Cart API

Creating a REST API

Task: Use Spring Boot to create an API server that provides a RESTful API for a collection of cart items in MySQL. You can test your API using https://gc-express-tester.surge.sh.

Setup:
Create a new Spring project with a MySQL database. However, it will not have any views. name to project **cart-api**.

Build Specifications:

1. Set up a table in MySQL.
    1. table name: **cart_items**
    2. table fields: **id** (long), **product** (String), **price** (Double), **quantity** (Integer)
    3. Use any schema you like.
2. Use Spring Boot to create your server. Do not include any views, only REST controller(s).
3. Test your endpoints using Postman.
4. Also test your finished API using https://gc-express-tester.surge.sh. (See guide below.)

Endpoints:
The API should have the following endpoints.

- `GET /cart-items`
    - Action: None
    - Response: a JSON array of all cart items
    - Response Code: `200` (OK)
    - Query string parameters: the request may have one of the following or it may have none. (See test cases below for examples.)
        1. `product` - if specified, only includes cart items that have this exact product name.
        2. `maxPrice` - if specified, only include cart items that are at or below this price.
        3. `prefix` - if specified, only includes cart items that start with the given string in the response array.
        4. `pageSize` - if specified, only includes up to the given number of items in the response array. For example, if there are ten items total, but pageSize=5, only return an array of the first five items.
- `GET /cart-items/{id}`
    - Action: None
    - Response: a JSON object of the item with the given ID

    - Response Code: `200` (OK)
    - However, if the item with that ID cannot be found in the array, return a string response “ID Not Found” with response code `404` (Not Found)

- `POST /cart-items`
    - Action: Add a cart item to the array using the JSON body of the request. Also generate a unique ID for that item.
    - Response: the added cart item object as JSON.
    - Response Code: `201` (Created)
- `PUT /cart-items/{id}`
    - Action: Update the cart item in the array that has the given id. Use the JSON body of the request as the new properties.
    - Response: the updated cart item object as JSON.
    - Response Code: `200` (OK).
- `DELETE /cart-items/{id}`
    - Action: Remove the item from the array that has the given ID.
    - Response: Empty
    - Response Code: `204` (No Content)
- `GET /cart-items/total-cost`
    - Action: Obtain the total price of the cart. (Multiply each item’s count by its price, and add up this number for each item. Then add 6% on sales tax by multiplying that sum by 1.06.)
    - Response: The total price as a double.
    - Response Code: `200` (OK)
- `PATCH /cart-items/{id}/add`
    - Action: Updates the quantity of an existing item in the cart whose id matches the given id.
    - Response: The updated cart item as JSON
    - Response Code: `200` (OK)
    - Required query parameters:
        1. count: The amount to add to the cart. For example, if the cart presently has 3 apples and this API call has a count of 2, the cart will be updated to contain 5 apples.

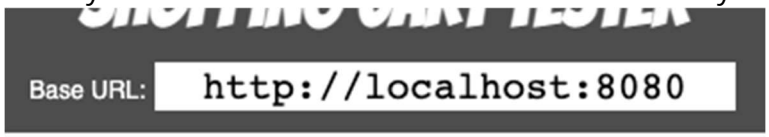
Testing With Client Application

When your API is finished, you plug it into this premade application (https://gc-express-tester.surge.sh) and see if it works!

1. In order for the web application to access your API from the browser, you must configure your RestController with CORS. Add this annotation to the top of your controller.
   `@CrossOrigin`

2. Visit the web application URL: https://gc-express-tester.surge.sh

3. Enter your API URL in the "Base URL" field. It's usually http://localhost:8080



Extended Challenges:

1. Add a **page** query string parameter. This should be combined with pageSize to respond with a specific page of results.
2. In the above instructions, GET /cart-items takes any one of the query string parameters: maxPrice, prefix, or pageSize. Make it also allow any combination of those. e.g. `/cart-items?prefix=A&maxPrice=20.0&pageSize=5`

Example Test Cases (to test manually with Postman):

1. `GET /cart-items` - responds with a JSON array of all cart items
2. `GET /cart-items` - responds with status code **200**
3. `GET /cart-items?product=Cheese` - responds with a JSON array of only the cart items that have `product` equal to "Cheese".
4. `GET /cart-items?maxPrice=3.0` - responds with a JSON array of only the cart items that have `price` <= 3.0
5. `GET /cart-items?prefix=Fancy` - responds with a JSON array of only the cart items that have `product` starting with "Fancy".
6. `GET /cart-items?pageSize=10` - responds with a JSON array of all cart items, but if there are more than ten items, the response includes only the first ten.
7. `GET /cart-items/:id` - responds with a JSON object of the item with the given ID
8. `GET /cart-items/:id` - responds with status code **200**
9. `GET /cart-items/:id` - responds with status code **404** when not found
10. `POST /cart-items` - add a cart item to the array using the JSON body of the request. Also generates a unique ID for that item.
11. `POST /cart-items` - responds with the added cart item object as JSON and status code **201**.
12. `PUT /cart-items/:id` - Updates the cart item in the array that has the given id.
13. `PUT /cart-items/:id` - Responds with the updated cart item as JSON and status code **200**.
14. `DELETE /cart-items/:id` - Removes the item from the array that has the given ID.
15. `DELETE /cart-items/:id` - Responds with no content and status code **204**.
16. `GET /total-cost` - Responds with the total cost of the cart. You will want to manually calculate everything in your collection to make sure this number is correct.
17. `PATCH /cart-items/:id/add?count=3` - Adds three items to the given cart.