DEV.BUILD WORKBOOK WEEK 11

LAB 15.1: CONSUMING AN API

Task: Work with an API that generates JSON data to display information on a web page.

Build Specifications:

1. Work with the Deck of Cards API, documented at https://deckofcardsapi.com/
2. In a Web MVC project, make a call to the API to generate a new deck. Capture the deck ID returned.
3. Draw 5 cards from the deck and display their names and images inside a view.

#2 and 3 can be done inside a single controller action if you'd like. You can do this in the Home controller of a new project, but it might be handy to do it on an action other than the index.

Notes: If you want to see a tree representation of the JSON data, use this website: http://jsonviewer.stack.hu/

Grading Standards:

- Application calls the Deck of Cards API to generate a new deck: 1 point
- Application includes a model class for the deck: 1 point
- Application captures the deck ID correctly: 1 point
- Application calls the Deck of Cards API to draw 5 cards: 1 point
- Application includes a model class for a hand of 5 cards: 1 point
- Application includes a model class for a card: 1 point
- Application correctly stores the 5 drawn cards in the model: 1 point
- Application displays the names (value and suit) of 5 drawn cards in the view: 1 point
- Application displays the images of the 5 drawn cards in the view: 1 point

Extended Challenges:

1. On the page that displays the cards, have a button to draw the next 5 cards from the same deck. You'll need to pass the ID along, either coded into the URL, as a hidden field, or using sessions or cookies.

2. Add "Keep" functionality to the cards view. When the user clicks the draw again button, keep the cards they chose to keep and only draw to replace the others. (Checkboxes might be the easiest way to accomplish the keep.)

LAB 15.2: CREATING A REST API

Task: Create a Web API that will provide movie data.

What will the application do?
Design an API that will provide the following endpoints:

1. Get a list of all movies
2. Get a list of all movies in a specific category
    - User specifies category as a query parameter
3. Get a random movie pick
4. Get a random movie pick from a specific category
    - User specifies category as a query parameter
5. Get a list of random movie picks
    - User specifies quantity as a query parameter
6. Get a list of all movie categories
7. Get info about a specific movie
    - User specifies title as a query parameter
8. Get a list of movies which have a keyword in their title
    - User specifies title as a query parameter

Build Specifications:

- Use the Dapper framework. Create a small movie database
- Set your API to include actions, and use the JSON Formatter for text/html requests.
- Note that if you use any special characters in queries to test your API, you need to encode them. See https://www.w3schools.com/tags/ref_urlencode.asp for a list of encodings. For instance, if your database had Science Fiction as a category (with a space), your test query might look like /listByCategory?category=Science%20Fiction
- Complete the first 4 tasks above, then complete as many as you can tasks 5-7, starting with whatever you want as extended challenges.
- You can decide the key names for your query parameters.
- API queries which return no results should return properly formatted empty JSON (for instance, a search for a category that doesn't exist).
- You can use Postman to test your API.

Grading Standards:

- 2 points for a working API project
- 2 points each for items 1-4 above

LAB 15.3: DESIGNING AND CREATING A REST API

Task: Design and implement a web API based around the Northwind database.

What will the application do?
As a group, review the tables in Northwind. Prioritize the information that you see being most important for a client and build an API to provide it, subject to the build specs below.

Build Specifications:

- API must have at least 3 controllers (3 points, 1 per working controller)
- API must have at least 2 GET endpoints per controller, whether there's an additional path added on (/api/Products/ByCategory) or you've got an optional query parameter (/api/Products?category=Food) (6 points, 1 per endpoint)
- API must have at least 2 POST endpoints (total, not per controller) which allow information to be added to a database table (4 points, 2 per endpoint)
- API must have at least 1 DELETE endpoint (total, not per controller) which allows records to be deleted. ***Choose wisely!*** You have to be careful deleting when another table's foreign key relationships point to records in your table. (2 points)

Extended Challenges:

- Implement PUT and PATCH methods as well for altering records.
- Refactor your code with a DAL, IDAL, and dependency injection.