Unit 7: Express & APIs: Lab 1 – Cart API with Array (Express)

Assigned: 01/11/23

Due: 01/12/23

Completed: 01/12/23

**LMS:** <https://lms.grandcircus.co/mod/assign/view.php?id=22806>

**Google Doc:** <https://docs.google.com/document/d/1LwQvBu5XKaZT7M7i82lYAs6aZ3qhfozXQ-DCz-DxOng/preview>

**GitHub:**

**Task:** Use Express to create an API server that provides a RESTful API for a collection of cart items. You can test your API using <https://gc-express-tester.surge.sh>.

**Build Specifications:**

1. Use Express to create your server.

import express, { Application, Request, Response } from 'express';

const app:Application = express();

const port = 3000;

app.use(express.json());

app.use(express.urlencoded({extended: true}));

// Listen takes two parameter: what port do you want me to run on, when started up successfully then what do you want me to do

// In our case we are using port 3000 (defined above) and we console.log the message shown

app.listen(port, ():void=> {

    console.log(`Listening on port ${port}`);

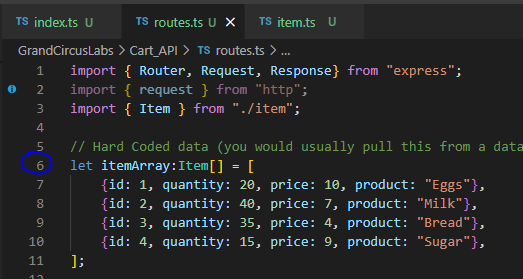
});

1. Require the module that will contain the routes you have created.

Found in index.ts



1. Start your server out with a hard-coded array of cart items, each including id, product, price, and quantity.



1. Test your endpoints using Postman.

Text

Description automatically generated

1. Also test your finished API using <https://gc-express-tester.surge.sh>. To do so, you must enable CORS support using the node cors package.

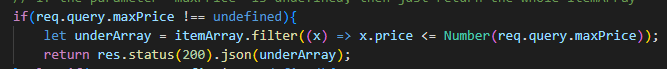
**Endpoints:**

The API should have the following endpoints.

1. **GET /cart-items**

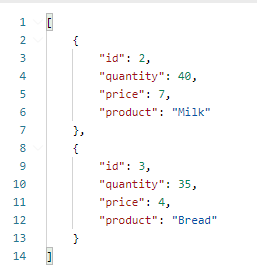


* 1. Action: None
  2. Response: a JSON array of all cart items
  3. Response Code: **200** (OK)
  4. Query string parameters: the request may have one of the following or it may have none. (See test cases below for examples.)
     1. **maxPrice** - if specified, only include products that are at or below this price.

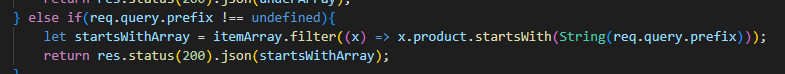


Graphical user interface

Description automatically generated with low confidence



* + 1. **prefix** - if specified, only includes products that start with the given string in the response array.



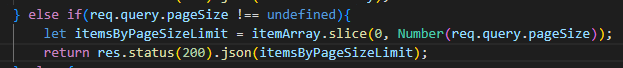
Text

Description automatically generated

Text

Description automatically generated with low confidence

* + 1. **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.



Text

Description automatically generated with medium confidence

Text

Description automatically generated

1. **GET /cart-items/:id**
   1. Action: None
   2. Response: a JSON object of the item with the given ID
   3. Response Code: **200** (OK)
   4. 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)

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated



1. **POST /cart-items**
   1. Action: Add a cart item to the array using the JSON body of the request. Also generate a unique ID for that item.
   2. Response: the added cart item object as JSON.
   3. Response Code: **201** (Created)

We created a new item in Postman by going back to our main URI, then clicking into the Body tab and selecting ‘raw’, then change the output to JSON.

Graphical user interface, application, website

Description automatically generated

A picture containing text

Description automatically generated



Graphical user interface, text, application, Word

Description automatically generated

I created a second item and you can see the ID increased incrementally to ‘6’ because of the helper function we wrote.

Graphical user interface, text, application

Description automatically generated

If I type a string (Hello) for the price instead of a number, then we get an error

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated

If we do a GET, then we can see those items are added to our cart-items

Text

Description automatically generated with medium confidence

1. **PUT /cart-items/:id**
   1. Action: Update the cart item in the array that has the given id. Use the JSON body of the request as the new properties.
   2. Response: the updated cart item object as JSON.
   3. Response Code: 200 (OK).
2. DELETE /cart-items/:id
   1. Action: Remove the item from the array that has the given ID.
   2. Response: Empty
   3. Response Code: **204** (No Content)

Because the way this is written, there is no validation checking to make sure someone provided a new value. I first tried it with only Quantity and it changed the Quantity, but resulted in “null” for Price and “undefined” for Product.

We would need to separate out the changes to each property OR add IF / ELSE statements to make sure that the user has provided a value and IF YES, then use that new value. IF NO, then leave the existing value.

Text

Description automatically generated

Graphical user interface, application

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Whereas entering all values changed the Quantity and Price, leaving the Product as is.

Graphical user interface

Description automatically generated with low confidence



1. **DELETE /cart-items/:id**

Started with the following in our array

A picture containing text

Description automatically generated

Then used DELETE to delete id: 4

Graphical user interface, text

Description automatically generated with medium confidence

The way this is written it responds with a message that the item was remove

Text

Description automatically generated



**Extended Challenges:**

1. 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
2. Build a JavaScript app to call your API and display results.

**Example Test Cases (to test manually with Postman, not Jest):**

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?maxPrice=3.0 - responds with a JSON array of only the cart items that have price <= 3.0
4. GET /cart-items?prefix=Fancy - responds with a JSON array of only the cart items that have product starting with "Fancy".
5. 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.
6. GET /cart-items/:id - responds with a JSON object of the item with the given ID
7. GET /cart-items/:id - responds with status code **200**
8. GET /cart-items/:id - responds with status code **404** when not found
9. 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.
10. POST /cart-items - responds with the added cart item object as JSON and status code **201**.
11. PUT /cart-items/:id - Updates the cart item in the array that has the given id.
12. PUT /cart-items/:id - Responds with the updated cart item as JSON and status code **200**.
13. DELETE /cart-items/:id - Removes the item from the array that has the given ID.
14. DELETE /cart-items/:id - Responds with no content and status code **204**.