EventSource & CORS

Estimated time for completion: **30 minutes**

Overview:

In this lab you will use HTML5's EventSource to provide push notifications from the server to the browser. You will also allow CORS by issuing the proper response headers from server code.

Goals:

- Use EventSource for server notifications to browser
- Issue Access-Control-Allow-Origin header to allow CORS

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

EventSource

When a user is browsing the movie details page and another user adds a movie review we want the movie details page's review information to be updated.

Criteria:

- Use EventSource for server notifications to browser

Steps:

1. In this part of the lab you are going to enhance the movie details page to allow server notifications. The notification we're interested in is when the movie has a new review. While viewing the page is a new review is submitted we want the number of reviews and the average stars to be updated (without requiring the user to hit refresh).

   Right now the movie details page is doing this but in a less than optimal manner. Let's take a look at this now.

   a. Open the solution from `~/EventSource/before`.
   b. Run the application and visit the details page for any movie.
   c. Open up a new window or tab and submit a review for this movie. You should see the number of reviews updated on the movie details page.
   d. Now open the developer tools for your browser and start a network trace. You can see how the movie details page is being updated – it's continuously polling the server.

2. This is not ideal and so we'll change this implementation to use `EventSource` to be slightly more efficient.
   a. Open `~/Areas/Movies/Views/Home/View.cshtml`.
   b. Scroll down to the `<script>` section and inspect the current code. Notice there are a couple of functions to update the UI for new movie review data. Also notice the `pollServer` function which uses `window.setTimeout` to periodically poll the server for the current review data. Also notice the other function `pollServerWithEventSource`. This is the function you will implement to use `EventSource`.
   c. For newer browsers we can use `pollServerWithEventSource`, but for browsers that do not support `EventSource` we want to fall back to the `pollServer` approach.

      Before the call to `pollServer`, check to see if `typeof EventSource` is `"undefined"`. If so, call `pollServer` otherwise call `pollServerWithEventSource`.

```
if (typeof EventSource == 'undefined') {
    pollServer();
}
else {
    pollServerWithEventSource();
}
```

   d. Now inside of `pollServerWithEventSource` notice the URL for the server point is already defined. After this line of code create an instance of an `EventSource`.
   e. Register for the `open` event and inside the callback function use `console.log` to log that the event source was opened.
   f. Register for the `error` event and inside the callback function use `console.log` to log that there was an error.
   g. Register for the `message` event and inside the callback function use `console.log` to log the value of `event.data`.

```
function pollServerWithEventSource() {
    var url = '@Url.Action("GetMovieReviewInfoEvent", "MovieMessage")';

    var es = new EventSource(url);

    es.addEventListener("open", function (e) {
        console.log("EventSource Opened:" + e);
    }, false);

    es.addEventListener("error", function (e) {
        console.log("EventSource Error. Event Phase: " + e.eventPhase);
    }, false);

    es.addEventListener("message", function (e) {
        console.log("EventSource Message:" + e.data);
    }, false);
}
```

3. Run the application and visit one of the movie's details page. Use your browser's developer tools and open the console window. Then, open another tab or windows and enter a review for the movie. Hopefully you see 1) no polling and 2) the updated movie review data being sent back from the server when a new review is created.
4. Right now we're just logging the data, so the last thing is to actually update the UI.
   a. Back in the `"message"` event handler, Use `JSON.parse` on the `event.data`.
   b. Pass the resultant object to `updateReviewInfo`.

```
es.addEventListener("message", function (e) {
    console.log("EventSource Message:" + e.data);
    var updateData = JSON.parse(e.data);
    updateReviewInfo(updateData);
}, false);
```

5. Run the application again and now you should see the number of reviews (and potentially the number of stars) change when you create a new review.
6. If you're curious on how the server piece is implemented, you can examine the code in `~/Areas/Movies/Controllers/MovieMessageController.cs`.
7. One other problem with our design and implementation is that the server is notifying all clients when any movie review is created. So if you open a movie details page for one movie and create a new review for a different movie then the wrong movie details page will get updated. To solve this, we'll use the "event" feature of event source to indicate which movie the notification is being performed for.
   a. We need to change the server code to add the "event" prefix. Open `~/Areas/Movies/Controllers/MovieMessageController.cs`.

   b. Around line 106, notice the call to `Response.Output.WriteLine("data:" + json)`. This is where the data is being sent back to the client.

      The line above is commented out and renders `Response.Output.WriteLine("event:" + movieID.ToString())`. Uncomment the line. The server will now render the movie ID as the event.

```
Response.Output.WriteLine("event:" + movieID.ToString());
Response.Output.WriteLine("data:" + json);
```

   c. Back in the client JavaScript where you're registering for the `"message"` event. Change it from `"message"` to the movie ID. The movie ID is available in a variable named `thisMovieID`. This will now filter the client to only raise the event when the movie that's being displayed has an update.

```
es.addEventListener(thisMovieID.toString(), function (e) {
    console.log("EventSource Message:" + e.data);
    var updateData = JSON.parse(e.data);
    updateReviewInfo(updateData);
}, false);
```

8. Run the application again and browse to a movie details page. Open another tab or window for a different movie and submit a new review. You should not see the movie detail update. If you add a review for the same movie and then you should see the movie review data update.

Cross Origin Resource Sharing

To allow CORS the server must issue the proper HTTP headers in the response to allow other origins to use its resources.

Criteria:

- Issue Access-Control-Allow-Origin header to allow CORS

Steps:

1. In this part you'll be using resources from a different origin than the movie review application.

   a. Run the movie review application. Notice on the home page there is a "chat" window. The chat window uses Ajax to go back to the chat server which is a separate application.

   b. We need to start the chat server. In your solution there is another project called "ChatServer". Right-click the chat server project and click "Set as Startup Project" and then run it without debugging (Ctrl-F5).

Notice the URL in the browser – it's `"http://localhost:8888/"`. This is different than the movie review application, which is `"http://localhost:7777/"`.

c. Set the movie review application back to the startup project, run it and visit the home page.

d. Open your browser's developer tool and start a network trace.

e. In the name field enter your name.

f. In the message filed enter a message and click "Say".

g. Watch the network trace and notice some Ajax calls were made but they should fail. Inspect the HTTP headers on the response. They do not include the `Access-Control-Allow-Origin` header.

h. To fix this, open the server code in the `ChatServer` project at `~/Controllers/ChatController.cs`.

i. Around line 32 notice the `ChatController` class. The line above it is commented out. Uncomment it. This enables the result filter which issues the CORS header to the Origin specified. Notice it is set to `"http://localhost:7777"`. You could also just set it to `"*"` to allow any origin.

2. 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.

Solutions:

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