Browser clients

Estimated time for completion: **45 minutes**

Overview:

In this lab you will write a browser-based WebAPI client using JavaScript and you will use Ajax to invoke the WebAPI movie service.

Goals:

- Write a browser-based movie service client

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

Browser-based client

In this part of the lab you will build a browser-based client of the movie service. It will display the list of the movies in a table. When a movie in the table is selected you will populate an editable section of the page with the movie details and the user can submit changes back to the server

Criteria:

- Use Ajax to invoke the movie service

Steps:

1. In this part you will be building a browser-based client of the movie service.
   a. Open the solution from `~/BrowserClients/before/service`.
   b. Notice there is a new file in the project called `~/MovieListing.htm`. Open it and start to get familiar with it.

      This is where the client code will be written. It will have the same basic functionality as the desktop client used in earlier labs. It will show the movie listings and allow updating, creating and deleting of individual movies.

    c. If you run the solution now this HTML page is set as the default page.

2. In `MovieListing.htm` there is placeholder function called `loadAllMovies`. This is called to populate the `<table>` with the movie listings. Implement the movie listing functionality.
    a. Implement `loadAllMovies` by making an Ajax call to the movie service.
    b. Handle the result and populate the `<tbody>` with the appropriate movie data.
3. In `MovieListing.htm`, once the `<table>` is populated with movie data there is code to allow the user to select the rows. When the user selects a movie the details are supposed to populate the `<form>` on the right. This is performed the `loadMovie` function is invoked and the movie ID is passed as a parameter. Implement the movie selection functionality.
    a. Implement `loadMovie` to make an Ajax call to the movie service passing the correct id to retrieve the correct movie.
    b. Handle the result and populate the `<form>` with the appropriate data.
4. There are three buttons beneath the `<form>` for invoking create, update and delete operations. You will implement these operations now.
    a. Devise a scheme to know when any of these buttons is clicked. It will be important to identify which button was clicked so the correct method can be invoked on the service.
    b. Use Ajax to invoke the correct method on movie service and pass the contents of the `<form>` as `"application/json"` content type.
    c. Simply `alert` the user upon success (or failure) of the Ajax call.

Jpeg media type formatter

In this part of the lab you will allow the browser client to request an image for a movie. This will be done with a custom Jpeg media type formatter.

Criteria:

- Implement a Jpeg media type formatter
- Display movie posters in the browser client

Steps:

1. In this part you will also populate the movie poster into the movie listings table. Provide an implementation in the movie service to render the movie poster to the browser.
    a. To render the image a custom media type formatter will be necessary. Since we are already using the built-in `JsonMediaTypeFormatter` for the `Movie` objects, we need a different data type to differentiate how the `Movie` will be rendered.

The `Movie` class implements an interface called `IHaveJpgImage` and supports writing the movie poster image to a `Stream`.

This should be sufficient information for designing an approach for rendering movie posters from `Movie` objects (and the `MoviesController` won't have to change).

2. Update the client code to show the movie poster.
    a. When rendering the movie listing table, add a new cell to render an `<img>` for the movie poster.

       One problem you might run into is that some browsers (Chrome for one) will send `Accept: */*` for `<img>` HTTP requests and the image media type formatter won't be chosen by the WebAPI content negotiation framework. To address this problem you will need another mechanism for the client to indicate the format desired. The typical approach is to pass a query string.

       Hint: Look at into the `AddQueryStringMapping` extension method for your image media type formatter to enable query string content negotiation support.

Solutions:

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