NWEN304 Final Project [Group12]
Banana Online Shopping Website

The system (Josh)

Our website functions in the same way that most shopping websites do - a user has the ability to create an account, and then log in and browse the items. Each user has a 'cart' that they can add and remove items to as they please, and then from within this cart they can 'checkout' and place the order for their items.

Error handling (Josh)

In terms of error handling in the application, we have a few things in place. For the most part the application simply catches an error as it happens and sends an error message through the browser.

Database Design and RESTful API (Andy Yang)

Our project initially had a plan to have three separate tables to store user details, item/product details and shopping cart details respectively. Even though we had discovered some better ways to redesign our database nearly one day (few hours) before our presentation, we still could not make a huge change on it because it can affect all the webpages using those database query. Therefore, currently we are just using cart_table mainly which contains: cart_id (arbitrarily set # to 999), item_id (for referencing its product name), and item_quantity (upon user's constant updates). Everyone logged in should share a same universal cart, and should proceed to checkout page to finalise the payment to clear the cart database for the next user. Besides, this web application is reasonable when serving a single logged in user at a time and that user needs to clear the cart and pay before leaving.

<u>Restful API we have got:</u>

1. GET
2. POST (Adding is only performed when there is no existing such item type in the cart yet)

```
//******Andy DB*****POST Request**************/
app.post('/api/products',urlencodedParser, async (req,res)=>{
  try {
    const client = await pool.connect();

    var r= await client.query('select item_id from  cart_table where item_id='+req.body.item_id+';');
    if((r.rows).length==0){
      var result = await client.query('insert into cart_table values ('+req.body.cart_id+', '+req.body.item_id+', '+req.body.item_quantity+');
    }else{  //if there are rows with same item_id
      var old_value= await client.query('select item_quantity from cart_table where item_id='+req.body.item_id+';');
      res.send(old_value);
    }
    if (!result) {
        return res.send("POST Failure");
      } else {
        console.log("successful");
      }
      res.send(result.rows);
      client.release();
    } catch (err) {
      console.error(err);
      res.send("Error " + err);
  }
});
```

3. PUT (*for avoiding adding unnecessary duplicate item rows into the cart!)
4. DELETE (Clearing database purpose/ or to support delete button in the cart.html page)

Curl command Test script (Andy Yang):

- **GET (api/logged): $ curl -i https://fathomless-tor-48342.herokuap.com/api/logged**
  → return '0' (user is not yet logged in)

```
0yangjunande-MacBook:~ andy-yang$ curl -i  https://fathomless-tor-48342.herokuap
.com/api/logged
HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 1
Etag: W/"1-tlifxqsNyCzxIJnRwtQKuZToQQw"
Set-Cookie: connect.sid=s%3AE6cEmTyh_S3HA-rba-apbaqibfcdw7Re.tq%2Fj2DwcUQScB7OPB
8EPdiBS2D2xUTQ32IrQvOQ%2BIMw; Path=/; HttpOnly
Date: Thu, 15 Nov 2018 23:59:47 GMT
Via: 1.1 vegur

0yangjunande-MacBook:~ andy-yang$
```

- **GET (api/products):**
  $ curl -i https://fathomless-tor-48342.herokua.com/api/products
  → if the cart currently has items inside, it returns all items in a JSON file format

```
[]yangjunande-MacBook:~ andy-yang$ curl -i  https://fathomless-tor-48342.herokua
.com/api/products
HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 139
Etag: W/"8b-rei5H183C8SHH54iOLE8s3fr5HA"
Set-Cookie: connect.sid=s%3Atb8O6wNGEH-xuYjGHh_Dqe1YCL8hJIMJ.U2uguN6Zqs183LGblxA
vSQ%2Bc74VXHLXESWdoMV2HVv8; Path=/; HttpOnly
Date: Fri, 16 Nov 2018 00:04:51 GMT
Via: 1.1 vegur

[{"cart_id":999,"item_id":4,"item_quantity":1},{"cart_id":999,"item_id":5,"item_
quantity":1},{"cart_id":999,"item_id":3,"item_quantity":2}]yangjunande-MacBook:~
 andy-yang$
```

- **POST (api/products): $ curl -d "cart_id=999&item_id=1&item_quantity=100" https://fathomless-tor-48342.herokuapp.com/api/products**

→ e.g. I add 100 Cute Banana (with idem_id=1) and assumed that the cart does not already exist any Cute Banana:

```
yangjunande-MacBook:~ andy-yang$ curl -d "cart_id=999&item_id=5&item_quantity=10
0" https://fathomless-tor-48342.herokuapp.com/api/products
{"command":"SELECT","rowCount":1,"oid":null,"rows":[{"item_quantity":1}],"fields
":[{"name":"item_quantity","tableID":6534967,"columnID":3,"dataTypeID":23,"dataT
ypeSize":4,"dataTypeModifier":-1,"format":"text"}],"_parsers":[null],"RowCtor":n
ull,"rowAsArray":false}yangjunand
yangjunande-MacBook:~ andy-yang$ curl -d "cart_id=999&item_id=1&item_quantity=10
0" https://fathomless-tor-48342.herokuapp.com/api/products
[]yangjunande-MacBook:~ andy-yang
```

→ result response:

[{"cart_id":999,"item_id":4,"item_quantity":1},{"cart_id":999,"item_id":5,"item_quantity":1},{"cart_id":999,"item_id":3,"item_quantity":2},{"cart_id":999,"item_id":1,"item_quantity":100}] → We can see that it was added successfully.

- **PUT (api/products): $ curl -X PUT -d "item_quantity=299 &item_id=1" https://fathomless-tor-48342.herokuapp.com/api/products**

  → altering the table value of item_id=1 (cute banana) to become new value 299!

```
yangjunande-MacBook:~ andy-yang$  curl -X PUT -d "item_quantity=299 &item_id=1"
https://fathomless-tor-48342.herokuapp.com/api/products
[]yangjunande-MacBook:~ andy-yangcurl -i  https://fathomless-tor-48342.herokuapp
.com/api/products
HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 49
Etag: W/"31-qQTiw4UKDcRv6/tsw8LTr6UG7bI"
Set-Cookie: connect.sid=s%3ABhvz53Y79pZH66zIBzAzwcAprXEK8bqi.1hK70%2FfzN08KLproE
0FJBxncFuEs7vOTz0FuE644fbM; Path=/; HttpOnly
Date: Fri, 16 Nov 2018 03:37:23 GMT
Via: 1.1 vegur
```

→ result response:

[{"cart_id":999,"item_id":1,"item_quantity":299}]

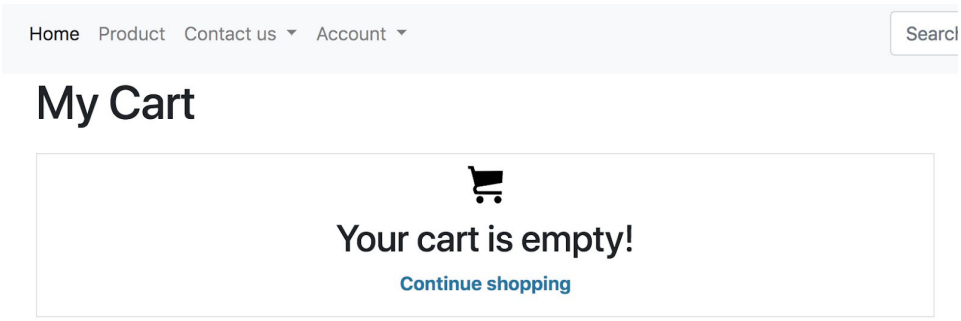
- **DELETE (api/clear):**

  $curl -X DELETE https://fathomless-tor-48342.herokuapp.com/api/clear

  →response:

```
yangjunande-MacBook:~ andy-yang$ curl -X DELETE https://fathomless-tor-48342.her
okuapp.com/api/clear
[]yangjunande-MacBook:~ andy-yang$ curli  https://fathomless-tor-48342.herokuapp
.com/api/products
HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 2
Etag: W/"2-l9Fw4VUO7kr8CvBlt4zaMCqXZ0w"
Set-Cookie: connect.sid=s%3AdoInZP6kSnA5TWDDFbP6Gjhg2Kzy7VI8.x2nr2fNJLjGRsCmf4LH
wbdQwPmoFHCWy0Z2qcTx4sOc; Path=/; HttpOnly
Date: Fri, 16 Nov 2018 10:10:12 GMT
Via: 1.1 vegur

[]yangjunande-MacBook:~ andy-yang$
```



And the database is all cleared!

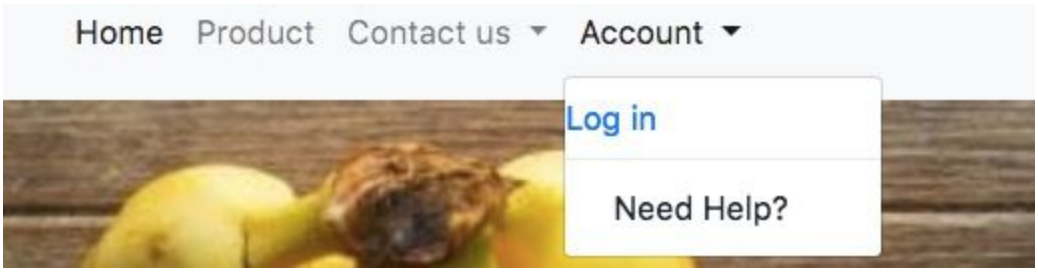
Authentication (Login/Sign up) Design (Josh)

Our application is currently using Okta to handle user authorisation and login capabilities. We experimented with passport.js also, but had issues with connecting to the local mongo database. The Okta API used allows us to control the complexity of a users password when they sign up for an account, as well as providing functionality to reset a users password if they forget it. Passport did seem like a more elegant solution, and it is unfortunate that we did not have enough time to properly implement it in our system.

Introduction of Web Application functions and how to use it (Zihui)

- LogIn and SignUp

  There is an Account drop-down menu at the head of the website. Click the Login link to jump to the Login/Signup page. If you are a registered user, you can log in directly with your account and password, and the web page will jump directly to the store page. If you are a new user, you need to use the email address on the registration page for new user registration. After clicking the registration button, the email will receive the activation account email sent by okta, click the activation button, and the page will also jump to the store interface.



- Browse and categorize

  When you log in to the store page, the store home page can browse the products for sale,

and at the top of the product page, there is a line of filter bars to filter out the products that match the attribute according to the product attributes. In addition, the sidebar can also display existing products according to the classification method of size, color and flavour.

- Add to cart and item page

  Click the Add To Cart button to pop up a dialog box where you can select the number of items to add to the shopping cart, then click the confirm button, and the item will be successfully added to the shopping cart.
  If you want to know the details of an item, double-click on the item name to go to the item page for that item. The item page contains specific information about the item, such as price, inventory, specific parameters, and detailed photos.

- View my cart and cart page

  When you log in, the store page will display a “View My Cart” button, click this button to go to the shopping cart page. There will be a list in the shopping cart page containing the previously placed items and the total price. Each different item is a line, and there is a delete button at the end of each line to remove the item from the shopping cart. If you are sure to purchase the items in the cart, click the "check out" button to go to the "checkout" page.

- Checkout page and Payment success page
  On the settlement page, customers are required to fill in their own mailing address, choose the delivery method and payment method. The floating box in the upper right corner shows the price of the order, the postage and the total. The lower right button supports returning to the shopping cart to modify the order information, if confirmed. Click the checkout button to pay.
  After the payment is successful, it will jump to the payment success page, at the same time, the shopping cart list is cleared. Click the "Back to Home" button to return to the product interface for the next purchase.