Building the Game

Joel Neubeck
http://joel.Neubeck.net
joel@neubeck.net

pluralsight
hardcore developer training

SignalR into MVC 4

- **Prerequisites**
  - Visual Studio 2010 SP1 or 2012
  - Microsoft ASP.NET and Web Tools 2012.2.

- **<u>Alternative</u>**
  - “Install-Package Microsoft.AspNet.SignalR”

The Hub

```
using Microsoft.AspNet.SignalR;
namespace Demo2
{
    public class GameHub : Hub
    {
        public void Hello()
        {
            Clients.All.hello(DateTime.Now.ToString("T"));
        }
    }
}
```

Broadcasting of Messages

- **Everyone**
  - Clients.All.hello();
- **Caller**
  - Clients.Caller.hello();
- **Group of Clients**
  - Clients.Group(**groupName)**.hello();
- **Specific Connections**
  - Clients.Client(**connectionId**).hello();

Client Side Code

```
<script src="/Scripts/jquery-1.8.2.min.js" ></script>
<script src="/Scripts/jquery.signalR-1.0.0.js"></script>
<script src="/signalr/hubs"></script>
<script type="text/javascript">
    $(function() {
        var hub = $.connection.gameHub;
        hub.client.hello = function(message) {
            $('#results').append('<b>' + message + '</b>');
        };

        $.connection.hub.start().done(function () {
            hub.server.hello();
        });
    });
</script>
```

GameState - Singleton

```
public class GameState
{
   // Singleton instance
  private readonly static Lazy<GameState> _instance =
         new Lazy<GameState>(() => new GameState());

  private GameState()
  {
  }

  public static GameState Instance
  {
      get { return _instance.Value; }
  }
}
```

GameState - Singleton

```
var player =  GameState.Instance.GetPlayer(userName);
```

GameState - Singleton

```
public class GameState
{
    . . .

    private readonly ConcurrentDictionary<string, Player> _players =
       new ConcurrentDictionary<string,
            Player>(StringComparer.OrdinalIgnoreCase);

    private readonly ConcurrentDictionary<string, Game> _games =
       new ConcurrentDictionary<string,
           Game>(StringComparer.OrdinalIgnoreCase);
 . . .
}
```

GameState – Access to the Hub

- **GlobalHost**
  - This is a static signalR class that give you access to the HubContext through the IConnectionManager interface.

```
var myHub =
   GlobalHost.ConnectionManager.GetHubContext<GameHub>();
```

GameState - Constructor

```
private readonly static Lazy<GameState> _instance =
  new Lazy<GameState>(() =>
      new GameState(
         GlobalHost.ConnectionManager.GetHubContext<GameHub>()
      )
  );

private GameState(IHubContext context)
{
    Clients = context.Clients;
    Groups = context.Groups;
}

private IHubConnectionContext Clients { get; set;}
private IGroupManager Groups { get; set; }
```

GameState - Constructor

```
private readonly static Lazy<GameState> _instance =
       new Lazy<GameState>(() => new GameState());
```

Managing Players

- **CreatePlayer(string userName)**
  - This method will create a new player and add them to the players collection
- **GetPlayer(string userName)**
  - This method will find a player by name which already exists in the players collections
- **GetNewOpponent(Player player)**
  - This method will search for active players who are **NOT** playing in a match
- **GetOpponent(Player player, Game game)**
  - This method will get the opponent of a player for a given game

Managing Games - CreateGame

- **CreateGame(Player player1, Player player2)**
  - This method is used to place two players into a game and assign them a new game board.

```
var game = new Game
{
    Player1 = player1,
    Player2 = player2,
    Board = new Board()
};
```

Managing Games - CreateGame

```
var group = Guid.NewGuid().ToString("d");
_games[group] = game;

player1.IsPlaying = true;
player1.Group = group;
player2.IsPlaying = true;
player2.Group = group

//add them to a group
Groups.Add(player1.ConnectionId, group);
Groups.Add(player2.ConnectionId, group);
```

Managing Games - FindGame

- **FindGame(Player player, out Player opponent)**
  - This method is used to find a game that a person is playing.

Managing Games - FindGame

```
if (player.Group == null)
    return null;

Game game;
_games.TryGetValue(player.Group, out game);
if (game != null)
{
    if (player.Id == game.Player1.Id)
    {
        opponent = game.Player2;
        return game;
    }
    opponent = game.Player1;
    return game;
}
return null;
```

Managing Games – ResetGame

- **ResetGame(Game game)**
  - Method used to reset two players after completing a match.