WebSocket tutorial for C#

Dr. Balázs Simon (sbalazs@iit.bme.hu), BME IIT, 2023

1 Introduction

This document describes how to create a Hello World WebSocket service in Visual Studio using ASP.NET core.

2 Application

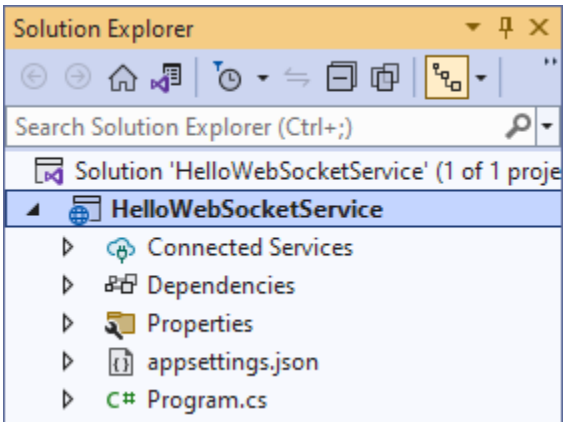
Open a command prompt from your project directory and issue the following commands to create a new web application (the ↛ symbol means that the command is continued in the next line, and the whole command should be typed as a single line):

```
c:\Users\[user]\source\projects>mkdir HelloWebSocketService

c:\Users\[user]\source\projects>dotnet new web -n HelloWebSocketService ↛
  -o HelloWebSocketService
```

(Make sure to use the simple command prompt and not PowerShell, since PowerShell handles the dash switches differently.)

Replace the **HelloWebSocketService.csproj** in the newly created **HelloWebSocketService** folder with the one attached to this tutorial (**HelloWebSocketService\HelloWebSocketService.csproj**).

Open the project in Visual Studio:



Create a new class called **HelloEndpoint** inside the project with the following content. This contains the application logic of the server:

```
using System.Net.WebSockets;

namespace HelloWebSocketService
{
    public class HelloEndpoint
    {
        public async Task Open(WebSocket socket)
        {
            Console.WriteLine("WebSocket opened.");
        }

        public async Task Close(WebSocket socket)
        {
            Console.WriteLine("WebSocket closed.");
        }

        public async Task Error(WebSocket socket, Exception ex)
        {
            Console.WriteLine("WebSocket error: " + ex.Message);
        }

        public async Task<string> Message(WebSocket socket, string message)
        {
            Console.WriteLine($"WebSocket message: {message}");
            return $"Hello: {message}";
        }
    }
}
```

Create a new class called **StringEncoder** inside the project with the following content. This helps converting the message format between strings and byte arrays:

```
using System.Net.WebSockets;
using System.Text;

namespace HelloWebSocketService
{
    public class StringEncoder
    {
        public static async Task<(WebSocketReceiveResult result, string? message)>
ReceiveAsync(WebSocket socket)
        {
            var buffer = new byte[1024 * 4];

            var result = await socket.ReceiveAsync(buffer: new
ArraySegment<byte>(buffer), cancellationToken: CancellationToken.None);

            if (result.MessageType == WebSocketMessageType.Text)
            {
                var text = Encoding.UTF8.GetString(buffer, 0, result.Count);
                return (result, text);
            }
            return (result, null);
        }
```

```
        public static async Task SendAsync(WebSocket socket, string message)
        {
            var buffer = new ArraySegment<byte>(Encoding.ASCII.GetBytes(message), 0,
message.Length);

            await socket.SendAsync(buffer: buffer,
                messageType: WebSocketMessageType.Text,
                endOfMessage: true,
                cancellationToken: CancellationToken.None);
        }

    }
}
```

Create a new class called **WebSocketMiddleware** inside the project with the following content. This forwards low-level web socket operations to our application logic:

```
using System.Net.WebSockets;

namespace HelloWebSocketService
{
    public class WebSocketMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly HelloEndpoint _server;

        public WebSocketMiddleware(RequestDelegate next, HelloEndpoint server)
        {
            _next = next;
            _server = server;
        }

        public async Task Invoke(HttpContext context)
        {
            if (!context.WebSockets.IsWebSocketRequest) return;

            var socket = await context.WebSockets.AcceptWebSocketAsync();
            await _server.Open(socket);

            try
            {
                while (socket.State == WebSocketState.Open)
                {
                    await HandleMessage(socket);
                }
            }
            catch (Exception ex)
            {
                await _server.Close(socket);
                await socket.CloseAsync(WebSocketCloseStatus.InternalServerError,
                                        ex.Message, CancellationToken.None);
                throw;
            }
        }

        private async Task HandleMessage(WebSocket socket)
```

```
        {
            var request = await StringEncoder.ReceiveAsync(socket);
            if (request.message is not null)
            {
                var response = await _server.Message(socket, request.message);
                if (response is not null)
                {
                    await StringEncoder.SendAsync(socket, response);
                }
            }
            else if (request.result.MessageType == WebSocketMessageType.Close)
            {
                await _server.Close(socket);
                await socket.CloseAsync(WebSocketCloseStatus.NormalClosure,
                                        null, CancellationToken.None);
            }
        }

    }
}
```

Update the main **Program.cs** with the following content. This allows html files to be published on the server, registers the web socket middleware and the service's logic:

```
using HelloWebSocketService;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddSingleton<HelloEndpoint>();

var app = builder.Build();
app.UseStaticFiles();
app.UseWebSockets();
app.UseMiddleware<WebSocketMiddleware>();

app.Run();
```

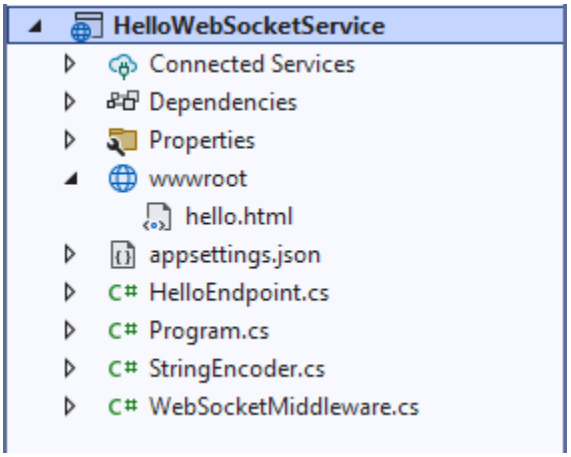
Create a new folder called **wwwroot** and add a new HTML file called **hello.html** under this folder with the following content. This is our web socket client:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Hello World WebSocket client</title>
    <script>
        var wsUrl = getRootUri() + "/hello";
        var websocket = null;

        function getRootUri() {
            return "ws://" +
    (document.location.hostname == "" ? "localhost" : document.location.hostname)
    + ":" +
    (document.location.port == "" ? "8080" : document.location.port);
```

```
}

function init() {
    output = document.getElementById("output");
}

function initWebSocket() {
    websocket = new WebSocket(wsUrl);
    websocket.onopen = function(evt) {
        onOpen(evt);
        doSend();
    };
    websocket.onmessage = function(evt) {
        onMessage(evt);
    };
    websocket.onerror = function(evt) {
        onError(evt);
        websocket = null;
    };
    websocket.onclose = function(evt) {
        onClose(evt);
        websocket = null;
    };
}

function send_message() {
    if (websocket == null) {
        initWebSocket();
    } else {
        doSend();
    }
}

function onOpen(evt) {
    writeToScreen("Connected to endpoint.");
}



function onMessage(evt) {
    writeToScreen("Message received: " + evt.data);
}

function onError(evt) {
    writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
}

function onClose(evt) {
    writeToScreen("Connection closed.");
}

function doSend() {
    message = textID.value;
    websocket.send(message);
    writeToScreen("Message Sent: " + message);
```

```
        }

        function writeToScreen(message) {
            var pre = document.createElement("p");
            pre.style.wordWrap = "break-word";
            pre.innerHTML = message;

            output.appendChild(pre);
        }

        window.addEventListener("load", init, false);

    </script>
</head>
<body>
    <h1 style="text-align: center;">Hello World WebSocket client</h1>

    <br/>

    <div style="text-align: center;">
        <form action="">
            <input onclick="send_message()" value="Send" type="button"/>
            <input id="textID" name="message" value="me" type="text"/><br/>
        </form>
    </div>
    <div id="output"></div>
</body>
</html>
```

The project should look like this:



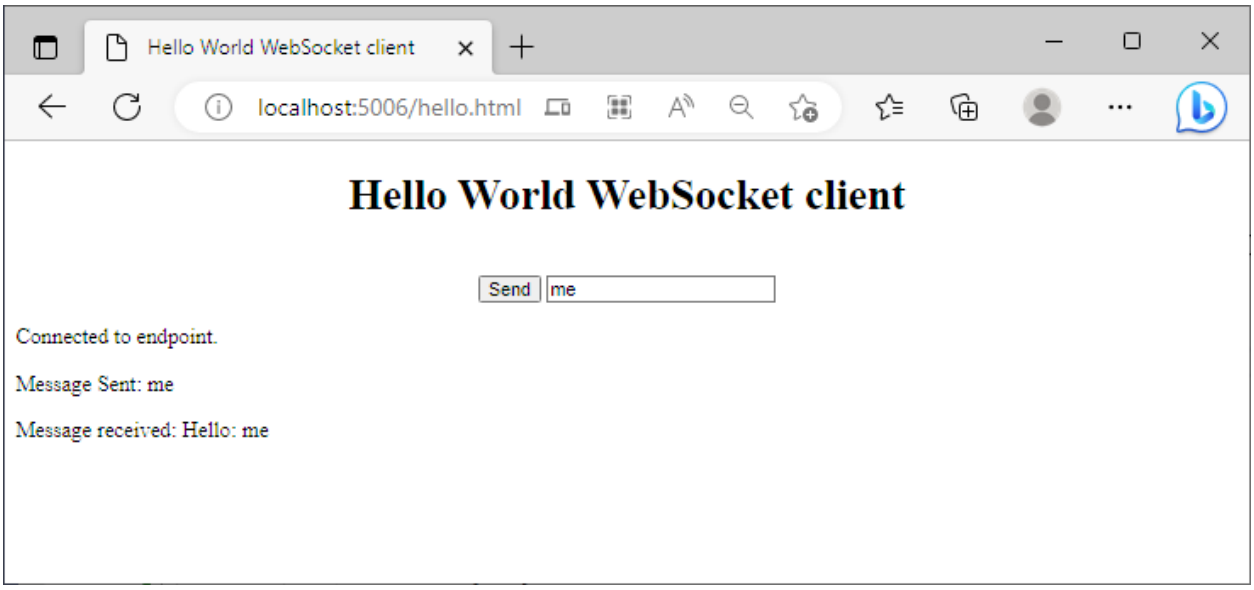
Run the application with **Ctrl+F5**.

To test the service, type the following URL into a browser (FireFox, Chrome, IE, etc.):

`http://localhost:[port]/hello.html`

Make sure to use HTTP and not HTTPS. The **[port]** number should be the one in the **Properties/ launchSettings.json** file.

Type something in the input field and click on the **Send** button:



Also, in the server log there should be: