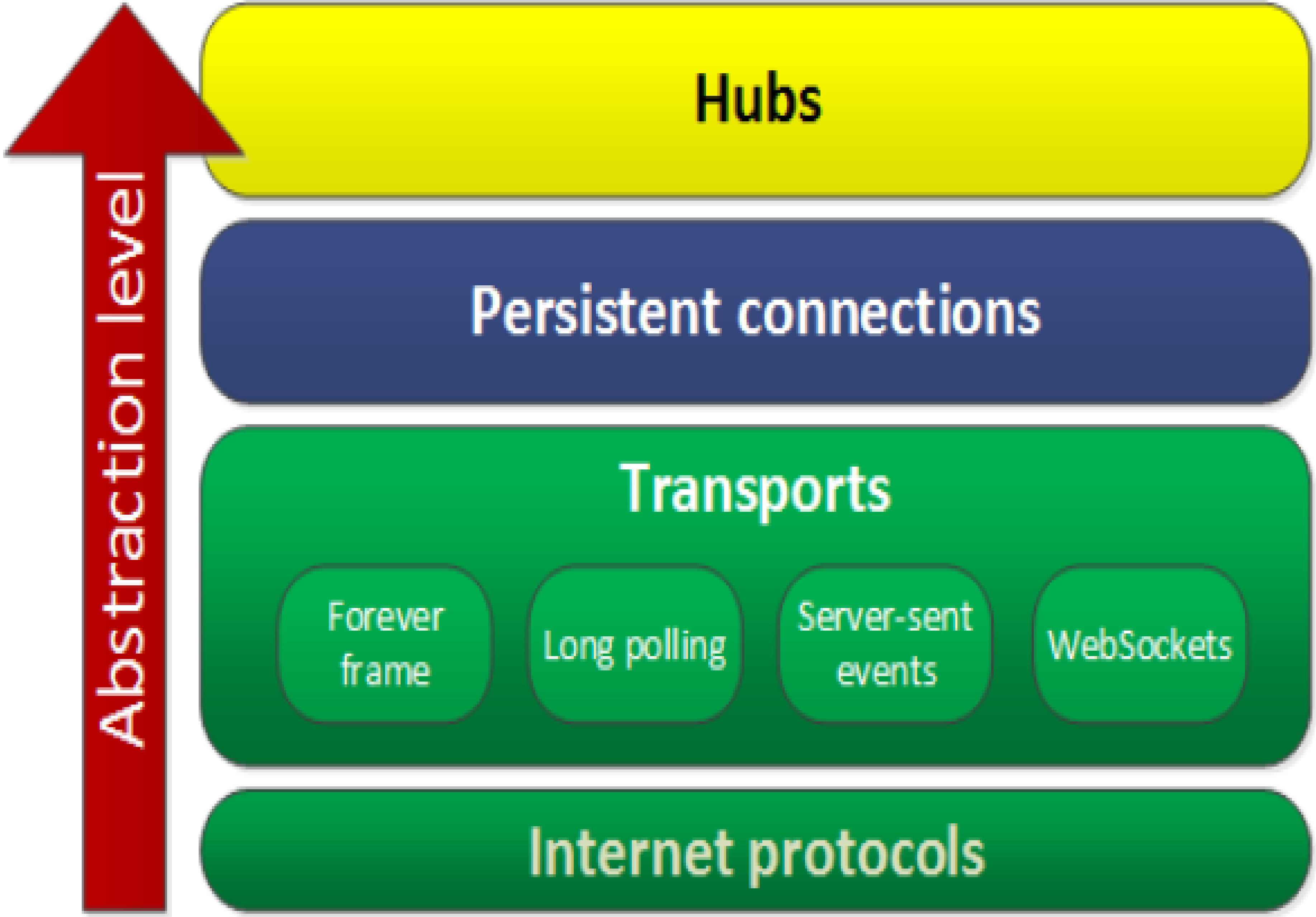
Hub

SignalR


Abstraction level
Hubs
Persistent connections
Transports
Forever frame
Long polling
Server-sent events
WebSockets
Internet protocols

SignalR Hubs

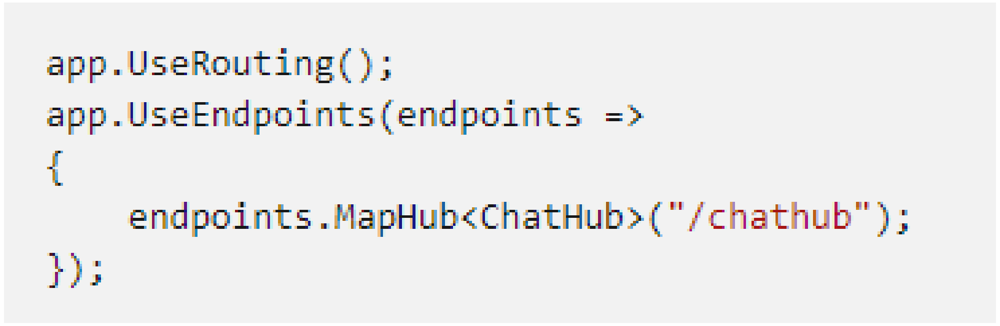
- The SignalR Hubs API enables you to call methods on connected clients from the server.
- In the server code, you define methods that are called by client.
- In the client code, you define methods that are called from the server.
- SignalR takes care of everything behind the scenes that makes real-time client-to-server and server-to-client communications possible.

Hubs

- Provide a higher level RPC framework over PersistentConnection
- ***remote procedure call*** : is a mechanism that enables methods on a system/computer/component to be called by an external or independent system/ computer/ component
- Hubs are per call, that is , each call from the client to hub will create a **new hub instance**
- recommended when we need to send different types of messages with various structures between the client and the server.

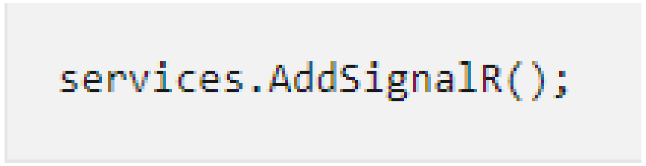
Configure SignalR hubs

- Add SignalR Middleware in Configure Method



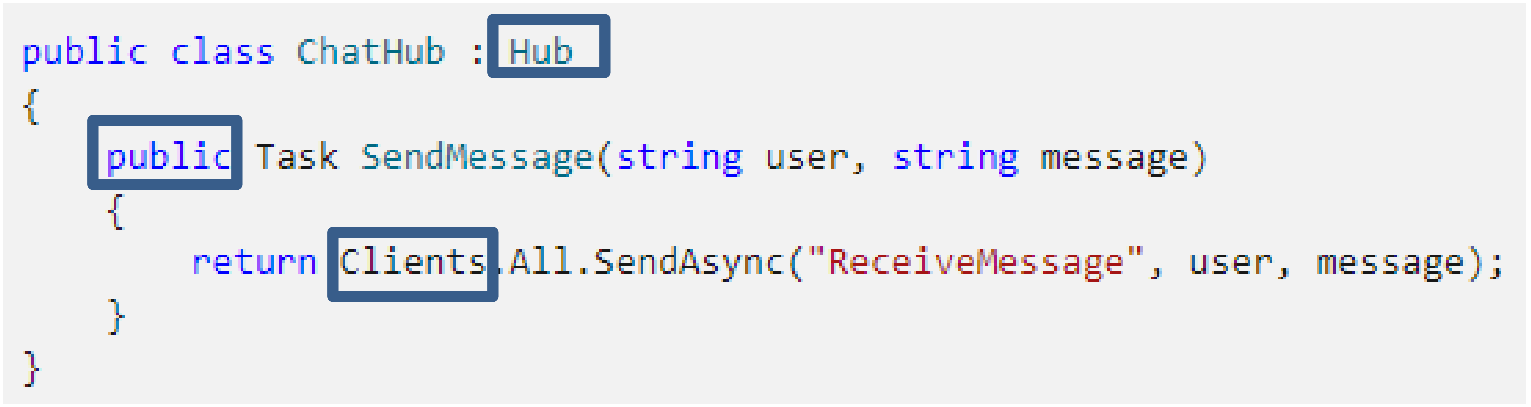
```
app.UseRouting();
app.UseEndpoints(endpoints =>
{
    endpoints.MapHub<ChatHub>("/chathub");
});
```

- The SignalR middleware requires some services, which are configured by calling services.AddSignalR.



```
services.AddSignalR();
```

Create and use hubs



```
public class ChatHub : Hub
{
    public Task SendMessage(string user, string message)
    {
        return Clients.All.SendAsync("ReceiveMessage", user, message);
    }
}
```

The Clients object

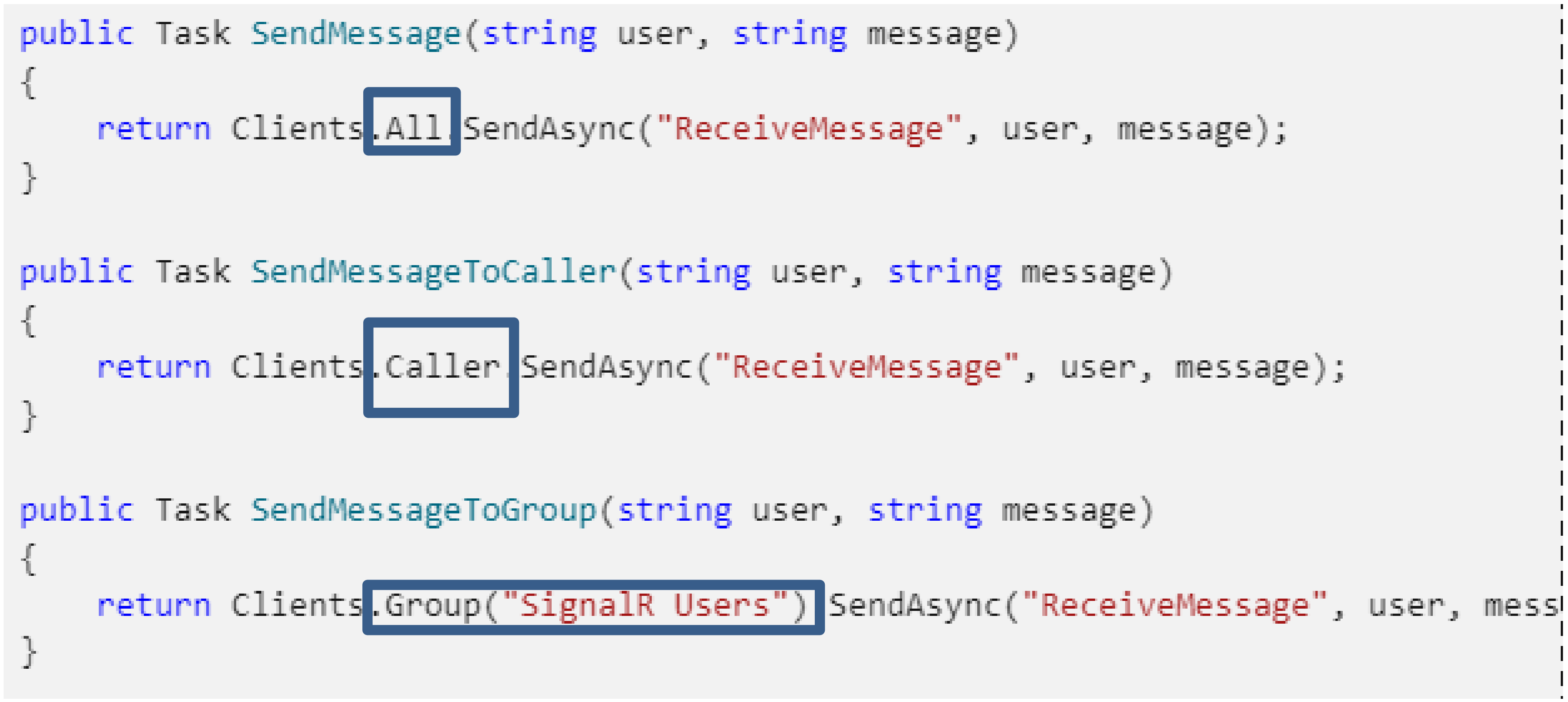
- The Hub class has a Clients property that contains the following properties for communication between server and client:

Property	Description
All	Calls a method on all connected clients
Caller	Calls a method on the client that invoked the hub method
Others	Calls a method on all connected clients except the client that invoked the method

Hub.Clients Methods

Method	Description
AllExcept	Calls a method on all connected clients except for the specified connections
Client	Calls a method on a specific connected client
Clients	Calls a method on specific connected clients
Group	Calls a method on all connections in the specified group
GroupExcept	Calls a method on all connections in the specified group, except the specified connections
Groups	Calls a method on multiple groups of connections
OthersInGroup	Calls a method on a group of connections, excluding the client that invoked the hub method
User	Calls a method on all connections associated with a specific user
Users	Calls a method on all connections associated with the specified users

Demo



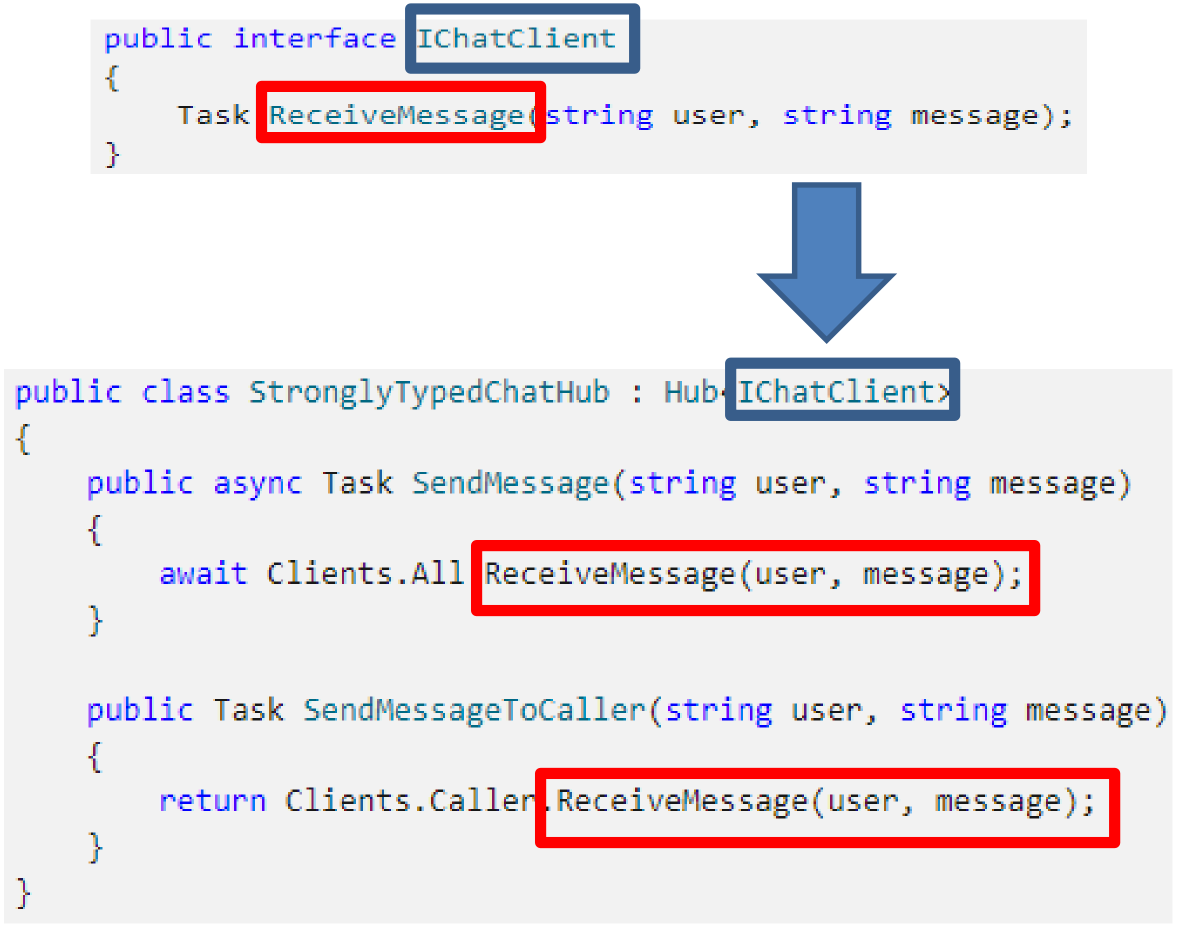
```
public Task SendMessage(string user, string message)
{
    return Clients.All.SendAsync("ReceiveMessage", user, message);
}

public Task SendMessageToCaller(string user, string message)
{
    return Clients.Caller.SendAsync("ReceiveMessage", user, message);
}

public Task SendMessageToGroup(string user, string message)
{
    return Clients.Group("SignalR Users").SendAsync("ReceiveMessage", user, mess
}
```

Strongly Typed Hubs

- A drawback of using **SendAsync** is that it relies on a magic string to specify the client method to be called.
- This leaves code open to **runtime errors** if the method name is misspelled or missing from the client.

Strong Typed Hubs

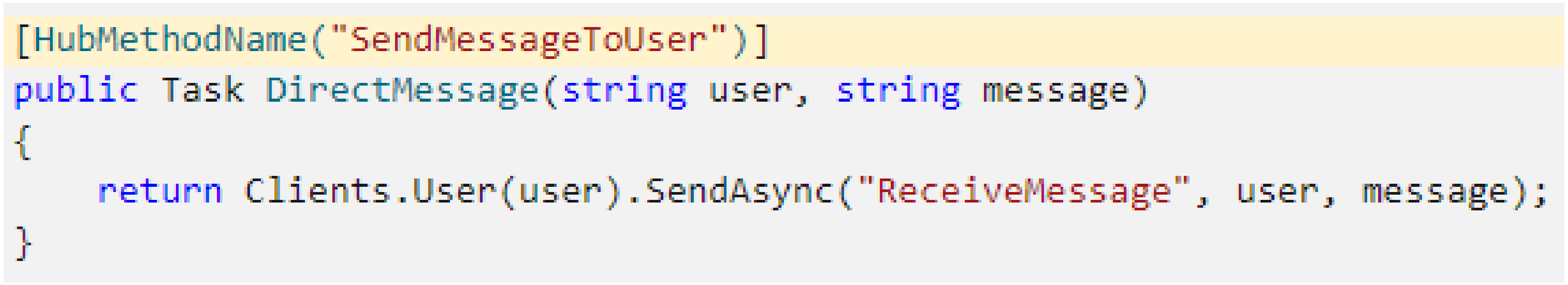
Using Hub<IChatClient> enables compile-time checking of the client methods



```
public interface IChatClient
{
    Task ReceiveMessage(string user, string message);
}
```

```
public class StronglyTypedChatHub : Hub<IChatClient>
{
    public async Task SendMessage(string user, string message)
    {
        await Clients.All ReceiveMessage(user, message);
    }

    public Task SendMessageToCaller(string user, string message)
    {
        return Clients.Caller.ReceiveMessage(user, message);
    }
}
```

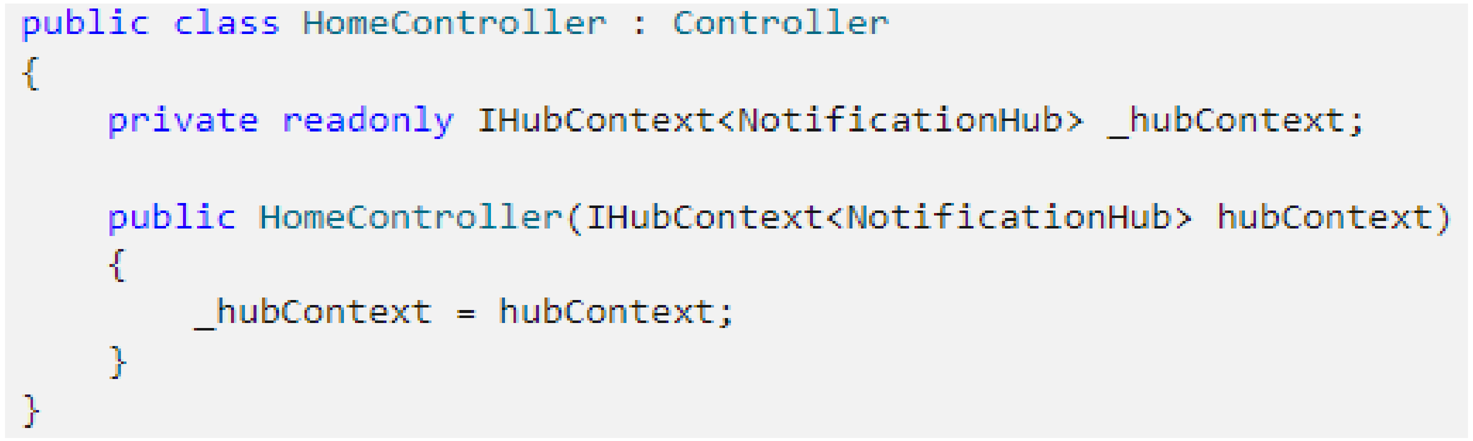
Change a hub method name



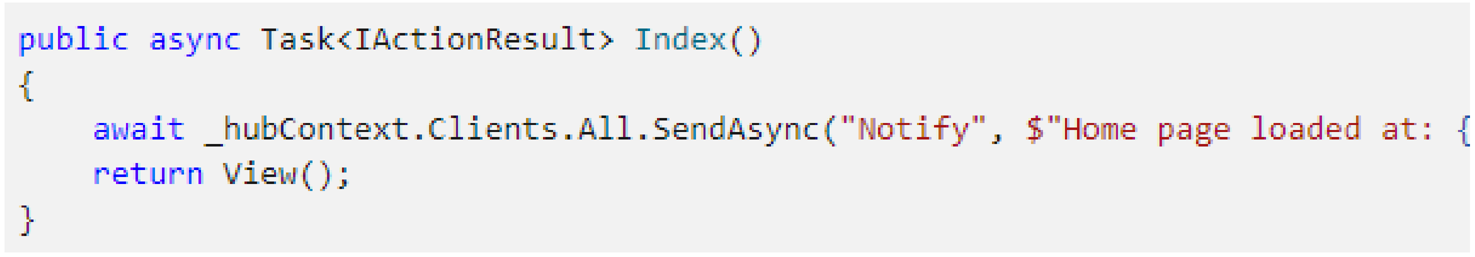
```
[HubMethodName("SendMessageToUser")]
public Task DirectMessage(string user, string message)
{
    return Clients.User(user).SendAsync("ReceiveMessage", user, message);
}
```

IHubContext Instance

- In ASP.NET Core SignalR, you can access an instance of IHubContext via dependency injection. You can inject an instance of IHubContext into a controller



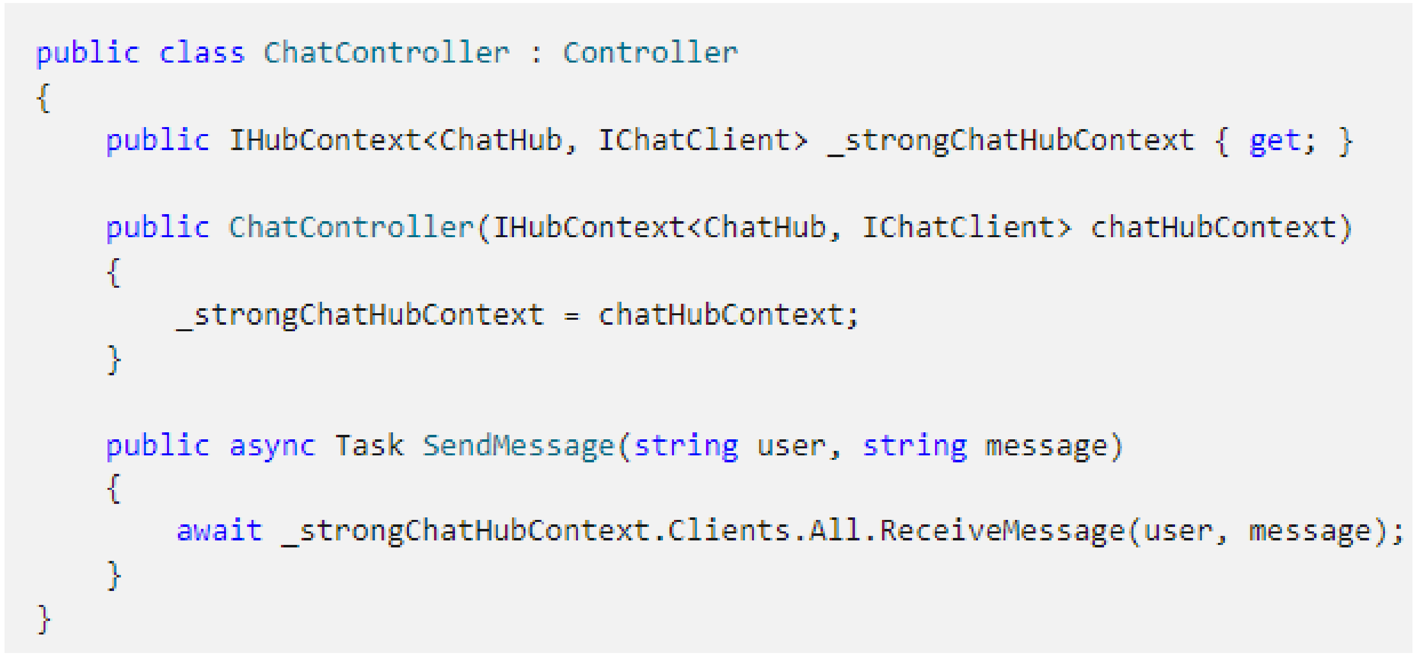
```
public class HomeController : Controller
{
    private readonly IHubContext<NotificationHub> _hubContext;

    public HomeController(IHubContext<NotificationHub> hubContext)
    {
        _hubContext = hubContext;
    }
}
```



```
public async Task<IActionResult> Index()
{
    await _hubContext.Clients.All.SendAsync("Notify", $"Home page loaded at: {
    return View();
}
```

Inject a strongly-typed HubContext

- To inject a strongly-typed HubContext, ensure your Hub inherits from Hub<T>.
- Inject it using the IHubContext<THub, T> interface rather than IHubContext<THub>



```
public class ChatController : Controller
{
    public IHubContext<ChatHub, IChatClient> _strongChatHubContext { get; }

    public ChatController(IHubContext<ChatHub, IChatClient> chatHubContext)
    {
        _strongChatHubContext = chatHubContext;
    }

    public async Task SendMessage(string user, string message)
    {
        await _strongChatHubContext.Clients.All.ReceiveMessage(user, message);
    }
}
```

ASP.NET Core SignalR JavaScript client

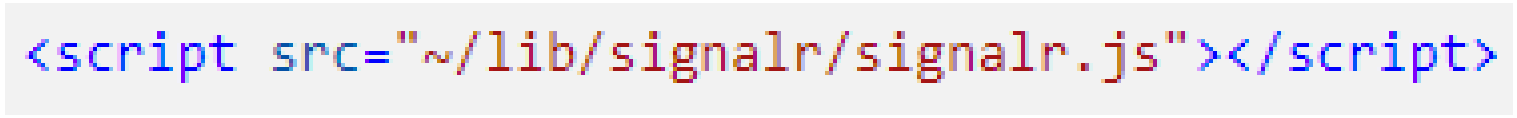
Install the SignalR client package

- npm installs the package contents in the *node_modules\@microsoft\signalr\dist\browser* folder.
- Create The *wwwroot/lib/signalr* folder.
- Copy the *signalr.js* file to the *wwwroot/lib/signalr* folder.

```
npm install @microsoft/signalr
```

Using Script file

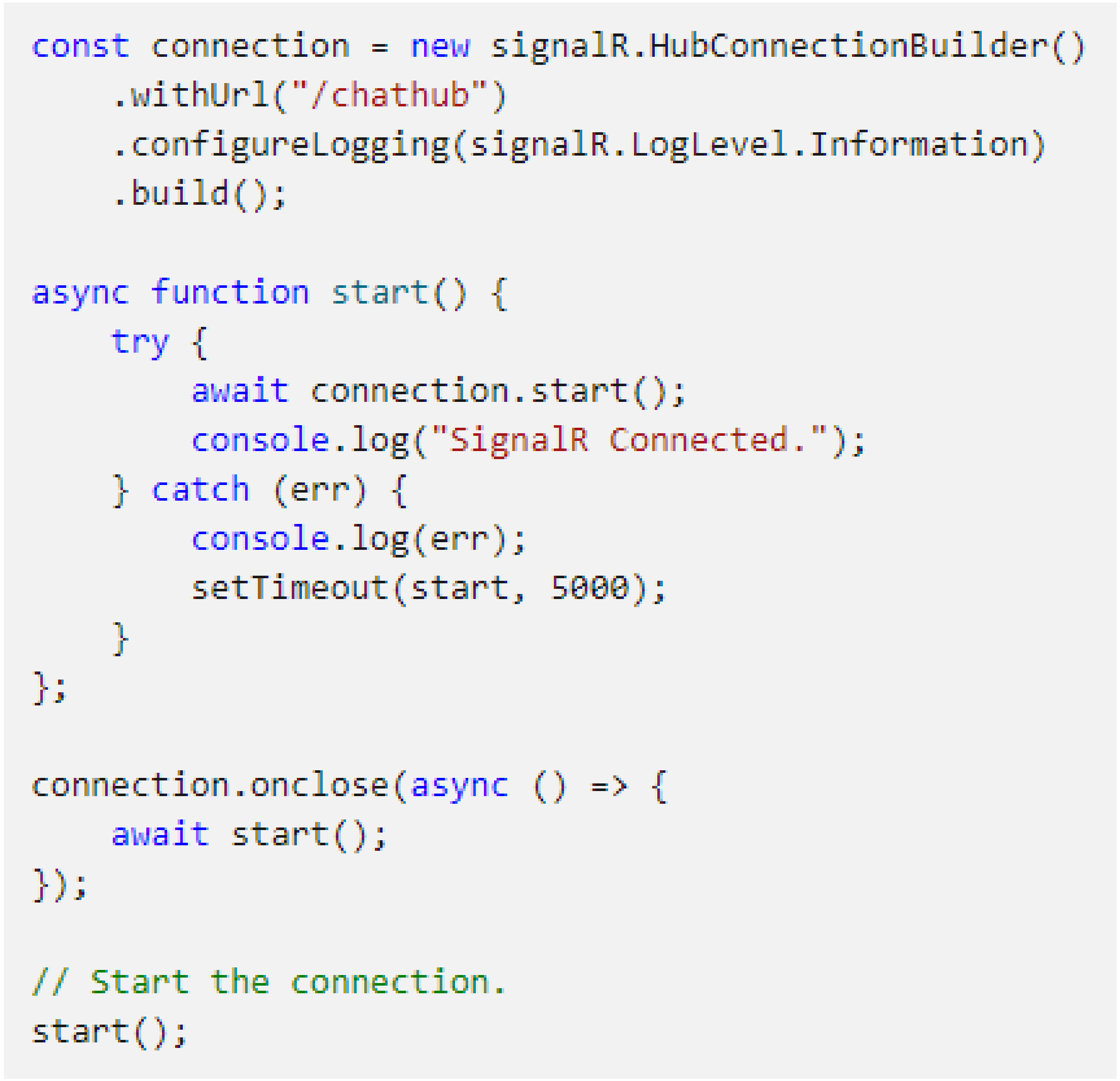
- Link to local script files



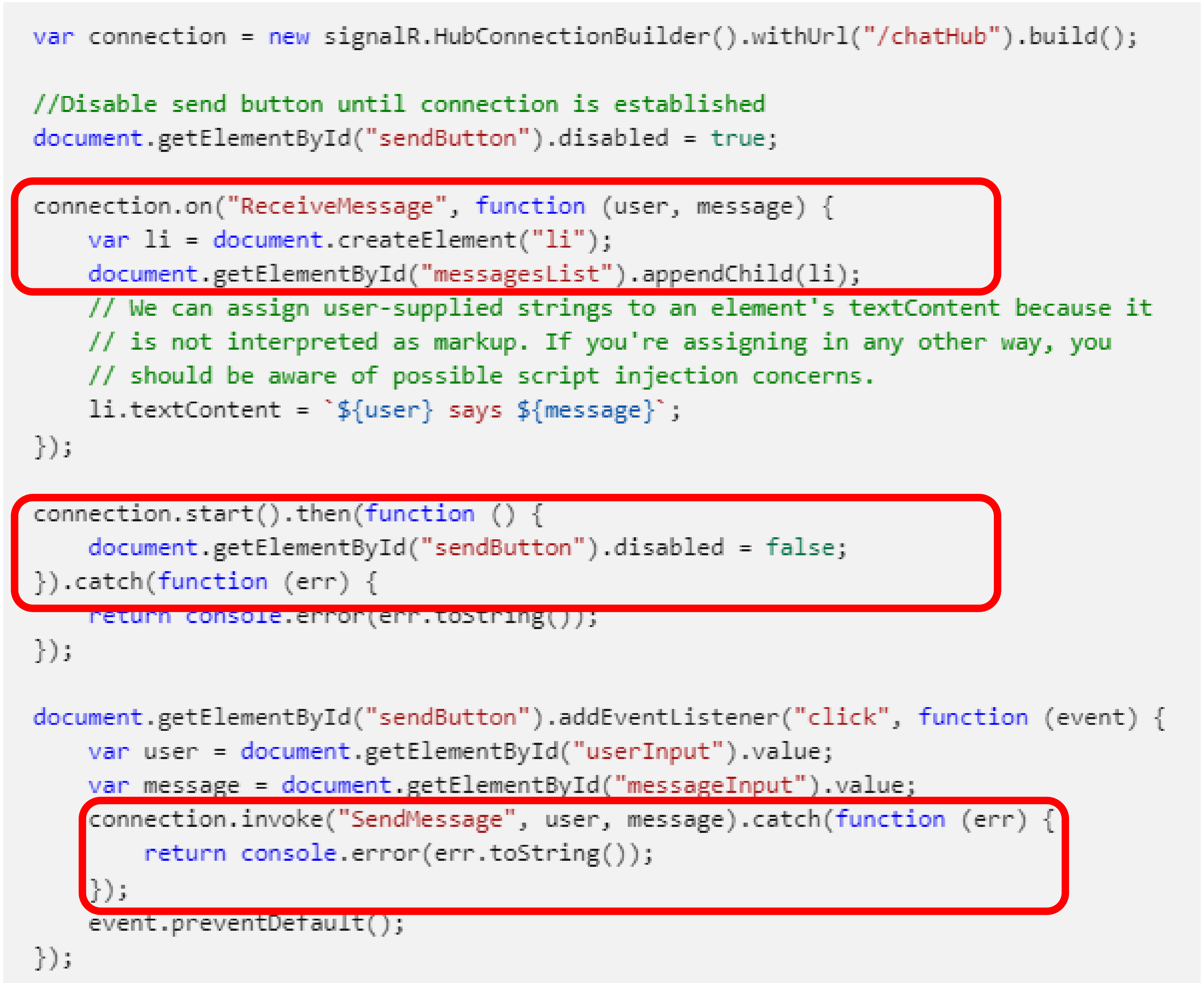
```
<script src="~/lib/signalr/signalr.js"></script>
```

- Or Using CDN

<script src="https://cdnjs.cloudflare.com/ajax/libs/microsoft-signalr/3.1.7/signalr.min.js"></script>



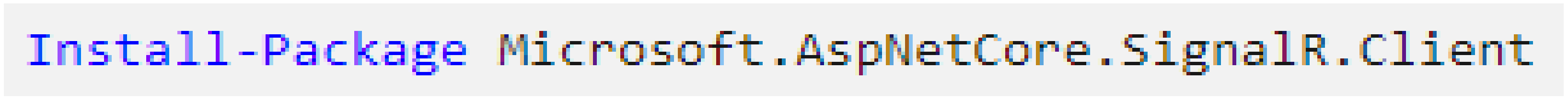
```javascript
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/chathub")
    .configureLogging(signalR.LogLevel.Information)
    .build();

async function start() {
    try {
        await connection.start();
        console.log("SignalR Connected.");
    } catch (err) {
        console.log(err);
        setTimeout(start, 5000);
    }
};

connection.onclose(async () => {
    await start();
});

// Start the connection.
start();
```



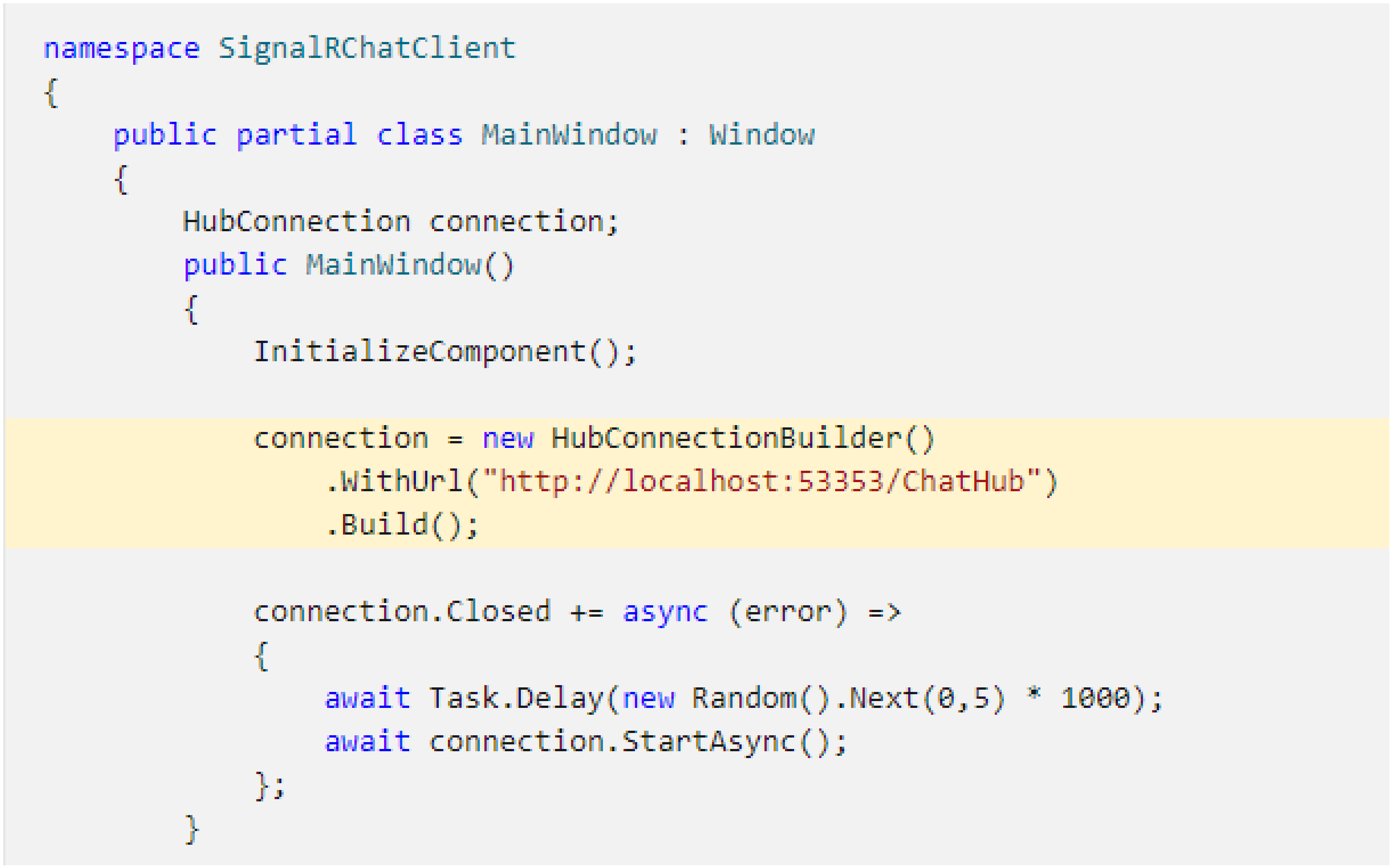
```
var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();

//Disable send button until connection is established
document.getElementById("sendButton").disabled = true;

connection.on("ReceiveMessage", function (user, message) {
    var li = document.createElement("li");
    document.getElementById("messagesList").appendChild(li);
    // We can assign user-supplied strings to an element's textContent because it
    // is not interpreted as markup. If you're assigning in any other way, you
    // should be aware of possible script injection concerns.
    li.textContent = `${user} says ${message}`;
});

connection.start().then(function () {
    document.getElementById("sendButton").disabled = false;
}).catch(function (err) {
    return console.error(err.toString());
});

document.getElementById("sendButton").addEventListener("click", function (event) {
    var user = document.getElementById("userInput").value;
    var message = document.getElementById("messageInput").value;
    connection.invoke("SendMessage", user, message).catch(function (err) {
        return console.error(err.toString());
    });
    event.preventDefault();
});
```

ASP.NET Core SignalR .NET Client

- The ASP.NET Core SignalR .NET client library lets you communicate with SignalR hubs from .NET apps

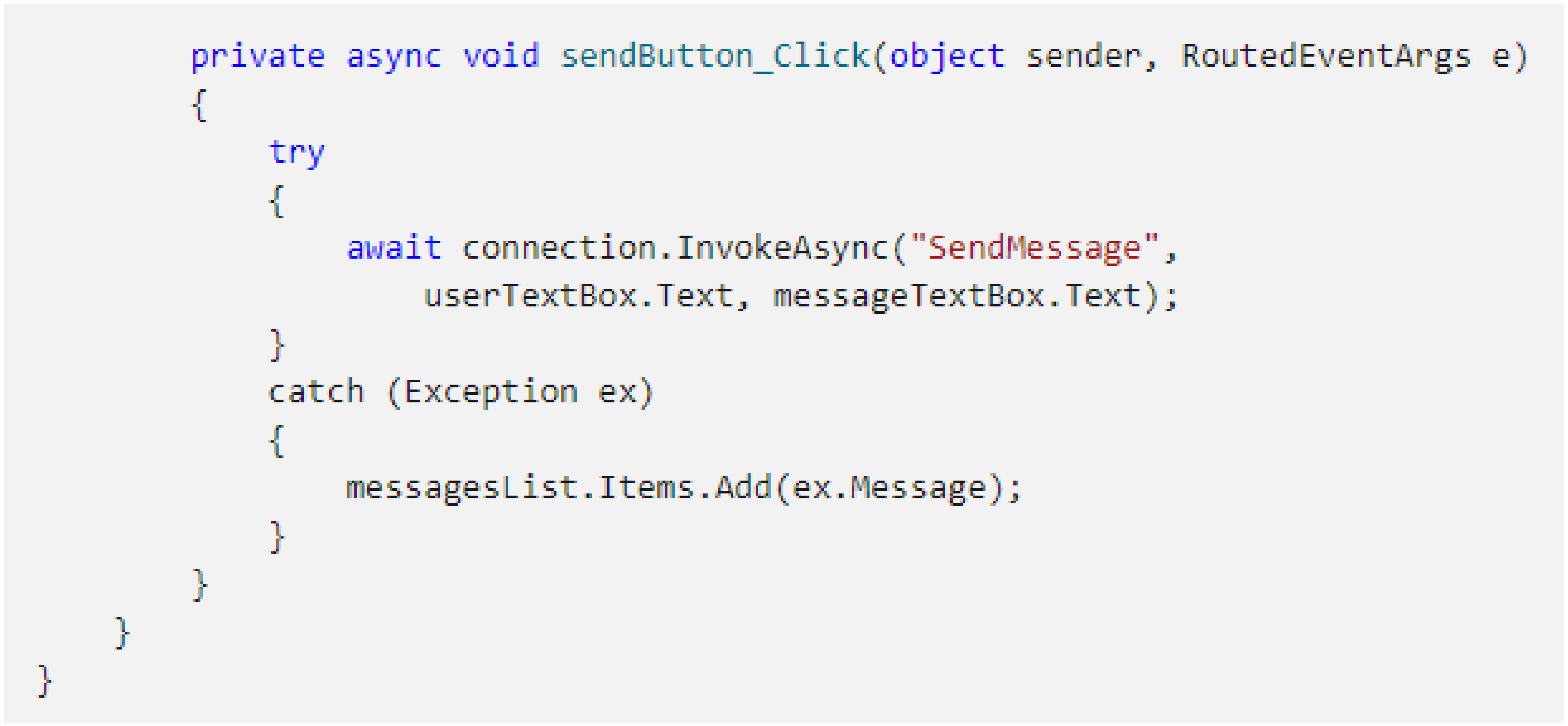


```
Install-Package Microsoft.AspNetCore.SignalR.Client
```



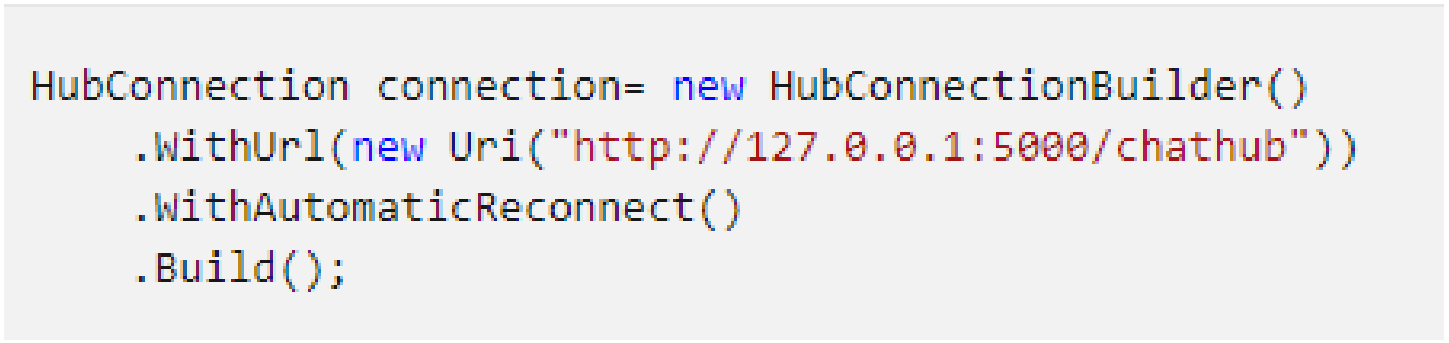
```
namespace SignalRChatClient
{
    public partial class MainWindow : Window
    {
        HubConnection connection;
        public MainWindow()
        {
            InitializeComponent();

            connection = new HubConnectionBuilder()
                .WithUrl("http://localhost:53353/ChatHub")
                .Build();

            connection.Closed += async (error) =>
            {
                await Task.Delay(new Random().Next(0,5) * 1000);
                await connection.StartAsync();
            };
        }
```

```
private async void connectButton_Click(object sender, RoutedEventArgs e)
{
    connection.On<string, string>("ReceiveMessage", (user, message) =>
    {
        this.Dispatcher.Invoke(() =>
        {
           var newMessage = $"{user}: {message}";
           messagesList.Items.Add(newMessage);
        });
    });

    try
    {
        await connection.StartAsync();
        messagesList.Items.Add("Connection started");
        connectButton.IsEnabled = false;
        sendButton.IsEnabled = true;
    }
    catch (Exception ex)
    {
        messagesList.Items.Add(ex.Message);
    }
}
```



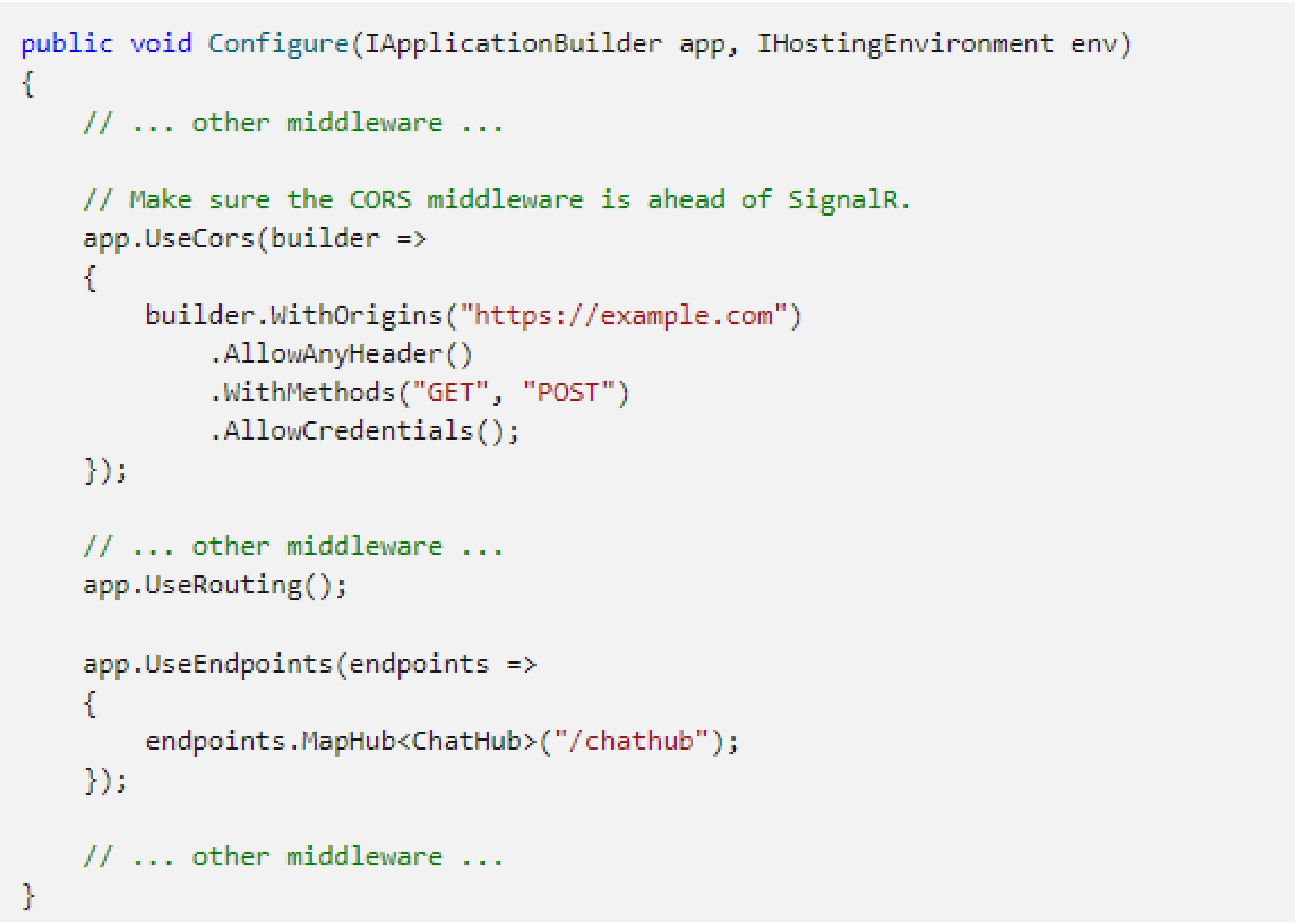
```
        private async void sendButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                await connection.InvokeAsync("SendMessage",
                    userTextBox.Text, messageTextBox.Text);
            }
            catch (Exception ex)
            {
                messagesList.Items.Add(ex.Message);
            }
        }
    }
}
```

Automatic Reconnect



```
HubConnection connection= new HubConnectionBuilder()
    .WithUrl(new Uri("http://127.0.0.1:5000/chathub"))
    .WithAutomaticReconnect()
    .Build();
```

CORS



```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // ... other middleware ...

    // Make sure the CORS middleware is ahead of SignalR.
    app.UseCors(builder =>
    {
        builder.WithOrigins("https://example.com")
            .AllowAnyHeader()
            .WithMethods("GET", "POST")
            .AllowCredentials();
    });

    // ... other middleware ...
    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapHub<ChatHub>("/chathub");
    });

    // ... other middleware ...
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddSignalR();

    services.AddCors(options =>
    {
        options.AddDefaultPolicy(builder =>
        {
            builder.WithOrigins("https://example.com")
                .AllowCredentials();
        });
    });
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
    }

    app.UseStaticFiles();
    app.UseRouting();

    app.UseCors();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
        endpoints.MapHub<ChatHub>("/chathub");
    });
}
```

THANK YOU