SOFT8025 Scalable Microservices Assignment 1

Completion Date: 25th October 2019

Value: 22 marks

On completion please zip up your files. Upload to Canvas.

Notes

You can backup your database with:

mysqldump -u root -p shop > database.sql

You can then restore the database by:

1. Open a command prompt and run

   mysql -u root -p

2. executing the following commands (from the mysql prompt):

   `mysql>` source database.sql;

   These commands will destroy an existing shop database

Part 1

You will need to create the shop database on your laptop (install mysql). You will need to give mysql an root password

You may need to change the PATH environment variable to point to mysql so that   you can run the mysql client. In windows the path to mysql is something like C:\Program Files\MySQL\MySQL Server 5.7\bin

Download the shopmicroservice.zip file and extract it

Open a command prompt and enter the following

```
mysql -u root -p
enter your root password
source [path to the database.sql file] ;
exit
```

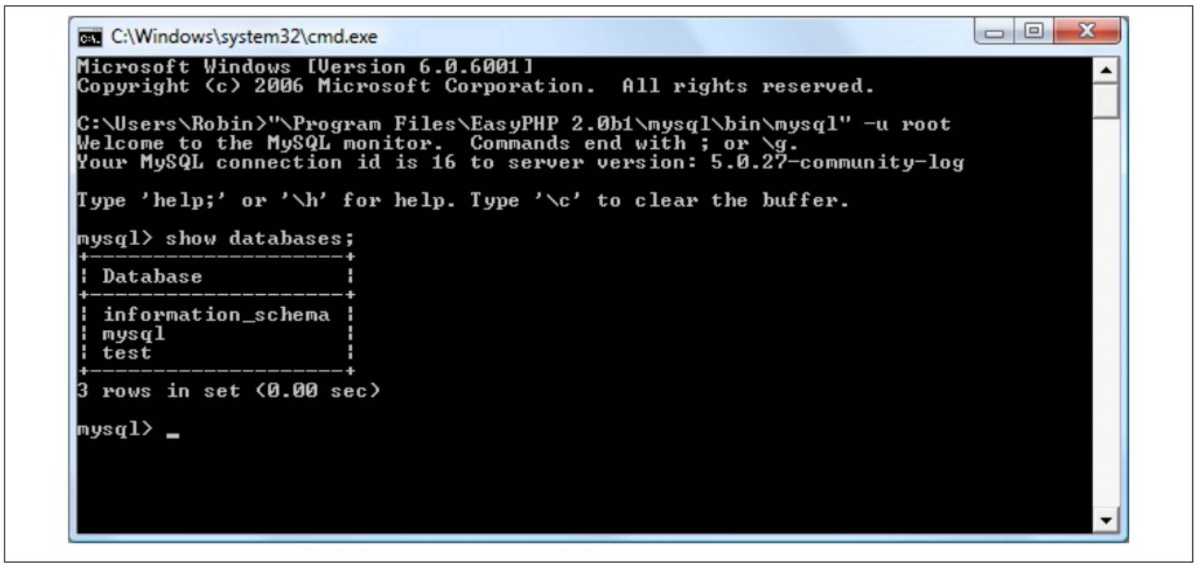
Part 2

To verify that the state of the database at any time.

Open a command prompt and run

mysql -u root -p

mysql> show databases;



C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Users\Robin>"\Program Files\EasyPHP 2.0b1\mysql\bin\mysql" -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16 to server version: 5.0.27-community-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
3 rows in set (0.00 sec)

mysql> _
```

mysql> use shop;

mysql> select * from products;
mysql> exit;

Part 3 – create new product form

a) Write the code for a creating a product. Obviously we should not have a new product button available to customers. So this functionality should only be available to admin users and is in here for demo purposes.

Update the newProductForm in index.html in the front-end microservice so that it includes the fields from the product table

```html
<div id="newProduct">
  <h1>New Product</h1>
  <p>
  <form id="newProductForm" action="">
    Name:<br>
    <input type="text" name="name" value="">
    <br>
    <input type="submit">
  </form>
</div>
```

b) Handle the new product in the catalogue service – see the shopdb monolith code.

You will need to write javascript code for the *newProduct* end point in the catalogue microservice to add the form data into the database. This will be a POST request with the data in query string format e.g. name=car1&description=good car.

The javascript function below in the index.html is invoked when the user clicks on the newproduct button. It currently communicates with a newProduct endpoint in the front-end microservice - this end point does not exist.

```
$('#newProductForm').on( "submit",function(event) {

    event.preventDefault();
    $("#newProduct").hide();
    var fd = $('#newProductForm').serialize();

    $.post(
            "/newProduct",
            fd,
            function (data) {
                console.log(data);
                $('#logonmessage').html(data);
            }
    );


});
```

The fd variable in the code above is currently passed as a query string to the server endpoint. See the following code in shopdb which uses the querystring module to parse the data from the browser. That is

```
var qs = require('querystring');


…..

……..

case "/newProduct":
    var body = '';
    console.log("add ");
    request.on('data', function (data) {
        body += data;
    });

    request.on('end', function () {
        var product = qs.parse(body);

        console.log('new Product');
        console.log(JSON.stringify(product, null, 2));


    });

    break;
```

Part 4 – Shopping Cart fixes

a) Combine the items in the cart if the user orders a product more than once. You need to modify the *add* endpoint in the cart microservice to achieve this. Currently when a user orders car1 twice there are two items in his/her cart for each selection.

b)
Add some styling to the overall total in the cart. Currently the total is on top of the checkout and has a default html style applied.

c)
Fix the delete button in the cart so that the selected item is deleted from the customer’s cart. Currently the function deleteCartItem is invoked in index.html of the front-end service when the delete button is clicked. The cartid is passed to the function.

```
out += '<td> <button onclick="deleteCartItem(' + cart[i].cartid;
out1 = ")" + '">Delete</button></td>';
```

You will need to “update” the code in api/cart/index.js. This end-point expects a HTTP delete method with the cart id in the url e.g. /cart/2 means delete cart item 2

```
app.delete("/cart/:id", function (req, res, next) {
  if (req.params.id == null) {
    return next(new Error("Must pass id of item to delete"), 400);
  }

  console.log("Delete item from cart: " + req.url);

  var custId = helpers.getCustomerId(req, app.get("env"));
```

The front-end service will invoke the cart microservice with a DELETE HTTP method and the url defined in endpoints.js (currently http://localhost:3003). So for example http://localhost:3003/1/items/2 means to delete cart item 2 from the cart of the customer with id 1.

i.e.

```
endpoints.cartsUrl + "/" + custId + "/items/" + req.params.id.-
toString()
```

There is an array of cart items for each customer in the cart. If a customer has not logged on then the custId is the session id.