Web Sockets

Estimated time for completion: 15 minutes

Overview:

In this lab you will use Web Sockets to build the classic sample which is a chat server.

Goals:

- Use WebSockets to send and receive data from the server

Lab Notes:

The lab uses a movie review website as its base. Registered users can view movies and reviews. Reviewers can write reviews. Administrators can do most anything. The accounts (username and passwords are the same value) for the application are:

- **Registered Users:** user1, user2 and user3
- **Reviewers:** reviewer1, reviewer2 and reviewer3
- **Administrator:** admin

Web Sockets

WebSockets allow for the server to asynchronously push data from the server to the client (as well as sending data from the client to the server). You will use this to build a two-way chat feature.

Criteria:

- Use WebSockets to send and receive data from the server

Steps:

1. In this part you'll be building a two-way chat feature into the movie review application.
   a. Open the solution from `~/WebSockets/before`.
   b. Run the movie review application. Notice on the home page there is a "chat" window. This is where the chat feature will be implemented.

      The server piece is already implemented using Fleck. If you're interested all the code is in `global.asax.cs`.

      You will simply implement the client code to connect, send and receive messages.
2. Now implement the client code.

a. Open `~/Views/Home/Index.cshtml`.

b. Examine the JavaScript in the file. You will implement `initWebSocket` and `sendChatToServer`.

c. Start with `sendChatToServer`. This assumes the `socket` variable has already been initialized. Use the `socket` variable to `send` the `chatName` and `message` values to the web socket.

   Since the web socket only accepts a string, you should put these two values into an object and use `JSON.stringify` to serialize the data. Pass the JSON string to the `send` function on the web socket.

```
function sendChatToServer(chatName, message) {
    var data = JSON.stringify({ Name: chatName, Message: message });
    socket.send(data);
}
```

3. Now implement `initWebSocket`. This needs to create an instance of the `WebSocket` object and assign it to the `socket` variable. When you create the web socket pass `"ws://localhost:5555 "` as the server to connect to.

   Next, register for the `onmessage` event. In the event handler, access the `data` property of the event arguments object. This will have the JSON data sent from the `sendChatToServer` you just implemented. You will need to use JSON.parse to convert from JSON into an object. Access the name and message properties from the object and pass them as parameters to the `updateChatFromServer` function.

```
function initWebSocket() {
    socket = new WebSocket('ws://localhost:5555');
    socket.onmessage = function (e) {
        var msg = JSON.parse(e.data);
        updateChatFromServer(msg.Name, msg.Message);
    };
}
```

4. Rebuild the solution and rerun the movie review web application home page. You should now be able to enter a chat message and see the entry show up in the chat window.

   Also try using two different browsers at the same time.

Solutions:

The final solution for this lab is available in the `~/after` directory.