HOSTEL HELP DESK

DESCRIPTION

The hostel help desk system functions in such a way that the hostellers can put forth their queries which will be answered by theis fellowmates. They can order their for their T-shirts.They can see both veg and non veg menus.

ENTITES INVLOVED

Hostellers-They can ask help for any doubts.They can order the T-shirts.The can see the Hostel day menu.

FUNCTIONS USED:

Interface:

frame()-Used as a geometry master for other widgets

button()-used to attach function or a merhod to a button

pack()-Manages widgets in blocks before placing them in parent widget

label()-Used to display text or image and it is just a view

entry()-Used to accept single line text from user

To display multiple windows at the same time we used popen() in os module.

DataBase used-Sqlite3

db.connect()-to connect to database

db.commit()-to save the performed actions in the database

Socket Programming:

socket()-It creates a end point for network connection.

bind()-Attaching to an IP and Port

listen()-Wait for connection

accept()-Accept new connection from new clients trying to connect to the server

connect()-Called by the client to connect to the server port

send()-used to send the message

recv()-used to receive the message.

TOOLS USED:

Pthon3

SQLite3

tkinter

ALGORITHM:

Server.py

```
1. accept_incoming_connections() //function to accept client request
    1.1. Repeat //Loop to accept client
        1.1.1. client,client_address←SERVER.accept()
        1.1.2. print(The user is connected)
        1.1.3. client.send(You are connected)
        1.1.4. addresses[client]←client_address //storing corresponding client address
               in dictionary with client name as key
        1.1.5. Thread(handle_client) //connect and disconnect client
2. Handle_client(client)
    2.1. name←client.name
    2.2. client.send(welcome)
    2.3. msg←user joined the chat
    2.4. broadcast(msg) //it will send to all clients
    2.5. clients[client]←name //to store name of the client in dictionary
    2.6. Repeat //Receive the message send by client
        2.6.1. msg←client.recv
        2.6.2. if (msg!=quit)
            2.6.2.1. broadcast(msg)
        2.6.3. else
            2.6.3.1. client.send(quit)
            2.6.3.2. client.close()
            2.6.3.3. del clients[client] //client name and address will be deleted
            2.6.3.4. broadcast(the user left the chat)
            2.6.3.5. break
3. broadcast(msg)
```

```
        3.1. for sock in clients:
                3.1.1.  sock.send(msg)
4.  clients←{} // to store client name
5.  addresses←{} //to store client address
6.  host←' '
7.  port←3300
8.  bufsiz←1024
9.  addr←(host,port)
10. SERVER←socket(AF_INET,SOCK_STREAM)
11. SERVER.bind(addr)
12. SERVER.listen(5)
13. Print(waiting for connection)
```

Client.py

```
1.  Client_socket←socket(AF_INET,SOCK_STREAM)
2.  Receive()
    2.1. repeat
        2.1.1.  try:
        2.1.2.  msg=client_socket.recv(msg)
        2.1.3.  msg_list.insert(msg)
        2.1.4.  except OS error: //Client may left the chat
        2.1.5.  break
3.  send(event=None) //it is passed by binders
    3.1. msg←my_msg.get()
    3.2. my_msg.set("")
    3.3. client_socket.send(msg)
    3.4. if (msg==quit)
    3.5. client_socket.close()
    3.6. quit()
```

```
4.  on_closing(event=None)
    4.1. my_msg.set(quit)
    4.2. send()
5.  host←input(enter host)
6.  port←3300
7.  bufsiz←1024
8.  addr←(host,port)
9.  client_socket.connect(addr)
10. receice_thread←Thread(target=receive)
11. receive_thread.start()
```