Lab: Implementing a chat using SignalR

# Scenario

The Adventures Works board and managers are pleased with the Photo Sharing application, but have requested that interactivity should be maximized to encourage users to register and participate fully in the community. Therefore, you have been asked to add chat functionality to the application.

Users should be able to start and stop a chat on a particular photo from the Display view. Chat rooms for each photo should be separated from each other. Users in the chat room should be able to send a message to all other users in that chat room, and they should be able to see all the messages that have been sent since they joined the chat room.

You have decided to use SignalR to implement the chat room over Web Sockets.

You will give the user the chance to join or leave a group. The group name will depend on the id of the Photo. Whenever a user sends a message to the SignalR Hub, the Hub will dispatch the message to every client in the same group.

# Objectives

After completing this lab, you will be able to:

* Create and implement a SignalR Hub that uses groups to manage chat rooms
* Create and implement a SignalR JavaScript client

Estimated Time: 60 minutes

# Exercise 1: Create and implement a SignalR Hub

## Scenario

Users should be able to start and stop a chat on a particular photo. Chat rooms for each photo should be separated from each other. Users in the chat room should be able to send a message to all other users in that chat room, and they should be able to see all the messages that have been sent since they joined the chat room.

In this exercise, you will:

* Create and implement a SignalR Hub.

The main tasks for this exercise are as follows:

1. Create a SignalR hub to push content to clients.
2. Modify the Startup class and configure the app.

### Task 1: Create a SignalR hub to push content to clients.

1. In Visual Studio, add a SignalR folder
2. Under the SignalR folder, add a ChatHub class
3. Derive the class from Hub the Model folder
4. Create an AddToGroup method, accepting a string userName and an int photoId
   1. Construct a group name by concatenating a “group-{photoId}”
   2. Add the user to the group by invoking the AddToGroupAsync method of the Groups object, passing the ConnectionId property of the Context object and the group name
   3. Let every member of the group know that the user joined by invoking the SendAsync method of the Group(groupName) of the Clients object, passing “MessageFromServer” and a string “{userName} joined”
5. Create a RemoveFromGroup, accepting a string username and an int photoId
   1. Construct a group name by concatenating “group-{photoId}”
   2. Remove the user from the group by invoking the RemoveFromGroupAsync method of the Groups object, passing the ConnectionId property of the Context object and the group name
   3. Let every member of the group know that the user left by invoking the SendAsync method of the Group(groupName) of the Clients object, passing “MessageFromServer” and a string “{userName} left”
   4. Send to the Caller a message “bye.”
6. Create a DispatchMessage method, accepting a string username, an int photoId and a string message
   1. Construct a group name by concatenating “group-{photoId}”
   2. Send a message to all Clients of the Group(groupName) by invoking the SendAsync and passing “MessageFromServer” and a string with “{userName}: {message}”

### Task 2: Modify the Startup class.

1. To configure a SignalR project, modify the project's Startup.ConfigureServices method by invoking services.AddSignalR method
2. Add signalR to the middleware pipeline and configure the routes to your hubs using

app.UseSignalR(routes => routes.MapHub<ChatHub>("/chathub"));

**Results**: At the end of this exercise, you will have created a SignalR Hub.

# Exercise 2: Create the JavaScript SignalR client

## Scenario

Users should be able to

* Start a chat on a particular photo from the Display view
* Stop a chat on a particular photo from the Display view
* Send a message to all other users in that chat room
* See all the messages that have been sent since they joined the chat room

To implement this feature you decide to write a JavaScript client that will handle the user interactions, invoke the SignalR Hub and dynamically build the user interface client side.

In this exercise, you will:

* Import the npm package for signalR
* Modify the Photo Details view to include
  + An empty container for the chat messages
  + An HTML template for a chat message
  + A Button to Join a chat room
  + A Button to leave a chat room
  + HTML input fields and a button to send a chat message
  + Javascript code to create an instance of a javascript Chat class
* Create a JavaScript Chat class, with methods to
  + Join a chat room
  + Leave a chat room
  + Send a message to the SignalR Hub
  + Handle messages received from the SignalR Hub

The main tasks for this exercise are as follows:

1. Modify the package.json file
2. Modify the Details View
3. Create and implement a Chat class

### Task 1: Modify the package.json file.

### Open package.json

1. Under the “dependencies” node, add a new element

"@aspnet/signalr": "~1.0.0"

**NOTE: Your version may vary.**

1. Save package.json

### Task 2: Modify the Details View.

The server will send static HTML and some JavaScript. The browser will then execute the JavaScript to establish a connection with the SignalR Hub and complete the rendering with the results.

Your View needs to contain:

* An empty container for the chat messages
* An HTML template for a chat message
* A Button to Join a chat room
* A Button to Leave a chat room
* HTML input fields and a button to Send a chat message
* Javascript code to create an instance of a javascript Chat class

Open the Details view of the Photo Controller

1. At the end of the existing content, add an UL element that will act as container for the comments. Set its id attribute to “chat-messages”
2. Add an INPUT field of type TEXT. Set its id attribute to “username”
3. Add a BUTTON. Set its id to “join”
4. Add a BUTTON. Set its id to “leave”
5. Add an INPUT field of type TEXT. Set its id attribute to “message”
6. Add a BUTTON. Set its id to “send”
7. Add a TEMPLATE tag. Set its id attribute to “chat-message-template”
8. In the template tag, add a LI element. Set its class to “chat-message”
9. Add a SCRIPT tag. Set its src attribute to "~/lib/signalr/signalr.js"
10. Add a SCRIPT tag. Set its type to “module”
11. Import the Chat class from the “/js/chat.js” module. You’re going to create this file in an upcoming task.
12. Create an instance of the Chat class, passing the @Model.Id value as a parameter

Task 3: Create and implement a Chat class

The Chat class:

* Handles the click event of all the chat-related buttons in the view and invokes the signalR hub to send messages to the server
* Handles messages from the server and dynamically builds the view with the message content

1. Under the wwwroot / js folder, create a chat.js file
2. Create a Chat class
3. Add a constructor accepting a photoId parameter
   1. Initialize a photoId property setting its value to the photoId parameter
   2. Get the document element whose id is “userName” and save it in a new “user” property
   3. Get the document element whose id is “chat-message-template” and save it in a new “template” property
   4. Get the document element whose id is “chat-messages” and save it in a new “target” property
   5. Get the document element whose id is “message” and save it in a new “message” property
   6. Create a new instance of signalR.HubConnectionBuilder, withUrl “/chatHub”, configure logging at Information level and Build the hub. Save the result in a new “connection” property.
   7. Get the document element whose id is “join” and add an event listener for the click event by invoking a this.join method. You will create this method at a later step.
   8. Get the document element whose id is “leave” and add an event listener for the click event by invoking a this.leave method. You will create this method at a later step.
   9. Get the document element whose id is “send” and add an event listener for the click event by invoking a this.send method. You will create this method at a later step.
   10. Handle the “MessageFromServer” event of the connection property by invoking a onMessageReceived method. You will create this method at a later step.
4. Add an async join method
   1. Start the connection and asynchronously wait
   2. Invoke the “AddToGroup” method of the connection passing the user name and the photo id
5. Add an async leave method
   1. Invoke the “RemoveFromGroup” method of the connection passing the user name and the photo id
   2. Stop the connection
6. Add an async send method
   1. Invoke the “DispatchMessage” on the connection, passing the user name, the photo id and the message
7. Add an onMessageReceived method, accepting a message parameter
   1. Find the this.template element whose class is “chat-message” and set its textContent to the message parameter
   2. Invoke the importNode method of the document object, passing this.template.content as first parameter and true as second parameter to indicate a deep import
   3. Append the result to this.target as a child node

Run the application on two separate browser tabs, chose the same photo and view its details on both tabs. Join a chat on both tabs using different user names. Send messages on both tabs and notice how the messages appear on both tabs. Leave the chat on one tab and notice how the messages do not appear on that tab if a message is sent from the other tab.

**Result: After completing this exercise, you will be able to build a JavaScript client that uses SignalR and dynamically build the HTML view client side.**