Creating a Remove Player Button

The next step is to create a method for the “Remove player” button.

To begin, delete the `remove` function from the remove event and replace it with a `Meteor.call` statement. Pass through a value of `removePlayer`, as the first argument. That is the name of the method that we’ll set up in a moment. Then, as the second argument, pass through the `selectedPlayer` variable.

This will allow us to access the ID of the selected player from inside the method. Scroll down to the `Meteor.methods` block and create the `removePlayer` method. Just like with helpers and events, we can create multiple methods ina single block of code by separating them with commas.

```
Meteor.methods = {
    removePlayer: function(selectedPlayer) {
        PlayersList.remove({_id: selectedPlayer});
    }
}
```

Remove Function

Because of this code, the button will work as expected, but there’s still a couple of things for us to consider. First, we have to check to see if the `selectedPlayer` variable is a string,w hich can be once again done using the `check` function. This will prevent the wrong type of data from being passed into the method.

Second, we need to prevent logged out users from being able to execute this method which can be done by creating a `currentUserId` variable and by using the same conditional we covered in the previous video. But there is another problem: if a user is logged into the application, they’re able to call the `removePlayer` method and pass through the ID of any player’s document, even if that document doesn’t belong to them. This means a user could delete players from another user’s leaderboard. To solve this problem, change the query inside the remove function, so that it will only remove a document from the collection if that document belongs to the currently logged in user:

```
Meteor.methods = {
    removePlayer: function(selectedPlayer) {
        PlayersList.remove({_id: selectedPlayer, createdBy: currentUserId});
    }
}
```

This means there are multiple layers of security that ensure users can only interact with the database exactly as we want them to.