WSSG Add new functionality step by step guide

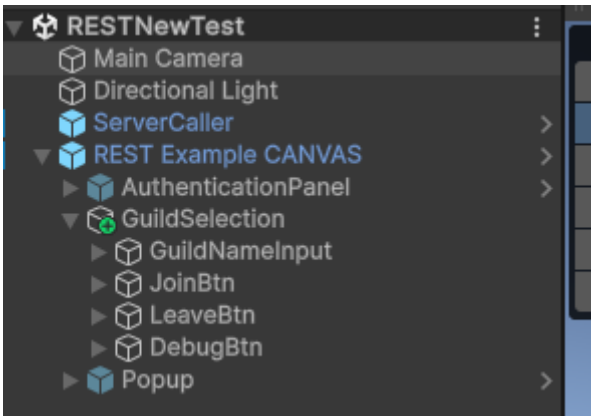
In this example I provide a "step by step" guide how to add functionality to the template.
in this case adding joining, leaving and a debug functionality to the Guild.
Depending on your usecase, it has different UI, names and functions!

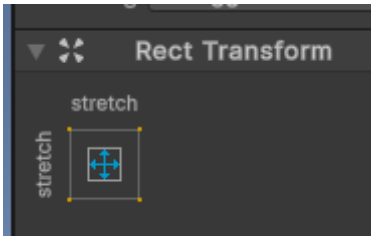
REST

Unity stuff

In `Assets\Scenes\Examples` copy the Template scene and rename it.
Note: The Templatescene could have no Eventsystem! Add it!

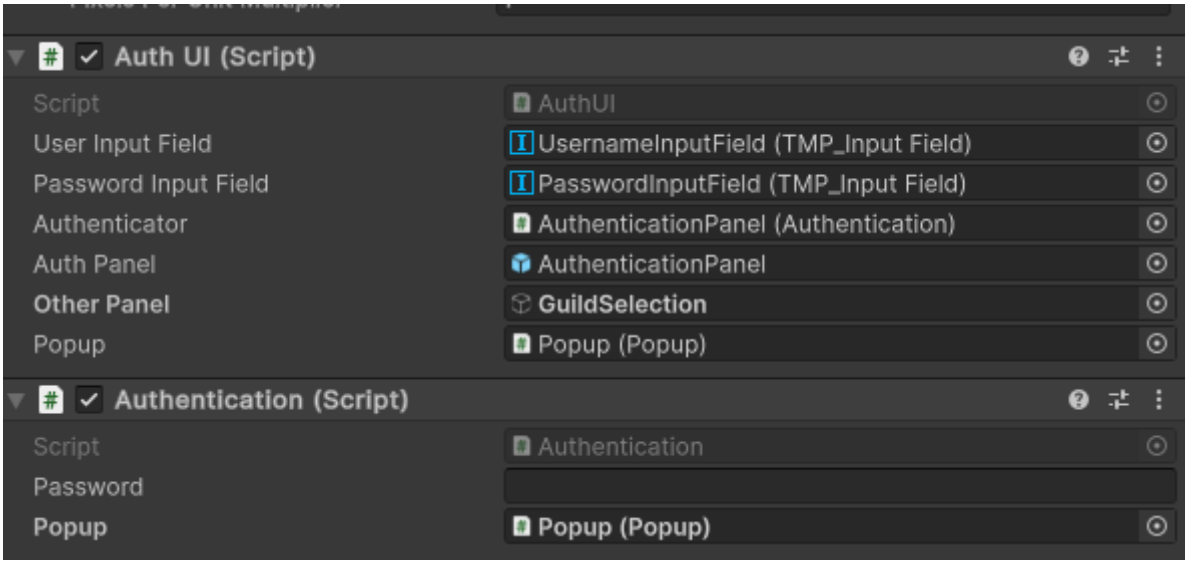
Add the UI components which you need.
In this case I only want to have the most simplest join, leave guild and a debug button for other things.



- GuildSelection is a Panel Object which is scaled for the whole canvas
  

- GuildNameInput: InputField - TextMeshpro
- JoinBtn: Button (TextMeshPro) for joining
- LeaveBtn: Button (TextMeshPro) for leaving
- DebugBtn: Button (TextMeshPro) for debugging
  UI Settings:
  Inputfield: 500 x 80 and the text set to autosize

Btns: 300 x 50 and text set to autosize



do not forget to set the inspector variables in AuthenticationPanel

Add New Script to the **GuildSelection** Gameobject. I named it GuildRest.cs

For this Script we want 3 methods, which are responsible of sending a joining, leaving and debugging call to the server.
For visualization it also has a reference to Popup.
To use the Inputfield we simply have a refence to it.

```
[SerializeField]
private Popup _popup;
[SerializeField]
private TMP_InputField _inputField;
public async void JoinGuildServerCall()
{
}
public async void LeaveGuildServerCall()
{
}
public async void DebugGuildServerCall()
{
}
```

And set the Popup/InputField in the inspector.

Implementation

At first we need to add the urls for the guild in **django/python**

Create a new python file in `WSSGTemplate\rest_scripts\` named `views_guild_functions.py`
The boilerplate code for `views_guild_functions.py` is:

```
from django.http import HttpResponse

import utilities
from ServerClass import ServerClass
from WSSGTemplate.models import *

def join_guild(request):
    print("join guild")

def leave_guild(request):
    print("leave guild")

def debug_guild(request):
    print("debug guild")
```

Go to `WSSGTemplate\rest_scripts\urls_rest.py`
Add the `views_guild_functions` to the imports
and add the specific urls to method mapping.
in this example they are

```
# urls for Guild
path("join_guild/", views_guild_functions.join_guild),
path("leave_guild/", views_guild_functions.leave_guild),
path("debug_guild/", views_guild_functions.debug_guild),
```

To each of these urls we map a function to the url.
DO NOT FORGET THE / on the url or that the methods do NOT have ()!!

now switching back to unity side.
In `GuildRest.cs` we now add the code to connect to the server and the data which should be send.
For Join and Leave we only send over which Guild we want to leave/join and our player name.

- To get the Guild name, we simply read out the text inside the TMP_InputField.
  so add `[SerializeField] private TMP_InputField _inputField;` to the attributes of this file.
- To get the username we simply use `TemplateSettings.username`

Then using the general structure which is provided in Quickreference, we input the things we need to send or change.

And for DebugGuild we only need 1 line which is:
`await ServerCaller.Instance.GenericRequestAsync("rest/debug_guild/", _popup.DefaultCallback);`

The whole file is this:

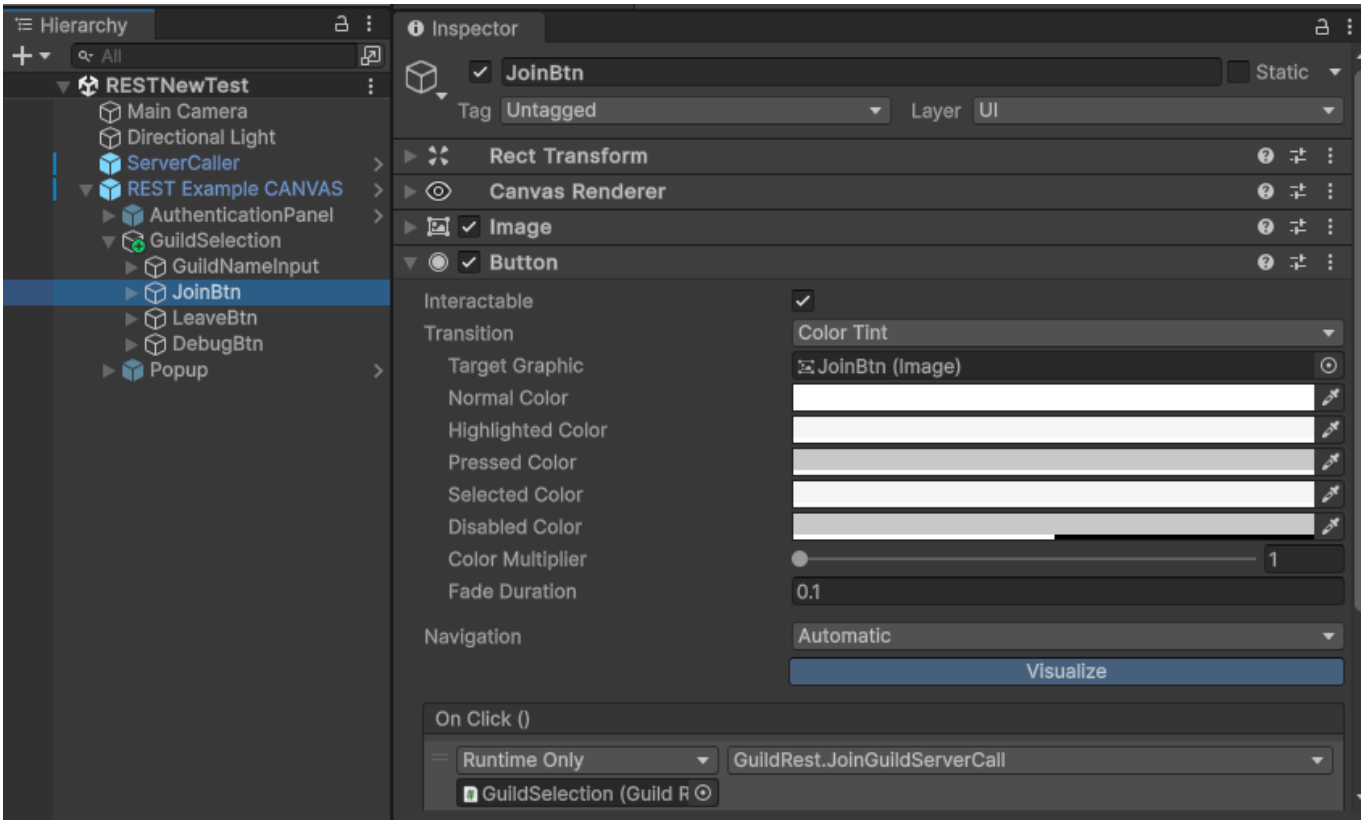
```
using System.Collections.Generic;
using ServerClasses;
using TMPro;
using UnityEngine;

public class GuildRest : MonoBehaviour
{
    [SerializeField]
    private Popup _popup;

    [SerializeField] private TMP_InputField _inputField;
    public async void JoinGuildServerCall()
    {var dataToSend = new Dictionary<string, object>
        {            { "id",TemplateSettings.username },
            { "guildname", _inputField.text },
          };
        await ServerCaller.Instance.GenericSendAsync("rest/join_guild/",
dataToSend, _popup.DefaultCallback);
    }    public async void LeaveGuildServerCall()
    {var dataToSend = new Dictionary<string, object>
        {            { "id",TemplateSettings.username },
            { "guildname", _inputField.text },

        };
        await ServerCaller.Instance.GenericSendAsync("rest/leave_guild/",
dataToSend, _popup.DefaultCallback);
    }    public async void DebugGuildServerCall()
    {await ServerCaller.Instance.GenericRequestAsync("rest/debug_guild/",
_popup.DefaultCallback);
    }
}
```

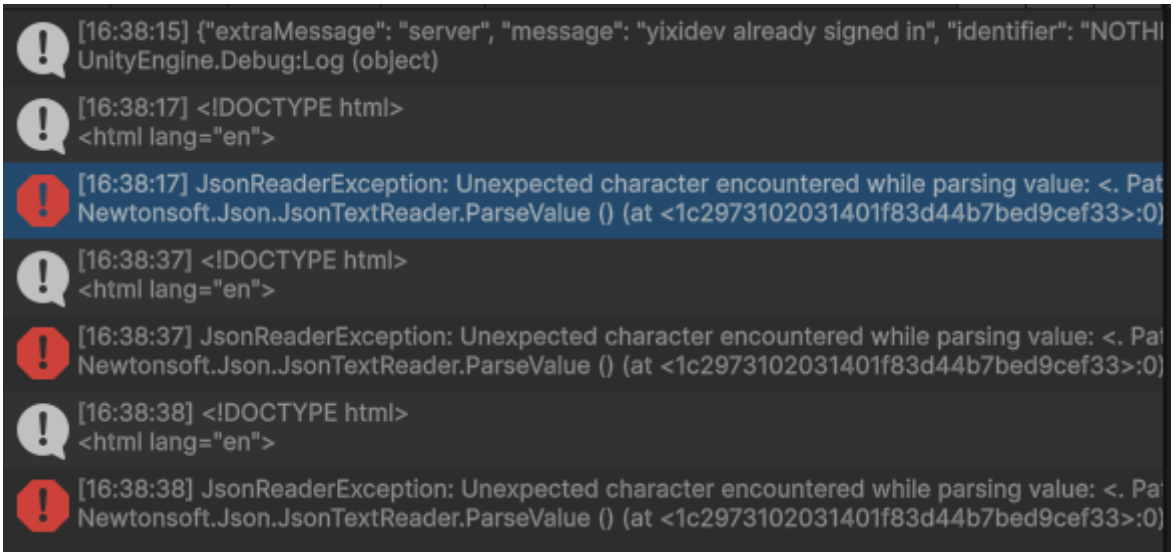
To test the script we need to add the calls to the respective button.



Ensure that the AuthenticationPanel is visible and the GuildSelectionPanel is NOT visible!

After login in and clicking all 3 buttons we get errors on unity side. but we also get the following output in the console, where our server is running

unity:



django:

```
join guild
leave guild
leave guild
```

This issue lies in the structure of "messaging system". Nonetheless we already got a connection to the server. Now lets fix this issue and write the code for the guild on the serverside!

Adding the line `return utilities.server_message_response(message="YOURMESSAGE", identifier="MESSAGE", status=200)` fixes the errors in unity side!

> **messaging structure**
>
> To help with the structure we only communicate in json. so we send json back and forth from client to server and back. Having some required paramters in it, also helps with other stuff.
> For example error messages like "No guild exists" or data, which is once again json.

for the guild we will be using the Guild model, which is already in `WSSGTemplate\models.py`

```
class Guild(models.Model):
    genericID = models.AutoField(primary_key=True)
    guildPlayers = models.ManyToManyField(Player, related_name='guilds')
    guildOwner = models.OneToOneField(Player, on_delete=models.CASCADE,
related_name='owned_guild_backref')
    guildName = models.CharField(max_length=1000, default="guild")
```

To join a guild we only need to add the player to the guildPlayers field.

Firstly we gonna insert following code

```
result_from_check = utilities.basic_request_check(request, ["id",
"guildname"])
if isinstance(result_from_check, HttpResponse):
    return result_from_check
player_id = result_from_check["id"]
guild_name = result_from_check["guildname"]
```

it basically just extracts the REQUIRED data which we sends from unity to django.
Note that you could also add the `ignore_missing_values=True` parameter, but then more

careful!

If one of these required parameters (e.g id or guildname) is missing it should return an error message to unity side!
If missing or wrong written, you will get following message in the unity console:
`{"extraMessage": "server", "message": "no guildname is given", "identifier": "ERROR"}`

Now we need to extract the correct guild and add the player to the guild.guildPlayers.

first to extract the guild OBJECT we can use a provided function
`utilities.search_object_by_attribute()`
to get the object according the the attributes (`guildName` in this case)

```
guild_obj=utilities.search_object_by_attribute(ServerClass.Guild,guildName=gui
ld_name)
print(guild_obj)
if guild_obj is None:
    return utilities.server_message_response(message=f"no guild with name
{guild_name} was found!")
```

just as a safety measure, if there is no guild or the guildname is wrong. a simple None check.
`utilities.search_object_by_attribute()` returns None, if there is nothing.

Now we do the same with the Player, so search after the name of ServerClass.Player

```
player_obj=utilities.search_object_by_attribute(ServerClass.Player,name=player
_id)
if player_obj is None:
    return utilities.server_message_response(message=f"no player with name
{guild_name} was found!")
```

We need the player object since we cannot add the player just by the string player_id.
(We could in theory, but that would potentially make the code more unreadable for others!)

Finally we have all the data and simply call the add function and save it, since otherwise it would not be in the database. And we return a "everything is ok" response

```
guild_obj.guildPlayers.add(player_obj)
guild_obj.save()
```

complete code:

```
def join_guild(request):
    print("join guild")
    result_from_check = utilities.basic_request_check(request, ["id",
"guildname"])
    if isinstance(result_from_check, HttpResponse):
        return result_from_check
    player_id = result_from_check["id"]
    guild_name = result_from_check["guildname"]

    #extract guild

guild_obj=utilities.search_object_by_attribute(ServerClass.Guild,guildName=gui
ld_name)
    print(guild_obj)
    if guild_obj is None:
        return utilities.server_message_response(message=f"no guild with name
{guild_name} was found!")

    # get player object !!

player_obj=utilities.search_object_by_attribute(ServerClass.Player,name=player
_id)
    if player_obj is None:
        return utilities.server_message_response(message=f"no player with name
{guild_name} was found!")

    #add player

    guild_obj.guildPlayers.add(player_obj)
    guild_obj.save()

    return utilities.server_message_response(message="join guild",
identifier="MESSAGE", status=200)****
```

Now for the leave_guild function we do the same thing, we only remove it from the list

complete code:

```
def leave_guild(request):
    print("leave guild")
    result_from_check = utilities.basic_request_check(request, ["id",
"guildname"])
    if isinstance(result_from_check, HttpResponse):
        return result_from_check
    player_id = result_from_check["id"]
```

```
    guild_name = result_from_check["guildname"]

    # extract guild
    guild_obj = utilities.search_object_by_attribute(ServerClass.Guild,
guildName=guild_name)
    print(guild_obj)
    if guild_obj is None:
        return utilities.server_message_response(message=f"no guild with name
{guild_name} was found!")

    # get player object !!
    player_obj = utilities.search_object_by_attribute(ServerClass.Player,
name=player_id)
    if player_obj is None:
        return utilities.server_message_response(message=f"no player with name
{guild_name} was found!")

    # add player

    guild_obj.guildPlayers.remove(player_obj)
    guild_obj.save()

    return utilities.server_message_response(message="leave guild",
identifier="MESSAGE", status=200)
```

For the Debug method we just gonna return every guild and the contents of it. Just for fun

the output from this would be with 1 guild:

```
{
  "extraMessage": "server",
  "message": "[{\"guildOwner\": 1, \"guildName\": \"guild\", \"guildPlayers\":
[1]}]",
  "identifier": "MESSAGE"
}
```

Note that the players are displayer via their genericID (their primary key) if you want to have the names you could use `convert_ids_to_list_of_names` or `convert_server_class_id_to_name`

WS (Websocket)

So now we want to implement the same feature but with websockets. so we have a REAL TIME communication between the client and the server!

for the unity UI stuff refer to WSSG Add new functionality-DESKTOP-GOM3D4L > Unity stuff
we just gonna change the methods

First we need to add the rounting stuff.
For that go to `WSSGTemplate\websockets_scripts`
Create a new File e.g `consumer_guild_functions`
here's the code:

```
import utilities
from ServerClass import ServerClass
from WSSGTemplate.models import Guild
from WSSGTemplate.websocket_scripts.consumers_basic_layer import
BasicWSServerLayer

class WSGuildfunction(BasicWSServerLayer):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.commands = {
            "COMMAND": self.handle_command,
        }
    def handle_command(self):
        self.send_broadcast_message(message="Command",identifier="MESSAGE",)
```

under `routing.py` add `re_path(r'ws/GuildWS/(?P<lobby_name>\w+)/$', consumer_guild_function.WSGuildfunction.as_asgi()),` to the urlpatterns.
do not forget to import the class!

Notes:

- the lobby_name in the repath is a hardcoded value! which will be used in `consumers_basic_layer.py`
- We only refernce to the CLASS and not single methods.
- We can now send broadcast_messages aka server communicates with EVERY CONNECTION AT ONCE
- If only a message to the current connected client is needed use `send_message_to_client`

since we only reference the class and not single methods like in WSSG Add new functionality-DESKTOP-GOM3D4L > REST. We gonna do it more manually by adding this mapping in the self.commands dictionary.

code:

```
def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)

    self.commands = {
        "join": self.handle_join,
        "leave": self.handle_leave,
        "debug": self.handle_debug,
    }
    async def handle_join(self, data):
            await
self.send_broadcast_message(message="Command",identifier="MESSAGE")

        async def handle_leave(self, data):
            await
self.send_broadcast_message(message="Command",identifier="MESSAGE")

        async def handle_debug(self, data):
            await
self.send_broadcast_message(message="Command",identifier="MESSAGE")
```

Now switching to unity to make the connection with WS work!

First copy the attributes from `GuildRest.cs` to `GuildWS.cs`
Change the inheritance from `Monobehaviour` to `CommunicationWS`
in the CommunicationWS.cs file there are some variables and shortcut functions.
like `url` . this is the url to connect to. So we gonna set it. e.g `url = "GuildWS/placeholder" ;`
The placeholder in this case correlates to the `lobby_name` of `re_path(r'ws/GuildWS/(?P<lobby_name>\w+)/$', consumer_guild_function.WSGuildfunction.as_asgi())` in `routing.py`

We just gonna pretend that we have one public Lobby to do stuff.
If you have a lobby or in game with other players. USE THE LOBBY NAME or another ID!

With the line `ConnectToUrl(GenerateURLfromPath(url));` we can connect to the WS on the server.

After starting it we see following output:
unity: **Websocket Connected!** and **Server message(WebSocketSharp.WebSocket): {"extraMessage": "broadcast", "message": "Command", "identifier": "MESSAGE"}**
python: **debug {"test":"hello world"}**

With the connection done, we just gonna implement the methods.

first on unity side (again) 3 methods for join, leave and debug
It is pretty similar with the only changes of the sending call
from `await ServerCaller.Instance.GenericSendAsync("rest/join_guild/", dataToSend, _popup.DefaultCallback);` to `SendMessageAsync(GenerateServerMessage(dataToSend, "debug"));`

code:

```
public void JoinGuild()
{
    var dataToSend = new Dictionary<string, object>
    {        { "id",TemplateSettings.username },
        { "guildname", _inputField.text },
      };
    SendMessageAsync(GenerateServerMessage(dataToSend, "join"));

}

public void LeaveGuild()
{
    var dataToSend = new Dictionary<string, object>
    {        { "id",TemplateSettings.username },
        { "guildname", _inputField.text },

    };
    SendMessageAsync(GenerateServerMessage(dataToSend, "leave"));

}

public void DebugGuild()
{
    var dataToSend = new Dictionary<string, object>
    {        { "test", "hello world" }
    };    SendMessageAsync(GenerateServerMessage(dataToSend, "debug"));
}
```

Note: `SendMessageAsync(GenerateServerMessage(dataToSend,"debug"))` needs this dictionary. Since I forgot to make an "request version" just input nothing in the dictionary or dummy data ^^'

Do not forget to set the methods in the buttons via the inspector!

Now we change change to the python side to implement the logic!

in Join we need the guild and player object. since it is a async method we use some methods needs an `await` infornt of them.

for join we just gonna grab the data directly

```
player_id = data["id"]
guild_name = data["guildname"]
```

get the Guild object

```
guild_obj = await  utilities.asearch_object_by_attribute(ServerClass.Guild,
guildName=guild_name)
if guild_obj is None:
    await self.send_broadcast_message(message=f"no guild with name
{guild_name} was found!", identifier="ERROR")
    return
```

Note: tha asearch_object_by_attribute. is just a sync to async wrapper! aka make the sync function to a async one

get the player object

```
#get player object
player_obj = await  utilities.asearch_object_by_attribute(ServerClass.Player,
name=player_id)
if player_obj is None:
    await self.send_broadcast_message(message=f"no player with name
{player_id} was found!", identifier="ERROR")
    return
```

add player to list:

```
# add player
guild_obj.guildPlayers.add(player_obj)
guild_obj.save()
```

complete code for method:

```
async def handle_join(self, data):

    player_id = data["id"]
    guild_name = data["guildname"]

    #get guild object
    guild_obj = await
utilities.asearch_object_by_attribute(ServerClass.Guild, guildName=guild_name)
    if guild_obj is None:
        await self.send_broadcast_message(message=f"no guild with name
{guild_name} was found!", identifier="ERROR")
        return

    #get player object
    player_obj = await
utilities.asearch_object_by_attribute(ServerClass.Player, name=player_id)
    if player_obj is None:
        await self.send_broadcast_message(message=f"no player with name
{player_id} was found!", identifier="ERROR")
        return

    #add player
    guild_obj.guildPlayers.add(player_obj)
    guild_obj.save()

    await self.send_broadcast_message(message="Added player to guild",
identifier="MESSAGE")
```

like the rest part we just gonna copy and replace the add with a remove

as an example one could return the list of guild for the debug function this way:

```
model_instances = await sync_to_async(list)(Guild.objects.all())
res = utilities.get_json_from_instances(model_instances)

await self.send_broadcast_message(message=f"{res}", identifier="MESSAGE")
```

Complete codes

Rest

urls_rest

```
from django.urls import path

from WSSGTemplate.rest_scripts import views_basicviews_guild_functions

urlpatterns = [

    path("signin/", views_basic.signin),
    path("signout/", views_basic.signout),
    path("signup/", views_basic.signup),
        ...
    # urls for Guild
    path("join_guild/", views_guild_functions.join_guild),
    path("leave_guild/", views_guild_functions.leave_guild),
    path("debug_guild/", views_guild_functions.debug_guild),
]
```

views_guild_function.py

```
from django.http import HttpResponse

import utilities
from ServerClass import ServerClass
from WSSGTemplate.models import *
from WSSGTemplate.rest_scripts import views


def join_guild(request):
    print("join guild")
    result_from_check = utilities.basic_request_check(request, ["id",
"guildname"])
    if isinstance(result_from_check, HttpResponse):
        return result_from_check
    player_id = result_from_check["id"]
    guild_name = result_from_check["guildname"]

    #extract guild

guild_obj=utilities.search_object_by_attribute(ServerClass.Guild,guildName=gui
ld_name)
    print(guild_obj)
    if guild_obj is None:
        return utilities.server_message_response(message=f"no guild with name
{guild_name} was found!")
```

```
    # get player object !!

player_obj=utilities.search_object_by_attribute(ServerClass.Player,name=player
_id)
    if player_obj is None:
        return utilities.server_message_response(message=f"no player with name
{guild_name} was found!")

    #add player

    guild_obj.guildPlayers.add(player_obj)
    guild_obj.save()

    return utilities.server_message_response(message="join guild",
identifier="MESSAGE", status=200)


def leave_guild(request):
    print("leave guild")
    result_from_check = utilities.basic_request_check(request, ["id",
"guildname"])
    if isinstance(result_from_check, HttpResponse):
        return result_from_check
    player_id = result_from_check["id"]
    guild_name = result_from_check["guildname"]

    # extract guild
    guild_obj = utilities.search_object_by_attribute(ServerClass.Guild,
guildName=guild_name)
    print(guild_obj)
    if guild_obj is None:
        return utilities.server_message_response(message=f"no guild with name
{guild_name} was found!")

    # get player object !!
    player_obj = utilities.search_object_by_attribute(ServerClass.Player,
name=player_id)
    if player_obj is None:
        return utilities.server_message_response(message=f"no player with name
{player_id} was found!")

    # add player

    guild_obj.guildPlayers.remove(player_obj)
    guild_obj.save()
```

```
    return utilities.server_message_response(message="leave guild",
identifier="MESSAGE", status=200)


def debug_guild(request):
    print("debug")
    # get all from view script
    model_instances = views.generic_get_all(server_class=ServerClass.Guild)
    #convert to json
    res = utilities.get_json_from_instances(model_instances)
    return utilities.server_message_response(status=200, identifier="MESSAGE",
message=f"{res}")
```

GuildRest.cs

```
using System.Collections.Generic;
using ServerClasses;
using TMPro;
using UnityEngine;

public class GuildRest : MonoBehaviour
{
    [SerializeField]
    private Popup _popup;

    [SerializeField] private TMP_InputField _inputField;
    public async void JoinGuildServerCall()
    {var dataToSend = new Dictionary<string, object>
        {            { "id",TemplateSettings.username },
            { "guildname", _inputField.text },
          };
        await ServerCaller.Instance.GenericSendAsync("rest/join_guild/",
dataToSend, _popup.DefaultCallback);
    }    public async void LeaveGuildServerCall()
    {var dataToSend = new Dictionary<string, object>
        {            { "id",TemplateSettings.username },
            { "guildname", _inputField.text },

        };
        await ServerCaller.Instance.GenericSendAsync("rest/leave_guild/",
dataToSend, _popup.DefaultCallback);
    }    public async void DebugGuildServerCall()
```

```
    {await ServerCaller.Instance.GenericRequestAsync("rest/debug_guild/",
_popup.DefaultCallback);
    }
}
```

WS (Websockets)

routing.py

```
from django.urls import re_path

from WSSGTemplate.websocket_scripts import consumer_guild_function

# the websocket urls.

websocket_urlpatterns = [

    re_path(r'ws/GuildWS/(?P<lobby_name>\w+)/$',
consumer_guild_function.WSGuildfunction.as_asgi()),
]
```

consumer_guild_function.py

```
from asgiref.sync import sync_to_async

import utilities
from ServerClass import ServerClass
from WSSGTemplate.models import *
from WSSGTemplate.websocket_scripts.consumers_basic_layer import
BasicWSServerLayer


class WSGuildfunction(BasicWSServerLayer):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.commands = {
            "join": self.handle_join,
            "leave": self.handle_leave,
```

```
            "debug": self.handle_debug,
        }

    async def handle_join(self, data):

        player_id = data["id"]
        guild_name = data["guildname"]

        #get guild object
        guild_obj = await
utilities.asearch_object_by_attribute(ServerClass.Guild, guildName=guild_name)
        if guild_obj is None:
            await self.send_broadcast_message(message=f"no guild with name
{guild_name} was found!", identifier="ERROR")
            return

        #get player object
        player_obj = await
utilities.asearch_object_by_attribute(ServerClass.Player, name=player_id)
        if player_obj is None:
            await self.send_broadcast_message(message=f"no player with name
{player_id} was found!", identifier="ERROR")
            return

        #add player
        guild_obj.guildPlayers.add(player_obj)
        guild_obj.save()

        await self.send_broadcast_message(message="Added player to guild",
identifier="MESSAGE")

    async def handle_leave(self, data):
        player_id = data["id"]
        guild_name = data["guildname"]

        # get guild object
        guild_obj = await
utilities.asearch_object_by_attribute(ServerClass.Guild, guildName=guild_name)
        if guild_obj is None:
            await self.send_broadcast_message(message=f"no guild with name
```

```
{guild_name} was found!", identifier="ERROR")
            return

        # get player object
        player_obj = await
utilities.asearch_object_by_attribute(ServerClass.Player, name=player_id)
        if player_obj is None:
            await self.send_broadcast_message(message=f"no player with name
{player_id} was found!", identifier="ERROR")
            return

        # add player
        guild_obj.guildPlayers.remove(player_obj)
        guild_obj.save()

        await self.send_broadcast_message(message="removed player to guild",
identifier="MESSAGE")



    async def handle_debug(self, data):
        print(data)
        model_instances = await sync_to_async(list)(Guild.objects.all())
        res = utilities.get_json_from_instances(model_instances)

        await self.send_broadcast_message(message=f"{res}",
identifier="MESSAGE")
```

GuildWS.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using WS;

public class GuildWS : CommunicationWS
{
    [SerializeField]
    private Popup _popup;
```

```
    [SerializeField] private TMP_InputField _inputField;

    private string placeholderForURL = "placeholder";
    // Start is called before the first frame update
    void Start()
    {        url = "GuildWS/placeholder" ;
        ConnectToUrl(GenerateURLfromPath(url));
        var dataToSend = new Dictionary<string, object>
        {            { "test", "hello world" }
        };        SendMessageAsync(GenerateServerMessage(dataToSend,
"debug"));

    }
    public void JoinGuild()
    {        var dataToSend = new Dictionary<string, object>
        {            { "id",TemplateSettings.username },
            { "guildname", _inputField.text },
          };
        SendMessageAsync(GenerateServerMessage(dataToSend, "join"));

    }
    public void LeaveGuild()
    {        var dataToSend = new Dictionary<string, object>
        {            { "id",TemplateSettings.username },
            { "guildname", _inputField.text },

        };
        SendMessageAsync(GenerateServerMessage(dataToSend, "leave"));

    }
    public void DebugGuild()
    {        var dataToSend = new Dictionary<string, object>
        {            { "test", "hello world" }
        };        SendMessageAsync(GenerateServerMessage(dataToSend,
"debug"));
    }
}
```