SignalR runs via RPC(remote procedure calls) which means that it consists of a number of simple commands on both client and server, but that it has the ability to invoke other functions on the client/server. This makes it very flexible, but requires the client and the server to have agreed upon a common set of commands.

In the client commands shown here, the string after the invoke command reflects the C# method name on the server and any alteration clientside must also be altered on the server.

Connection

Calls to the SignalR section of the server must access the /chat route and carry the users bearer token as shown in the example below:

```
const chatLocalUrl = "http://localhost:49873/chat";
this.state.connection = new signalr.HubConnection(chatLocalUrl+"?
token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZXN0dXNlciI
sImVtYWlsIjoidGVzdHVzZXJAaG90bWFpbC5jb20iLCJqdGkiOiI0NDdiM2ZmNi0xZ
GJmLTQ2YjktODNmZC05MjVlYjI4MGE4MTUiLCJyb2xlcyI6InVzZXIiLCJleHAiOjE
1MjQ0MjkzMTEsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6NjM5MzkvIiwiYXVkIjoia
HR0cDovL2xvY2FsaG9zdDo2MzkzOS8ifQ.ep2n5ZPNR6dTRdDETalHuLpPbYu56Boj
UgRn_-EeccQ");
```

Carrying the bearer token in the url is far from ideal, but at present is the only workable solution. In the future it may be possible to exchange the token as a SignalR message.

Connections are given an id and currently stored in an in-memory dictionary with username as key on the server.

Chats and Joining groups

In the finished product, every conversation should take the form of a group conversation (even if there are only two participants) and must have a name. Other methods providing user to user direct messaging and user to all direct messaging are implemented for test purposes. Groups are stored in-memory for quick access, but also backed up to database, so a user logging in after a logout should still be a participant in the same groups.

All groups must have a name. The server provides a group name to avoid security problems with accidentally matching group names and this is relayed to the client.

Method invocations for joining groups

Below are several methods used to create and add users to groups. **These are under construction and are subject to change**.

JoinGroup(string groupName)

To create a group, the calling function must specify a group name which does not presently exist. If the group does exist, the user will join this group.

```
        this.state.connection.invoke("joinGroup", groupName);
```

This adds the user to the group on the server, commits the changes to database and then responds by calling invoking a client method according to the following code:

```
Clients.Group(groupName).InvokeAsync("userAddedToGroup", new[] {
name, groupName });
```

The clientside code to handle this is defined as follows:

```
this.state.connection.on('userAddedToGroup', (name, group) => {
          //Code here
        });
```

AddClientToGroup(string groupName, string usernameToAdd)
Designed to add a single user to existing group.

StartGroupChatWithMultipleClients(string[] usernames)
Designed to create a group chat from scratch with multiple users.

LeaveGroup(string groupName)
Removes the calling user from a group.

Sending Messages

A message is sent as shown below.

```
       this.state.connection.invoke("sendMessageToGroup",
groupName, message);
```

The server will then send the message to the group by invoking a client method.

```
Clients.Group(groupName).InvokeAsync("messageSentToGroup", new[] {
groupName, name, message });
```

which in turn is defined on the client as:

```
   this.state.connection.on('messageSentToGroup',
(group,senderName,message) => {
            //Code here
        });
```

Sending Messages - Convenience Methods

For testing convenience the following methods can also be called on the server:

SendMessageToUser

```
        this.state.connection.invoke("sendMessageToUser",
username, message);
```

calls client according to:

```
Clients.Client(connectionId).InvokeAsync("messageSentToSpecificUse
r", new[] { senderName, message });
```

SendMessageToAllConnectUsers

```
this.state.connection.invoke("sendMessageToAllConnectedUsers",
this.state.message);
```

calls client according to:

```
    Clients.All.InvokeAsync("messageSentToAllConnectedUsers",
new[] { senderName, message });
```

Receiving Messages

The following format for client methods called by server is currently defined by serverside code.

```
        //Various info messages groupwise
        this.state.connection.on('addInfoMessageFromGroup',
(group, message) => {
          //Code here
        });

        //Called when a user connects or disconnects, and so can
be used to
        //update a users online status.
        this.state.connection.on('alterFriendStatus', (name,
group, status) => {
           //Code here
        });

        //Useful for initial testing
        this.state.connection.on('messageSentToSpecificUser',
(name, message) => {
           //Code here
        });

        //Useful for initial testing
        this.state.connection.on('messageSentToAllConnectedUsers',
(name, message) => {
          //Code here
        });

        this.state.connection.on('messageSentToGroup',
(group,senderName,message) => {
        //Code here
        });

        this.state.connection.on('userAddedToGroup', (name, group)
=> {
           //Code here
        });

        this.state.connection.on('userLeftGroup', (name, group) =>
{
          //Code here
        });
```