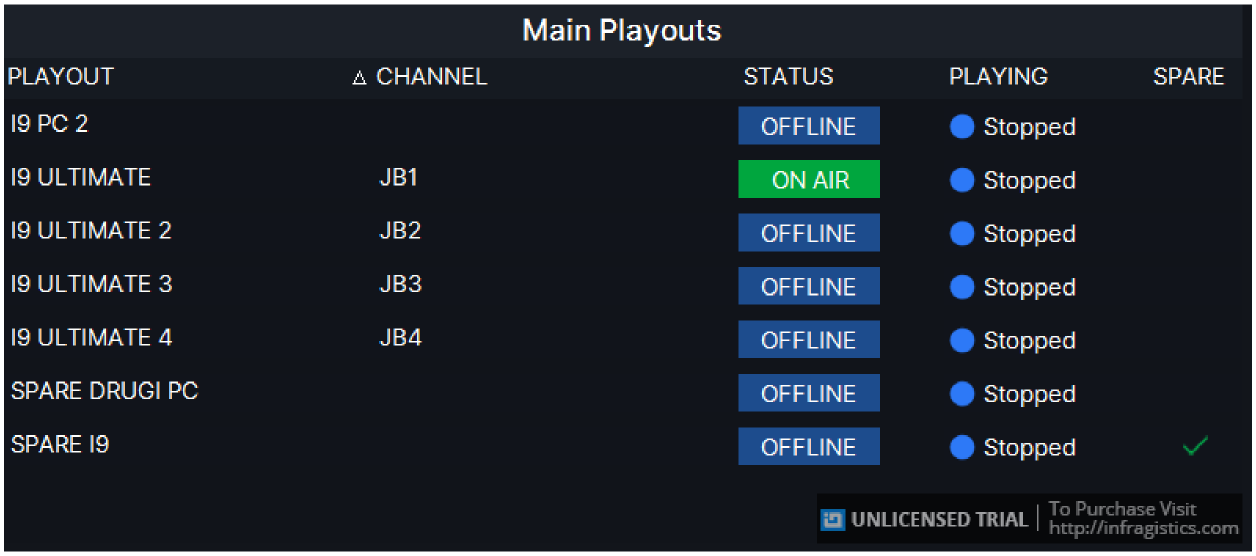
API CALLS

LIST OF ALL PLAYOUTS

`Playouts/short`



Main Playouts

PLAYOUT	△ CHANNEL	STATUS	PLAYING	SPARE
I9 PC 2		OFFLINE	● Stopped	
I9 ULTIMATE	JB1	ON AIR	● Stopped	
I9 ULTIMATE 2	JB2	OFFLINE	● Stopped	
I9 ULTIMATE 3	JB3	OFFLINE	● Stopped	
I9 ULTIMATE 4	JB4	OFFLINE	● Stopped	
SPARE DRUGI PC		OFFLINE	● Stopped	
SPARE I9		OFFLINE	● Stopped	✓

UNLICENSED TRIAL | To Purchase Visit http://infragistics.com

Returns PlayoutListInfo. Now I used Binding List for this, not sure if you can use it for DevExpress grid, but anyway here are both options

```
public class PlayoutListInfo
```

Or if you only want to use class, without notifications

```
public class PlayoutListInfo2
```

For Status column, possible values are

```
"0", "OFFLINE"
"1", "ON AIR"
"2", "UNKNOWN"
```

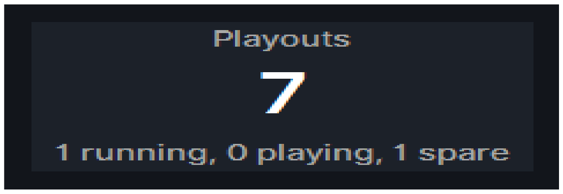
Color for “2” should be some dark yellow, I used 122, 95, 32

For Playing column, possible values are

```
"0", "Stopped"
"1", "Playing"
```

You can use as it is in Figma, Green = playing, Blue = stopped. Picture I sent up is desktop version, you do as it is in Figma

Once playouts are loaded, we send request via SignalR to all apps to send their status.



Playouts text is fixed. Number in the middle get from this same api call. For text below, I used this method

```
internal string ReturnPlayoutStatusNo()
{
    int spare = Playouts.Where(p => p.Spare).Count();

    return Playouts.Where(p => !p.Spare && p.IsOnline == 1).Count().ToString() + "
running, " + Playouts.Where(p => !p.Spare && p.IsPlaying == 1).Count().ToString() +
" playing" + (spare > 0 ? ", " + spare.ToString() + " spare" : string.Empty);
}
```

`Playouts is List<PlayoutListInfo>`

`SignalR`

`Endpoint is /broadcastHub`

Id for signalR is random GUID

Group for signalR is “WebDashboard”

```
await _connection.InvokeAsync("RegisterClient", clientId, group);
```

Connection uses JWT from login.

And from desktop apps, I do Ping on server every 5 seconds

We have 2 events

```
_signalR.CommandReceived += OnCommandReceived;
_signalR.ConnectionStateChanged += OnConnectionStateChanged;
```

`OnConnectionStateChanged`

```
switch (obj)
{
    case "Connected":
        foreach (var playout in playoutControl1.Playouts)
        {
            playout.IsOnline = 0;
        }
        break;
    case "Closed":
        foreach (var playout in playoutControl1.Playouts)
        {
            playout.IsOnline = 2; //set Unknows status when there is no connection
        }
        break;
}
```

We use CommandDto to send via SignalR.

So after playouts are loaded, we send this command to signalR to ask all apps about their status. Probably I will upgrade this to send only to Playouts (I can change this later)

```
public async Task SendRequest(string ClientId)
{
    if (_connection == null)
        return;

    if (_connection.State == HubConnectionState.Connected)
    {
        try
        {
            await _connection.InvokeAsync("SendToAllExcept", ClientId, new 
CommandDto
            {
                command = ServiceMessages.RequestStatusSync.ToSignal(),
                clientId = ClientId
            });
        }
        catch (Exception ex)
        {
            // može i logger
            Console.WriteLine("Ping error: " + ex.Message);
        }
    }
}
```

OnCommandReceived

And in OnCommandReceive, we get status of apps like this

```
private async void OnCommandReceived(object data)
 {
     if (InvokeRequired)
     {
         try
         {
             Invoke(new System.Action(() => OnCommandReceived(data)));
         }
         catch { }
         return;
     }

     CommandDto? comm = data.ToObject<CommandDto>();

     if (comm != null)
     {
         switch (comm.command.FromSignal<ServiceMessages>())
         {
             case ServiceMessages.ClientStatusChanged:
                 if (comm.group != null && comm.group.Length > 0)
                 {
                     switch (comm.group.FromSignal<ClientType>())
                     {
                         case ClientType.PlayoutServer:
                             playoutControl1.PlayoutStatusChanged(comm);
                             GetPlayoutNo();
                             break;
                         case ClientType.RemoteControl:
                         case ClientType.Scheduler:
                         case ClientType.CgControl:
                             onlineControl1.SetStatus(comm);
                             break;
                     }
                 }
                 break;
```

PlayoutStatusChanged just update Status (isOnline value), and change status from Offline to On Air.

For Playing column, we will do that later, when playout is finished, so what you can do for test is to change that value also in PlayoutStatusChanged (when playout is online, it’s also playing, when it’s offline, then it’s stopped)

SignalR Interface, if you need it

```
Task RegisterClientAsync(string clientId, string clientType, string connectionId,
string userid, string username, string name);
 Task HandleDisconnectAsync(string connectionId);
 Task UpdatePingAsync(string clientId);

 Task<string> NotifyClientStatusAsync(string clientId, string clientType, string
userid, string username, string name, string status);
 Task NotifyEntityChangedAsync(object data);

 Task<string> SendToClientAsync(string clientId, object data);
 Task<string> SendToGroupAsync(string groupName, object data, string
exceptConnectionId, bool except);
 Task<string> SendToGroupsAsync(IEnumerable<string> groupNames, object data);
 Task SendToAllAsync(object data);
 Task SendToAllExceptAsync(string id, object data);
 Task<IReadOnlyCollection<ClientSession>> GetClientsSnapshot();

 // Maps and tracking
 ConcurrentDictionary<string, string> ClientConnections { get; }
 ConcurrentDictionary<string, ClientSession> AllClients { get; }
 ConcurrentDictionary<string, DateTime> LastPing { get; }
 Dictionary<string, HashSet<string>> GroupsMap { get; }
```