Individual Report - Callum Copping

Description of code)

```
# Add new item to order
try:
    menu_item = request.form['menu_item']
    # Construct the name of the quantity field
    quantity_field_name = f"{menu_item} quantity"
    quantity = int(request.form.get(quantity_field_name))
except:
    # Handle errors when parsing form data
    return render_template('home.html', error="An error occured adding an item to your order sorry :(")

# Update the order and total price
order_items = session['order_items']
if menu_item in order_items:
    order_items[menu_item] += quantity
else:
    order_items[menu_item] = quantity      You, 4 weeks ago • Add quantities of items to order, …

session['order_items'] = order_items
session['tot_price'] += fetch_price(menu_item) * quantity
session['tot_price'] = round(session['tot_price'], 2)
```

The above code snippet handles adding an item from the menu to a customer's cart, it first retrieves the item from the form data, it then uses this to construct a variable for the quantity variable, this is particularly useful as it allows for dynamic retrieval of quantities regardless of the menu. The code has a try-except block to handle the possibility of an error occurring whilst retrieving the form data. Once the item and quantity have been retrieved this information is then used to update the session variable "order_items" which is a dictionary, if the ordered item is already in the customer's cart the key-value pair is incremented by the quantity otherwise it is created with the quantity chosen by the customer. The session variable is then updated and the price for the item is fetched and used to update the total price of the order, this value is then rounded to 2 decimal places.

```
# Fetch the price of the menu item from the database
try:
    menu_item_db = db.Menu.query.filter_by(name=menu_item).first()
    item_price = float(menu_item_db.price)
    return item_price
except:
    print("Error: Menu item ", {menu_item}, " not found in database.")
```

The above code snippet handles the fetching of an items price mentioned previously, it first fetches the item that matches the given item, it then converts the listed price of the item to a float which is then returned. The code is inside of a try-except block to handle two possible causes of errors either the item is not in the database or somehow the stored price isn't a float. The query is done using SQLAlchemy's query API to find the items name in the name column, this allows for efficient retrieval of data from the database.

Function Point Analysis)

The following are the identified External Inputs (EI), External Outputs (EO), External Inquiries (EQ), Internal Logical Files (ILF), and External Interface Files (EIF):

EI: 1

- User registration (a name, date of birth, address, phone number, email address.)
    - DETs: 5 (a name, date of birth, address, phone number, email address.)
    - FTRs: 2 (Client table, input)
    - Average complexity, FP = 4

EO: 1

- Retrieve total spent by all clients within a set of age ranges (Amount spent, age range)
    - DETs: 2 (Amount spent, age range)
    - FTRs: 3 (Input, Client table, Output)
    - Low complexity, FP = 4

EQ: 1

- Retrieve all orders (order ID, table number, time placed, time of last change, notes, order status ID)
    - DETs: 6 (order ID, table number, time placed, time of last change, notes, order status ID)
    - FTRs: 2 (Orders, Output)
    - Average complexity, FP = 4

ILF: 3

- Client table (name, date of birth, address, phone number, email address, amount spent)
    - RETs: 1 (Client table)
    - DETs: 6 (name, date of birth, address, phone number, email address, amount spent)
    - Low complexity, FP = 7
- Order_status table (ID, name, description)
    - RETs: 1 (Order_status table)
    - DETs: 3 (ID, name, description)
    - Low complexity, FP = 7
- Order table (ID, table number, time placed, time of last change, notes, order_status ID)
    - RETs: 2 (Order table, order_status table)
    - DETs: 6 (ID, table number, time placed, time of last change, notes, order_status ID)
    - Low complexity, FP = 7

EIF: 0

UFC = 4+4+4+7+7+7 = 33

What I've learnt developing software in a group project)

Developing software using scrum was a valuable learning experience, especially as group projects during my first year were very chaotic. Frequent meetings ensure everyone is always working towards a common goal. As an agile development methodology, scrum allowed our group to be very reactive which lead to us producing a good final product, for example in each sprint if someone finished their task earlier than expected during our meeting we would look at the current application and see what was missing or didn't make sense and assign the necessary changes to those who had finished their work for the sprint even if these were often minor styling changes. One issue I ran into within scrum was that as I was assigned scrum master, I unintentionally found myself in somewhat of a leadership role despite trying to inform my group that wasn't my job nor what we were told to do, I think this occurred due to the project's intimidating nature and it was my group members instincts. Another issue was some members were not fully open with if they had any problems completing their task which on occasion led to features being incomplete and left to someone else to finish or taking longer than expected which could have been avoided. Aside from the technical skills, participating in this group project highlighted the importance of ethics, such as prioritizing the users' privacy, security, and accessibility. This reinforced my belief that technology should be developed with ethics in mind.

Robust Code)

I found out fairly quickly in this project that developing robust code is even more important when working with a team, this is as you never know exactly how the members of the group will implement their tasks which can lead to unexpected interactions in the program which will almost always lead to an error. I have always tested code I write quite rigorously however I learnt in this project that before merging a feature into the working branch it is important to not only test the code, I had just implemented but test all of the surrounding code to make sure that there are no potentially fatal interactions. This led to me implementing a defensive programming style, surrounding a large portion of my code in try-except blocks so that our program would handle errors in a graceful fashion, an example of this would be when the customer goes from the home page to the ordering page they have to input a table number, despite the text box being limited to only accepting positive integers above 0 I still chose to check that the input was an integer within python before using it. In hindsight, I think it may have eased development if we as a group implemented an error logging system such as pythons built-in logging module, as this would allow us to more easily track down the causes of any errors and remove them or handle them gracefully.