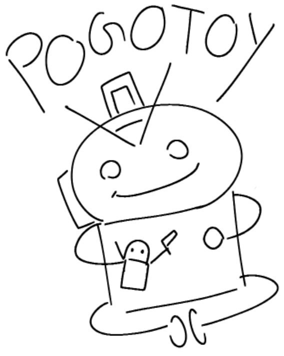
ITCS212 Web Programming

Project Phase 2

PogoToy



Submitted By

Patsathorn Chadbanterng	Reg no. 6588012
Arus Thienmee	Reg no. 6588108
Thanathorn Thongsuk	Reg no. 6588109
Panya Trakoolgerntong	Reg no. 6588110

Section 3 Group 1

Faculty of ICT, Mahidol University

.

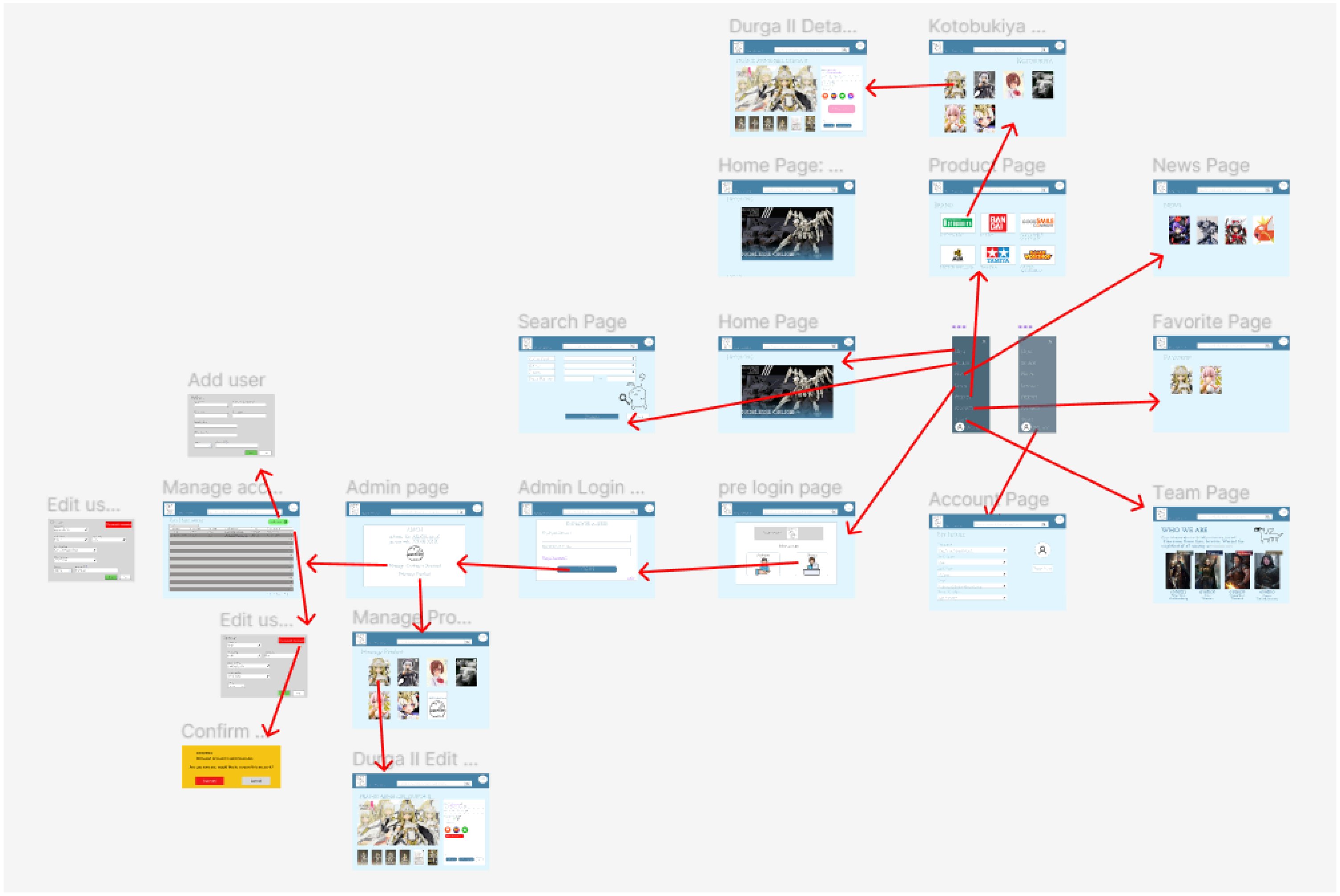
Project Description

The business domain focused on this project is a toy store. The business plan is to have a storefront and online store on Shopee. This toy store will also have a website as a web catalog for available products of the store. In addition, customers can register an account to use it to mark their favorite product, so that they will be able to keep in touch with the product availability, though the product availability can be viewed without an account.

This site takes inspiration from https://www.gunplavillage.com/. As seen on the referred website, the website should be integrated with a payment and delivery feature, which will be added to this business's website for possible business expansion.

Wireframe

This is the wireframe of the website:

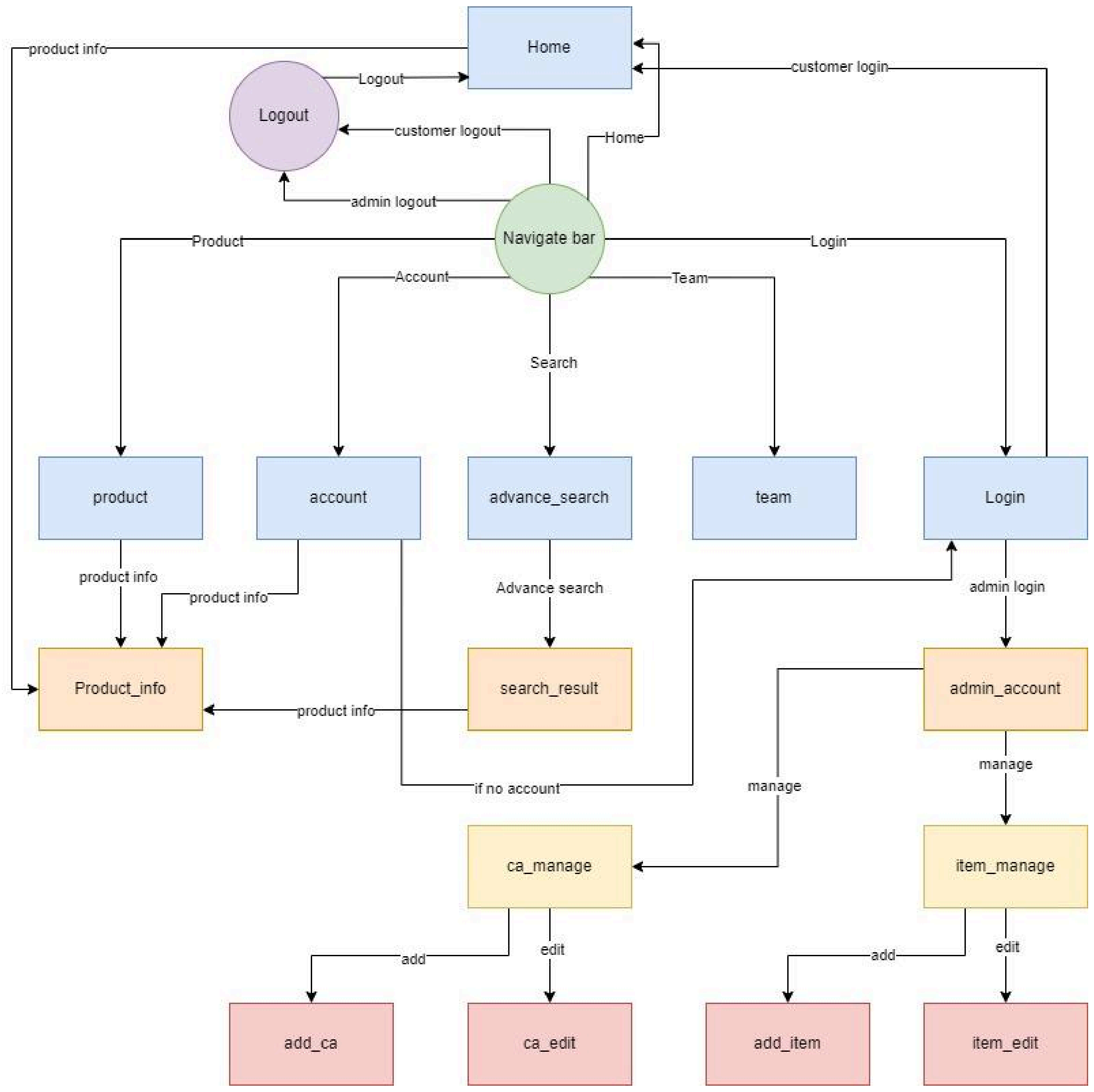


The prototype wireframe can also be viewed in Figma: https://www.figma.com/file/X8hfRCPODiUzxJQXZgJlBZ/ArusMonnEarthSiaPOG?type=design&node-id=500%3A908&mode=design&t=YSfcNBVNAl8e9zW9-1

Overview

From the finished user interface prototype designed in Figma, attempts of implementation happen to adjust the design to fit our frontend developer's ability. The database is also reimplemented to clean up the unnecessary and add ease of usage in query calling, which will lessen the backend's perplexity

Navigation diagram of the website (Diagram):



Detail of web app and code (frontend)

(noted that the frontend server will be mentioned as Client)

<Style>:

Overall cascade design will be done using the Bootstrap framework for the sake of responsiveness of the webpage. However, in some cases, the style from the framework will be overwritten by adding style attributes in the specific tags or using our own CSS. In the early stage of the workflow, implementing outer CSS for the login page and sign-in page since both pages use the same style, causes those pages not to be flexible, regardless of the functionality that is still functional. The DOM in this website will mostly be manipulated directly through innerHTML.

<Script type = "javascript">

- **nav.js**

As can be seen in the navigation diagram, the main navigation bar will be a terminal that connects everything in place, so separating it into a separate Javascript file is the solution to reduce code redundancy. By implementing the Bootstrap class "offcanvas", the navigation bar will be able to disguise as a floating button on the top-right of the page. Clicking that button will expand the navigation bar from the right. The string created in the script will finally be appended into the document as an innerHTML of the tag "navInit."

- **product_card.js**

Using Bootstrap class "card" to create a box showing a thumbnail picture of the product, the name, the price, and the link redirecting to the product page. Function createCol( ) will receive the product's data and then return a string of HTML scripts that will be put in another function called cardInit( ). The function cardInit( ) will find the document tag with an ID "productGen" and put the script string generated by createCol( ) into the innerHTML of that tag. The column generated from these functions also utilizes the Bootstrap grid system. The grid system will be created by using the Bootstrap class “row” on a tag and then putting tags with class “col” which stands for column inside the row, then the space occupied by each column in each row will be determined by a number behind col such as “col-4.”

Since the Bootstrap template column has 12 columns available per 1 row, the number following the col- will determine how many template columns the defined column occupies. For example, if it is “col-4”, that tag will use up one-third of the row (basic mathematics 4/12 = 1/**3**). Despite that, our code uses this:

```
<div class="col-lg-3 col-6 col-xl-3">
```

It means that on a large screen(col-lg width >= 992px) and larger screen (col-xl width >= 1200px), it will be displayed as 4 cards per row (3/12 = 1/**4**), and on the smallest setting (col- the default and have no maximum width set), it will display 2 cards per row (6/12 = 1/**2**). The row of the product card also applies “g-5” which means gutter with size 5. A gutter, according to bootstrap.com, is the padding between columns, used to align content in the grid system. The size of spacing can be looked up according to the given code:

(source: https://getbootstrap.com/docs/5.0/layout/gutters/)

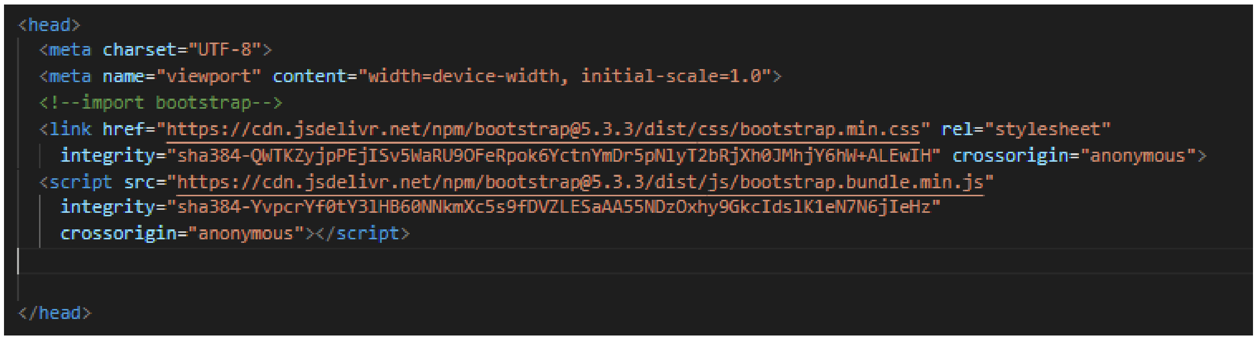
```
$grid-gutter-width: 1.5rem;
$gutters: (
  0: 0,
  1: $spacer * .25,
  2: $spacer * .5,
  3: $spacer,
  4: $spacer * 1.5,
  5: $spacer * 3,
);
```

- **searchbar.js**

Using the same technique of putting script strings from Javascript to every HTML file by putting it as an innerHTML to the element with ID = “search”. The search bar will be used as a simple search option where it finds a product whose name matches an input query. Design-wise, Bootstrap class “input-group” and “form-control” are applied to the element to beautify the search bar with a sense of minimalism.

<Html>:

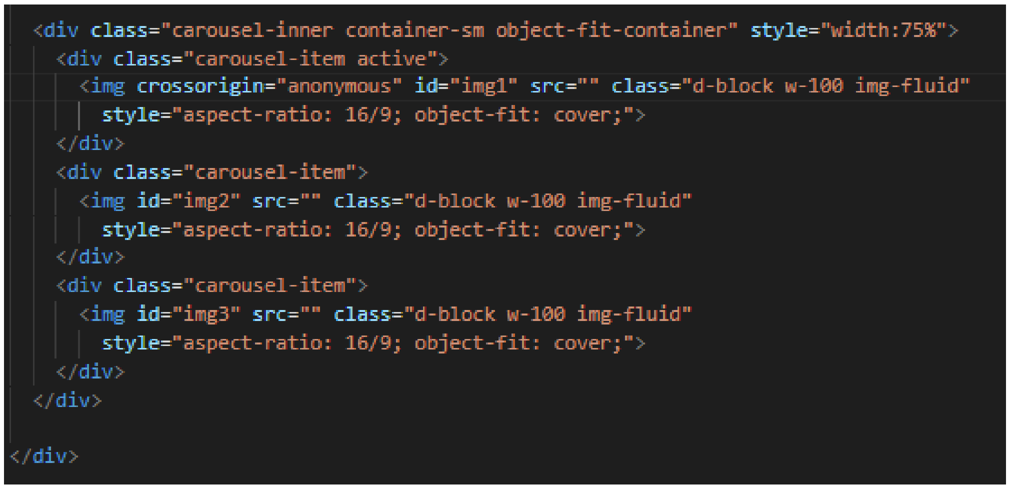
Since every page will hardly rely on the Bootstrap 5 framework, every page header will be included with CDN that will load the framework to the page:



```html
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!--import bootstrap-->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
    crossorigin="anonymous"></script>


</head>
```

- **Home.html**

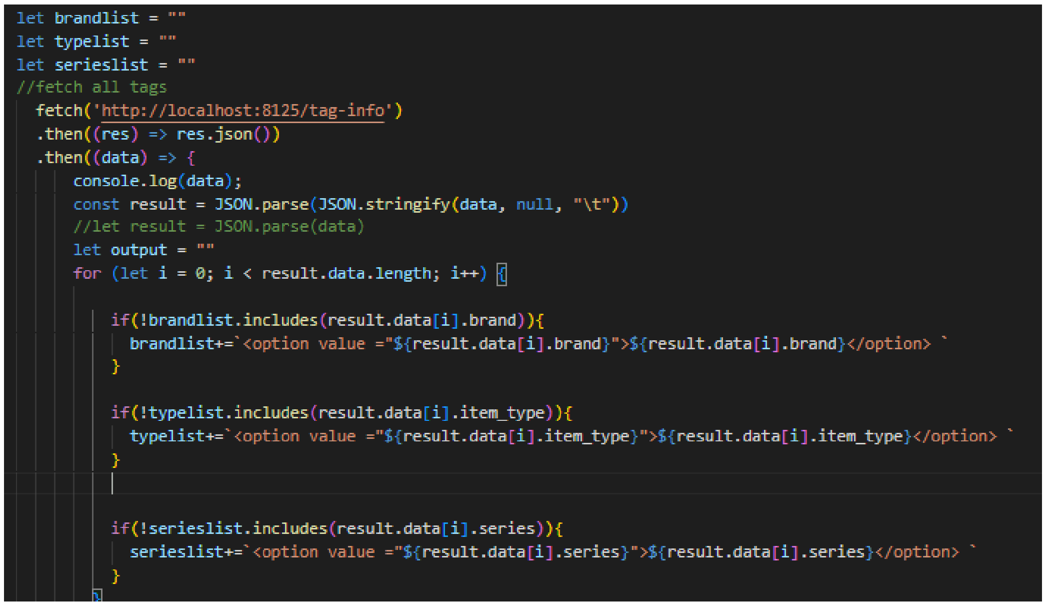
Implementing the Bootstrap class “carousel” and anything related to it, based on the official Bootstrap site tutorial, resulted in an automatic slideshow of the top 3 most recent products at times. Below the slideshow, the top 4 recent product cards will be shown.



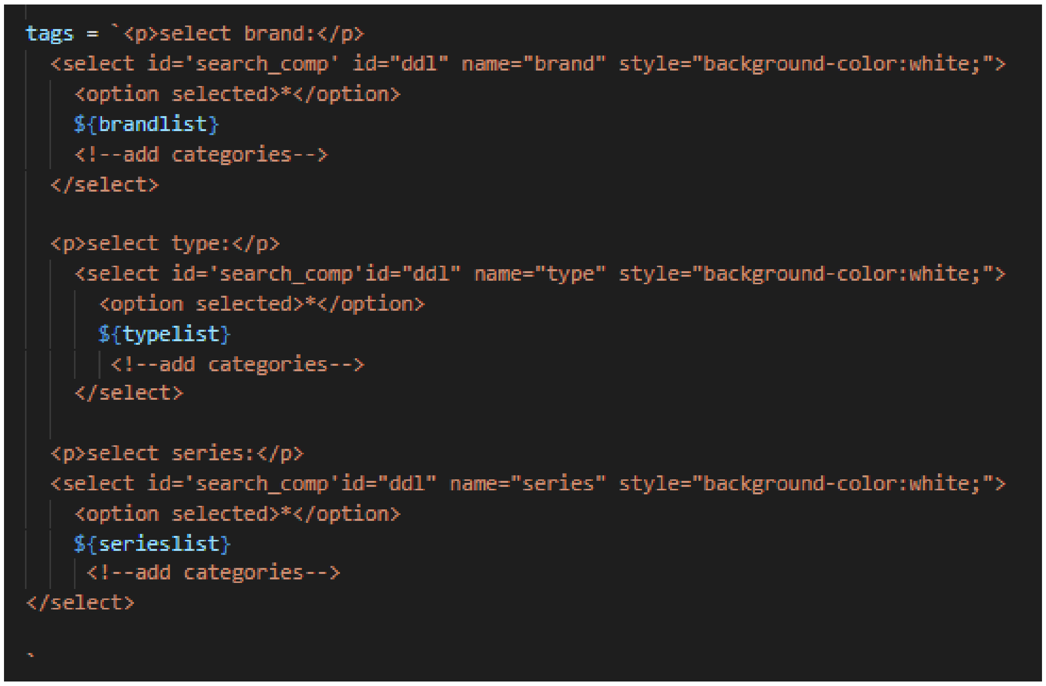
```html
  <div class="carousel-inner container-sm object-fit-container" style="width:75%">
    <div class="carousel-item active">
      <img crossorigin="anonymous" id="img1" src="" class="d-block w-100 img-fluid"
        style="aspect-ratio: 16/9; object-fit: cover;">
    </div>
    <div class="carousel-item">
      <img id="img2" src="" class="d-block w-100 img-fluid"
        style="aspect-ratio: 16/9; object-fit: cover;">
    </div>
    <div class="carousel-item">
      <img id="img3" src="" class="d-block w-100 img-fluid"
        style="aspect-ratio: 16/9; object-fit: cover;">
    </div>
  </div>

</div>
```

- **advance_search.html**

The advanced search can use up to 3 search criteria, namely brand, product type, and product series. Every possible tag from every criterion will be distinctly fetched from the database. After searching, the query sent will take a parameter of the applied tag to fetch a product according to the searching prompt and display using cardInit( ).



```
let brandlist = ""
let typelist = ""
let serieslist = ""
//fetch all tags
  fetch('http://localhost:8125/tag-info')
  .then((res) => res.json())
  .then((data) => {
      console.log(data);
      const result = JSON.parse(JSON.stringify(data, null, "\t"))
      //let result = JSON.parse(data)
      let output = ""
      for (let i = 0; i < result.data.length; i++) {

          if(!brandlist.includes(result.data[i].brand)){
            brandlist+=`<option value ="${result.data[i].brand}">${result.data[i].brand}</option> `
          }

          if(!typelist.includes(result.data[i].item_type)){
            typelist+=`<option value ="${result.data[i].item_type}">${result.data[i].item_type}</option> `
          }


          if(!serieslist.includes(result.data[i].series)){
            serieslist+=`<option value ="${result.data[i].series}">${result.data[i].series}</option> `
          }
        }
```



```
tags = `<p>select brand:</p>
  <select id='search_comp' id="ddl" name="brand" style="background-color:white;">
    <option selected>*</option>
    ${brandlist}
    <!--add categories-->
  </select>

  <p>select type:</p>
    <select id='search_comp'id="ddl" name="type" style="background-color:white;">
      <option selected>*</option>
      ${typelist}
       <!--add categories-->
    </select>

  <p>select series:</p>
  <select id='search_comp'id="ddl" name="series" style="background-color:white;">
    <option selected>*</option>
    ${serieslist}
     <!--add categories-->
</select>

`
```

- **Login_page.html and Signup_page.html**

In both pages, we have a main container to cover all of the content that is mainly focused on using input_box to allow users to input their email and password for login to our website and we use CSS or style to edit the web page layout.

- **product.html/ search_result.html**

Start by fetching all product possibles, users will see every product card generated by "product_card.js". Product information will be fetched and the loop will run through those pieces of information to extract them as a parameter for createCol( ) in order to call cardInit( ). After users click on a button that links to the specific product page, the website will redirect users to a product_detail

- **product_info.html**

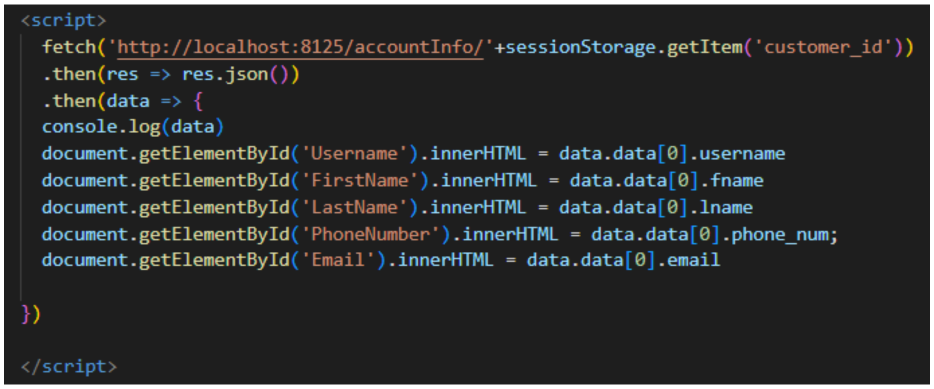
A specific product page will display information about the selected product, organized by Bootstrap class "list-group-horizontal". The product image will also be put into a "carousel" for a stylish slideshow display. The product content will be displayed in a Bootstrap "card".

- **account.html**

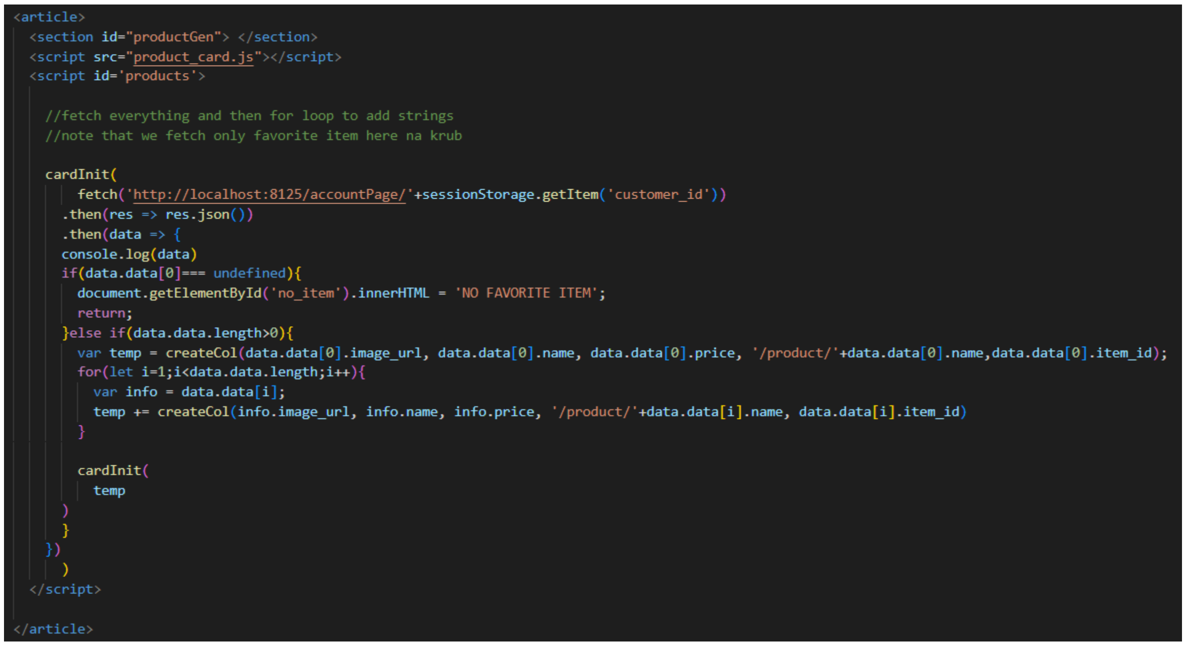
The account page will display information about the user and favorite products that the user adds by themselves. The data of the favorited item list will be fetched from the database to show user information (username, first-last name, phone number, and email) and favorite products. The page also displays customer information using the Bootstrap class "list-group"

- **admin_account.html**

When the user logs in as an admin, the admin will be redirected to this page. The page consists of 2 hyperlinks taking admins to the account managing and item managing page.



```
<script>
  fetch('http://localhost:8125/accountInfo/'+sessionStorage.getItem('customer_id'))
  .then(res => res.json())
  .then(data => {
  console.log(data)
  document.getElementById('Username').innerHTML = data.data[0].username
  document.getElementById('FirstName').innerHTML = data.data[0].fname
  document.getElementById('LastName').innerHTML = data.data[0].lname
  document.getElementById('PhoneNumber').innerHTML = data.data[0].phone_num;
  document.getElementById('Email').innerHTML = data.data[0].email

})

</script>
```



```
<article>
  <section id="productGen"> </section>
  <script src="product_card.js"></script>
  <script id='products'>

    //fetch everything and then for loop to add strings
    //note that we fetch only favorite item here na krub

    cardInit(
        fetch('http://localhost:8125/accountPage/'+sessionStorage.getItem('customer_id'))
      .then(res => res.json())
      .then(data => {
      console.log(data)
      if(data.data[0]=== undefined){
        document.getElementById('no_item').innerHTML = 'NO FAVORITE ITEM';
        return;
      }else if(data.data.length>0){
        var temp = createCol(data.data[0].image_url, data.data[0].name, data.data[0].price, '/product/'+data.data[0].name,data.data[0].item_id);
        for(let i=1;i<data.data.length;i++){
          var info = data.data[i];
          temp += createCol(info.image_url, info.name, info.price, '/product/'+data.data[i].name, data.data[i].item_id)
        }

        cardInit(
          temp
      )
      }
    })
      )
  </script>

</article>
```

- **ca_manage.html/ item_manage.html**

Show every customer account by using fetch to receive data for display information from the database in the table. Admin can access the ca_edit.html by clicking on the button “+ add new customer here” button and access the add_ca.html by clicking on “click here to manage”.

- **ca_edit.html/ item_edit.html**

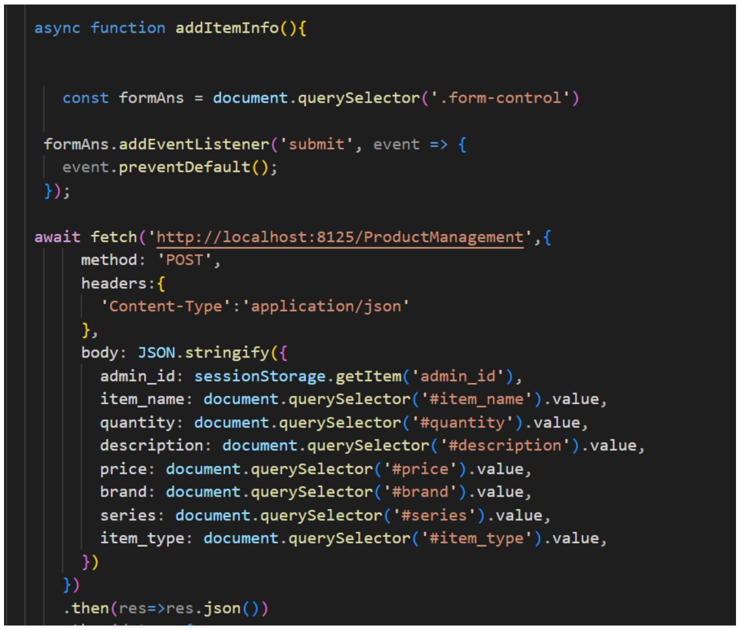
Admin can update or edit customer account details by inserting on <input> block and clicking “confirm change” to update new details. However, the edited information in item_edit does not cover the image part.

- **add_ca.html**

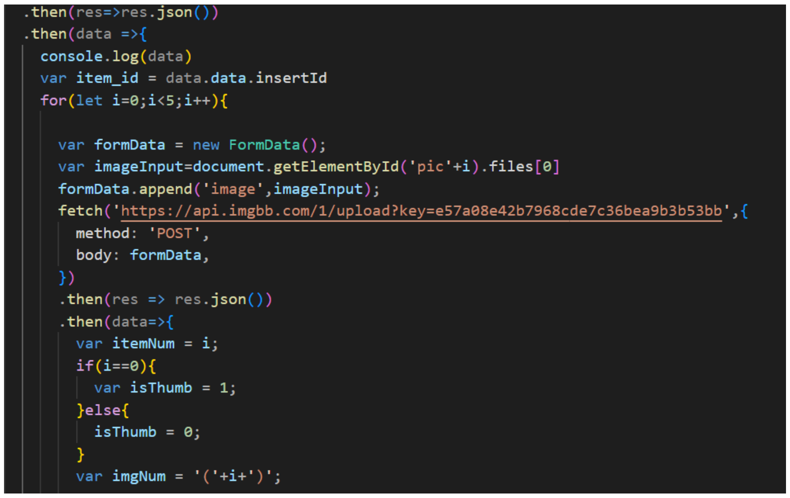
This page has only purpose of adding an account to the database. By utilizing the function “addAccountInfo”, the data will extracted from the input form and kept as a JSON body, ready to be sent to POST in the backend server.

- **add_item.html**

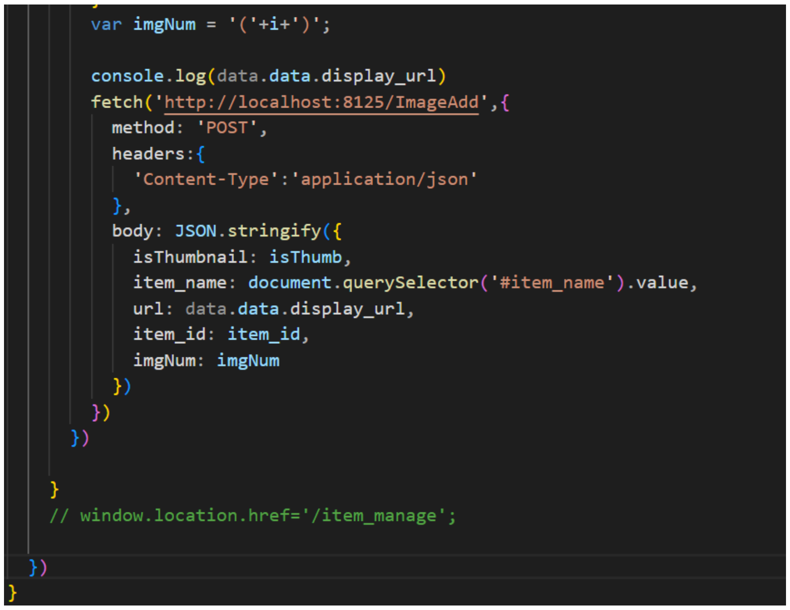
Admin can insert information about the new product on <input>. After inserting all information, the admin needs to click add for insert. We use fetch to send the new data to the query. However, this operation is special. With the implementation of an API from imgbb.com, the admin can upload the picture as a URL.



```
async function addItemInfo(){


   const formAns = document.querySelector('.form-control')

 formAns.addEventListener('submit', event => {
   event.preventDefault();
 });

await fetch('http://localhost:8125/ProductManagement',{
     method: 'POST',
     headers:{
       'Content-Type':'application/json'
     },
     body: JSON.stringify({
       admin_id: sessionStorage.getItem('admin_id'),
       item_name: document.querySelector('#item_name').value,
       quantity: document.querySelector('#quantity').value,
       description: document.querySelector('#description').value,
       price: document.querySelector('#price').value,
       brand: document.querySelector('#brand').value,
       series: document.querySelector('#series').value,
       item_type: document.querySelector('#item_type').value,
     })
   })
   .then(res=>res.json())
```



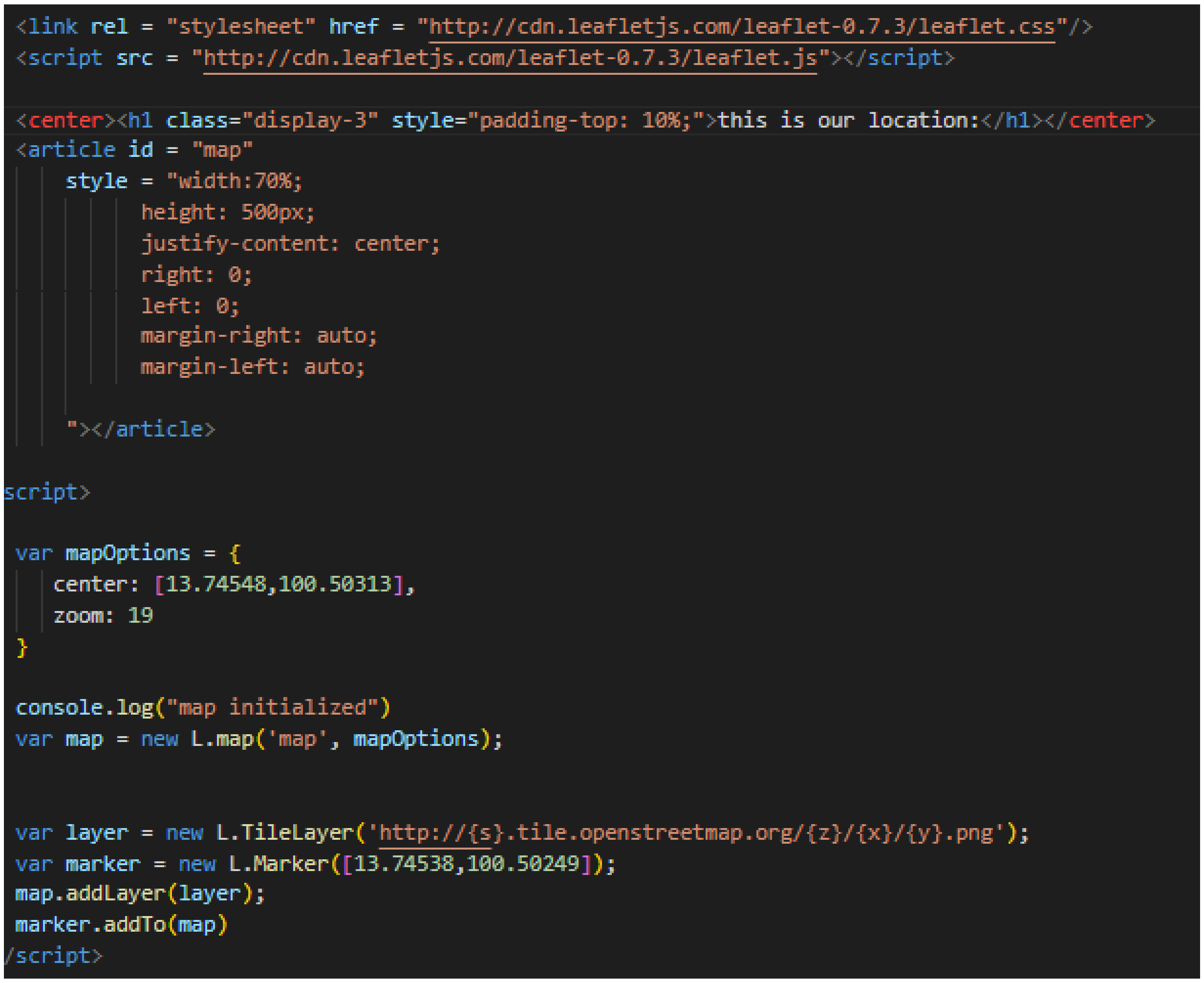
```
.then(res=>res.json())
.then(data =>{
  console.log(data)
  var item_id = data.data.insertId
  for(let i=0;i<5;i++){

    var formData = new FormData();
    var imageInput=document.getElementById('pic'+i).files[0]
    formData.append('image',imageInput);
    fetch('https://api.imgbb.com/1/upload?key=e57a08e42b7968cde7c36bea9b3b53bb',{
      method: 'POST',
      body: formData,
    })
    .then(res => res.json())
    .then(data=>{
      var itemNum = i;
      if(i==0){
        var isThumb = 1;
      }else{
        isThumb = 0;
      }
      var imgNum = '('+i+')';
```



```
      var imgNum = '('+i+')';

      console.log(data.data.display_url)
      fetch('http://localhost:8125/ImageAdd',{
        method: 'POST',
        headers:{
          'Content-Type':'application/json'
        },
        body: JSON.stringify({
          isThumbnail: isThumb,
          item_name: document.querySelector('#item_name').value,
          url: data.data.display_url,
          item_id: item_id,
          imgNum: imgNum
        })
      })
    })

  }
  // window.location.href='/item_manage';

})
}
```

- **Team.html**

Team page implementing Bootstrap class “card” and similarly applying grid system to the result of cardInit( ) function. Each card will show a gorgeous picture of the comrades who fought along each other’s side since the start of the semester. At the bottom of the page, there is also a map implemented using “Leaflet” extracting data from Open Street Map because one of the group members, not specified as the one named Arus, thought that it was using an API, but it was just a built-in library which displays a map, but it works as additional information to the team page so it will be kept like that.



```
<link rel = "stylesheet" href = "http://cdn.leafletjs.com/leaflet-0.7.3/leaflet.css"/>
<script src = "http://cdn.leafletjs.com/leaflet-0.7.3/leaflet.js"></script>

<center><h1 class="display-3" style="padding-top: 10%;">this is our location:</h1></center>
<article id = "map"
    style = "width:70%;
        height: 500px;
        justify-content: center;
        right: 0;
        left: 0;
        margin-right: auto;
        margin-left: auto;

    "></article>

script>

var mapOptions = {
    center: [13.74548,100.50313],
    zoom: 19
}

console.log("map initialized")
var map = new L.map('map', mapOptions);


var layer = new L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png');
var marker = new L.Marker([13.74538,100.50249]);
map.addLayer(layer);
marker.addTo(map)
/script>
```

- **404_page.html**

This page will feature a big plain text written “404 Not Found” and a hyperlink to the homepage.

Database implementation

For our database implementation, we changed the ERD diagram to be a table in MySQL.

Table

- LOGIN_INFO
    - login_id INT AUTO_INCREMENT PRIMARY KEY
    - username VARCHAR(30) [username account]
    - password VARCHAR(30) [password account]
    - email VARCHAR(30) [customer and admin email]
    - acc_time DATETIME [login time]
- Admin
    - admin_id INT AUTO_INCREMENT PRIMARY KEY
    - login_id INT [admin login_id from LOGIN_INFO table]
    - fname VARCHAR(30) [admin first name login information]
    - lname VARCHAR(30) [admin last name login information]
    - phone_num INT [admin phone number login information]
    - FOREIGN KEY (login_id) REFERENCES LOGIN_INFO(login_id)
- Customer_Account
    - customer_id INT AUTO_INCREMENT PRIMARY KEY
    - login_id INT [customer login_id from LOGIN_INFO table]
    - username VARCHAR(30) [customer username account]
    - fname VARCHAR(30) [customer first name login information]
    - lname VARCHAR(30) [customer last name login information]
    - email VARCHAR(30) [customer email]
    - password VARCHAR(30) [customer password]
    - phone_num INT [customer phone number]
    - pfp_link VARCHAR(255) [customer profile link]
    - FOREIGN KEY (login_id) REFERENCES LOGIN_INFO(login_id)

- ITEM
  - item_id INT AUTO_INCREMENT PRIMARY KEY
  - admin_id INT   [admin_id who editing]
  - name VARCHAR(255) [item name]
  - status BOOLEAN [status of item]
  - quantity INT [amount of item]
  - description VARCHAR(255) [item description]
  - price FLOAT [item price]
  - brand VARCHAR(255) [brand of item]
  - series VARCHAR(255)   [series of item]
  - item_type VARCHAR(255) [item type]
  - release_date VARCHAR(255) [releasing date of item]
  - FOREIGN KEY (admin_id) REFERENCES Admin(admin_id)
    [using admin_id from Admin table for edit item]
- Tag
  - tag_id INT AUTO_INCREMENT PRIMARY KEY   [id_tag for item]
  - tagname VARCHAR(255) [name tag]
  - tagtype VARCHAR(255) [tag of itemtype]

- Item_Tag
  - item_id INT
  - tag_id INT
  - update_date DATETIME   [time of item update by admin]
  - PRIMARY KEY (item_id, tag_id)
  - FOREIGN KEY (item_id) REFERENCES ITEM(item_id)
  - FOREIGN KEY (tag_id) REFERENCES Tag(tag_id)

- Favorite_Item
    - item_id INT
    - customer_id INT
    - date_added DATETIME   [date of item that add to favorite from customer]
    - PRIMARY KEY (item_id, customer_id)
    - FOREIGN KEY (item_id) REFERENCES ITEM(item_id)
    - FOREIGN KEY (customer_id) REFERENCES Customer_Account(customer_id)

- Image
    - image_id INT AUTO_INCREMENT PRIMARY KEY
    - isTumbnail BOOLEAN []
    - image_name VARCHAR(255) [image name]
    - image_url VARCHAR(255)   [image link]
    - item_id INT   [id item from ITEM table]
    - FOREIGN KEY (item_id) REFERENCES ITEM(item_id)

Detail of web service (backend)

Client-server Model

The Client-server Model is described to be a distributed application framework dividing tasks between servers and clients, which either reside in the same system or communicate through a computer network or the Internet. The client relies on sending a request to another program to access a service made available by a server. The server runs one or more programs that share resources with and distribute work among clients. The category of client-server we use is that of two-tier architecture, where we have a client and a server which connect through a protocol that links the two tiers. The Client-server uses express, path package to redirect the request to different HTML pages, JSON web token package to authenticate the login session, and dotenv package to call upon the access token string. The port used for the client is 8025.

Router redirect

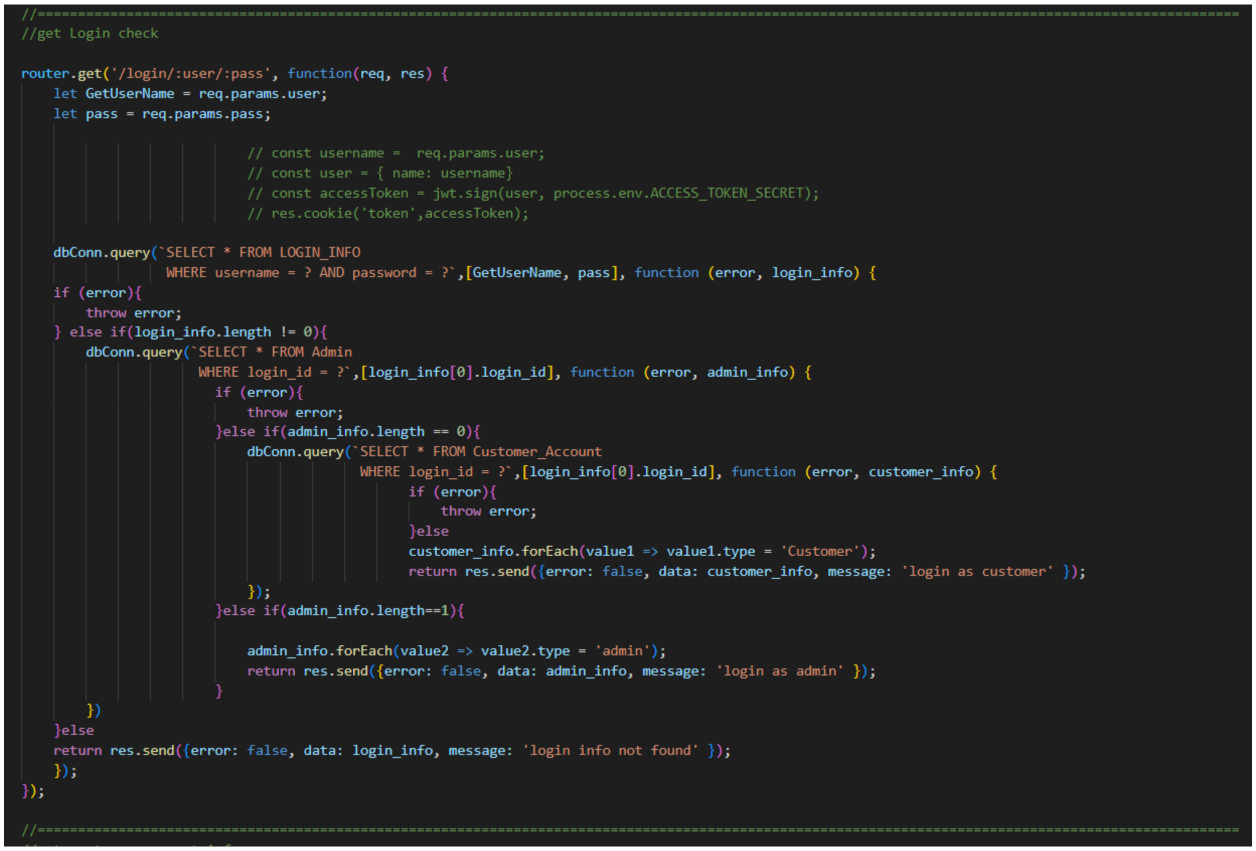
```
    router.get('/', (req,res) =>{

res.status(200).sendFile(path.join(`${__dirname}/src/html/home.html`))

    })
```

This router is the path that the user will be sent to when starting up the application directing the user to the home page.

```
router.get('/:id', (req,res) =>{

    let pageName = req.params.id;

    console.log("Request at "+ req.url);


res.status(200).sendFile(path.join(`${__dirname}/src/html/`+pageName+`.htm
l`))

})
```

The router takes id as parameter and like the user to the html page based on the parameter string. On use the name will be the same as the html file name. Due to this, when sending the wrong parameter, the web will display

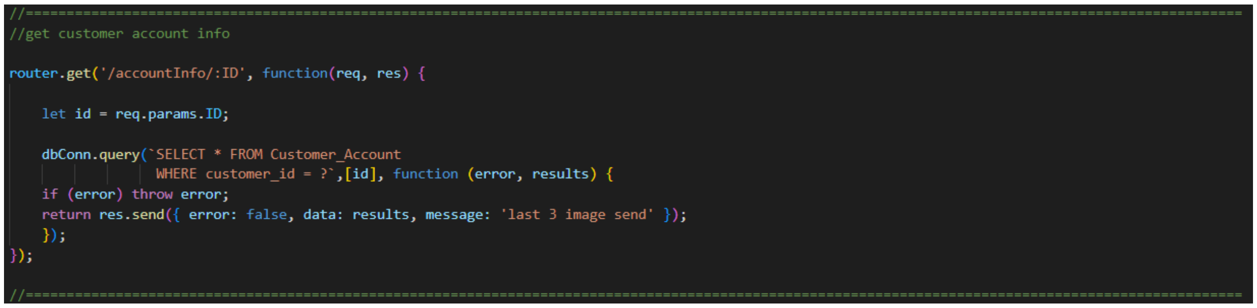
// Get Login Check



```
//==============================================================================================================================
//get Login check

router.get('/login/:user/:pass', function(req, res) {
    let GetUserName = req.params.user;
    let pass = req.params.pass;

                                // const username =  req.params.user;
                                // const user = { name: username}
                                // const accessToken = jwt.sign(user, process.env.ACCESS_TOKEN_SECRET);
                                // res.cookie('token',accessToken);

    dbConn.query(`SELECT * FROM LOGIN_INFO
                  WHERE username = ? AND password = ?`,[GetUserName, pass], function (error, login_info) {
    if (error){
        throw error;
    } else if(login_info.length != 0){
        dbConn.query(`SELECT * FROM Admin
                      WHERE login_id = ?`,[login_info[0].login_id], function (error, admin_info) {
                        if (error){
                            throw error;
                        }else if(admin_info.length == 0){
                            dbConn.query(`SELECT * FROM Customer_Account
                                          WHERE login_id = ?`,[login_info[0].login_id], function (error, customer_info) {
                                                if (error){
                                                    throw error;
                                                }else
                                                customer_info.forEach(value1 => value1.type = 'Customer');
                                                return res.send({error: false, data: customer_info, message: 'login as customer' });
                            });
                        }else if(admin_info.length==1){

                            admin_info.forEach(value2 => value2.type = 'admin');
                            return res.send({error: false, data: admin_info, message: 'login as admin' });
                        }
        })
    }else
    return res.send({error: false, data: login_info, message: 'login info not found' });
    });
});

//==============================================================================================================================
```

When a user login to our website, the back-end will check if the username and password combination exists in the database or not, then it will check the status of the user as a customer or admin. First we get a query from login_info and check if the length of login_info is not equal to 0, that means we have a customer or admin in our database. After that we will get a query from the Admin table by checking the length of the admin. If length of admin is not 0 that means our database has admin. But if it is not, we will get a query from customer_Account and respond to the data.

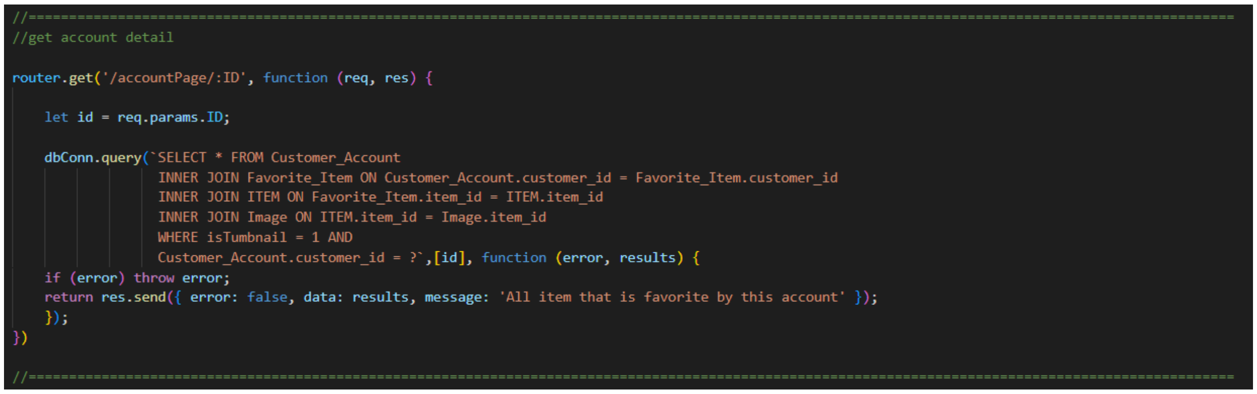
// Get Customer account info



```
//==========================================================================================================================
//get customer account info

router.get('/accountInfo/:ID', function(req, res) {

    let id = req.params.ID;

    dbConn.query(`SELECT * FROM Customer_Account
                  WHERE customer_id = ?`,[id], function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results, message: 'last 3 image send' });
    });
});

//==========================================================================================================================
```

This router is used when we need to update or edit that customer account. When we access the customer edit page, this router will get a query for the customer ID that we want to do and get every data.

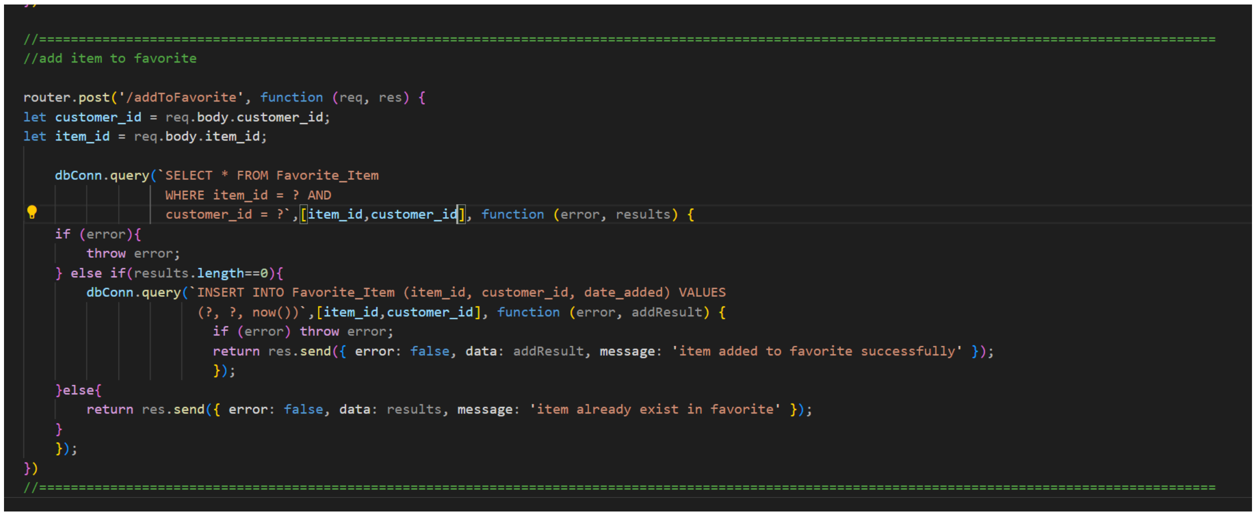
// Get Account Detail



```
//==========================================================================================================================
//get account detail

router.get('/accountPage/:ID', function (req, res) {

    let id = req.params.ID;

    dbConn.query(`SELECT * FROM Customer_Account
                  INNER JOIN Favorite_Item ON Customer_Account.customer_id = Favorite_Item.customer_id
                  INNER JOIN ITEM ON Favorite_Item.item_id = ITEM.item_id
                  INNER JOIN Image ON ITEM.item_id = Image.item_id
                  WHERE isTumbnail = 1 AND
                  Customer_Account.customer_id = ?`,[id], function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results, message: 'All item that is favorite by this account' });
    });
})

//==========================================================================================================================
```

This router is used when the user already login and this router will get the query to display the favorite product that user has already added to their favorite.

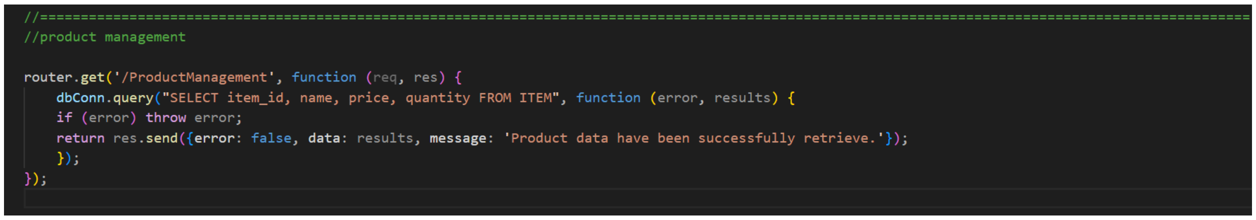
// Add Item to favorite



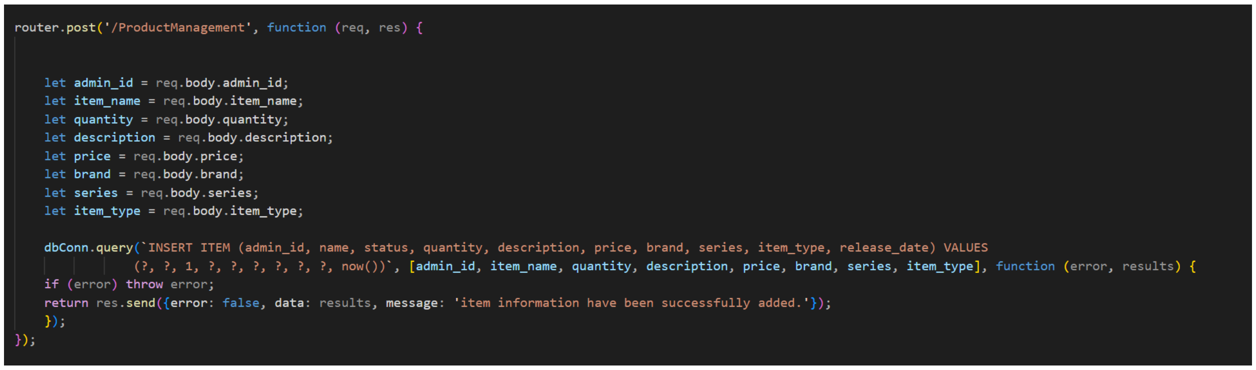
```
//==========================================================================================================================================
//add item to favorite

router.post('/addToFavorite', function (req, res) {
let customer_id = req.body.customer_id;
let item_id = req.body.item_id;

    dbConn.query(`SELECT * FROM Favorite_Item
                WHERE item_id = ? AND
                customer_id = ?`,[item_id,customer_id], function (error, results) {
    if (error){
        throw error;
    } else if(results.length==0){
        dbConn.query(`INSERT INTO Favorite_Item (item_id, customer_id, date_added) VALUES
                    (?, ?, now())`,[item_id,customer_id], function (error, addResult) {
                      if (error) throw error;
                      return res.send({ error: false, data: addResult, message: 'item added to favorite successfully' });
                      });
    }else{
        return res.send({ error: false, data: results, message: 'item already exist in favorite' });
    }
    });
})
//==========================================================================================================================================
```

Router for adding product to favorite. Before adding a product, we should check whether this account has already been added or not by checking the length of the result. If length is not equal to 0 that means this account already adds it. But if it is not the item will be added to the Favorite_Item table.

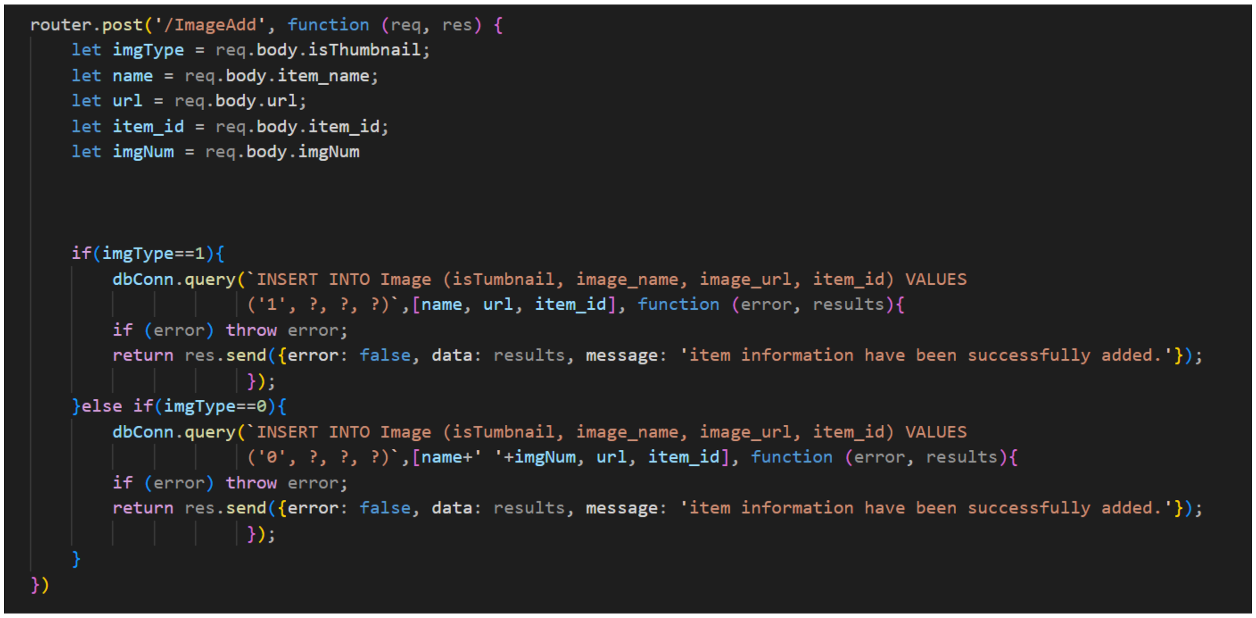
// Product Management



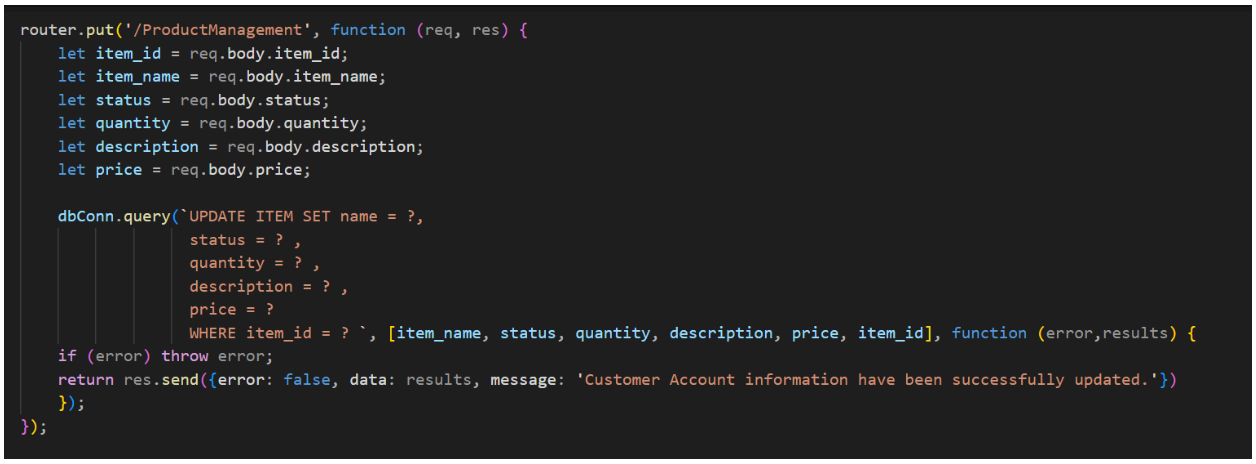
```
//===========================================================================================================================================
//product management

router.get('/ProductManagement', function (req, res) {
    dbConn.query("SELECT item_id, name, price, quantity FROM ITEM", function (error, results) {
    if (error) throw error;
    return res.send({error: false, data: results, message: 'Product data have been successfully retrieve.'});
    });
});
```



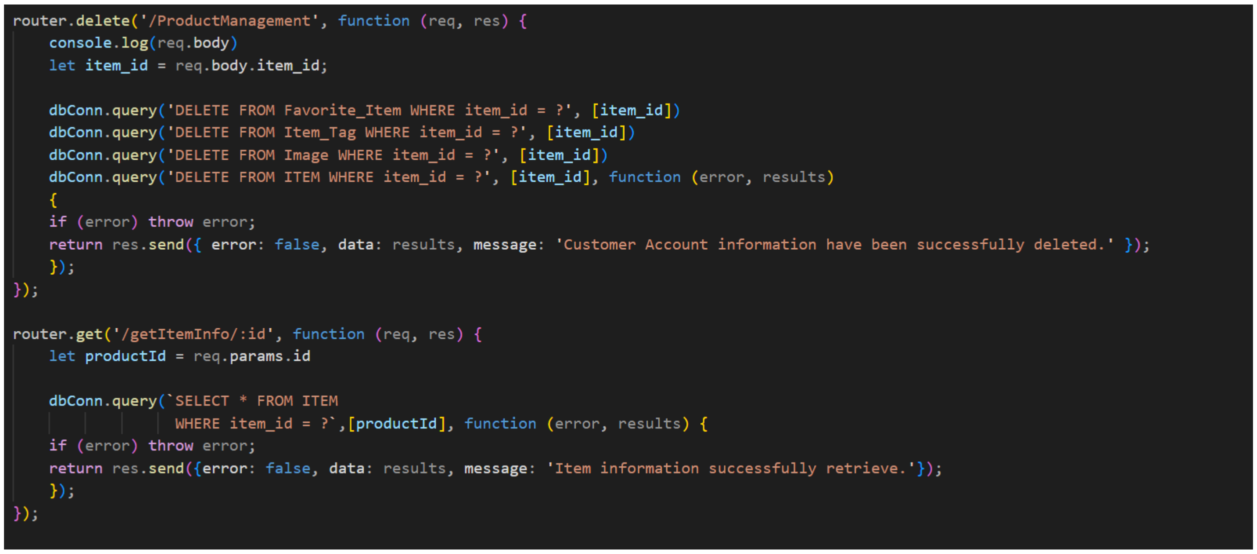
```
router.post('/ProductManagement', function (req, res) {


    let admin_id = req.body.admin_id;
    let item_name = req.body.item_name;
    let quantity = req.body.quantity;
    let description = req.body.description;
    let price = req.body.price;
    let brand = req.body.brand;
    let series = req.body.series;
    let item_type = req.body.item_type;

    dbConn.query(`INSERT ITEM (admin_id, name, status, quantity, description, price, brand, series, item_type, release_date) VALUES
                (?, ?, 1, ?, ?, ?, ?, ?, ?, now())`, [admin_id, item_name, quantity, description, price, brand, series, item_type], function (error, results) {
    if (error) throw error;
    return res.send({error: false, data: results, message: 'item information have been successfully added.'});
    });
});
```



```
router.post('/ImageAdd', function (req, res) {
    let imgType = req.body.isThumbnail;
    let name = req.body.item_name;
    let url = req.body.url;
    let item_id = req.body.item_id;
    let imgNum = req.body.imgNum



    if(imgType==1){
        dbConn.query(`INSERT INTO Image (isTumbnail, image_name, image_url, item_id) VALUES
                    ('1', ?, ?, ?)`,[name, url, item_id], function (error, results){
        if (error) throw error;
        return res.send({error: false, data: results, message: 'item information have been successfully added.'});
                    });
    }else if(imgType==0){
        dbConn.query(`INSERT INTO Image (isTumbnail, image_name, image_url, item_id) VALUES
                    ('0', ?, ?, ?)`,[name+' '+imgNum, url, item_id], function (error, results){
        if (error) throw error;
        return res.send({error: false, data: results, message: 'item information have been successfully added.'});
                    });
    }
})
```



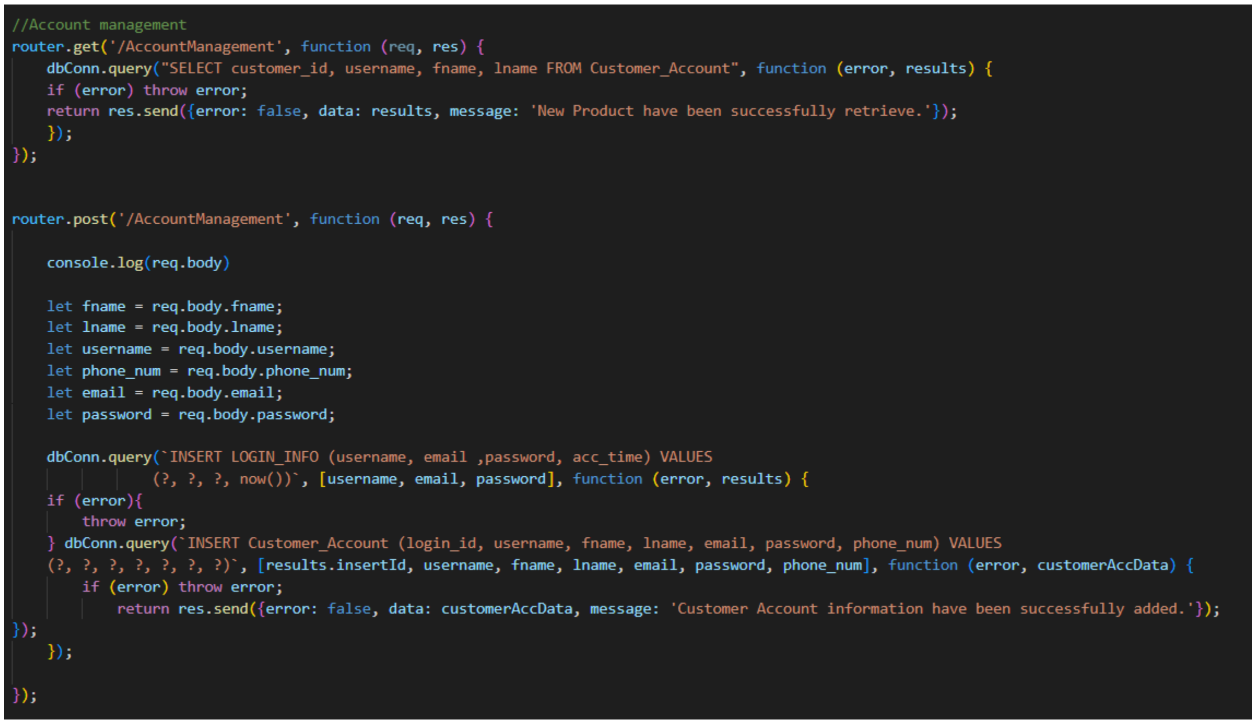
```
router.put('/ProductManagement', function (req, res) {
    let item_id = req.body.item_id;
    let item_name = req.body.item_name;
    let status = req.body.status;
    let quantity = req.body.quantity;
    let description = req.body.description;
    let price = req.body.price;

    dbConn.query(`UPDATE ITEM SET name = ?,
                status = ? ,
                quantity = ? ,
                description = ? ,
                price = ?
                WHERE item_id = ? `, [item_name, status, quantity, description, price, item_id], function (error,results) {
    if (error) throw error;
    return res.send({error: false, data: results, message: 'Customer Account information have been successfully updated.'})
    });
});
```



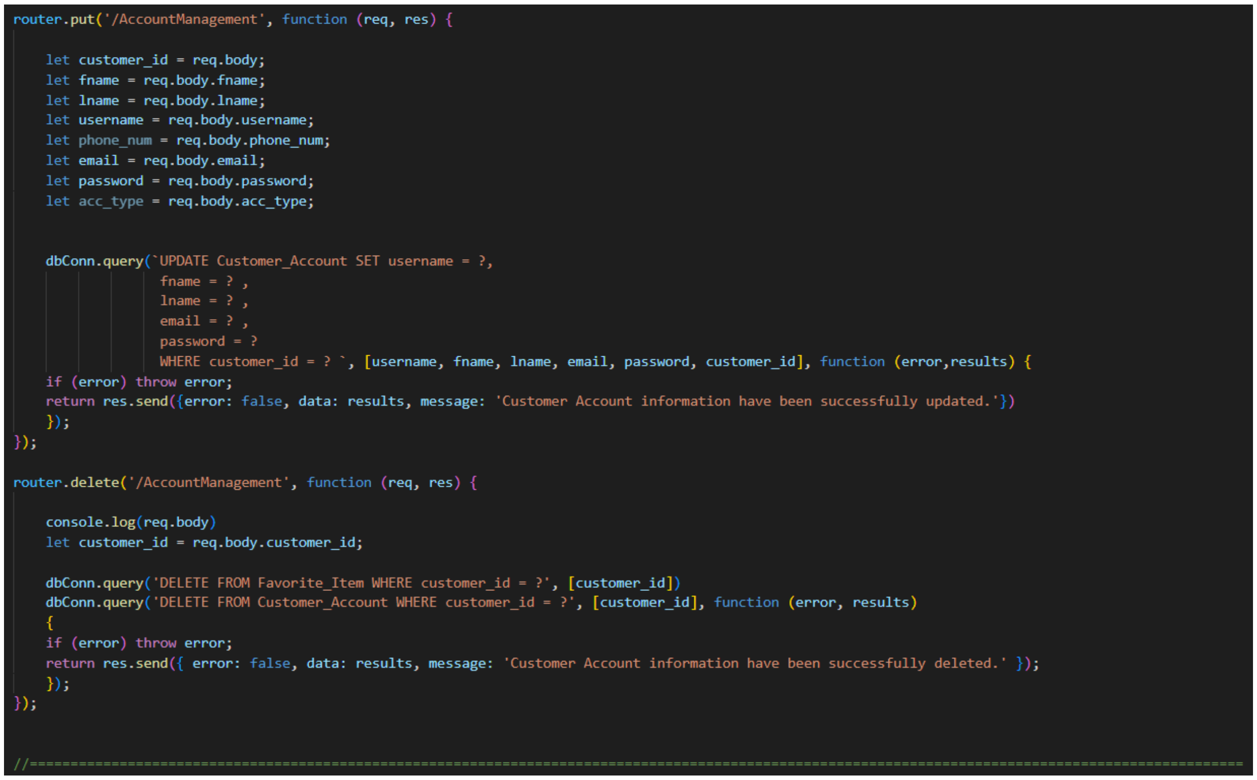
```
router.delete('/ProductManagement', function (req, res) {
    console.log(req.body)
    let item_id = req.body.item_id;

    dbConn.query('DELETE FROM Favorite_Item WHERE item_id = ?', [item_id])
    dbConn.query('DELETE FROM Item_Tag WHERE item_id = ?', [item_id])
    dbConn.query('DELETE FROM Image WHERE item_id = ?', [item_id])
    dbConn.query('DELETE FROM ITEM WHERE item_id = ?', [item_id], function (error, results)
    {
    if (error) throw error;
    return res.send({ error: false, data: results, message: 'Customer Account information have been successfully deleted.' });
    });
});

router.get('/getItemInfo/:id', function (req, res) {
    let productId = req.params.id

    dbConn.query(`SELECT * FROM ITEM
                  WHERE item_id = ?`,[productId], function (error, results) {
    if (error) throw error;
    return res.send({error: false, data: results, message: 'Item information successfully retrieve.'});
    });
});
```

This is the CRUD of the item management page. The create function is split into 2 parts, adding to the item table and adding to the image table. The update function is used in conjunction with the getItemInfo so by adding information into the update page, the admin won't have to input everything and only the part they would like to change. The delete feature requires removal from multiple tables due to foreign key restrictions.

// Account Management

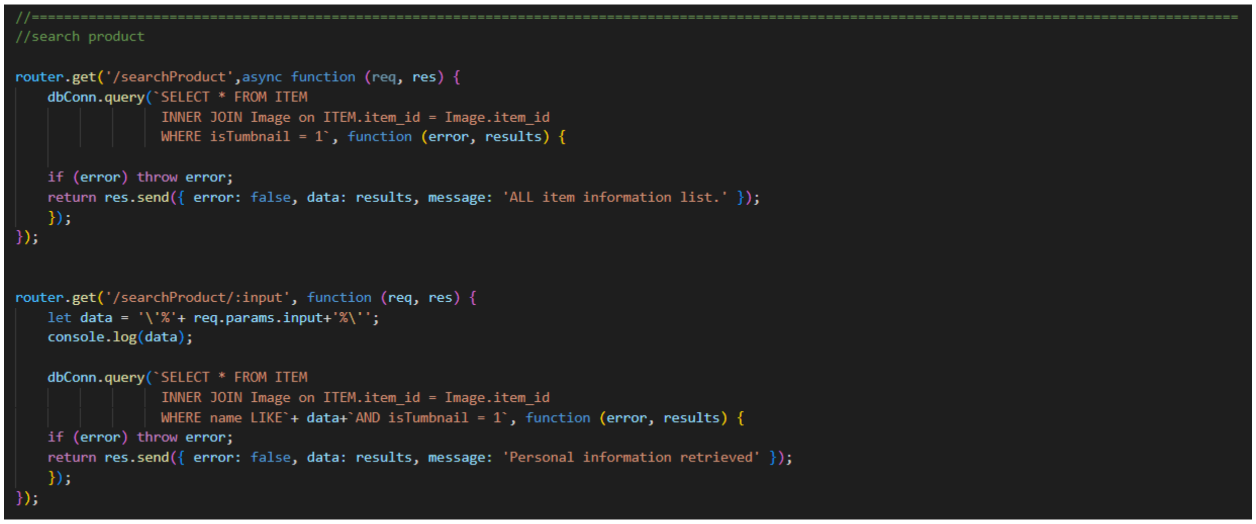


```
//Account management
router.get('/AccountManagement', function (req, res) {
    dbConn.query("SELECT customer_id, username, fname, lname FROM Customer_Account", function (error, results) {
    if (error) throw error;
    return res.send({error: false, data: results, message: 'New Product have been successfully retrieve.'});
    });
});


router.post('/AccountManagement', function (req, res) {

    console.log(req.body)

    let fname = req.body.fname;
    let lname = req.body.lname;
    let username = req.body.username;
    let phone_num = req.body.phone_num;
    let email = req.body.email;
    let password = req.body.password;

    dbConn.query(`INSERT LOGIN_INFO (username, email ,password, acc_time) VALUES
                (?, ?, ?, now())`, [username, email, password], function (error, results) {
    if (error){
        throw error;
    } dbConn.query(`INSERT Customer_Account (login_id, username, fname, lname, email, password, phone_num) VALUES
    (?, ?, ?, ?, ?, ?, ?)`, [results.insertId, username, fname, lname, email, password, phone_num], function (error, customerAccData) {
        if (error) throw error;
            return res.send({error: false, data: customerAccData, message: 'Customer Account information have been successfully added.'});
});
    });

});
```

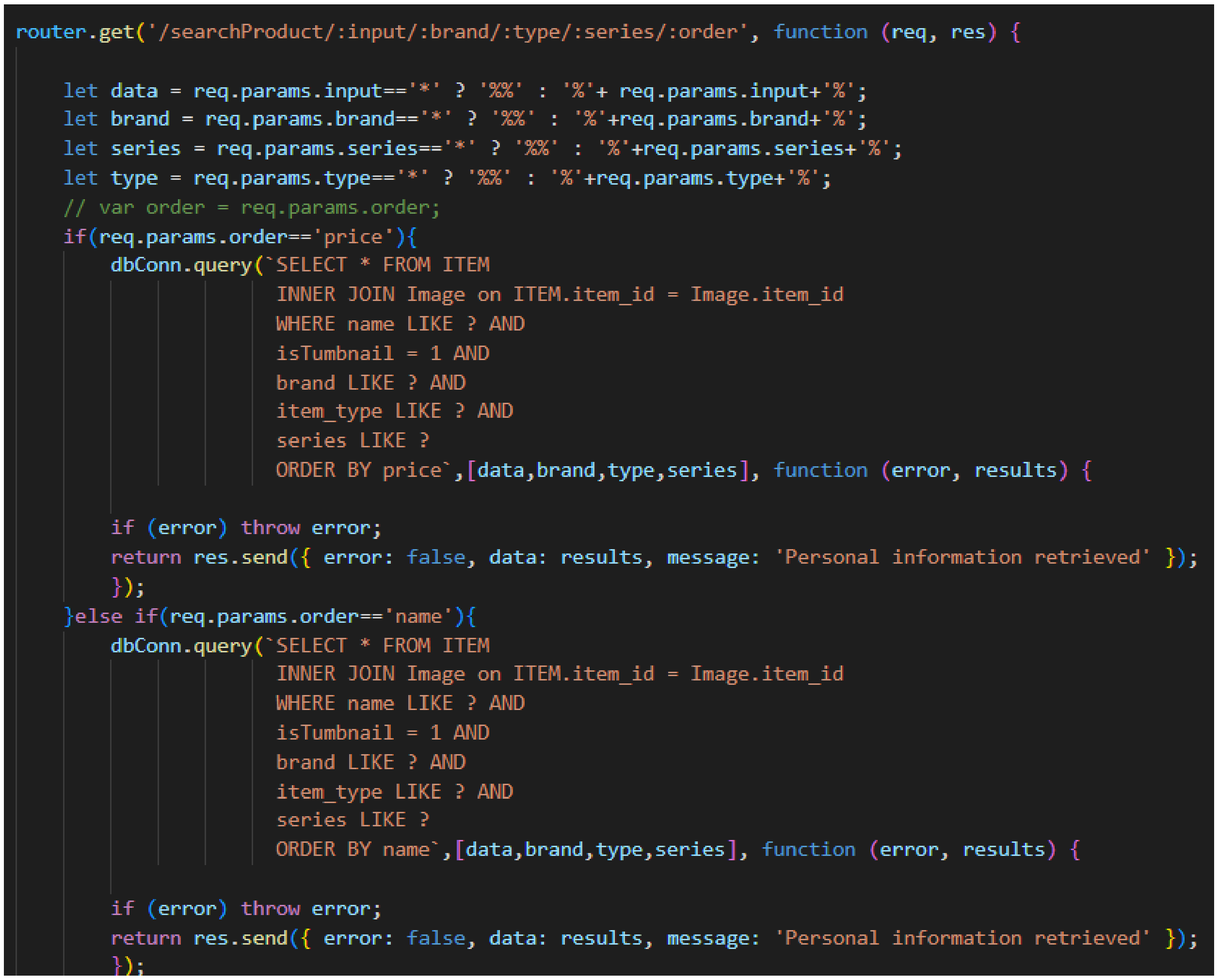


```
router.put('/AccountManagement', function (req, res) {

    let customer_id = req.body;
    let fname = req.body.fname;
    let lname = req.body.lname;
    let username = req.body.username;
    let phone_num = req.body.phone_num;
    let email = req.body.email;
    let password = req.body.password;
    let acc_type = req.body.acc_type;


    dbConn.query(`UPDATE Customer_Account SET username = ?,
                  fname = ? ,
                  lname = ? ,
                  email = ? ,
                  password = ?
                  WHERE customer_id = ? `, [username, fname, lname, email, password, customer_id], function (error,results) {
    if (error) throw error;
    return res.send({error: false, data: results, message: 'Customer Account information have been successfully updated.'})
    });
});

router.delete('/AccountManagement', function (req, res) {

    console.log(req.body)
    let customer_id = req.body.customer_id;

    dbConn.query('DELETE FROM Favorite_Item WHERE customer_id = ?', [customer_id])
    dbConn.query('DELETE FROM Customer_Account WHERE customer_id = ?', [customer_id], function (error, results)
    {
    if (error) throw error;
    return res.send({ error: false, data: results, message: 'Customer Account information have been successfully deleted.' });
    });
});


//=====================================================================================================================================
```

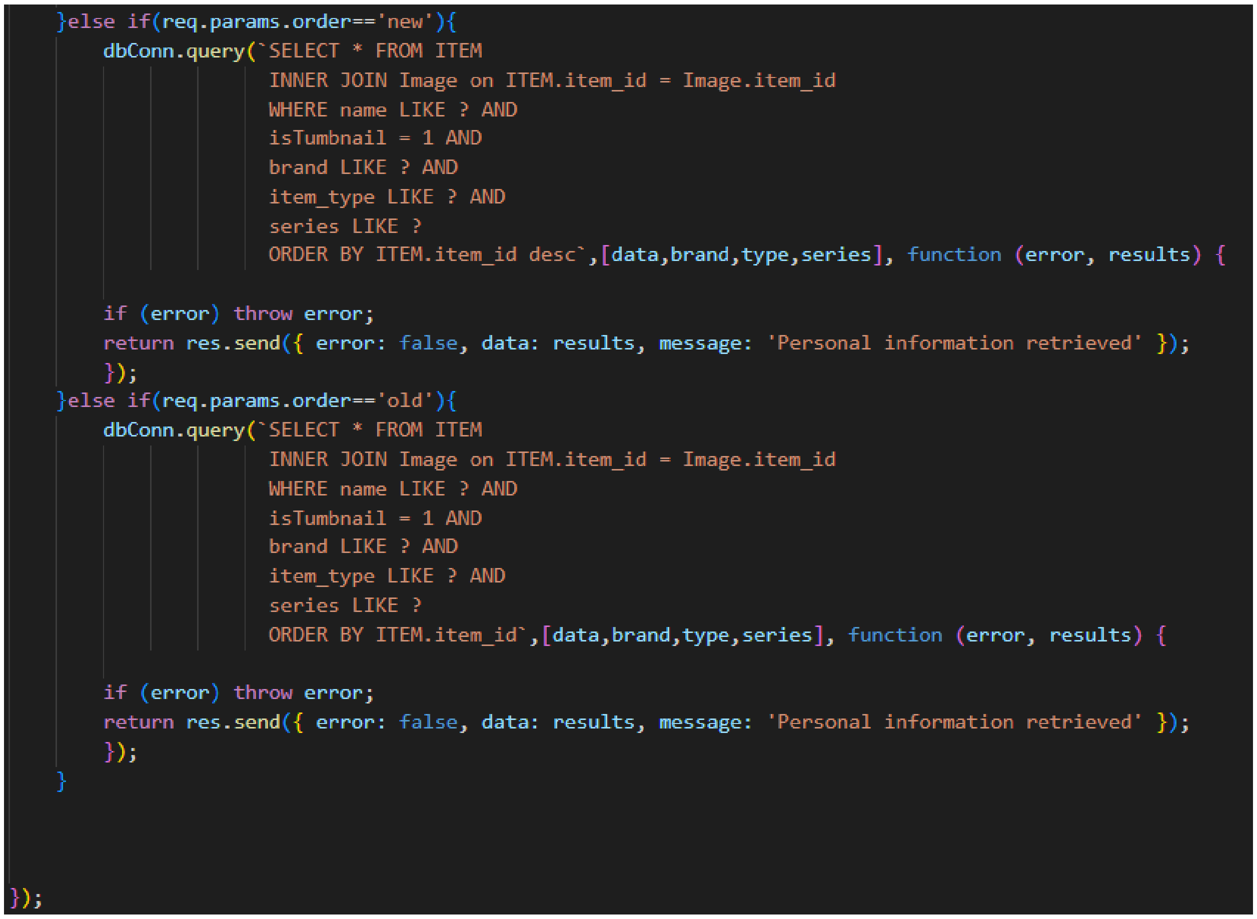
// Search Product



```
//=====================================================================================================================================
//search product

router.get('/searchProduct',async function (req, res) {
    dbConn.query(`SELECT * FROM ITEM
                  INNER JOIN Image on ITEM.item_id = Image.item_id
                  WHERE isTumbnail = 1`, function (error, results) {

    if (error) throw error;
    return res.send({ error: false, data: results, message: 'ALL item information list.' });
    });
});


router.get('/searchProduct/:input', function (req, res) {
    let data = '\'%'+ req.params.input+'%\'';
    console.log(data);

    dbConn.query(`SELECT * FROM ITEM
                  INNER JOIN Image on ITEM.item_id = Image.item_id
                  WHERE name LIKE`+ data+`AND isTumbnail = 1`, function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results, message: 'Personal information retrieved' });
    });
});
```

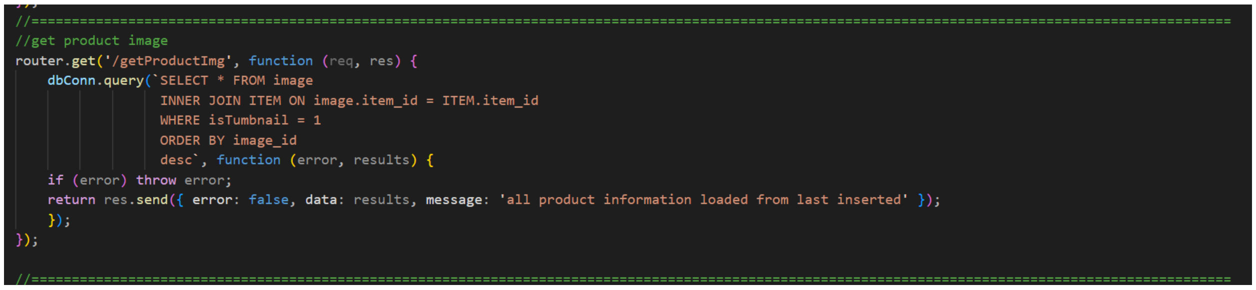


```
router.get('/searchProduct/:input/:brand/:type/:series/:order', function (req, res) {

    let data = req.params.input=='*' ? '%%' : '%'+ req.params.input+'%';
    let brand = req.params.brand=='*' ? '%%' : '%'+req.params.brand+'%';
    let series = req.params.series=='*' ? '%%' : '%'+req.params.series+'%';
    let type = req.params.type=='*' ? '%%' : '%'+req.params.type+'%';
    // var order = req.params.order;
    if(req.params.order=='price'){
        dbConn.query(`SELECT * FROM ITEM
                        INNER JOIN Image on ITEM.item_id = Image.item_id
                        WHERE name LIKE ? AND
                        isTumbnail = 1 AND
                        brand LIKE ? AND
                        item_type LIKE ? AND
                        series LIKE ?
                        ORDER BY price`,[data,brand,type,series], function (error, results) {

        if (error) throw error;
        return res.send({ error: false, data: results, message: 'Personal information retrieved' });
        });
    }else if(req.params.order=='name'){
        dbConn.query(`SELECT * FROM ITEM
                        INNER JOIN Image on ITEM.item_id = Image.item_id
                        WHERE name LIKE ? AND
                        isTumbnail = 1 AND
                        brand LIKE ? AND
                        item_type LIKE ? AND
                        series LIKE ?
                        ORDER BY name`,[data,brand,type,series], function (error, results) {

        if (error) throw error;
        return res.send({ error: false, data: results, message: 'Personal information retrieved' });
        });
```



```
    }else if(req.params.order=='new'){
        dbConn.query(`SELECT * FROM ITEM
                    INNER JOIN Image on ITEM.item_id = Image.item_id
                    WHERE name LIKE ? AND
                    isTumbnail = 1 AND
                    brand LIKE ? AND
                    item_type LIKE ? AND
                    series LIKE ?
                    ORDER BY ITEM.item_id desc`,[data,brand,type,series], function (error, results) {

        if (error) throw error;
        return res.send({ error: false, data: results, message: 'Personal information retrieved' });
        });
    }else if(req.params.order=='old'){
        dbConn.query(`SELECT * FROM ITEM
                    INNER JOIN Image on ITEM.item_id = Image.item_id
                    WHERE name LIKE ? AND
                    isTumbnail = 1 AND
                    brand LIKE ? AND
                    item_type LIKE ? AND
                    series LIKE ?
                    ORDER BY ITEM.item_id`,[data,brand,type,series], function (error, results) {

        if (error) throw error;
        return res.send({ error: false, data: results, message: 'Personal information retrieved' });
        });
    }



});
```

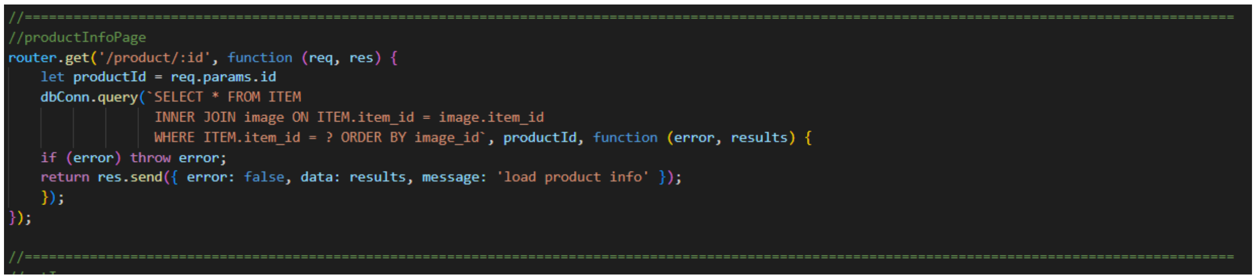
// Get product information



```
//==========================================================================================================================
//get product image
router.get('/getProductImg', function (req, res) {
    dbConn.query(`SELECT * FROM image
                INNER JOIN ITEM ON image.item_id = ITEM.item_id
                WHERE isTumbnail = 1
                ORDER BY image_id
                desc`, function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results, message: 'all product information loaded from last inserted' });
    });
});

//==========================================================================================================================
```

Get the thumbnail picture and send it to the product page in slide and also use in product info order by last data added into the database

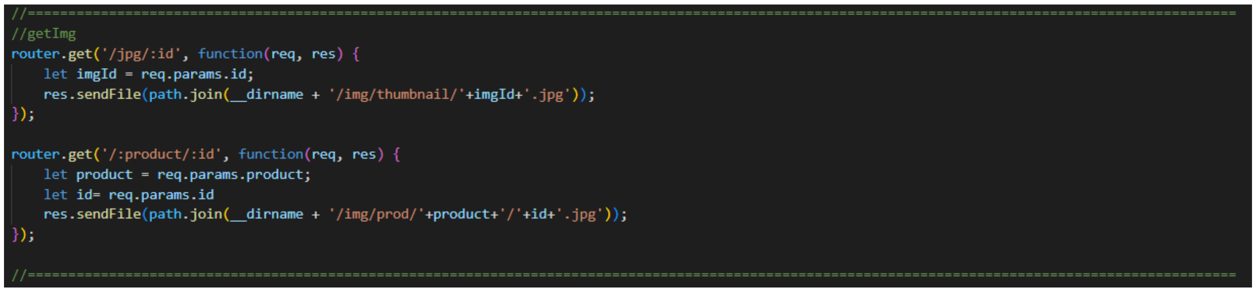
// Product Info Page



```
//==========================================================================================================================
//productInfoPage
router.get('/product/:id', function (req, res) {
    let productId = req.params.id
    dbConn.query(`SELECT * FROM ITEM
                  INNER JOIN image ON ITEM.item_id = image.item_id
                  WHERE ITEM.item_id = ? ORDER BY image_id`, productId, function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results, message: 'load product info' });
    });
});

//==========================================================================================================================
```

This service gets information about an item that is passed to by its parameter, this is used when going into the item information page, to get all the information the item has in detail.

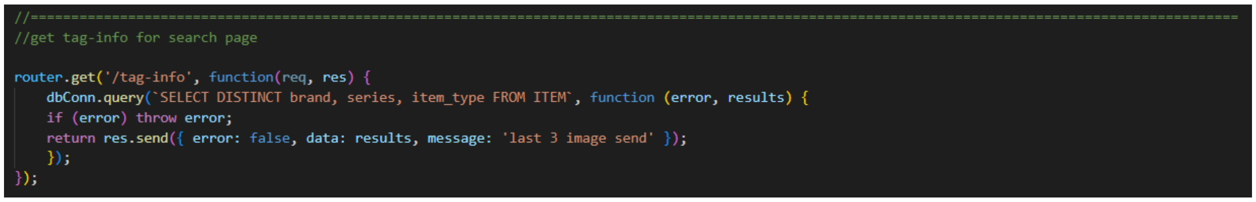
// Get Img



```
//==========================================================================================================================
//getImg
router.get('/jpg/:id', function(req, res) {
    let imgId = req.params.id;
    res.sendFile(path.join(__dirname + '/img/thumbnail/'+imgId+'.jpg'));
});

router.get('/:product/:id', function(req, res) {
    let product = req.params.product;
    let id= req.params.id
    res.sendFile(path.join(__dirname + '/img/prod/'+product+'/'+id+'.jpg'));
});

//==========================================================================================================================
```

The service above is used for accessing a thumbnail picture and the one with a parameter ":id" is used to get the additional informational images displayed as a slideshow on the product page.

// Get tag-info for the Search Page



```
//==========================================================================================================================
//get tag-info for search page

router.get('/tag-info', function(req, res) {
    dbConn.query(`SELECT DISTINCT brand, series, item_type FROM ITEM`, function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results, message: 'last 3 image send' });
    });
});
```

This service is used for retrieving unique brand, series, and item_type that's in the database and will be used in the advanced search page to filter out search options.

Postman

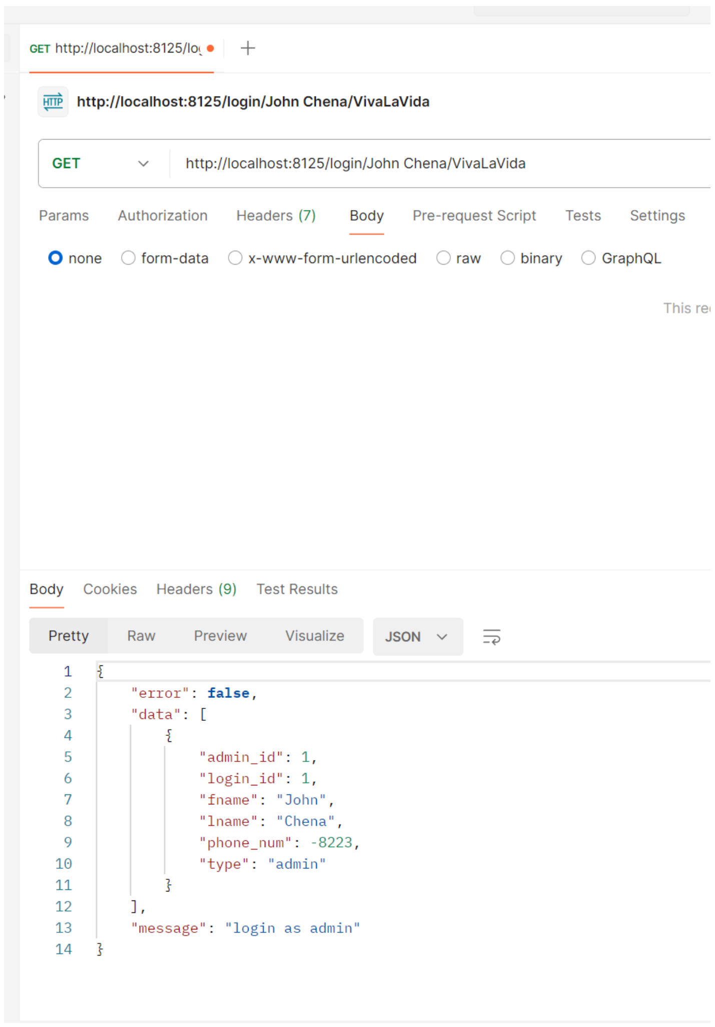
Authentication web service

- '/login/:user/:pass'

Testing for login as admin

Method: get

URL: http://localhost:8125/login/John Chena/VivaLaVida



GET http://localhost:8125/lo

http://localhost:8125/login/John Chena/VivaLaVida

GET http://localhost:8125/login/John Chena/VivaLaVida

Params Authorization Headers (7) Body Pre-request Script Tests Settings

[✓ none] form-data x-www-form-urlencoded raw binary GraphQL

This re

Body Cookies Headers (9) Test Results

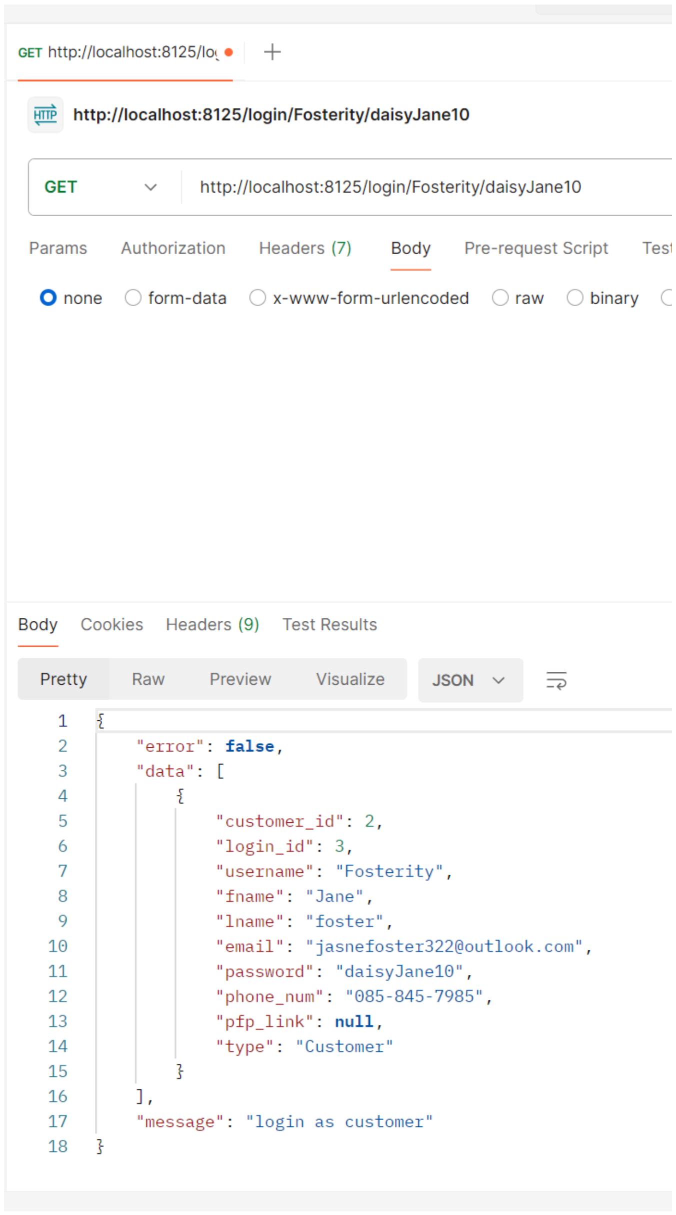
Pretty Raw Preview Visualize JSON

```
{
    "error": false,
    "data": [
        {
            "admin_id": 1,
            "login_id": 1,
            "fname": "John",
            "lname": "Chena",
            "phone_num": -8223,
            "type": "admin"
        }
    ],
    "message": "login as admin"
}
```

Testing for login as customer

Method: get

URL: http://localhost:8125/login/Fosterity/daisyJane10



GET http://localhost:8125/login/Fosterity/daisyJane10

Body: [✓ none] form-data x-www-form-urlencoded raw binary

```
{
    "error": false,
    "data": [
        {
            "customer_id": 2,
            "login_id": 3,
            "username": "Fosterity",
            "fname": "Jane",
            "lname": "foster",
            "email": "jasnefoster322@outlook.com",
            "password": "daisyJane10",
            "phone_num": "085-845-7985",
            "pfp_link": null,
            "type": "Customer"
        }
    ],
    "message": "login as customer"
}
```

==

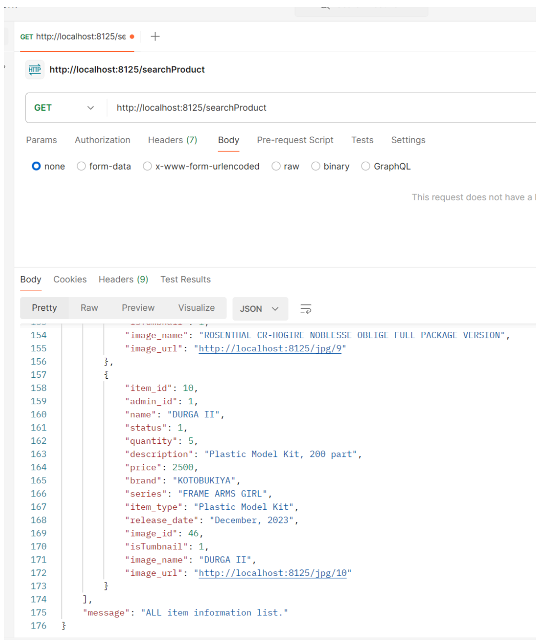
Product Search: no criteria search

- ‘/searchProduct’

Testing for search all product

Method: get

URL: http://localhost:8125/searchProduct



URL: http://localhost:8125/searchProduct

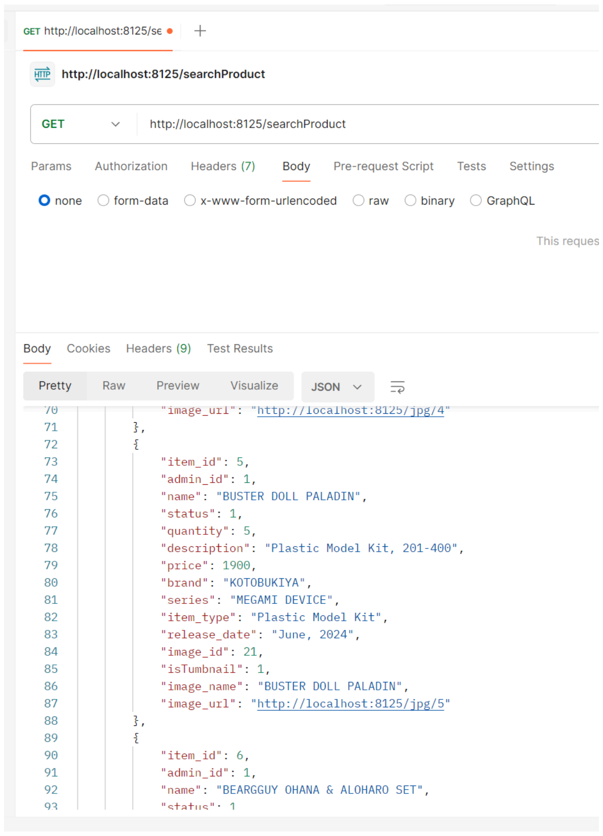
This request does not have a

```
            "image_name": "ROSENTHAL CR-HOGIRE NOBLESSE OBLIGE FULL PACKAGE VERSION",
            "image_url": "http://localhost:8125/jpg/9"
        },
        {
            "item_id": 10,
            "admin_id": 1,
            "name": "DURGA II",
            "status": 1,
            "quantity": 5,
            "description": "Plastic Model Kit, 200 part",
            "price": 2500,
            "brand": "KOTOBUKIYA",
            "series": "FRAME ARMS GIRL",
            "item_type": "Plastic Model Kit",
            "release_date": "December, 2023",
            "image_id": 46,
            "isTumbnail": 1,
            "image_name": "DURGA II",
            "image_url": "http://localhost:8125/jpg/10"
        }
    ],
    "message": "ALL item information list."
}
```

Testing for search all product

Method: get

URL: http://localhost:8125/searchProduct



GET http://localhost:8125/se

http://localhost:8125/searchProduct

GET http://localhost:8125/searchProduct

Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This reques

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
            "image_url": "http://localhost:8125/jpg/4"
        },
        {
            "item_id": 5,
            "admin_id": 1,
            "name": "BUSTER DOLL PALADIN",
            "status": 1,
            "quantity": 5,
            "description": "Plastic Model Kit, 201-400",
            "price": 1900,
            "brand": "KOTOBUKIYA",
            "series": "MEGAMI DEVICE",
            "item_type": "Plastic Model Kit",
            "release_date": "June, 2024",
            "image_id": 21,
            "isTumbnail": 1,
            "image_name": "BUSTER DOLL PALADIN",
            "image_url": "http://localhost:8125/jpg/5"
        },
        {
            "item_id": 6,
            "admin_id": 1,
            "name": "BEARGGUY OHANA & ALOHARO SET",
            "status": 1
```

==

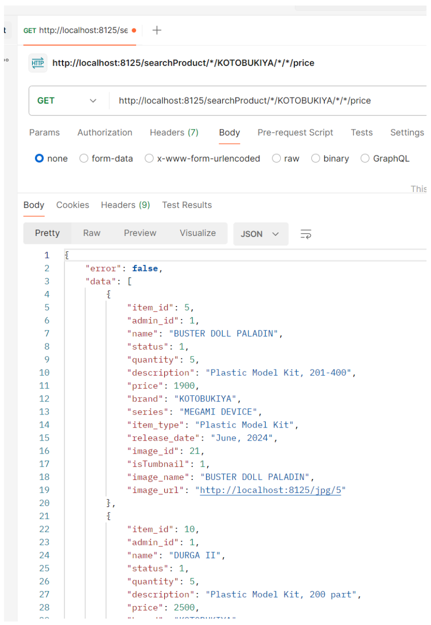
Product Search: criteria search

- '/searchProduct/:input/:brand/:type/:series/:order'

Testing for searching product brand kotobukiya and order by price

Method: get

URL: http://localhost:8125/searchProduct//KOTOBUKIYA/*/*/price*



http://localhost:8125/searchProduct/*/KOTOBUKIYA/*/*/price

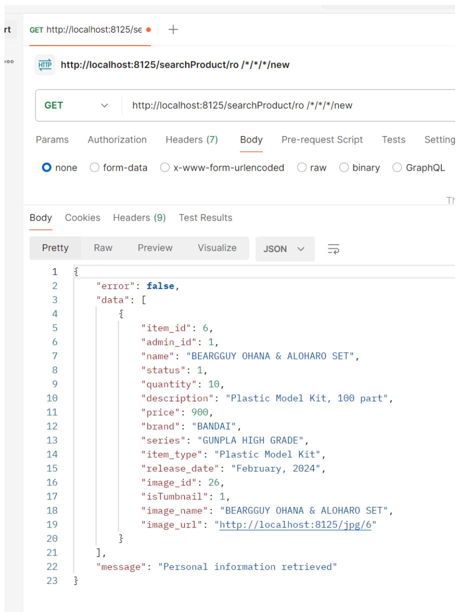
GET http://localhost:8125/searchProduct/*/KOTOBUKIYA/*/*/price

```
{
    "error": false,
    "data": [
        {
            "item_id": 5,
            "admin_id": 1,
            "name": "BUSTER DOLL PALADIN",
            "status": 1,
            "quantity": 5,
            "description": "Plastic Model Kit, 201-400",
            "price": 1900,
            "brand": "KOTOBUKIYA",
            "series": "MEGAMI DEVICE",
            "item_type": "Plastic Model Kit",
            "release_date": "June, 2024",
            "image_id": 21,
            "isTumbnail": 1,
            "image_name": "BUSTER DOLL PALADIN",
            "image_url": "http://localhost:8125/jpg/5"
        },
        {
            "item_id": 10,
            "admin_id": 1,
            "name": "DURGA II",
            "status": 1,
            "quantity": 5,
            "description": "Plastic Model Kit, 200 part",
            "price": 2500,
```

Testing for searching product with 'ro' in the name and order by newest add in database

Method: get

URL: http://localhost:8125/searchProduct/ro //*/*/new*



http://localhost:8125/searchProduct/ro /*/*/*/new

GET http://localhost:8125/searchProduct/ro /*/*/*/new

```
{
    "error": false,
    "data": [
        {
            "item_id": 6,
            "admin_id": 1,
            "name": "BEARGGUY OHANA & ALOHARO SET",
            "status": 1,
            "quantity": 10,
            "description": "Plastic Model Kit, 100 part",
            "price": 900,
            "brand": "BANDAI",
            "series": "GUNPLA HIGH GRADE",
            "item_type": "Plastic Model Kit",
            "release_date": "February, 2024",
            "image_id": 26,
            "isTumbnail": 1,
            "image_name": "BEARGGUY OHANA & ALOHARO SET",
            "image_url": "http://localhost:8125/jpg/6"
        }
    ],
    "message": "Personal information retrieved"
}
```

==

Account Management: insert data

- ‘/AccountManagement’

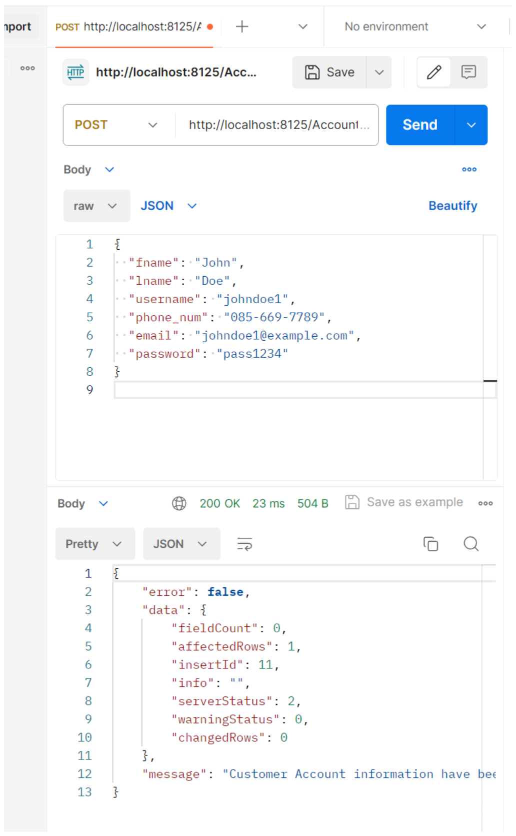
Testing for admin add customer account to database

Method: POST

URL: http://localhost:8125/AccountManagement

Body: raw JSON

```
{
  "fname": "John",
  "lname": "Doe",
  "username": "johndoe1",
  "phone_num": "085-669-7789",
  "email": "johndoe1@example.com",
  "password": "pass1234"
}
```



```
{
  "fname": "John",
  "lname": "Doe",
  "username": "johndoe1",
  "phone_num": "085-669-7789",
  "email": "johndoe1@example.com",
  "password": "pass1234"
}
```

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 11,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "Customer Account information have bee
}
```

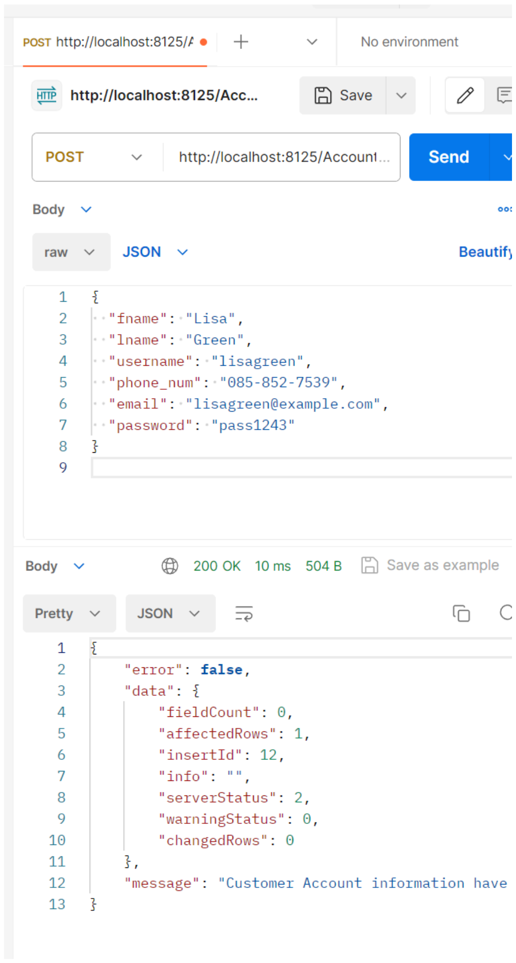
Testing for manage account as customer

Method: POST

URL: http://localhost:8125/AccountManagement

Body: raw JSON

```
{
  "fname": "Lisa",
  "lname": "Green",
  "username": "lisagreen",
  "phone_num": "085-852-7539",
  "email": "lisagreen@example.com",
  "password": "pass1243"
}
```



```
{
  "fname": "Lisa",
  "lname": "Green",
  "username": "lisagreen",
  "phone_num": "085-852-7539",
  "email": "lisagreen@example.com",
  "password": "pass1243"
}
```

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 12,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "Customer Account information have
}
```

==

Account Management : Update data

- '/AccountManagement'

Testing for updating account as admin

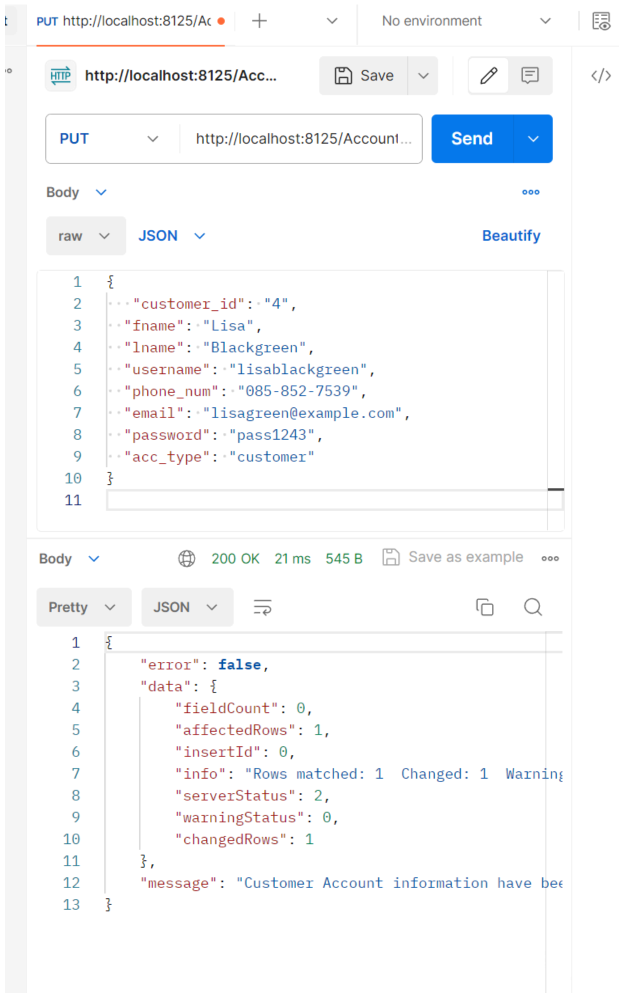
Change customer last name (lname) and username of customer ID 4

Method: PUT

URL: http://localhost:8125/AccountManagement

Body: raw JSON

```
{
   "customer_id": "4",
  "fname": "Lisa",
  "lname": "Blackgreen",
  "username": "lisablackgreen",
  "phone_num": "085-852-7539",
  "email": "lisagreen@example.com",
  "password": "pass1243",
  "acc_type": "customer"
}
```


PUT http://localhost:8125/Ac
No environment
http://localhost:8125/Acc...
Save
PUT
http://localhost:8125/Account...
Send
Body
raw
JSON
Beautify

```
{
  "customer_id": "4",
  "fname": "Lisa",
  "lname": "Blackgreen",
  "username": "lisablackgreen",
  "phone_num": "085-852-7539",
  "email": "lisagreen@example.com",
  "password": "pass1243",
  "acc_type": "customer"
}
```

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 0,
        "info": "Rows matched: 1  Changed: 1  Warnin
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 1
    },
    "message": "Customer Account information have bee
}
```

Testing for add account as admin

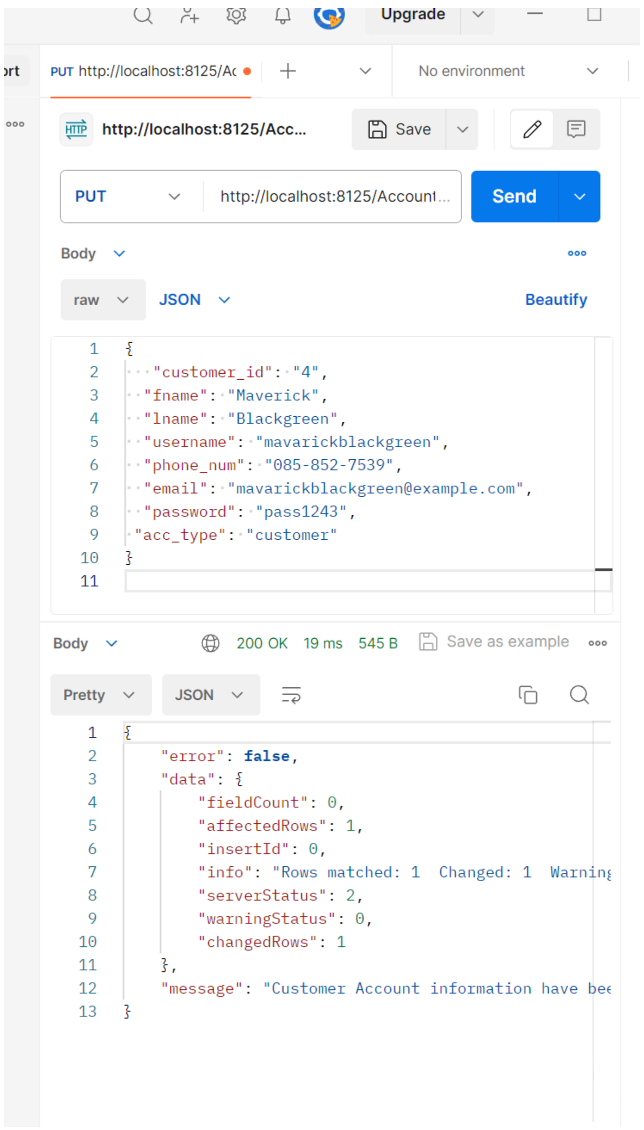
Change customer first name, username and email of customer ID 4

Method: PUT

URL: http://localhost:8125/AccountManagement

Body: raw JSON

```
{
    "customer_id": "4",
  "fname": "Maverick",
  "lname": "Blackgreen",
  "username": "mavarickblackgreen",
  "phone_num": "085-852-7539",
  "email": "mavarickblackgreen@example.com",
  "password": "pass1243",
 "acc_type": "customer"
}
```



PUT http://localhost:8125/Ac   No environment

http://localhost:8125/Acc...   Save

PUT   http://localhost:8125/Account...   Send

Body   raw   JSON   Beautify

```
{
   "customer_id": "4",
  "fname": "Maverick",
  "lname": "Blackgreen",
  "username": "mavarickblackgreen",
  "phone_num": "085-852-7539",
  "email": "mavarickblackgreen@example.com",
  "password": "pass1243",
 "acc_type": "customer"
}
```

Body   200 OK   19 ms   545 B   Save as example

Pretty   JSON

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 0,
        "info": "Rows matched: 1  Changed: 1  Warning
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 1
    },
    "message": "Customer Account information have bee
}
```

==

Account Management : Delete data

- '/AccountManagement'

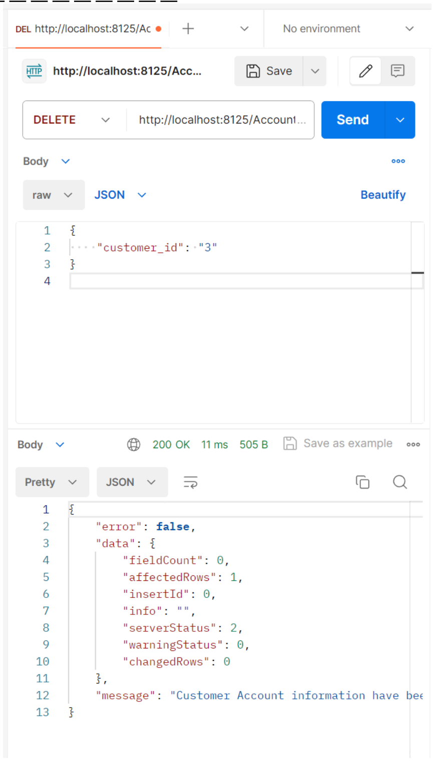
Testing for delete customer account id = 3 as admin

Method: DELETE

URL: http://localhost:8125/AccountManagement

Body: raw JSON

```
{
    "customer_id": "3"
}
```



```
{
    "customer_id": "3"
}
```

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 0,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "Customer Account information have bee
}
```

Account Management : Delete data

- '/AccountManagement'

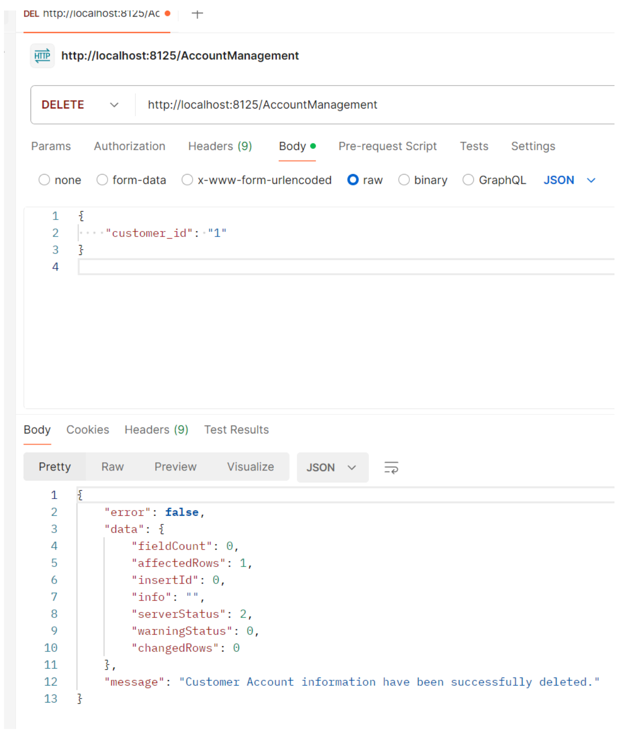
Testing for delete customer account id = 1 as admin

Method: DELETE

URL: http://localhost:8125/AccountManagement

Body: raw JSON

```
{
    "customer_id": "1"
}
```



```
{
    "customer_id": "1"
}
```

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 0,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "Customer Account information have been successfully deleted."
}
```

==

Product Management : insert data

- '/ProductManagement'

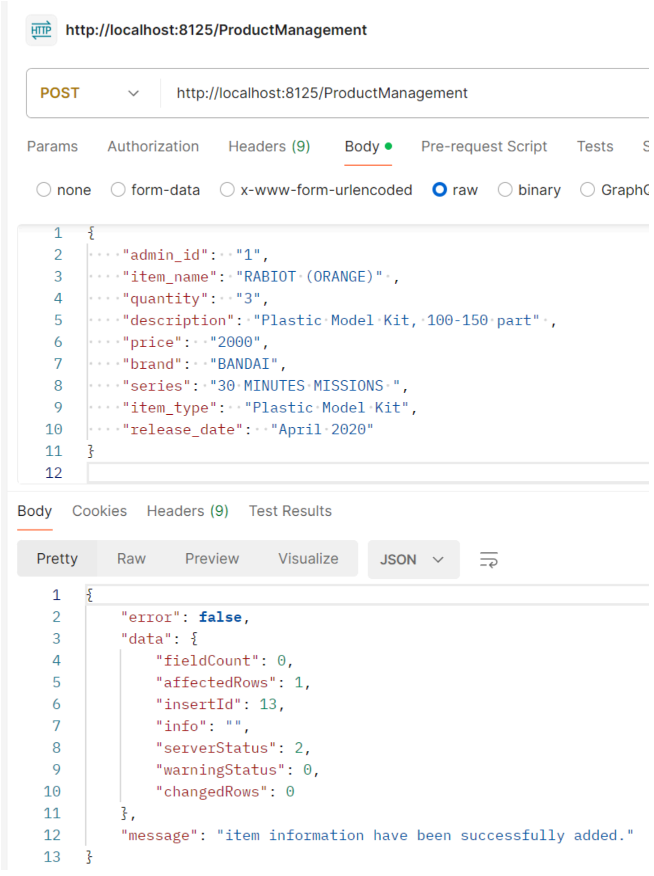
Testing for manage add new product as admin

Method: POST

URL: http://localhost:8125/ProductManagement

Body: raw JSON

```
{
    "admin_id":  "1",
    "item_name": "RABIOT (ORANGE)" ,
    "quantity":  "3",
    "description": "Plastic Model Kit, 100-150 part" ,
    "price":  "2000",
    "brand":  "BANDAI",
    "series": "30 MINUTES MISSIONS ",
    "item_type":  "Plastic Model Kit",
    "release_date":  "April 2020"
}
```


http://localhost:8125/ProductManagement

POST http://localhost:8125/ProductManagement

Params | Authorization | Headers (9) | Body ● | Pre-request Script | Tests

none | form-data | x-www-form-urlencoded | raw (selected) | binary | GraphQ

```
{
    "admin_id":  "1",
    "item_name": "RABIOT (ORANGE)" ,
    "quantity":  "3",
    "description": "Plastic Model Kit, 100-150 part" ,
    "price":  "2000",
    "brand":  "BANDAI",
    "series": "30 MINUTES MISSIONS ",
    "item_type":  "Plastic Model Kit",
    "release_date":  "April 2020"
}
```

Body | Cookies | Headers (9) | Test Results

Pretty | Raw | Preview | Visualize | JSON

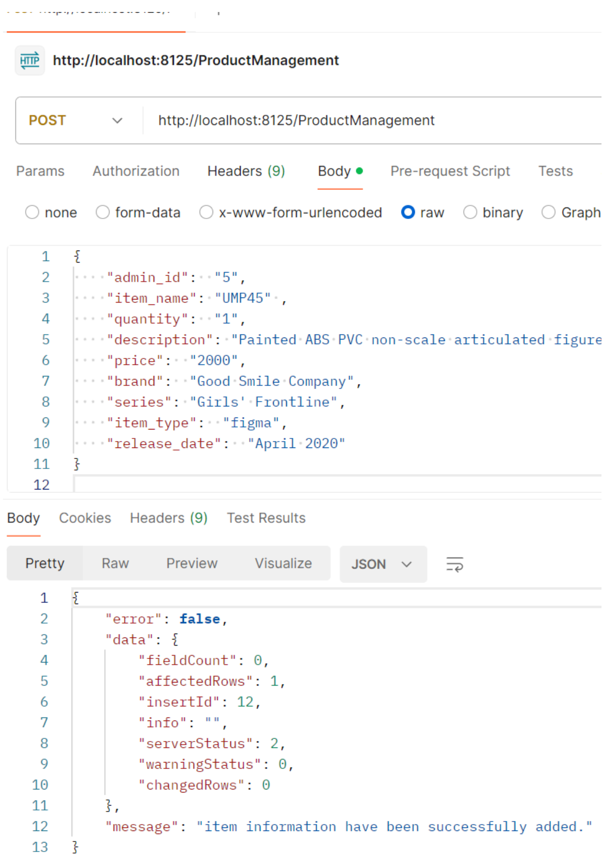
```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 13,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "item information have been successfully added."
}
```

Testing for manage product by adding new item into database

Method: POST

URL: http://localhost:8125/ProductManagement

```
{
    "admin_id":  "5",
    "item_name": "UMP45" ,
    "quantity":  "1",
    "description": "Painted ABS PVC non-scale articulated figure with stand included. Approximately 135mm in height" ,
    "price":  "2000",
    "brand":  "Good Smile Company",
    "series": "Girls' Frontline",
    "item_type":  "figma",
    "release_date":  "April 2020"
}
```



http://localhost:8125/ProductManagement

POST http://localhost:8125/ProductManagement

Params Authorization Headers (9) Body Pre-request Script Tests

none form-data x-www-form-urlencoded [✓ raw] binary Graph

```
{
    "admin_id":  "5",
    "item_name": "UMP45" ,
    "quantity":  "1",
    "description": "Painted ABS PVC non-scale articulated figure
    "price":  "2000",
    "brand":  "Good Smile Company",
    "series": "Girls' Frontline",
    "item_type":  "figma",
    "release_date":  "April 2020"
}
```

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 12,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "item information have been successfully added."
}
```

==

Product Management : Additional insert data

- *'/ImageAdd'*

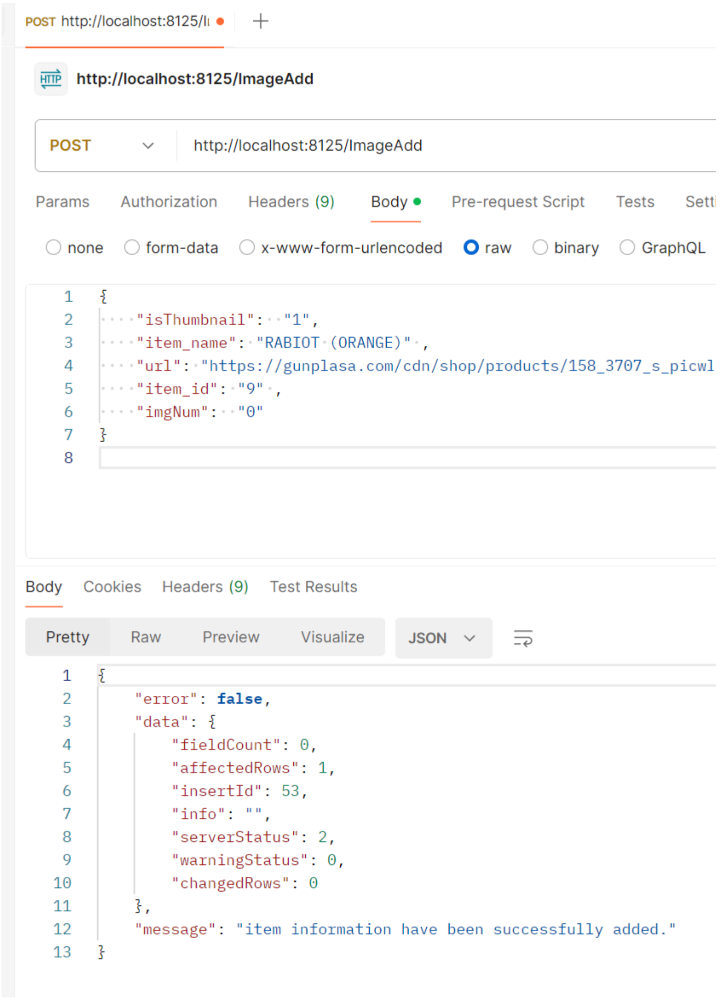
Testing for admin add item image that is a thumbnail image into item table

Method: POST

URL: http://localhost:8125/ImageAdd

Body: raw JSON

```
{
    "isThumbnail":  "1",
    "item_name": "RABIOT (ORANGE)" ,
    "url":
"https://gunplasa.com/cdn/shop/products/158_3707_s_picwlnvq40kt0q2aeqguuxry1jsk_1200x1
200.jpg?v=1661888549",
    "item_id": "9" ,
    "imgNum":  "0"
}
```



POST http://localhost:8125/ImageAdd

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL

```
{
    "isThumbnail":  "1",
    "item_name": "RABIOT (ORANGE)" ,
    "url": "https://gunplasa.com/cdn/shop/products/158_3707_s_picwl
    "item_id": "9" ,
    "imgNum":  "0"
}
```

JSON

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 53,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "item information have been successfully added."
}
```

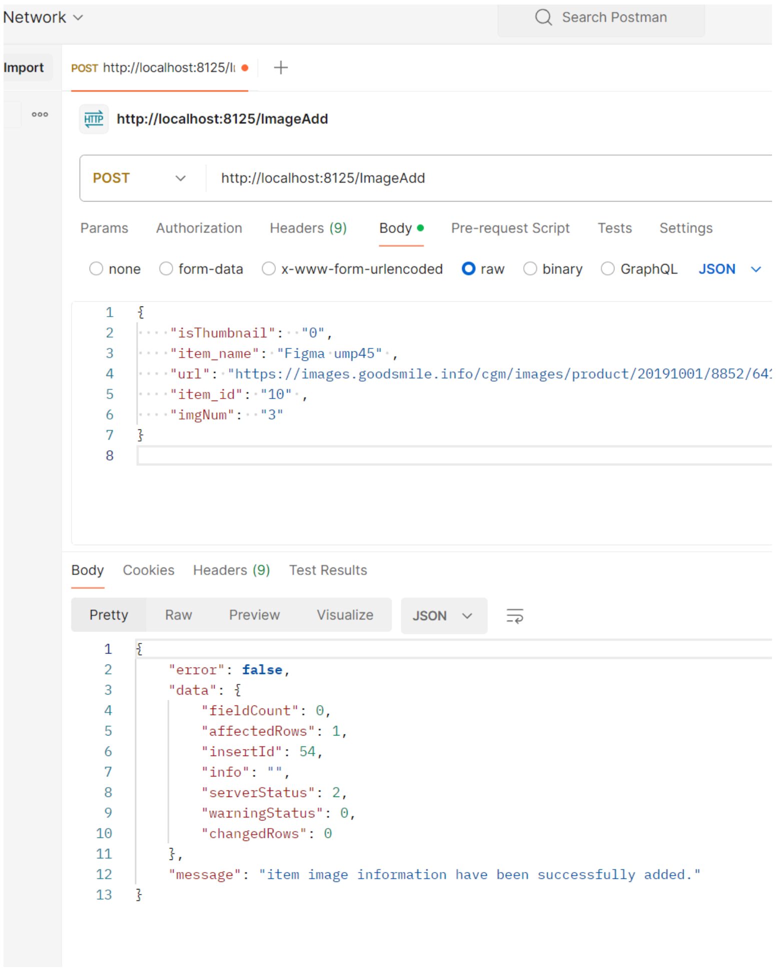
Testing for admin add item image that is for slideshow into item table

Method: POST

URL: http://localhost:8125/ImageAdd

Body: raw JSON

```
{
    "isThumbnail":  "0",
    "item_name": "Figma ump45" ,
    "url":
"https://images.goodsmile.info/cgm/images/product/20191001/8852/64135/large/1b865eec46
4b75da646a4557066c7f8f.jpg",
    "item_id": "10" ,
    "imgNum":  "3"
}
```



POST http://localhost:8125/ImageAdd

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```
{
    "isThumbnail":  "0",
    "item_name": "Figma ump45" ,
    "url": "https://images.goodsmile.info/cgm/images/product/20191001/8852/64
    "item_id": "10" ,
    "imgNum":  "3"
}
```

Body | JSON

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 54,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "item image information have been successfully added."
}
```

Product Management : Updating data

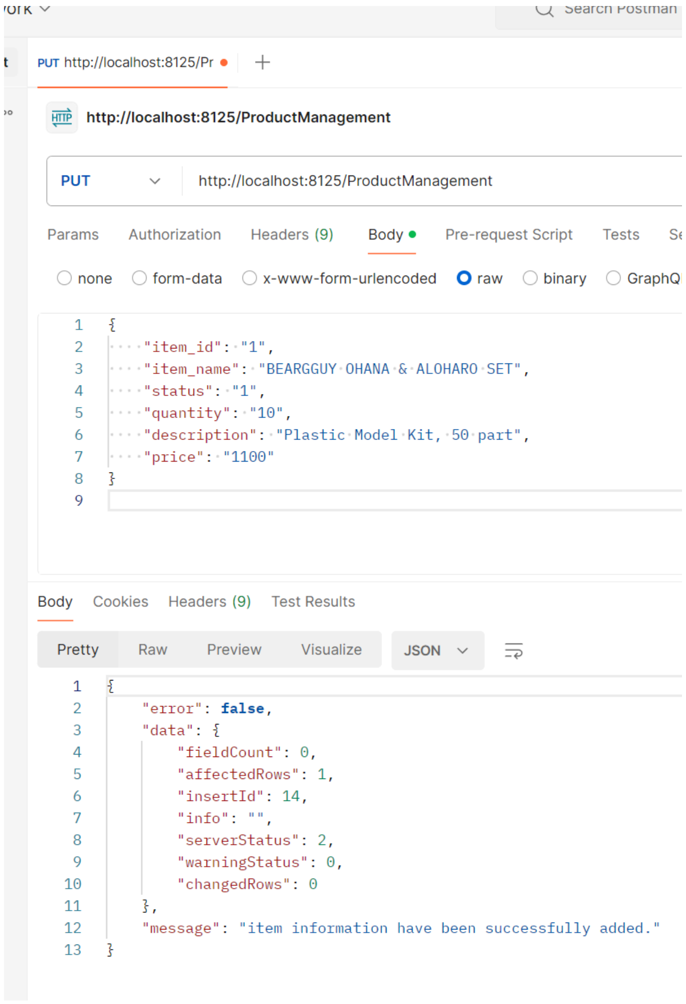
- '/ProductManagement'

Testing for update product as admin

Method: PUT

URL: http://localhost:8125/ProductManagement

```
{
    "item_id": "1",
    "item_name": "BEARGGUY OHANA & ALOHARO SET",
    "status": "1",
    "quantity": "10",
    "description": "Plastic Model Kit, 50 part",
    "price": "1100"
}
```



```
{
    "item_id": "1",
    "item_name": "BEARGGUY OHANA & ALOHARO SET",
    "status": "1",
    "quantity": "10",
    "description": "Plastic Model Kit, 50 part",
    "price": "1100"
}
```

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 14,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "item information have been successfully added."
}
```

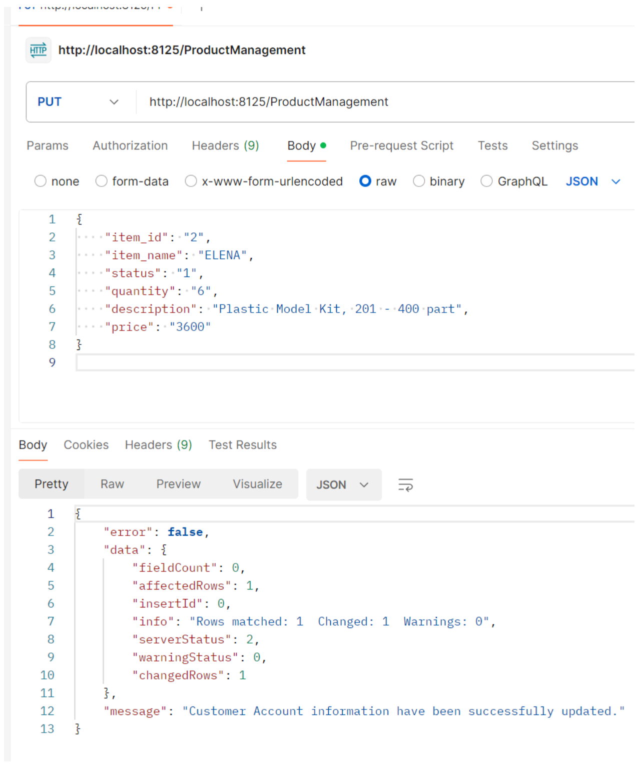
Testing for add product as admin

Change number of quantity and price

Method: PUT

URL: http://localhost:8125/ProductManagement

```
{
    "item_id": "2",
    "item_name": "ELENA",
    "status": "1",
    "quantity": "6",
    "description": "Plastic Model Kit, 201 - 400 part",
    "price": "3600"
}
```



http://localhost:8125/ProductManagement

PUT http://localhost:8125/ProductManagement

Params · Authorization · Headers (9) · Body · Pre-request Script · Tests · Settings

none · form-data · x-www-form-urlencoded · [✓ raw] · binary · GraphQL · JSON

```
{
    "item_id": "2",
    "item_name": "ELENA",
    "status": "1",
    "quantity": "6",
    "description": "Plastic Model Kit, 201 - 400 part",
    "price": "3600"
}
```

Body · Cookies · Headers (9) · Test Results

Pretty · Raw · Preview · Visualize · JSON

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 0,
        "info": "Rows matched: 1  Changed: 1  Warnings: 0",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 1
    },
    "message": "Customer Account information have been successfully updated."
}
```

==

Product Management : Delete data

- ‘/ProductManagement’

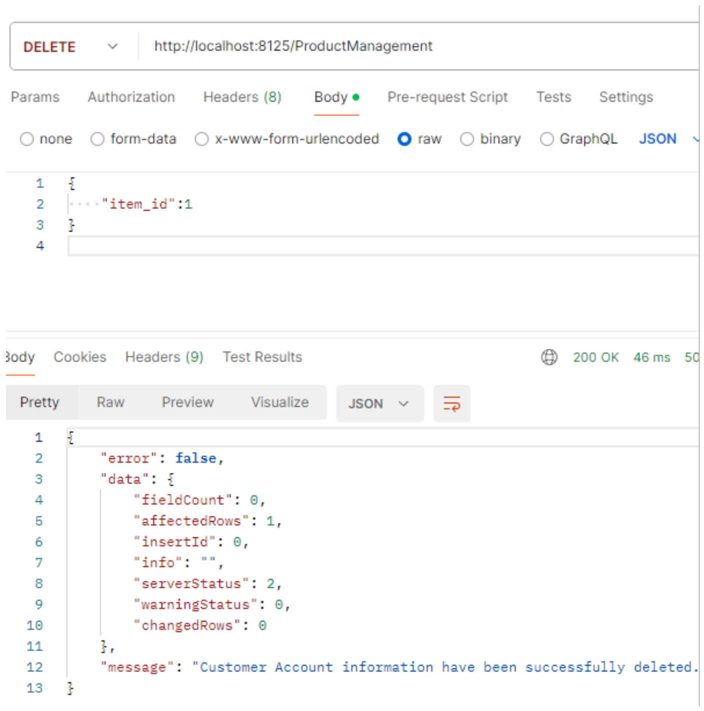
Testing for delete product as admin with item id = 1

Method: DELETE

URL: http://localhost:8125/ProductManagement

Body: raw JSON

```
{
    "item_id":1
}
```



DELETE http://localhost:8125/ProductManagement

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
····"item_id":1
}
```

Body Cookies Headers (9) Test Results 200 OK 46 ms 50

Pretty Raw Preview Visualize JSON

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 0,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "Customer Account information have been successfully deleted.
}
```

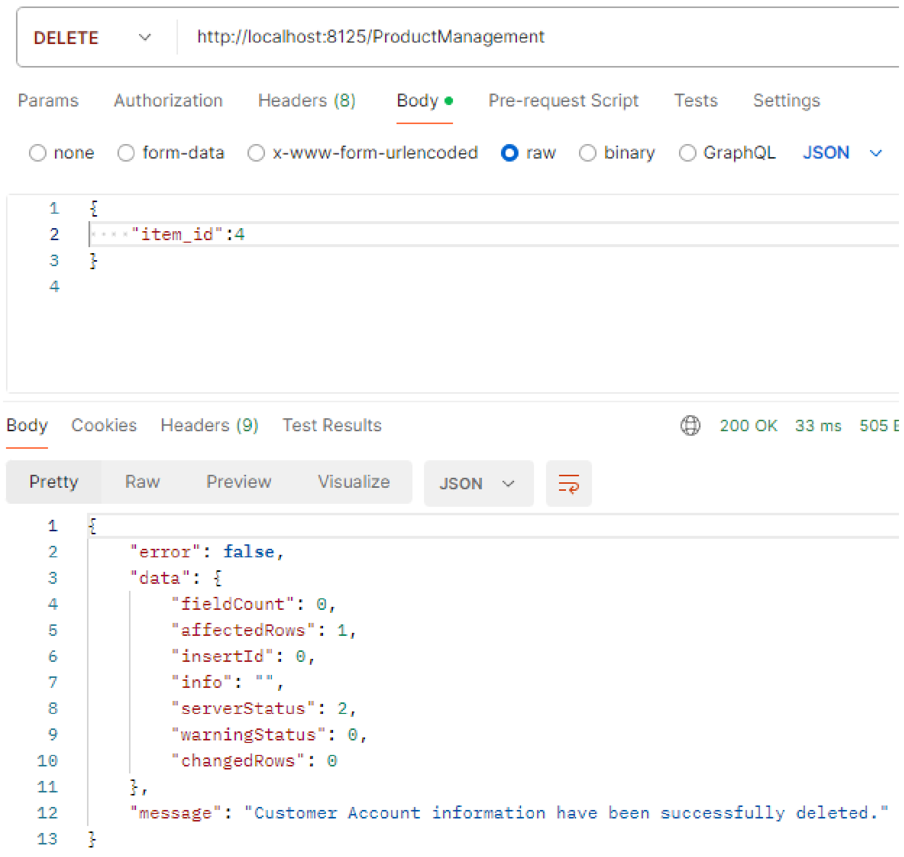
Testing for delete product as admin with item id = 4

Method: DELETE

URL: http://localhost:8125/ProductManagement

Body: raw JSON

```
{
    "item_id":4
}
```



DELETE http://localhost:8125/ProductManagement

```
{
    "item_id":4
}
```

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 0,
        "info": "",
        "serverStatus": 2,
        "warningStatus": 0,
        "changedRows": 0
    },
    "message": "Customer Account information have been successfully deleted."
}
```

==

Reference

Works Cited

Gunpla Village ปลีก-ส่ง Gundam,Nendoroid,Kotobukiya ลิขสิทธิ์แท้, https://www.gunplavillage.com/. Accessed 29 February 2024.

無限邂逅メガロマリアオフィシャルサイト｜ コトブキヤ, https://megalomaria.com/. Accessed 1 March 2024.

API — ImgBB, https://api.imgbb.com/. Accessed 24 April 2024.

"Documentation." *Leaflet*, https://leafletjs.com/reference.html. Accessed 24 April 2024.

"Frame Arms Girl - Kaemodel จำหน่ายโมเดลกันดั้ม โมจีน Model Gundam สั่งซื้อกันดั้ม Bundai ราคาถูก : Inspired by LnwShop.com." *แก่โมเดล*, http://www.kaemodel.com/category/96/kotobukiya-plastic-model-kit/frame-arms-girl. Accessed 1 March 2024.

"Introduction · Bootstrap v5.0." *Bootstrap*, https://getbootstrap.com/docs/5.0/getting-started/introduction/. Accessed 24 April 2024.

"KOTOBUKIYA | Figures・ Model Kits・Hobby." コトブキヤ, https://www.kotobukiya.co.jp/en/. Accessed 29 February 2024.