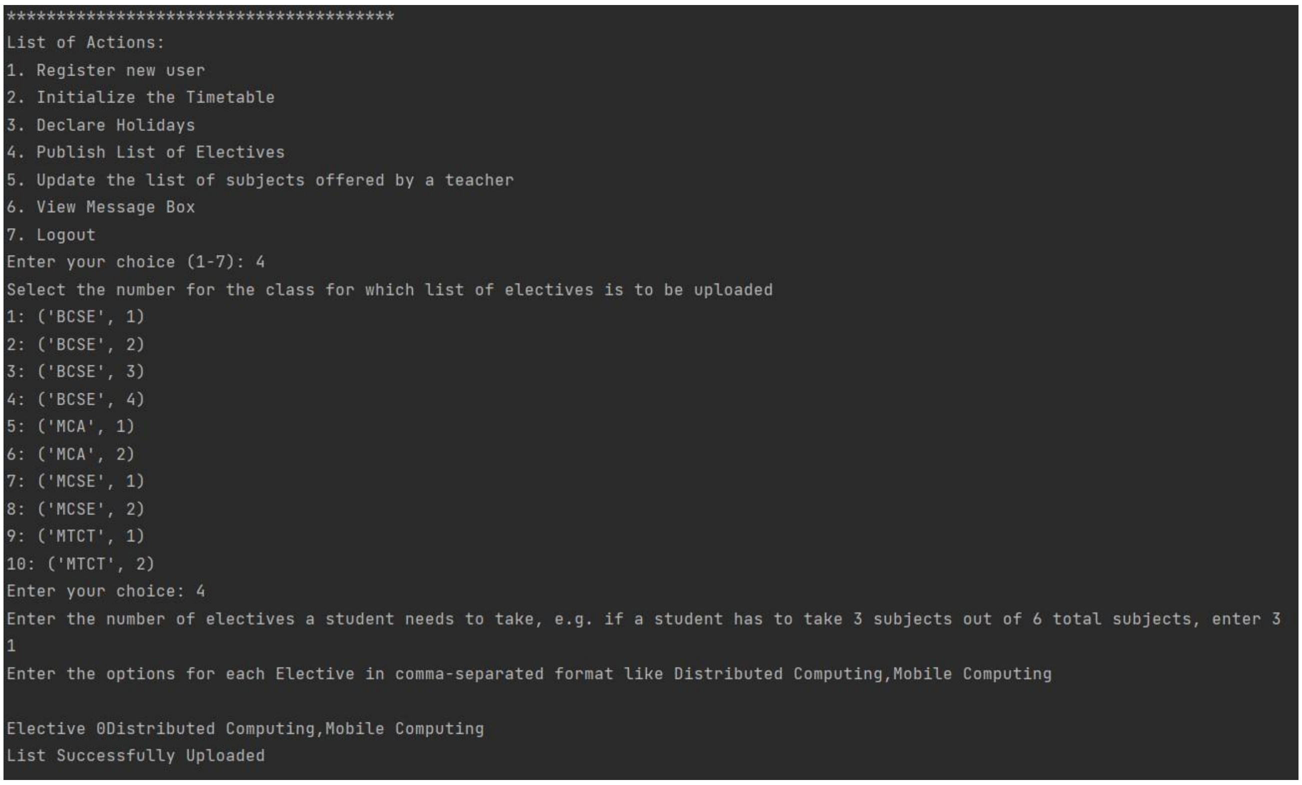


```
Hello  admin

***************************************
List of Actions:
1. Register new user
2. Initialize the Timetable
3. Declare Holidays
4. Publish List of Electives
5. Update the list of subjects offered by a teacher
6. View Message Box
7. Logout
Enter your choice (1-7): 3
Date (dd-mm-yyyy): 21-04-2023
Reason: Heat Wave
Do you want to declare more holidays? (y/n): n
Holidays Successfully Declared
```

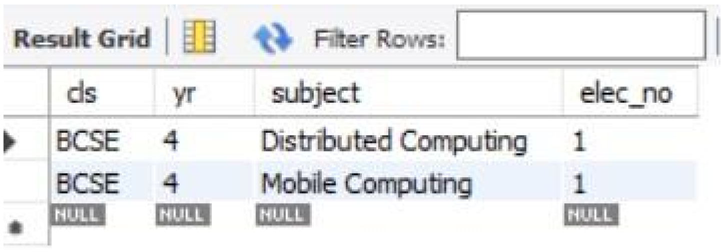


	date	name
▶	21-04-2023	Heat Wave
✱	NULL	NULL

Sample 5: To check whether list of electives is being uploaded

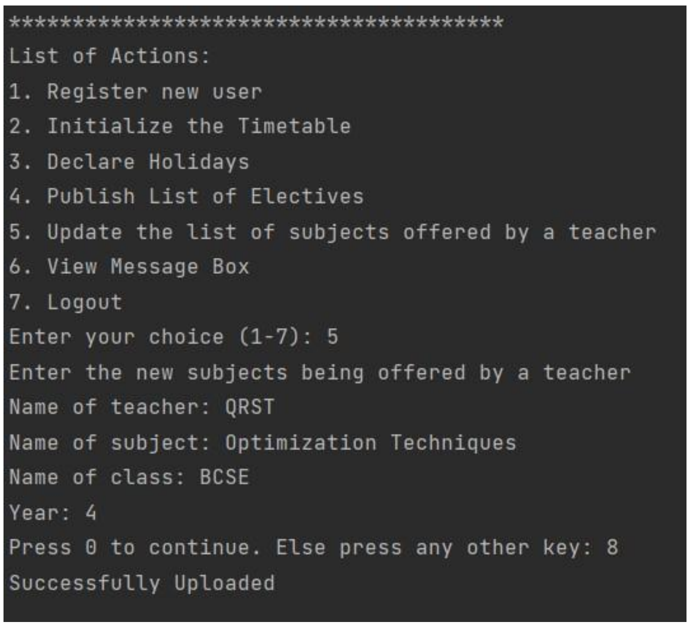


```
***************************************
List of Actions:
1. Register new user
2. Initialize the Timetable
3. Declare Holidays
4. Publish List of Electives
5. Update the list of subjects offered by a teacher
6. View Message Box
7. Logout
Enter your choice (1-7): 4
Select the number for the class for which list of electives is to be uploaded
1: ('BCSE', 1)
2: ('BCSE', 2)
3: ('BCSE', 3)
4: ('BCSE', 4)
5: ('MCA', 1)
6: ('MCA', 2)
7: ('MCSE', 1)
8: ('MCSE', 2)
9: ('MTCT', 1)
10: ('MTCT', 2)
Enter your choice: 4
Enter the number of electives a student needs to take, e.g. if a student has to take 3 subjects out of 6 total subjects, enter 3
1
Enter the options for each Elective in comma-separated format like Distributed Computing,Mobile Computing

Elective 0Distributed Computing,Mobile Computing
List Successfully Uploaded
```

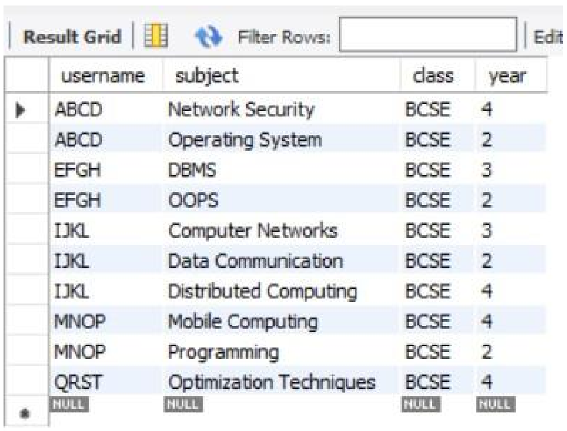


cls	yr	subject	elec_no
BCSE	4	Distributed Computing	1
BCSE	4	Mobile Computing	1
NULL	NULL	NULL	NULL

Sample 6: To check whether teacher-subject combinations are being uploaded

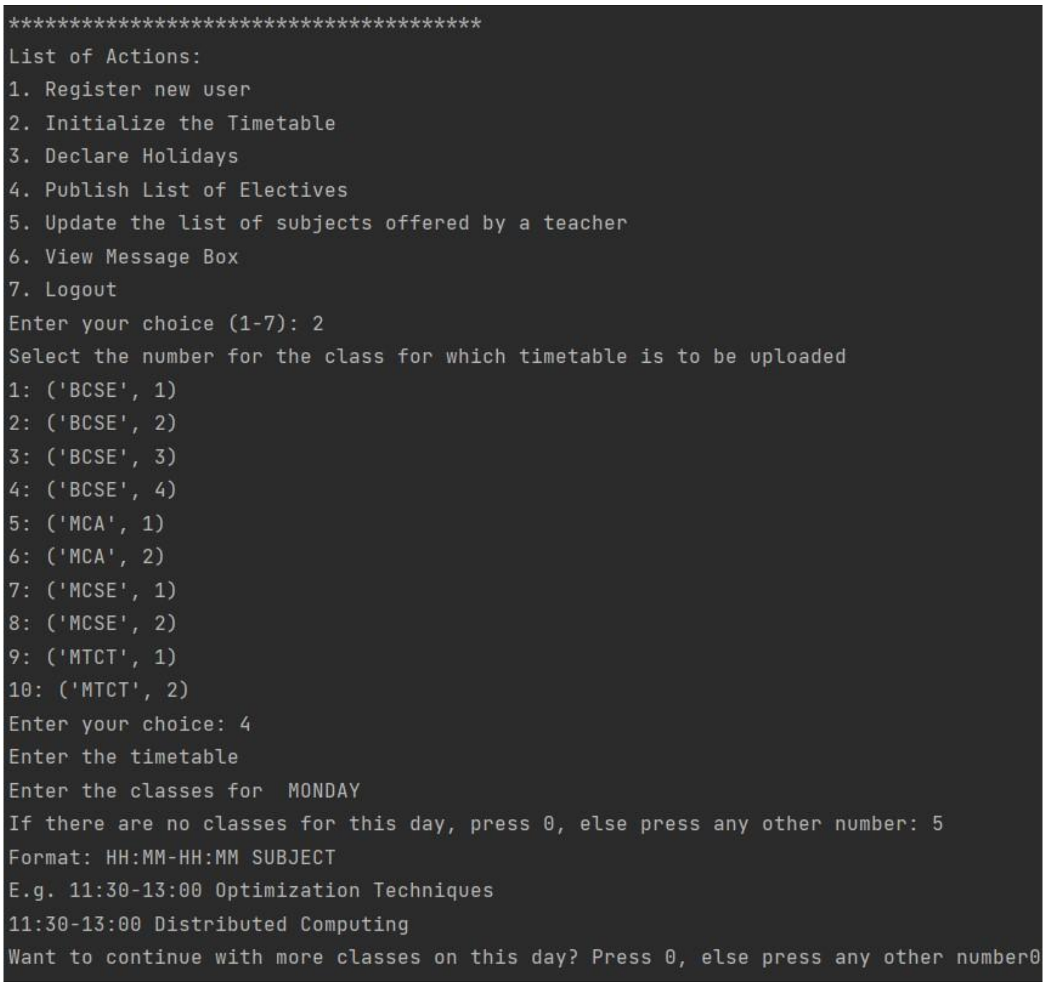


```
****************************************
List of Actions:
1. Register new user
2. Initialize the Timetable
3. Declare Holidays
4. Publish List of Electives
5. Update the list of subjects offered by a teacher
6. View Message Box
7. Logout
Enter your choice (1-7): 5
Enter the new subjects being offered by a teacher
Name of teacher: QRST
Name of subject: Optimization Techniques
Name of class: BCSE
Year: 4
Press 0 to continue. Else press any other key: 8
Successfully Uploaded
```

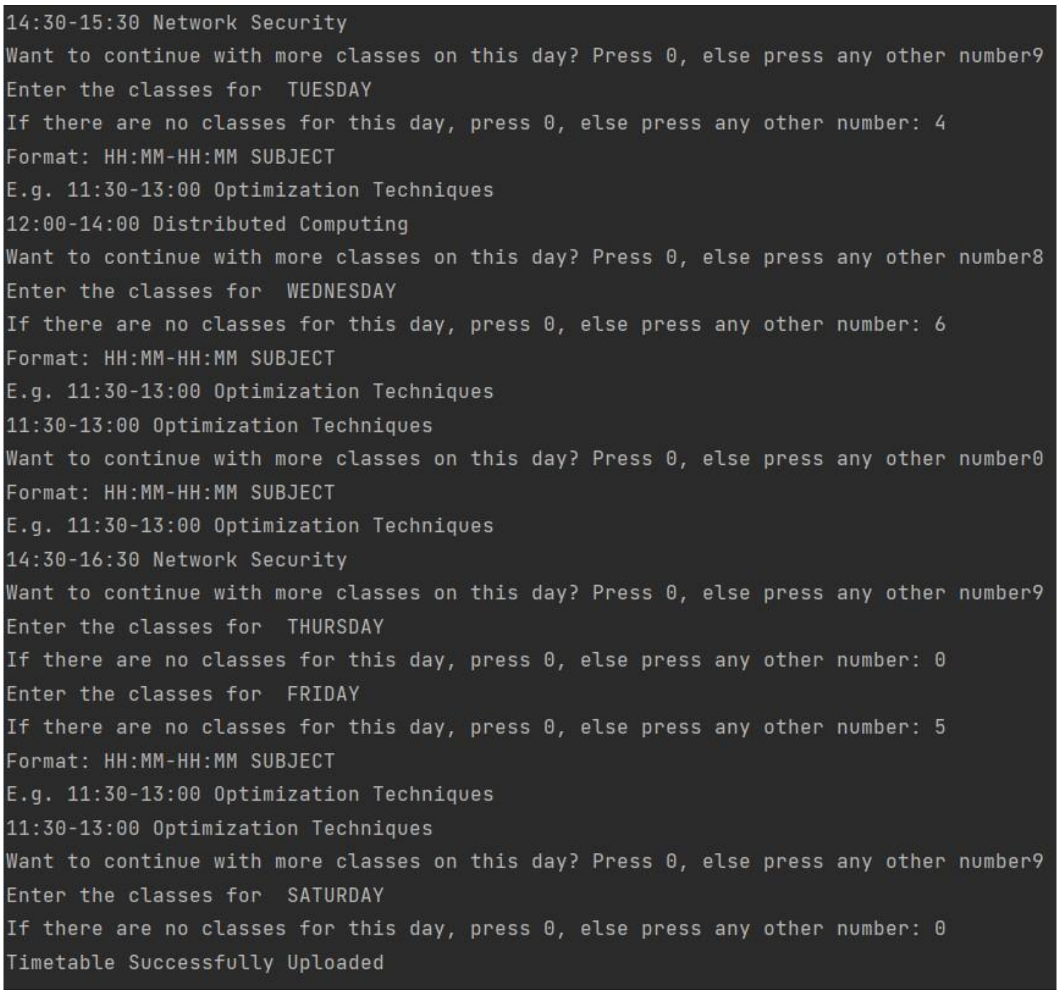


username	subject	class	year
ABCD	Network Security	BCSE	4
ABCD	Operating System	BCSE	2
EFGH	DBMS	BCSE	3
EFGH	OOPS	BCSE	2
IJKL	Computer Networks	BCSE	3
IJKL	Data Communication	BCSE	2
IJKL	Distributed Computing	BCSE	4
MNOP	Mobile Computing	BCSE	4
MNOP	Programming	BCSE	2
QRST	Optimization Techniques	BCSE	4
NULL	NULL	NULL	NULL

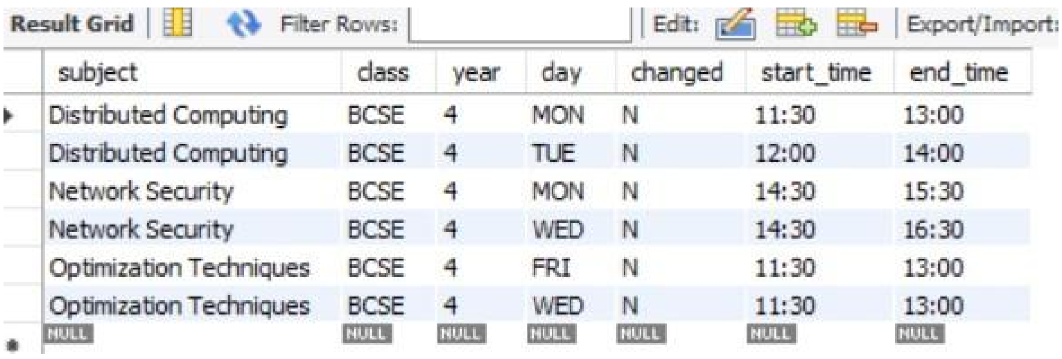
Sample 7: To check whether timetable can be uploaded by the admin



```
***************************************
List of Actions:
1. Register new user
2. Initialize the Timetable
3. Declare Holidays
4. Publish List of Electives
5. Update the list of subjects offered by a teacher
6. View Message Box
7. Logout
Enter your choice (1-7): 2
Select the number for the class for which timetable is to be uploaded
1: ('BCSE', 1)
2: ('BCSE', 2)
3: ('BCSE', 3)
4: ('BCSE', 4)
5: ('MCA', 1)
6: ('MCA', 2)
7: ('MCSE', 1)
8: ('MCSE', 2)
9: ('MTCT', 1)
10: ('MTCT', 2)
Enter your choice: 4
Enter the timetable
Enter the classes for  MONDAY
If there are no classes for this day, press 0, else press any other number: 5
Format: HH:MM-HH:MM SUBJECT
E.g. 11:30-13:00 Optimization Techniques
11:30-13:00 Distributed Computing
Want to continue with more classes on this day? Press 0, else press any other number0
```

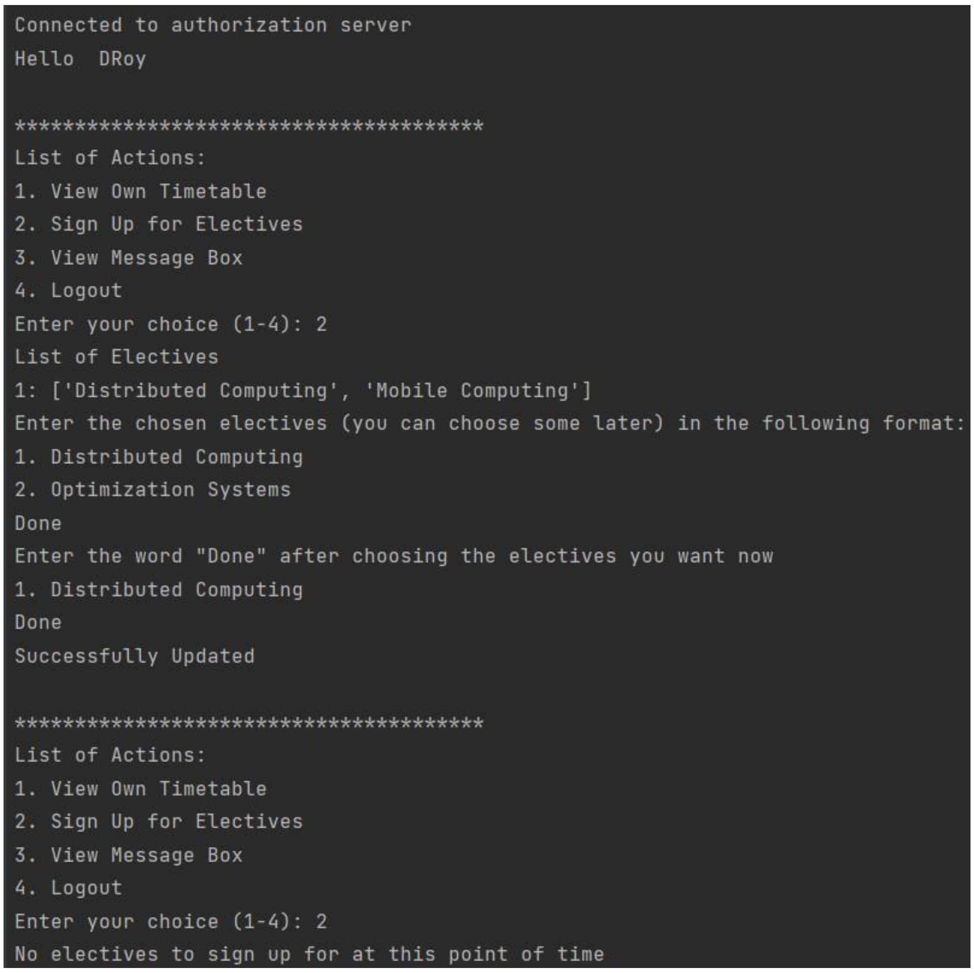


```
14:30-15:30 Network Security
Want to continue with more classes on this day? Press 0, else press any other number9
Enter the classes for  TUESDAY
If there are no classes for this day, press 0, else press any other number: 4
Format: HH:MM-HH:MM SUBJECT
E.g. 11:30-13:00 Optimization Techniques
12:00-14:00 Distributed Computing
Want to continue with more classes on this day? Press 0, else press any other number8
Enter the classes for  WEDNESDAY
If there are no classes for this day, press 0, else press any other number: 6
Format: HH:MM-HH:MM SUBJECT
E.g. 11:30-13:00 Optimization Techniques
11:30-13:00 Optimization Techniques
Want to continue with more classes on this day? Press 0, else press any other number0
Format: HH:MM-HH:MM SUBJECT
E.g. 11:30-13:00 Optimization Techniques
14:30-16:30 Network Security
Want to continue with more classes on this day? Press 0, else press any other number9
Enter the classes for  THURSDAY
If there are no classes for this day, press 0, else press any other number: 0
Enter the classes for  FRIDAY
If there are no classes for this day, press 0, else press any other number: 5
Format: HH:MM-HH:MM SUBJECT
E.g. 11:30-13:00 Optimization Techniques
11:30-13:00 Optimization Techniques
Want to continue with more classes on this day? Press 0, else press any other number9
Enter the classes for  SATURDAY
If there are no classes for this day, press 0, else press any other number: 0
Timetable Successfully Uploaded
```

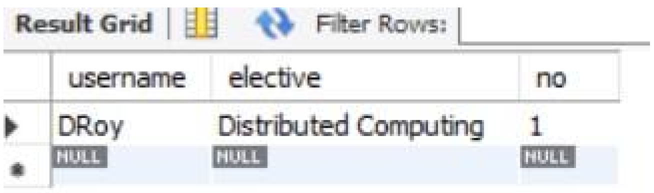


subject	class	year	day	changed	start_time	end_time
Distributed Computing	BCSE	4	MON	N	11:30	13:00
Distributed Computing	BCSE	4	TUE	N	12:00	14:00
Network Security	BCSE	4	MON	N	14:30	15:30
Network Security	BCSE	4	WED	N	14:30	16:30
Optimization Techniques	BCSE	4	FRI	N	11:30	13:00
Optimization Techniques	BCSE	4	WED	N	11:30	13:00
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Sample 8: To check whether students can select their elective

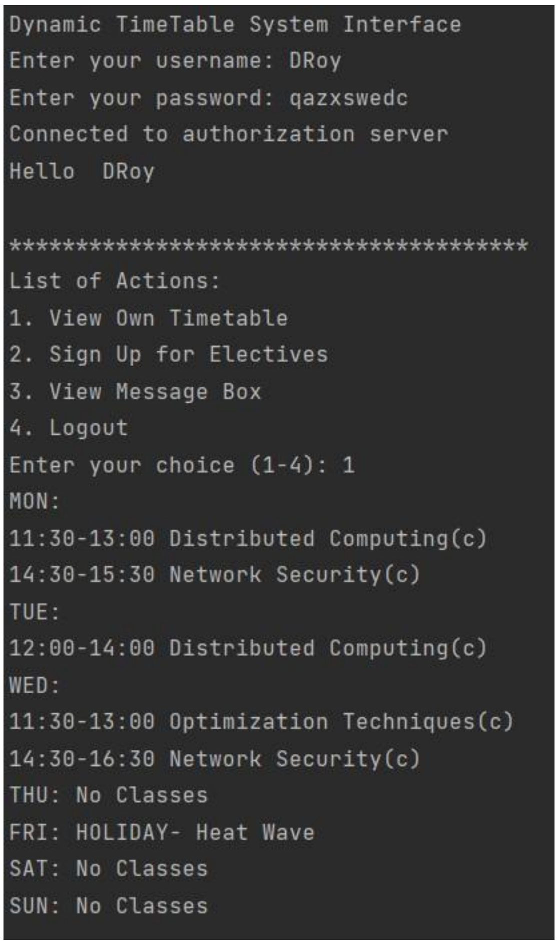


```
Connected to authorization server
Hello  DRoy

***************************************
List of Actions:
1. View Own Timetable
2. Sign Up for Electives
3. View Message Box
4. Logout
Enter your choice (1-4): 2
List of Electives
1: ['Distributed Computing', 'Mobile Computing']
Enter the chosen electives (you can choose some later) in the following format:
1. Distributed Computing
2. Optimization Systems
Done
Enter the word "Done" after choosing the electives you want now
1. Distributed Computing
Done
Successfully Updated

***************************************
List of Actions:
1. View Own Timetable
2. Sign Up for Electives
3. View Message Box
4. Logout
Enter your choice (1-4): 2
No electives to sign up for at this point of time
```



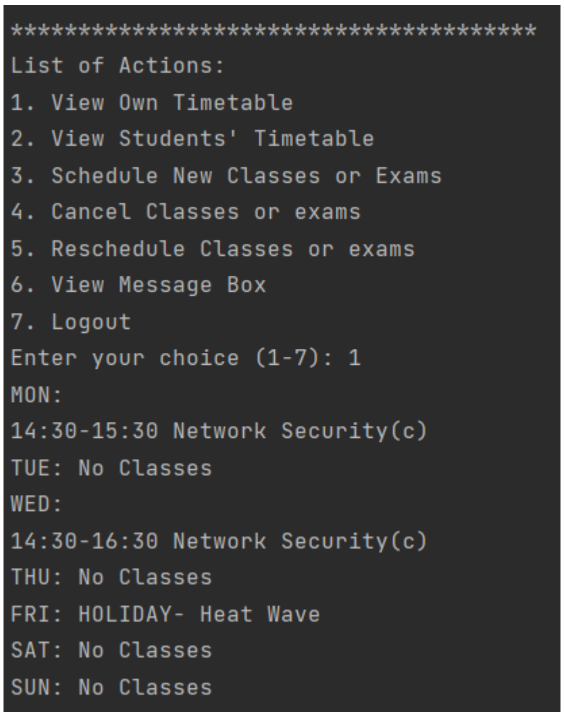
username	elective	no
DRoy	Distributed Computing	1
NULL	NULL	NULL

Sample 9: To check whether students can view their timetable



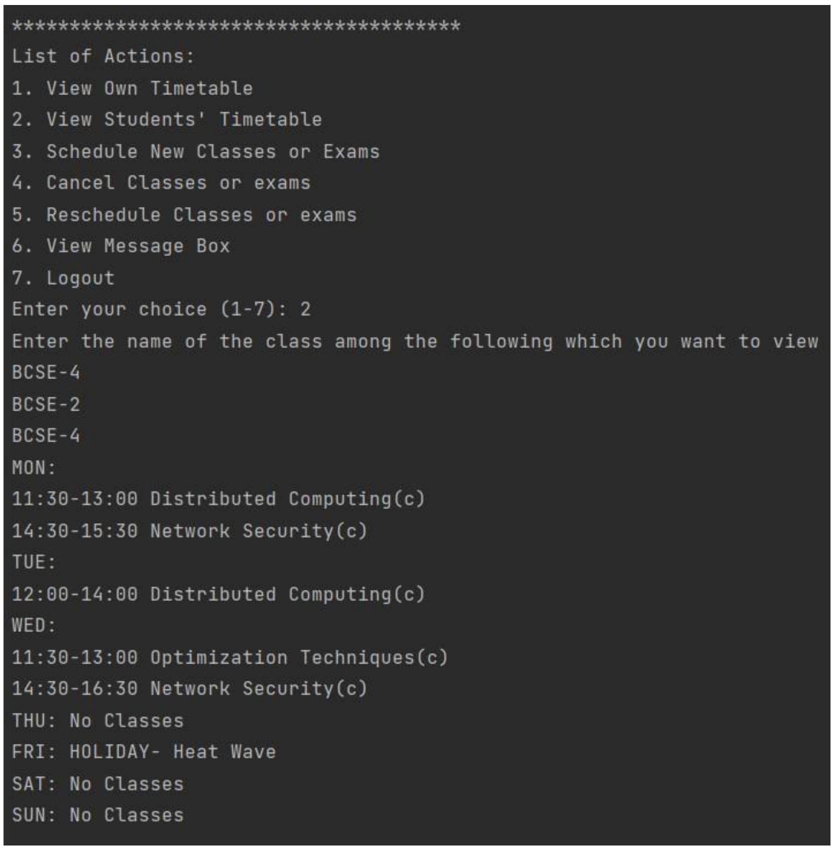
```
Dynamic TimeTable System Interface
Enter your username: DRoy
Enter your password: qazxswedc
Connected to authorization server
Hello  DRoy

***************************************
List of Actions:
1. View Own Timetable
2. Sign Up for Electives
3. View Message Box
4. Logout
Enter your choice (1-4): 1
MON:
11:30-13:00 Distributed Computing(c)
14:30-15:30 Network Security(c)
TUE:
12:00-14:00 Distributed Computing(c)
WED:
11:30-13:00 Optimization Techniques(c)
14:30-16:30 Network Security(c)
THU: No Classes
FRI: HOLIDAY- Heat Wave
SAT: No Classes
SUN: No Classes
```

Sample 10: To check whether teachers can view their own timetable



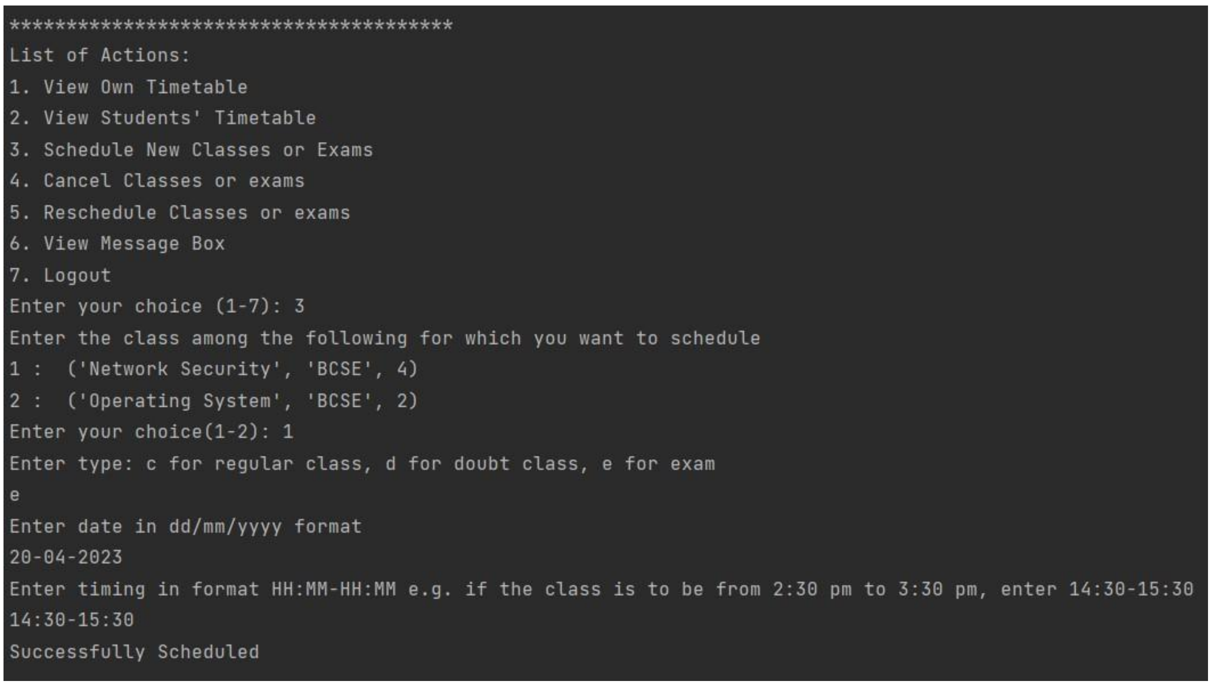
```
***************************************
List of Actions:
1. View Own Timetable
2. View Students' Timetable
3. Schedule New Classes or Exams
4. Cancel Classes or exams
5. Reschedule Classes or exams
6. View Message Box
7. Logout
Enter your choice (1-7): 1
MON:
14:30-15:30 Network Security(c)
TUE: No Classes
WED:
14:30-16:30 Network Security(c)
THU: No Classes
FRI: HOLIDAY- Heat Wave
SAT: No Classes
SUN: No Classes
```

Sample 11: To check whether teachers can view students' timetable



```
***************************************
List of Actions:
1. View Own Timetable
2. View Students' Timetable
3. Schedule New Classes or Exams
4. Cancel Classes or exams
5. Reschedule Classes or exams
6. View Message Box
7. Logout
Enter your choice (1-7): 2
Enter the name of the class among the following which you want to view
BCSE-4
BCSE-2
BCSE-4
MON:
11:30-13:00 Distributed Computing(c)
14:30-15:30 Network Security(c)
TUE:
12:00-14:00 Distributed Computing(c)
WED:
11:30-13:00 Optimization Techniques(c)
14:30-16:30 Network Security(c)
THU: No Classes
FRI: HOLIDAY- Heat Wave
SAT: No Classes
SUN: No Classes
```

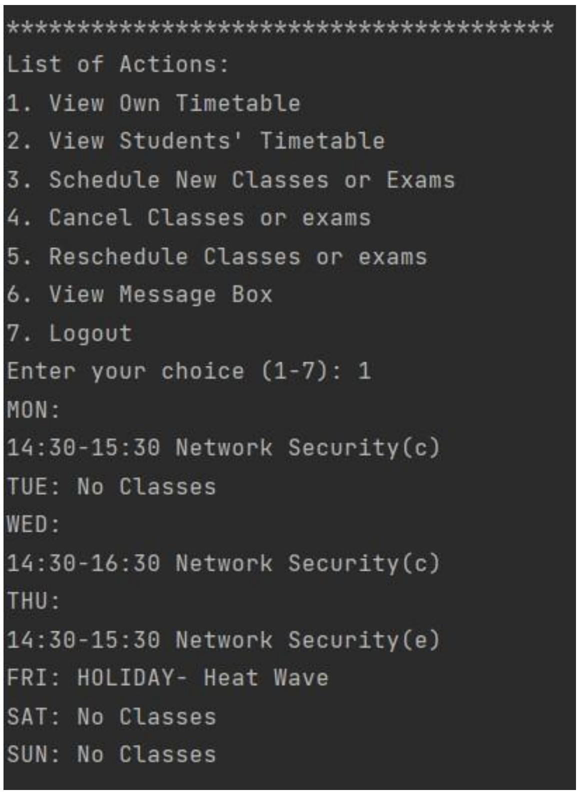
Sample 12: To check whether teachers can edit the timetable



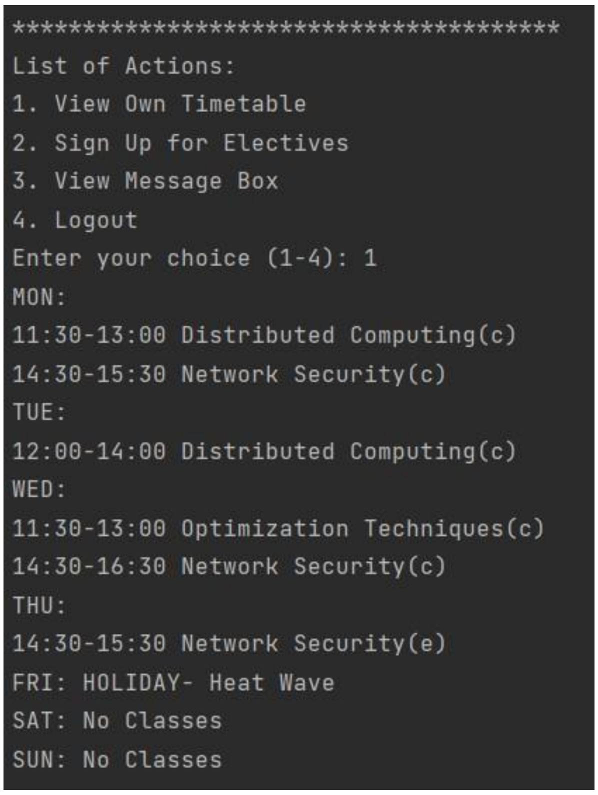
```
***************************************
List of Actions:
1. View Own Timetable
2. View Students' Timetable
3. Schedule New Classes or Exams
4. Cancel Classes or exams
5. Reschedule Classes or exams
6. View Message Box
7. Logout
Enter your choice (1-7): 3
Enter the class among the following for which you want to schedule
1 :  ('Network Security', 'BCSE', 4)
2 :  ('Operating System', 'BCSE', 2)
Enter your choice(1-2): 1
Enter type: c for regular class, d for doubt class, e for exam
e
Enter date in dd/mm/yyyy format
20-04-2023
Enter timing in format HH:MM-HH:MM e.g. if the class is to be from 2:30 pm to 3:30 pm, enter 14:30-15:30
14:30-15:30
Successfully Scheduled
```

Here, the teacher successfully scheduled an exam on 20-04-2023.

The changes are reflected in the teacher’s own timetable as well as the timetable of a student taking Network Security.

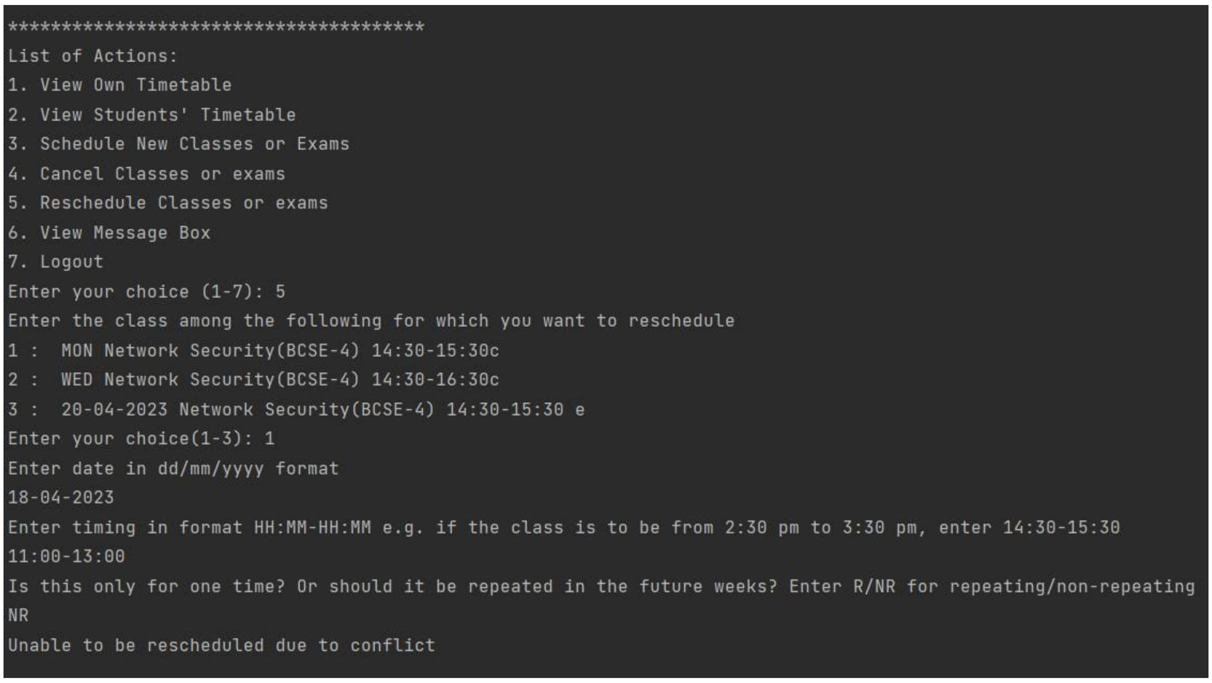


```
***************************************
List of Actions:
1. View Own Timetable
2. View Students' Timetable
3. Schedule New Classes or Exams
4. Cancel Classes or exams
5. Reschedule Classes or exams
6. View Message Box
7. Logout
Enter your choice (1-7): 1
MON:
14:30-15:30 Network Security(c)
TUE: No Classes
WED:
14:30-16:30 Network Security(c)
THU:
14:30-15:30 Network Security(e)
FRI: HOLIDAY- Heat Wave
SAT: No Classes
SUN: No Classes
```



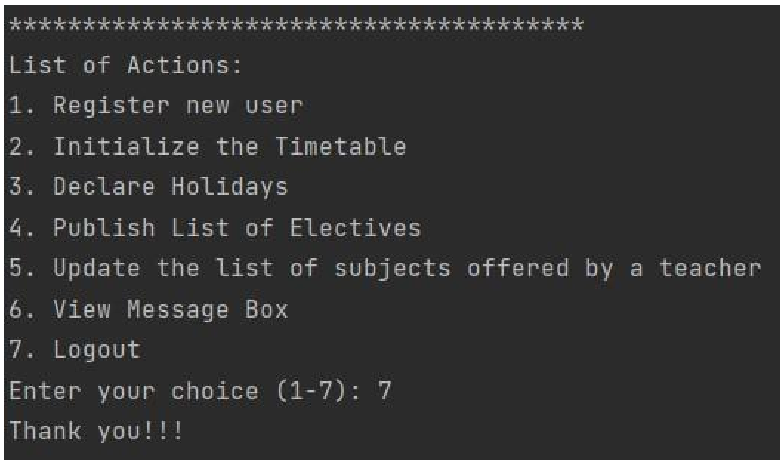
```
***************************************
List of Actions:
1. View Own Timetable
2. Sign Up for Electives
3. View Message Box
4. Logout
Enter your choice (1-4): 1
MON:
11:30-13:00 Distributed Computing(c)
14:30-15:30 Network Security(c)
TUE:
12:00-14:00 Distributed Computing(c)
WED:
11:30-13:00 Optimization Techniques(c)
14:30-16:30 Network Security(c)
THU:
14:30-15:30 Network Security(e)
FRI: HOLIDAY- Heat Wave
SAT: No Classes
SUN: No Classes
```

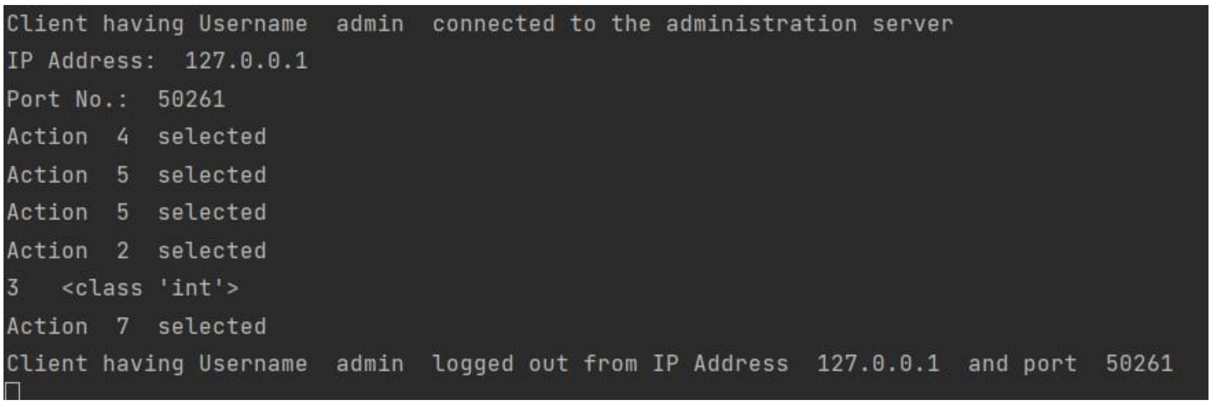
Sample 13: To check whether the conflict checking for classes is working



```
***************************************
List of Actions:
1. View Own Timetable
2. View Students' Timetable
3. Schedule New Classes or Exams
4. Cancel Classes or exams
5. Reschedule Classes or exams
6. View Message Box
7. Logout
Enter your choice (1-7): 5
Enter the class among the following for which you want to reschedule
1 :  MON Network Security(BCSE-4) 14:30-15:30c
2 :  WED Network Security(BCSE-4) 14:30-16:30c
3 :  20-04-2023 Network Security(BCSE-4) 14:30-15:30 e
Enter your choice(1-3): 1
Enter date in dd/mm/yyyy format
18-04-2023
Enter timing in format HH:MM-HH:MM e.g. if the class is to be from 2:30 pm to 3:30 pm, enter 14:30-15:30
11:00-13:00
Is this only for one time? Or should it be repeated in the future weeks? Enter R/NR for repeating/non-repeating
NR
Unable to be rescheduled due to conflict
```

Sample 14: To check whether Logout functionality is working


```
***************************************
List of Actions:
1. Register new user
2. Initialize the Timetable
3. Declare Holidays
4. Publish List of Electives
5. Update the list of subjects offered by a teacher
6. View Message Box
7. Logout
Enter your choice (1-7): 7
Thank you!!!
```


```
Client having Username  admin  connected to the administration server
IP Address:  127.0.0.1
Port No.:  50261
Action  4  selected
Action  5  selected
Action  5  selected
Action  2  selected
3   <class 'int'>
Action  7  selected
Client having Username  admin  logged out from IP Address  127.0.0.1  and port  50261
```

COMPARISON WITH CENTRALISED SOLUTION

A centralised solution with one server serving multiple clients can also be implemented for the problem.

- **Security**: As each server has only a partial view of the entire system, it can be configured such that each server gets access to only the part of the database it needs for its work. Security is not a big concern for most of the data — the timetables and the subjects can be accessed by all (admin, student, teacher) servers as it stands — but the main concern is the passwords for the usernames which need to be uncompromised. With a dedicated authorization server and an administrative server, the access can be secured (write access given to administrative server and only read access given to authorization server) unlike in a centralised server, where the whole server had to be given full access to the database.
- **Load Balancing and Scalability**: The distributed system ensures that clients get redirected to their own dedicated servers, ensuring that one server does not have to cater to all clients unlike in a centralised system. If more clients of a particular domain need to be served, then only that part (server) can be replicated instead of all the parts in a centralised system (a single server has all the parts).
- **Extensibility**: A distributed system enables one to easily add another service to the existing system or take down or modify a part which is no longer required. In a

distributed system, a new server can be set up for new functionality and any updates to the existing servers due to this can be done in much less down-time than if there was only one server (which would have to be down for a longer time to add new functionality and to test before being back online).

- **Availability**: A distributed system would be able to have less overall down-time for new updates and for bugs as it is highly unlikely that all the servers will fail at the same time. In a centralised system, if one service fails, the entire system will have to be brought down for updates and debugging.
- **Updates**: It is easier to update a single new feature in a distributed system than in a centralised system where we would need to rebuild and retest the entire system for it.
- **Separation of Concerns**: The different client domains have dedicated servers catering to them in a distributed system. Thus it becomes easier to understand and debug the code if necessary.
- **Point(s) of Failure**: The distributed system has multiple potential points of failure as opposed to a centralised system having a single point of failure.
- **Cost and Energy Consumption:** As multiple servers would be running at any point of time in a distributed system as opposed to a single server, the cost and energy consumption would be higher. Also, it is not as if the task is data-intensive or computing-intensive which would justify the extra energy consumption (by increasing computational or storage efficiency).
- **Software Complexity**: It is more complex and harder to develop a distributed system, so development costs would be higher and there is a greater chance of introducing errors so any new feature should be thoroughly tested before it comes online.
- **Network Issues and Speed**: Due to there being multiple servers and a need to connect to multiple servers (authorization and then the respective domain server), the speed suffers. Networks are also potential points of failure and have bandwidth limitations.
- **Synchronisation and Consistency**: The different servers connect to a database where they modify / view its contents. As there are multiple servers and each server can serve multiple clients simultaneously, synchronisation would become a bigger issue with time and growth.

CONCLUSION

A distributed computing solution was proposed to solve the identified problem. A prototype was developed based on the design and tested in various situations. Finally, it was compared to a centralised solution for the same problem. If cost, energy consumption, speed and developmental complexity are considered, a centralised solution is better, especially if the application caters to a limited number of people (like for a single university department as it is currently developed for) such that scalability does not become a concern. However, if one wants to expand the system to include a number of departments in the university, a distributed solution would be preferable. A distributed system is better if we want to separate the different client domains into different servers for easy debugging or updates or managing security for only a sensitive part of the database. A distributed system would also be more extensible and have a higher availability than a centralised system. Synchronisation and consistency are issues that we would have to consider whether the solution is distributed or centralised, but it would be more difficult for a distributed system. So, we can conclude that one should decide whether a distributed or a centralised approach is needed based on one's requirements.

SCOPE FOR IMPROVEMENTS: The interface right now is command-line based and it can be replaced by a graphical user interface for better user experience. It can also be made into a web-based application or a mobile application. More functionality like recording attendance in classes or exams and individual messaging feature and scheduling project work or placement commitments and extra-curricular activities that a student may be involved in can also be added. If it is expanded to include more departments (rather than the one department it is currently configured for), some servers (like the student server or the teacher server or authorization server) can be replicated to handle the increase in demand.