Due 7/14

EXPRESS LAB 2: CART API - NODE PG

Task: Start with a copy of Express Lab 1. Modify it to use a database instead of using the array to store cart items.

Build Specifications:

1. In pgAdmin, create a database called "ExpressShopDB" and a table called "shopping_cart". The table will have columns: `id`, `product`, `price`, and `quantity`. Don't forget to set `id` as the primary key with auto increment.
2. Construct the pg-connection-pool.js file that will contain all of the information allowing the server to communicate with the database.
3. Adjust your GET, POST, PUT, and DELETE requests in your routes module to include the appropriate queries for each of the five requests.
4. Test your endpoints with Postman to make sure the routing is set up.
5. Also test your finished API using https://gc-express-tester.surge.sh.

post and put: need to return object - students.js example is wrong

Hints:

- For the query string parameters maxPrice and prefix, use an SQL WHERE clause.
- For the query string parameter pageSize, use an SQL LIMIT clause.

Extended Challenges:

- Continue to build an Angular app to call your API and display results.
- Use a form to add items by calling your POST endpoint.
- Add buttons to the cart items. On click call your DELETE endpoint.
- Enable items to be edited (or perhaps just the quantity changed) and call the PUT endpoint.

Tests

1. Database is named **ExpressShopDB** and table is named **shopping_cart**.
2. shopping_cart table has four columns (`id`, `product`, `price`, and `quantity`).
3. `GET /cart-items` - responds with a JSON array of all cart items from database
4. `GET /cart-items` - responds with status code **200**
5. `GET /cart-items?maxPrice=3.0` - responds with a JSON array of only the cart items that have `price` <= 3.0
6. `GET /cart-items?prefix=Fancy` - responds with a JSON array of only the cart items that have `product` starting with "Fancy".
7. `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.
8. `GET /cart-items/:id` - responds with a JSON object of the item with the given ID
9. `GET /cart-items/:id` - responds with status code **200**
10. `GET /cart-items/:id` - responds with status code **404** when not found

11. `POST /cart-items` - add a cart item to the database using the JSON body of the request. Database generates a unique ID for that item.
12. `POST /cart-items` - responds with the added cart item object as JSON and status code **201**.
13. `PUT /cart-items/:id` - Updates the cart item in the database that has the given id.
14. `PUT /cart-items/:id` - Responds with the updated cart item as JSON and status code **200**.
15. `DELETE /cart-items/:id` - Removes the item from the database that has the given ID.
16. `DELETE /cart-items/:id` - Responds with no content and status code **204**.

for post, since need to return object created that includes id, normally when you do an insert it doesn’t return anything. you can use returning (SQL term) at the end of it
regular INSERT into at the beginning, then:
RETURNING *