SE 3200: Web Application Development I

Assignment: Resourceful

Requirements

Resource

- Define a single resource that your web application will manage. Also define a simple data model for your resource that includes at least five attributes (not including the unique identifer).

Server API

- Using Python, create a server web application which implements an API that supports full CRUD operations for your resource, according to the following specifications:
    - Five RESTful routes, each implemented using the appropriate request method and path, response headers, and response status codes (including any error conditions):
        - **List**: returns a JSON representation of the collection's members.
        - **Retrieve**: returns a JSON representation of a single member in the collection, as specified by the unique identifier given as a path parameter.
        - **Create**: creates a new member within the collection, as specificed by the data parameters given in the request body.
        - **Replace**: updates an existing member in the collection, as specified by the unique identifier given as a path parameter, and by the data parameters given in the request body.
        - **Delete**: removes an existing member from the collection, as specified by the unique identifier given as a path parameter.
    - CORS should be implemented server-wide in order to support Ajax requests from client applications. This should include support for both simple requests and preflighted requests, by implementing the `OPTIONS` method appropriately.
    - If a request is received that does not conform to the RESTful routes defined above, then the server should return an appropriate error response, with the correct status code, headers, and content body that properly explain the reason for the response. This should include requests for an invalid route (method/path pair), and requests that specify a nonexistent member of the collection.

Database

- Using SQLite, create a valid database schema sufficient to store data records according to the data model that you defined to represent your resource.
- Within your server application, create a Python class to encapsulate all database logic for your application. This class will then be utilized by each of the RESTful actions implemented by the server API to interface with the database for all CRUD operations.

Client

- Using JavaScript, create a client web application that communicates with your server application, using its API, on behalf of the user, with the following:
    - A list showing the collection of members for your resource, with at least two of the attributes displayed for each member in the list. The list should be able to clearly and effectively display many records. The data for the list should be requested using the appropriate API endpoint. The list should remain updated with any changes made via the functionality described below, using API requests as necessary to receive updated data.
    - A form used to create a new member in the collection, as well as to update an existing member. The form should include a field for each attribute belonging to your resource. When updating, the form

fields should be pre-filled such that the user may quickly make a change without re-entering the values. The data from the form should be sent using the appropriate API endpoints.

    - Buttons and/or links which accommodate any needed UI transitions, such as to display an empty form to create a new member, or to display a pre-filled form to update an existing member.
    - A delete button or link associated to each record that allows the user to remove a member from the collection, after confirming the action with the user (hint: use the `confirm()` method).
    - Optionally, a read-only list of all attributes and associated values for a single member in the collection, using data requested by the *retrieve* API endpoint. Alternatively, it is sufficient for the user to view the attributes of each member via the pre-filled form described above, using data requested by the *list* API endpoint.
    - All data sent to and received from the server API should be implemented using Ajax requests.

- You may take liberties to modify your application's features and purpose from that described above, but the overall specifications and structure listed above should still be met.
- Make your application look professional and presentable. Use valid HTML and CSS to structure and style your application.
- No third-party JavaScript or CSS libraries or frameworks may be used without prior instructor permission.

Documentation

- The following items should be clearly detailed and documented in the `README.md` within your Git repository:
    - The name of your resource, and the name of each of its attributes.
    - The database schema which represents your resource, documented as a valid SQLite `CREATE TABLE` query.
    - All REST endpoints implemented by your API server. Include the name, HTTP method, and path for each.
- Use Markdown to structure and style the content within your `README.md`. Go here to see an example.

Submission

1. Submit your project using Git and GitHub. Start by creating a repo for this assignment here.
2. Show your completed assignment to the instructor during class or office hours to receive credit.