Easy implementation

First of all, create a “console project” in “dotnet core”.

Add following namespaces

```
using TcpServerKit.Core.Tcp;
```

Then initialize server ip and port

```
Server.InitServerAsGuest("127.0.0.1", 3000);
```

Define two properties for client’s event

```
Server.NewClientJoined += (TcpClient client) =>
      {
      Console.WriteLine("new client join");
      };
      Server.ClientExited += (TcpClient client) =>
      {
      Console.WriteLine("client exit");
      };
```

Now your server is ready to start, but before that we need to add listeners

```
Server.AddListener("Login", Login);
```

In this example we add one listener with eventName “Login”.
After the client sends a message with the same “eventName”, the Login function is invoked.

```
void Login(string data, TcpClient client)
{


}
```

`data` is a string data which client sends to the server and `client` is a TcpClient object of who sends it.

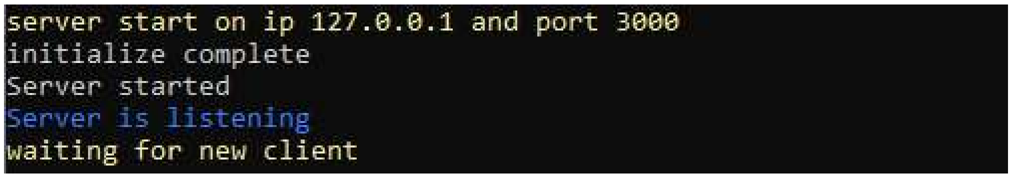
Very well. After adding all of the listeners we can start the server.

```
Server.StartServer();
```

There is it. All you need to run your server.

And don’t forget to put a `Console.ReadKey();` To prevent the console from closing.

See the result.



```
server start on ip 127.0.0.1 and port 3000
initialize complete
Server started
Server is listening
waiting for new client
```

Now we will work with users and rooms.
For that, create a new class with name `MyUser` and inherit from `User’ then implement constructors.
For that we need to import the following namespaces

```
using System;
using System.Net.Sockets;
using TcpServerKit.Core;


public class MyUser : User
{
      public MyUser()
      {
      }

      public MyUser(TcpClient tcpClient) : base(tcpClient)
      {
      }
}
```

So, create another class for Room and name it `MyRoom` and inherit from `Room`.
Implement constructor and functions.

See the sample

```
using System.Net.Sockets;
using TcpServerKit.Core;
using TcpServerKit.Core.Tcp;
using TcpServerKit.Manager;


    public class MyRoom : Room
    {

        public MyRoom(int roundCount) : base(roundCount)
        {

        }

        public override void GameCompleted()
        {
           // invoke when game completed
        }

        public override void NewUserJoined(User user)
        {
            // invoke when new user joined
        }

        public override void RoomReadyForStart(List<User> users)
        {
            // invoke when room ready for start
        }

        public override void RoundStarted(ushort roundId)
        {
            // invoke when new round started
        }

        public override void UserExited(User user)
        {
            // invoke when an user exit from room or his connection lost
        }

        public override void UserKicked(User user)
        {
            // invoke when an user kicked from room
        }
    }
```

So after creating my classes, we want to create a new `MyUser’ object

In the Login function create this.

```
Private void Login(string data, TcpClient client)
{
      var user = new MyUser(client);

      Console.WriteLine("new user with UniqueId{user.UniqueId} logined");
      Server.Send(user, "Login", "{ \"result\": true, id: " +
          user.UniqueId + "}");

}
```

In the first line create the user
Next line writes a message on the console and shows the user's unique id, `UniqueId` automatically generated.
And Users will automatically be added to the users list.

Next line we send a message to the client with the event name “Login” and a json message which sends the user`s unique id.

Event name must be implemented at the client side.

After this, we will join user to a room and start playing the game

Add a new listener and name it `Join`.

```
Private void Join(string data, TcpClient client)
{
      Var user = UserManager.FindUser(client);

      var result = RoomManager.Join(user);

      if (!result)
      {
          var room = new MyRoom(4);
          room.UsersCount = new TcpServerKit.Core.Range(2);
```

```
            room.AddUser(user);
        }
        Else
        {
            Server.Send(user, "Join", string.Empty);
        }
}
```

In the first line we find the user with his client.
Next line we will join the user to a random room and it returns “true” value if the user joins successfully.

Next line, if the result is “false”
Create a new room and then join the user to it.
Number `4` means the number of rounds of the game we're going to play, the default value is `2`.
Next line we specify how many users can join this room.
it can be a fixed value or a range of values,for example

```
room.UsersCount = new TcpServerKit.Core.Range(2, 4);
```

Next line we add the user to the room.

After that `NewUserJoined` will be invoked.

```
public override void NewUserJoined(User user)
{
       Console.WriteLine("new user with UniqueId{user.UniqueId} joined");
}
```

After that all users join the room `RoomReadyForStart` will be invoked.

```
public override void RoomReadyForStart(List<User> users)
{
       Console.WriteLine("RoomReadyForStart");
}
```

if we specify users count as a ranged value, in the previous example
Our range is between 2 and 4.
If users who join to room are 3 or 2, we can start the game but the room is not full yet.
For that we can check if all users joined the room with `AllUsersJoined`.

It returns true if all users joined and the room is full of users.

Server

OnlineClientsCounts : returns online users count.
NewClientJoined : invokes when new client joins.
ClientExited : invokes when client Disconnects.

InitServerAsGuest(string ip, int port) : set server ip and port.
InitServer(string licencePath, int port) loads licence and gets server ip from licence and sets specified port.
StartServer : start the server with specified ip and port.
CloseConnection : disconnects a client from the server.
Send(User user, string eventName, string message) : sends a string to user.
AddListener(string eventName, MuEvent event) : adds a listener for server.

muEvent :
```
MuEvent(string data, TcpClient client)
```

Room

UniqueId : Unique id of the room.
Password : password of the room.
UsersCount : range of users which can join the room.
Level : level of room, room can have level for simple match making, type is a range.
AccessMode : default value of access mode is `AccessMode.Public` which means any users can join it, `AccessMode.Private` is for no public Rooms and just with his `UniqueId` and users in it are available.
GameStarted : is the game already started or not.
Users : users who joined the room.
RoundsCount : returns number of game’s rounds
AllUsersJoined : returns true if all of users are joined the room.
AddUser : adds an user to the room.
StartRound : starts the first playable round, if not available `GameCompleted` event will be invoked.
CurrentRound : returns current playing round.
RemoveUser : removes an user from the room.
KickUser : kick an user from the room.
GameComplete : returns true if the game has been completed.
IsPlaying : returns true if any round is active and playing.

GameCompleted() : invokes when game is completed
NewUserJoined(User user) : invokes when a new user joins
RoomReadyForStart(List<User> users) : invokes when room is ready to start
RoundStarted(ushort roundId) : invokes when new round starts
UserExited(User user) : invokes when an user exits from the room
UserKicked(User user) : invokes when an user gets kicked from the room

User

AddScore : adds score to the user in current playing round., type is double
GetScore : gets score of the user in `currentRound` or specified round using the round index.
UpdateClient : updates `TcpClient` of an user.
Room : room which user is in.

Round

Index : returns index of the round.
RoundComplete : completes the round.

RoomManager Functions

```
    var result = RoomManager.Join(user);
```
Join a random room.

```
    var result = RoomManager.Join(id, user);
```
Join to a room with id `id` which have not password

```
    var result = RoomManager.Join(level, user);
```
Join to random room with level range `level`

```
    var result = RoomManager.Join(id, password, user);
```
Join to a room with id `id` and password `password’

```
    var room = FindRoom(user)
```
Find room which user is in

```
    var room = FindRoom(id)
```
Find room with room `UniqueId` id

UserManager Functions

```
    var user = FindUser(id)
```
Find user with `UniqueId`.

```
    var user = FindUser(client)
```
Find a user with its client.