CartServiceImpl.java

Inflexibility of program

- All variables in CartServiceImpl are local, and the class has no fields. This makes Mockito mock testing infeasible, and thus reduces our coverage.

Lack of documentation

- Intended behavior is hard to measure. For instance, what should we do if a negative amount of a product is added to a cart? What if the username is negative? There are many ambiguities that make it difficult to test

Inconsistent naming of variables and parameters

Inconsistent and incomplete programming / error handling

- Oftentimes, error behavior is indistinguishable from general functioning. For instance, the counting functions return 0 whenever an error occurs, which is identical behavior to an empty cart, for example. Some functions desperately need error states (i.e. return -1, throw Exception, etc.)
- Situations where products are not found are inconsistent. For instance, when comparing inputs and outputs using null product ID's, updateProductToCart() works fine, giving a success message, while addProductToCart() gives a failure message. This made testing more difficult and introduces general inconsistencies into the program's behavior.

getCartCount: 141-142 (if rs.next()....)

- Does not recognize a null column when it appears. rs.wasNull() does not check if rs.next() is null, but rather rs.getInt(). Inserting the below code prior to these lines prints true twice, indicating that the result is indeed null

```
- System.out.println(rs.next());
- count = rs.getInt(1);
- System.out.println(rs.wasNull());
```

- Does not have a response to indicate an error other than printing stacktrace
  - I.e. empty cart and cart with invalid NULL value both return 0, not even the Exception message itself
  - If no match for the username is found, it does not throw an error (debatable whether this is a bug)

getProductCount: 350-351

- Same as getCartCount 141-142

getCartItemCount:

- Same as getCartCount 141-142

<u>DemandServiceImpl.java</u>

Inflexibility of program

- All variables in DemandServiceImpl are local, and the class has no fields. This makes Mockito mock testing infeasible, and thus reduces our coverage.

Lack of documentation

- Intended behavior is hard to measure, and there is no explanation aside from comments on a few functions
- For instance, the return status of removeProduct is true for the branch that occurs when a demand is not found. Why would it return true if nothing was removed?

Inconsistent naming of variables and parameters

addAProduct: 41

- The comments in the function say that the flag should be returned true if the item is added to the database: specifically, if a demand of a certain username and product ID already exists, nothing is inserted into the database, as that branch is skipped entirely. However, the branch for when the demand already exists is to set the flag to true, which goes against this comment, since the new demand was not successfully added.

<u>OrderServiceImpl.java</u>

Lack of documentation

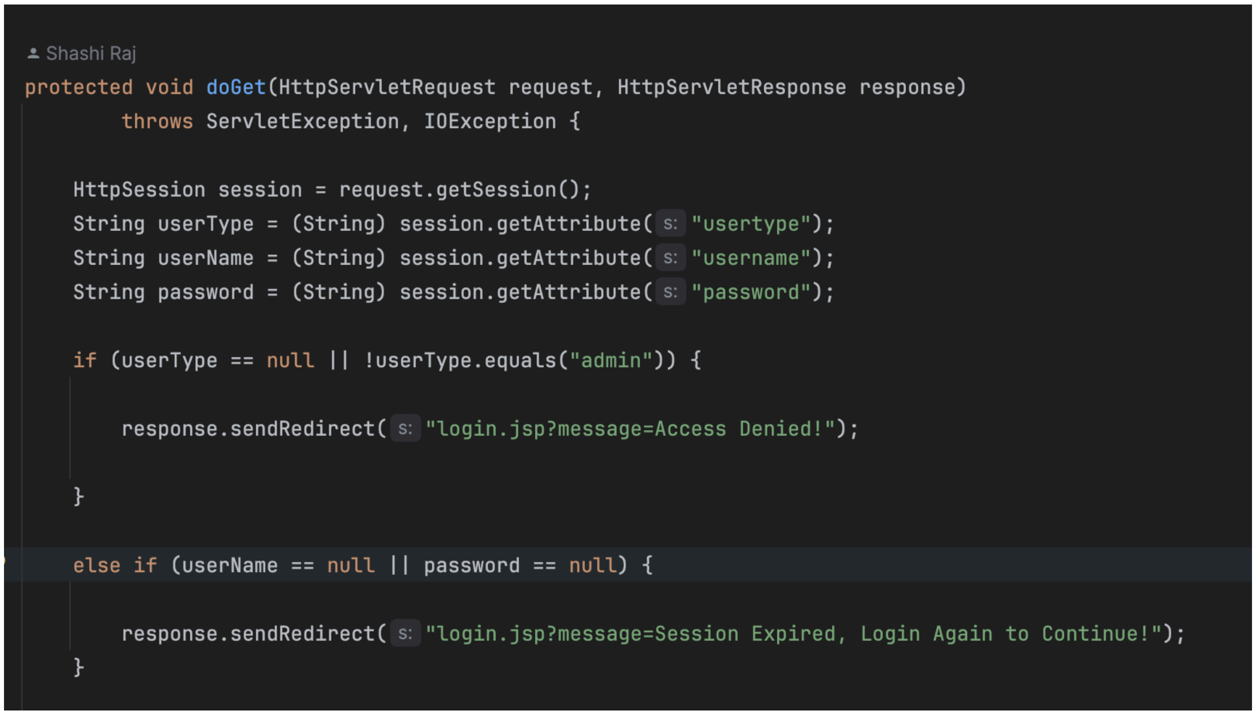
- Intended behavior is hard to measure, and there is no explanation, and this appears in some faults discovered

Inconsistent naming of variables and parameters

getOrdersByUserId: 216

- The query to select orders to match a transaction retrieves order objects, but tries to access the transid field, which is nonexistent. It should be orderid, but I believe the inconsistent naming between tables caused the original author to make this mistake and not test it.

<u>AddProductSrv, RemoveProductSrv, UpdateProductSrv</u>



```
Shashi Raj
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    HttpSession session = request.getSession();
    String userType = (String) session.getAttribute( s: "usertype");
    String userName = (String) session.getAttribute( s: "username");
    String password = (String) session.getAttribute( s: "password");

    if (userType == null || !userType.equals("admin")) {

        response.sendRedirect( s: "login.jsp?message=Access Denied!");

    }

    else if (userName == null || password == null) {

        response.sendRedirect( s: "login.jsp?message=Session Expired, Login Again to Continue!");
    }
```

Cannot reach the second if statement, as when logging in you automatically require that the username and password are not null, and you cannot set a cookie session with null attributes. Specifically, if a username, password, or usertype is passed in that is null, it will throw an exception when doing (String) session.getAttribute().

<u>Register</u>

When registering a user, you can register with empty fields, such as the address. This should not be allowed, and is also prohibited by the frontend, but there is no backend api checks.

<u>AddtoCart</u>

When adding a number of items into your cart with less being available, it does not stop you, and gives a success message, although only the lesser amount is actually added.

NOTE: TransServiceImpl.java, UserServiceImpl.java, and ProductServiceImpl.java are tested with exceptions that should be thrown, this leads to a lot more failed tests than the previous files. For example, if a null userId is passed in, these tests expect an IllegalArgumentException to be thrown instead of the empty string that is returned by the function in this case.

TransServiceImpl.java

Lack of documentation

- Intended behavior is hard to measure. What should we do if a a null transaction id is passed in?

Inconsistent naming of variables and parameters

- The function is named getUserId, which is misleading because userId is actually the username which is not unique to users unlike their email address.

Inconsistent and incomplete programming / error handling

- When passing in a transaction id of a transaction that does not exist, no error is thrown, an empty string is just returned instead of a user id. This also occurs when a null or empty string as the transaction id is passed in.

UserServiceImpl.java

Lack of documentation

- Intended behavior is hard to measure. For instance, what should we do if a negative phone numbers is registered for a user? What if the email is not an actual email? There are many ambiguities that make it difficult to test.

Inconsistent and incomplete programming / error handling

- When passing in any null or empty string as an argument, no checks or error handling are being conducted. This also occurs for email addresses and phone numbers that are not in a valid format (e.g. no @gmail.com at the end). Similarly, negative values for parameters such as pinCode and phone number are accepted and user ids for a user that does not exist.

Inconsistent naming of variables

- For the isValidCredential() function, email and password are taken is as credentials, however, the invalid credentials response says “Incorrect Username or Password” instead of “Incorrect Email or Password”

ProductServiceImpl.java

Lack of documentation

- Intended behavior is hard to measure. For instance, what should we do if a product is added or updated with a negative quantity? What if multiples of the same products are added but they just have different product ids? There are many ambiguities that make it difficult to test.

Inconsistent and incomplete programming / error handling

- When passing in any null or empty string as an argument, no checks or error handling are being conducted. This also occurs when an id of a product that does not exist is passed in. Similarly, negative values for parameters such as prodPrice and prodQuantity are accepted.

getImage() returns an array of bytes, however, when a product has an existing, valid image, a null array is returned.

sellNProduct(): This function does not handle cases where N is greater than the product's quantity and sells more product than is in inventory. Similarly, a negative amount of products can be sold.