
Buy OnDemand

**Electronic Trading System
Software Requirements Specification
For online Mini-eBay**

Version <2.0>

Buy OnDemand	Version: 2.0
Design Report	Date: 18 / Nov / 22
Second Draft	

Revision History

Date	Version	Description	Author
18 / Nov / 22	2.0	Collaboration Class Diagram, E/R Diagram	Emily
18 / Nov / 22	2.0	Collaboration Class Diagram, E/R Diagram, Use Case, Github Repository	Alice
18 / Nov / 22	2.0	System Screens	Lily
20 / Nov / 22	2.0	Use Case, ER Diagram, Memos of Group Meeting	Emily
20 / Nov / 22	2.0	Use Case, ER Diagram	Alice
20 / Nov / 22	2.0	Use Case, System Screens	Lily
21 / Nov / 22	2.0	System Screens, ER Diagram	Emily
21 / Nov / 22	2.0	Detailed Designs, ER Diagram	Lily
21 / Nov / 22	2.0	ER Diagram, Final Touches	Alice

Buy OnDemand	Version: 2.0
Design Report	Date: 18 / Nov / 22
Second Draft	

Table of Contents

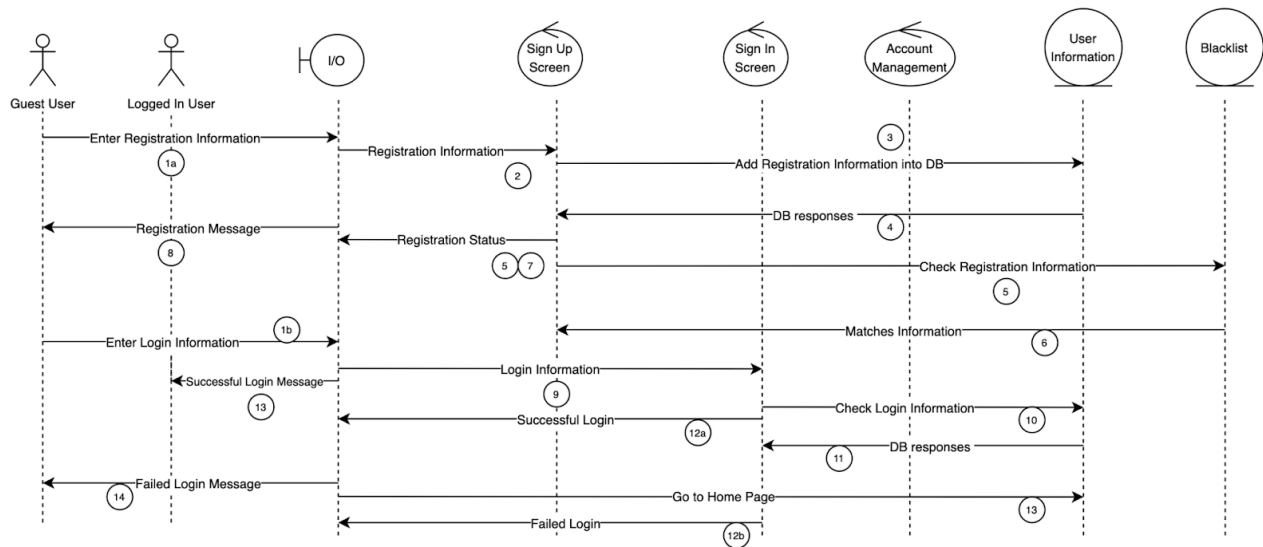
1.	Introduction	4
1.1	Collaboration Class Diagram	4
2.	Use-Cases	4
3.	E-R Diagram	9
4.	Detailed Design	9
5.	System Screens	13
5.1	Major GUI Screens	13
5.2	Sample Prototype	16
6.	Memos of Group Meetings	16
7.	Address of Github Repository	17

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

Design Report

1. Introduction

1.1 Collaboration Class Diagram



2. Use Cases

[Guests Use Case]

Use-Case: Guests apply to be ordinary users meaning to be registered customers

Description: They can only choose to either be a buyer or seller. They will have to be approved by the Admin.

Normal Case: The admin approves the request and the person now becomes a registered customer.

Exceptional Case: The admin rejects the user due to some possible reason, such as, the user's email is in the blacklist, or the email is invalid. This would bring them back to the sign-up page.

Use-Case: Browse the website

Description: Guests can browse the website and see the items being sold. They can filter the items based on ratings and price ranges. When in the "specific item page", if they click on the "add to cart" button, a special message will show up, telling the user to log in or sign up to be a registered ordinary user. There are no exceptional cases for this.

Use-Case: Report an item

Description: If they see an item stolen or anything suspicious they can report it to the Admin.

Normal Case: If proven true, Admin will remove the item and a report will be sent to the police.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

Exceptional Case: Admin will leave the item as is and no report will be sent if the item is authentic.

[Ordinary Users: Buyers Use Case]

Use-Case: Browse the website

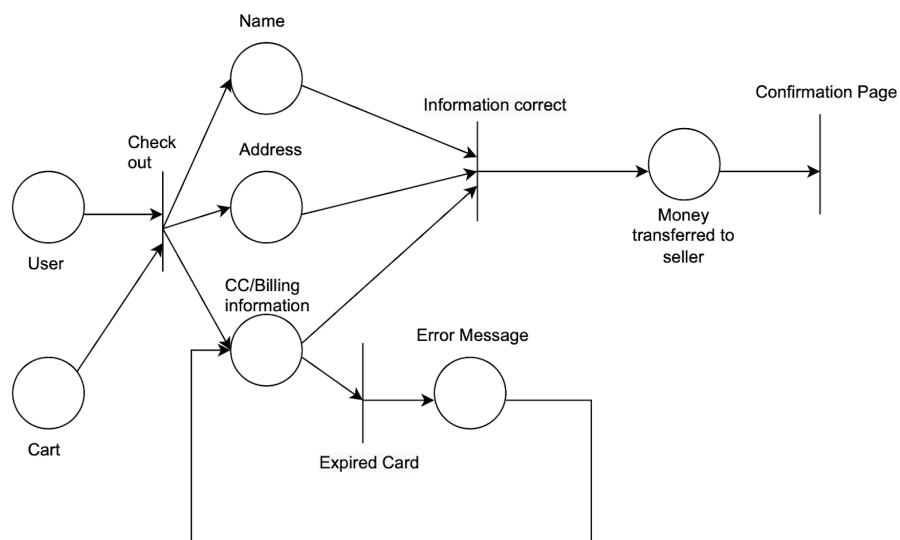
Description: Buyers can browse the website and see the items being sold. They can filter the items based on ratings and price ranges. There are no exceptional cases for this.

Use-Case: Order items and make purchase

Description: Buyers can add items to the cart and order.

Normal Case: Buyers will be brought to the checkout page where they will provide personal information such as name, address, and billing information. Once this is successful it'll bring them to the confirmation page and the buyer's money will be transferred to the seller.

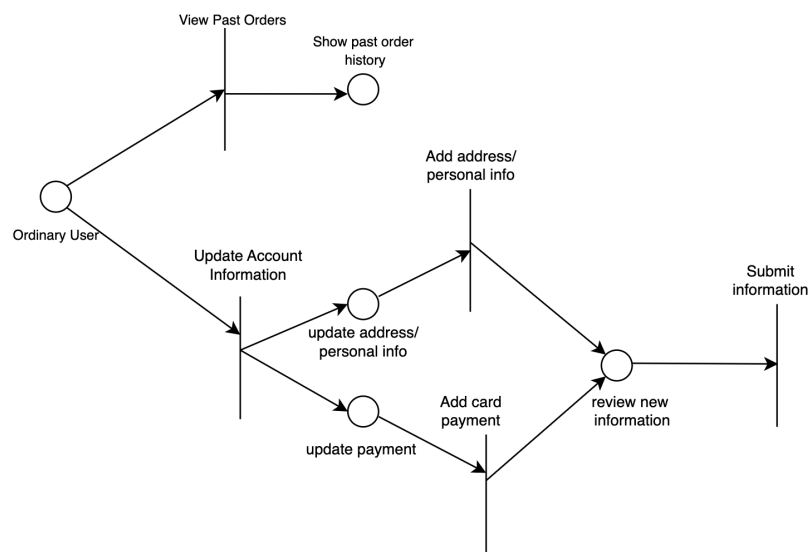
Exceptional Case: If the buyer perhaps used an expired card, they will be given an error message and asked to use a different card.



Use Case: View Order History

Description: The buyer can view their past purchase history. Including the total cost of the order, the quantity of each item, as well as see the rating they left for that order. There are no exceptional cases for this.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	



Use-Case: Post ratings

Description: They can rate their experience with the seller and on the item they purchased. Both ratings will range from 1 (worst) to 5 (best) stars.

Normal Case: If the user rates between 1 and 5, and clicks “post”, then the rating would be successfully posted.

Exceptional Case: If the user doesn’t rate between 1 and 5, say 0, and clicks “post”, then a message will pop up indicating the user to make a rating.

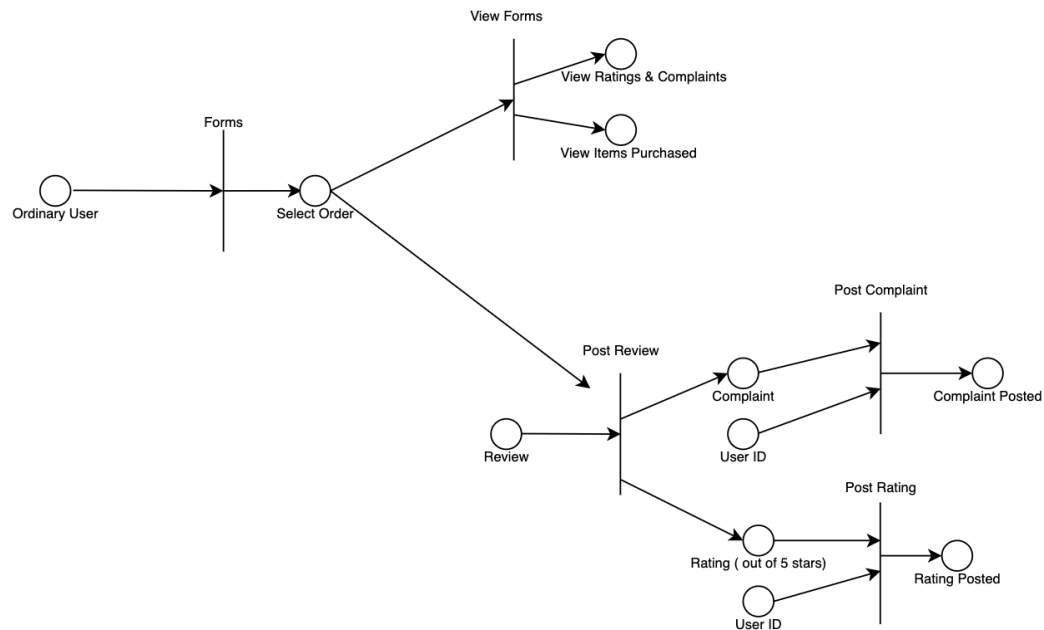
Use-Case: File a dispute

Description: If a complaint is made by the seller about the buyer, they have the right to dispute the complaint.

Normal Case: A dispute is made to a complaint and the Admin will take into account the dispute before making their decision.

Exceptional Case: The dispute will be disregarded.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	



Use Case: Change account information

Description: Buyer is able to change account information--name, password, phone number, and credit card number. There are no exceptional cases.

[Ordinary Users: Sellers Use Case]

Use Case: Add items to sell

Description: Sellers are allowed to add items to sell. This includes title, picture, video (optional), key words, price range, and description of the item.

Normal Case: The user provides a title, picture, price range, and description of the item.

Exceptional Case: If any of these aspects are missing, a special error message will indicate to the user to fill out the missing part.

Use Case: See all orders

Description: Sellers are able to see the history of orders including current and their status. There are no exceptional cases

Use Case: Dispute complaints

Description: If a complaint is made about the seller or product, they have the right to dispute the complaint.

Normal Case: A dispute is made to a complaint and the Admin will take into account the dispute before making their decision.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

Exceptional Case: The dispute will be disregarded.

Use Case: Change account information

Description: Seller is able to change account information--name, password, phone number, credit card number. There are no exceptional cases.

Use Case: Settle bids

Description: Decides if the transaction will proceed. If the seller proceeds, the seller may choose who to sell the item to.

Normal Case: The seller will sell their item to the highest bidder.

Exceptional Case: If the seller decides to choose a bidder that is not the highest, then there will be a reason that is provided from them to the Admin.

[Admin Use Case]

Use Case: Process items to sell

Description: Process items the seller wants to sell. Including bidding. Decides if it should be published for buyers to view and order.

Normal Case: Admin will accept the item(s) the seller is trying to sell. And in the case of bidding, if the buyer's justification of who to sell the item to (if buyer was not the highest bidder) is valid, the admin will accept the item(s) the seller is trying to sell. This means that it can be published for buyers to view and order.

Exceptional Case: Admin will communicate any issues that hinder the seller's item from being successfully processed. This includes bidding justification issues.

Use Case: Manage and settle disputes

Description: They're able to view the complaint(s). They'll ultimately decide if it's a legitimate complaint and what actions need to be made. There are no exceptional cases.

Use Case: Send warnings and removal of ordinary users

Description: Send warnings to both the buyers and sellers.

Normal Case: Admin sends a warning to sellers with 2 complaints and/or their overall average rating is lower than a 2 with at least 3 evaluators. Admin will also send warnings to buyers with invalid/problematic ratings.

Exceptional Case: Admin removes buyer and/or seller with 2 warnings from the system.

Use Case: Collect transaction statistics

Description: Collect transaction statistics, the total amount of charges earned/spent, on a weekly basis of ordinary users. There are no exceptional cases.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

```
def store(request):
    if request.user.is_authenticated:
        customer = request.user.customer
        order, created = Order.objects.get_or_create(customer=customer, complete=False)
        items = order.orderitem_set.all()
        cartItems = order.get_cart_items
    else:
        items = []
        order = {'get_cart_total':0, 'get_cart_items':0, 'shipping':False}
        cartItems = order['get_cart_items']

    products = Product.objects.all()
    context = {'products':products, 'cartItems':cartItems}
    return render(request, 'store/store.html', context)
```

cart: This is a function for the authenticated users to view the items they added in their cart. It checks if the user is authenticated so we access the customer to get an existing order/or create the object for it. Then set items that are all associated to that parent for that order. If the user is not authenticated then we create an empty cart for now for non-logged in users. The information about their items, order, and cartItems are rendered to the cart page.

```
def cart(request):
    if request.user.is_authenticated:
        customer = request.user.customer
        order, created = Order.objects.get_or_create(customer=customer, complete=False)
        items = order.orderitem_set.all()
        cartItems = order.get_cart_items
    else:
        items = []
        order = {'get_cart_total':0, 'get_cart_items':0, 'shipping':False}
        cartItems = order['get_cart_items']

    context = {'items':items, 'order':order, 'cartItems':cartItems}
    return render(request, 'store/cart.html', context)
```

checkout: This is a function that renders the same information from the cart view to the checkout view. This checks if the user is authenticated so we access the customer to get an existing order/or create the object for it. Then set items that are all associated to that parent for that order. If the user is not authenticated then we create an empty cart for now for non-logged in users. The information about their items, order, and cartItems are rendered to the checkout page.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

```
def checkout(request):
    if request.user.is_authenticated:
        customer = request.user.customer
        order, created = Order.objects.get_or_create(customer=customer, complete=False)
        items = order.orderitem_set.all()
        cartItems = order.get_cart_items
    else:
        items = []
        order = {'get_cart_total':0, 'get_cart_items':0, 'shipping':False}
        cartItems = order['get_cart_items']

    context = {'items':items, 'order':order, 'cartItems':cartItems}
    return render(request, 'store/checkout.html', context)
```

updateItem: This is a function that updates the quantity of items in the cart page. This parses our data so that it can set the value of data. We can access values as a python dictionary to print the action and the product. Then we query that customer and get the product we are passing in to either create or add to the order. We change the order that already exists whether they wanna add or remove the quantity of that item which then saves it. If the item the user wants to remove is less than or equal to 0, we remove it from the cart.

```
def updateItem(request):
    data = json.loads(request.body)
    productId = data['productId']
    action = data['action']
    print('Action:', action)
    print('Product:', productId)

    customer = request.user.customer
    product = Product.objects.get(id=productId)
    order, created = Order.objects.get_or_create(customer=customer, complete=False)

    orderItem, created = OrderItem.objects.get_or_create(order=order, product=product)

    if action == 'add':
        orderItem.quantity = (orderItem.quantity + 1)
    elif action == 'remove':
        orderItem.quantity = (orderItem.quantity - 1)

    orderItem.save()

    if orderItem.quantity <= 0:
        orderItem.delete()

    return JsonResponse('Item was added', safe=False)
```

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

processOrder: This is a function that processes the user's order after they checkout. This allows us to get the order or create the order and the total amount which then queries the items inside a python dictionary. If the total cost matches the order of the total, we then save the order which then creates an instance of the shipping address and then it responds with a json response saying that the order is complete. The user must be logged in order for them to add items to their cart.

```
def processOrder(request):
    #print('Data:', request.body)
    transaction_id = datetime.datetime.now().timestamp()
    data = json.loads(request.body)
    if request.user.is_authenticated:
        customer = request.user.customer
        order, created = Order.objects.get_or_create(customer=customer, complete=False)
        total = float(data['form']['total'])
        order.transaction_id = transaction_id

        if total == order.get_cart_total:
            order.complete = True
            order.save()

            ShippingAddress.objects.create(
                customer=customer,
                order=order,
                address=data['shipping']['address'],
                city=data['shipping']['city'],
                state=data['shipping']['state'],
                zipcode=data['shipping']['zipcode'],
            )
    else:
        print('User is not logged in')
    return JsonResponse('Payment submitted..', safe=False)
```

get_cart_total: This is a function to calculate the total cost in the user's cart. This gets the total of our cart value by running a loop to sum up the totals for each item in their cart which is then returned.

```
def get_cart_total(self):
    orderitems = self.orderitem_set.all()
    total = sum([item.get_total for item in orderitems])
    return total
```

get_total: This is a function used to calculate the amount in their cart page for each item. This is calculated by deriving the total price from the product price multiplied by the quantity.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

```
def get_total(self):
    total = self.product.price * self.quantity
    return total
```

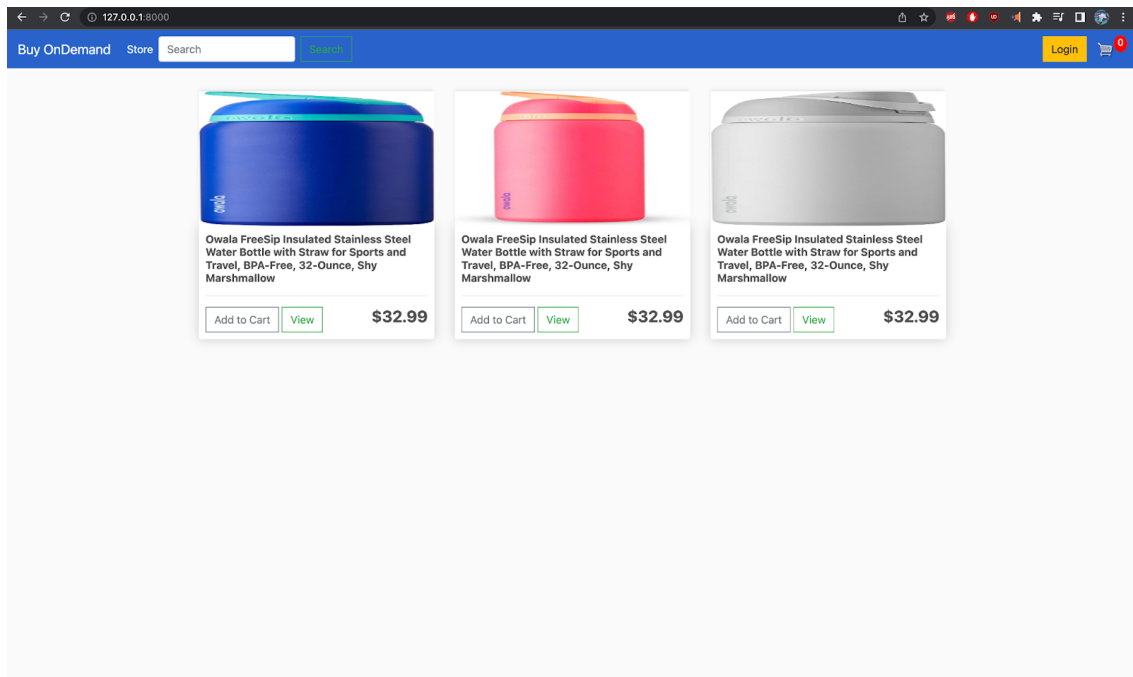
get_cart_items: This is a function to calculate the total number of items in a user's cart. This gets the total quantity of our cart value by running a loop to sum up the quantity for each item in their cart which is then returned.

```
def get_cart_items(self):
    orderitems = self.orderitem_set.all()
    total = sum([item.quantity for item in orderitems])
    return total
```

5. System Screens

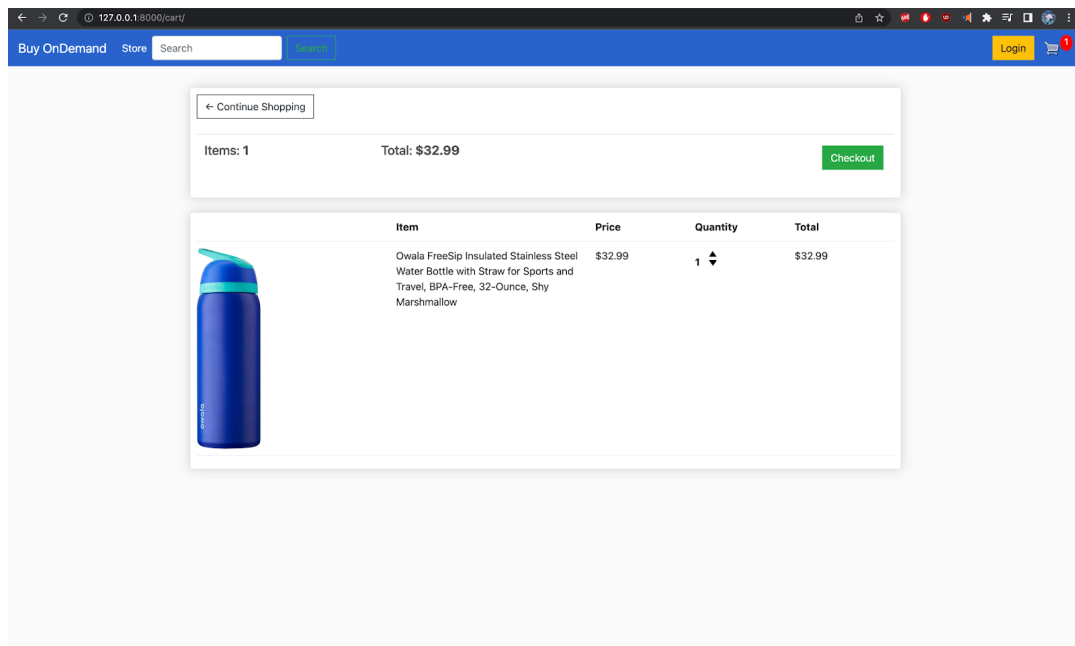
5.1 Major GUI Screens

The Landing/ Homepage of the Buy OnDemand website

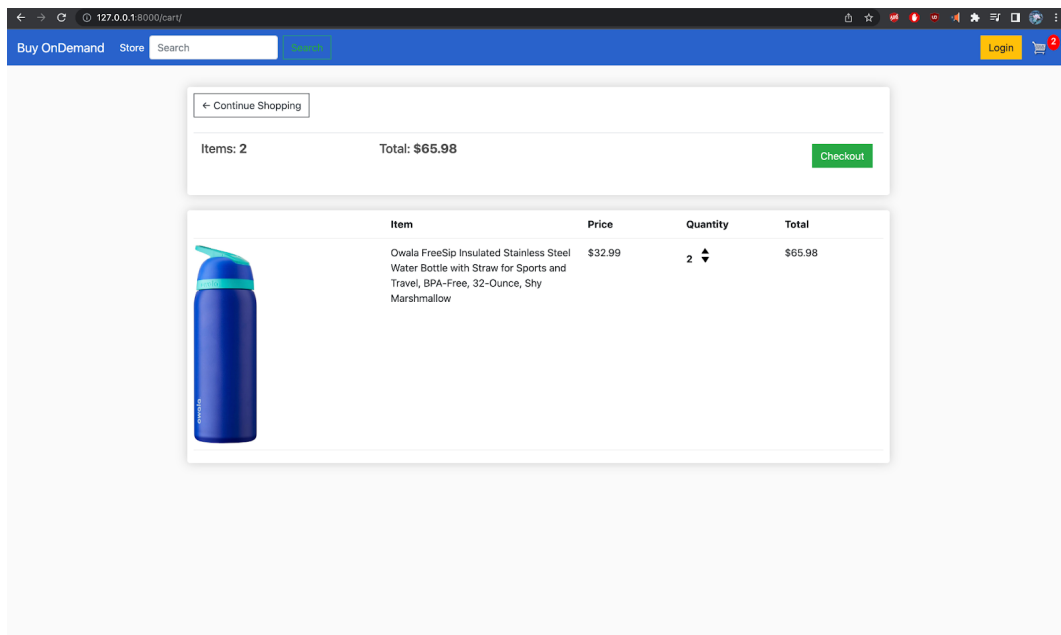


Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

Checkout Page

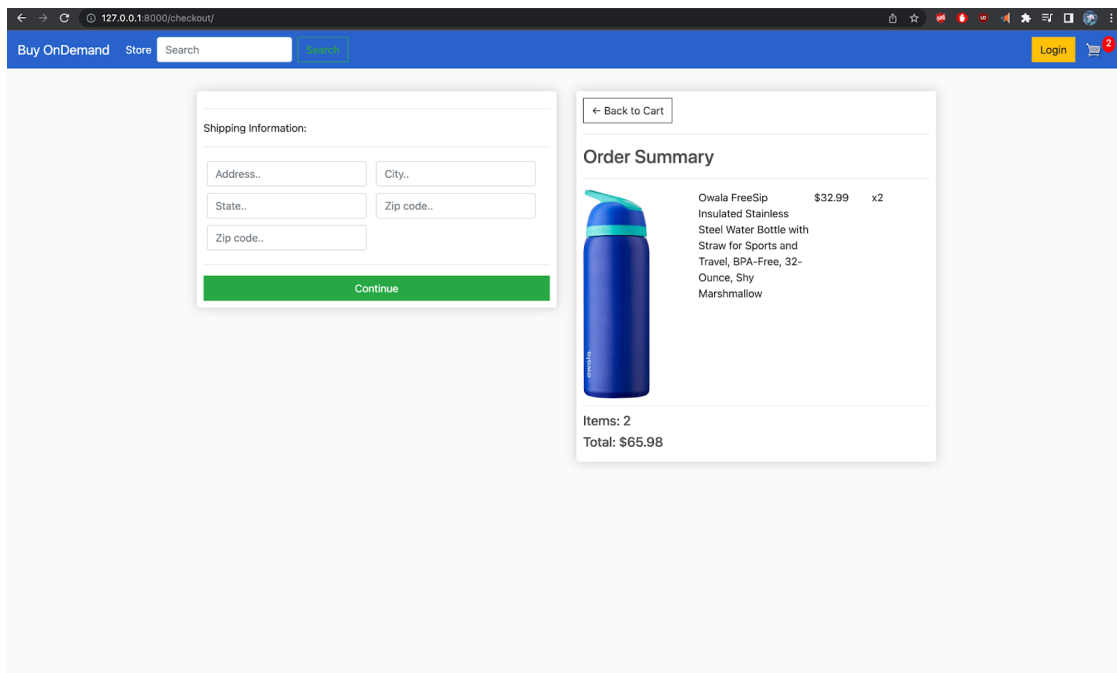


Editing Quantity of Item(s)



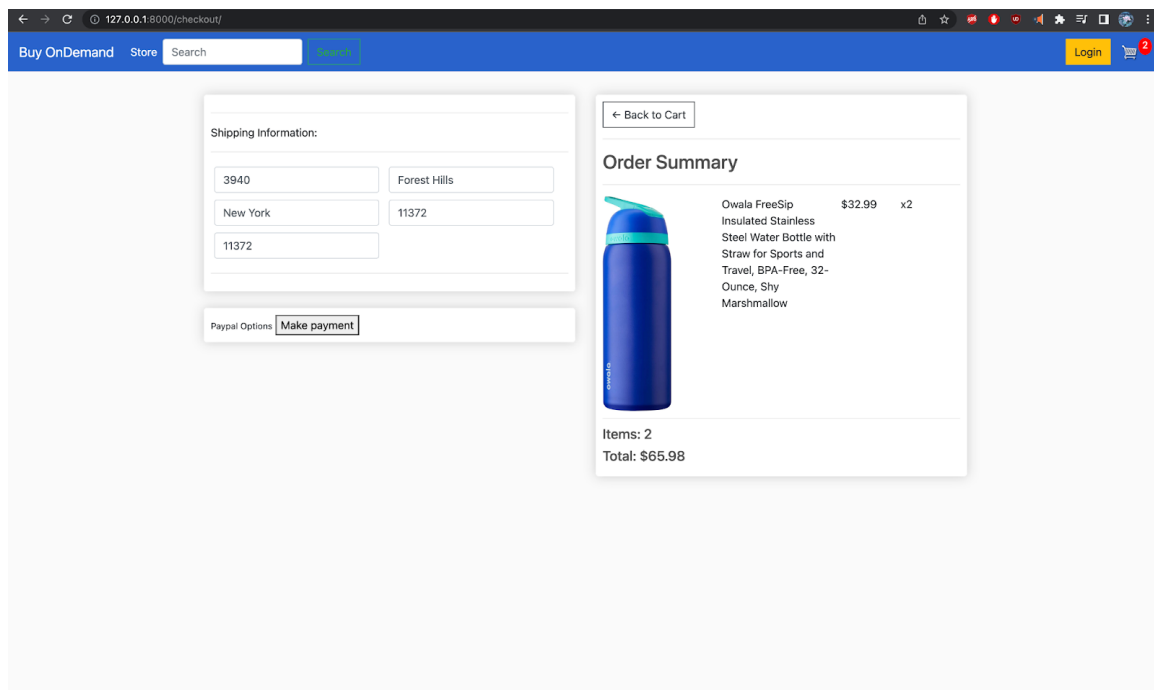
Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

Adding Shipping Information + Order Summary



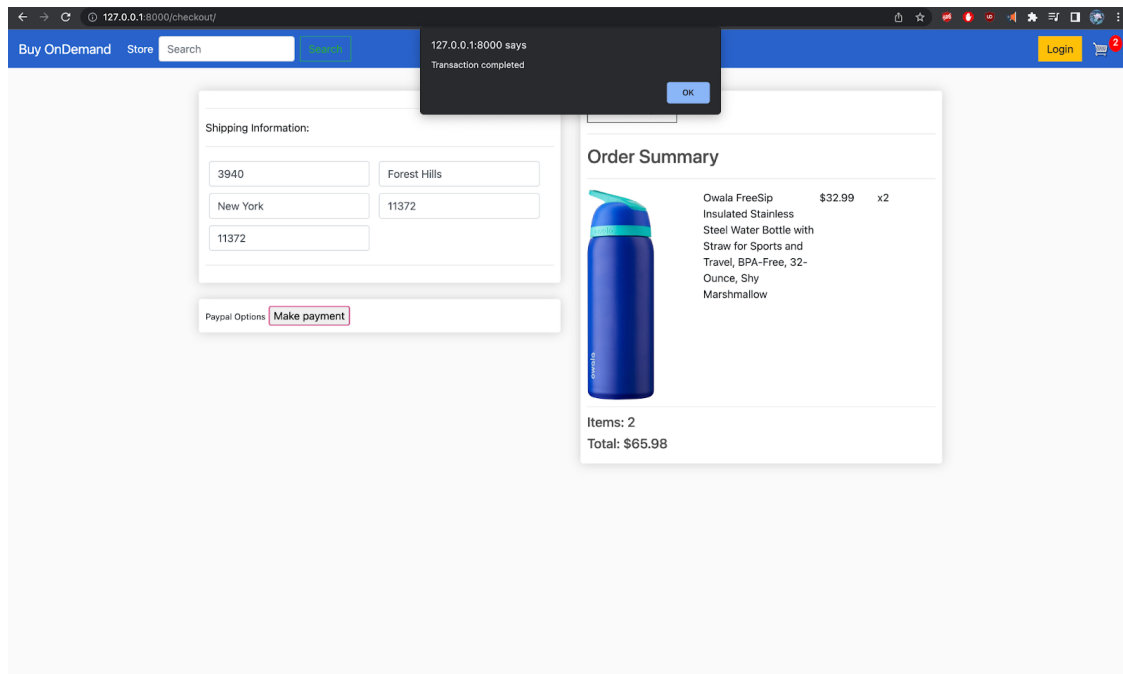
The screenshot shows a web browser window with the URL 127.0.0.1:8000/checkout/. The page has a blue header with "Buy OnDemand", "Store", a search bar, a "Search" button, a "Login" button, and a shopping cart icon with a red "2" badge. The main content area is divided into two columns. The left column is titled "Shipping Information:" and contains four input fields: "Address..", "City..", "State..", and "Zip code..". Below these fields is a green "Continue" button. The right column is titled "Order Summary" and features a "Back to Cart" button. It displays a blue water bottle with a green straw. The text next to the bottle reads: "Owala FreeSip Insulated Stainless Steel Water Bottle with Straw for Sports and Travel, BPA-Free, 32-Ounce, Shy Marshmallow". The price is "\$32.99" and the quantity is "x2". At the bottom of the summary, it says "Items: 2" and "Total: \$65.98".

Making A Payment



This screenshot is similar to the previous one but shows the "Making A Payment" stage. The "Shipping Information" section now has pre-filled values: "3940" for Address, "Forest Hills" for City, "New York" for State, and "11372" for Zip code. Below the shipping fields is a "Paypal Options" button and a "Make payment" button. The "Order Summary" section remains the same, showing the water bottle, price, and total of \$65.98.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	



5.2 Sample Prototype

We recorded a video of being on our landing page and the checkout process of the web application. The link to the video is provided here: [demo 1.mp4](#)

6. Memos of Group Meetings

The first meeting we had was at the beginning when the project was finalized by Professor Wei. We met, exchanged contact information with one another, and discussed what programming language and platforms we intended to employ for the project. We decided on using Django for the frontend and SQLite for the database. Then, the second meeting we had was to divide the phase 1 report to see what each person's task/responsibilities would be. We decided on the following:

Alice Liu: References, Overview, Use-Case Model Survey, Use-Case Reports, Use-Case Diagram.

Lily Liang: Scope, Definition, Acronyms, Abbreviations, Supplementary Requirements, Supporting Information, Use Case Diagram.

Emily Kim: Purpose, Overview, Use-Case Model Survey, Assumptions and Dependencies, Use-Case Diagram.

Then we had our last meeting about the phase 2 report for this project. We discussed what each person's task/responsibilities would be for this phase. We decided on the following:

Alice Liu: Collaboration Class Diagram, E/R Diagram, Use-Case, Github Repository.

Buy OnDemand	Version: 2.0
Software Requirements Specification	Date: 18 / Nov / 22
Second Draft	

Lily Liang: System Screens, Use-Case, Detailed Design, E/R Diagram.

Emily Kim: Collaboration Class Diagram, E/R Diagram, Use-Case, System Screens, Memos of Group Meetings.

We kept in touch via texts throughout the entire process, and if somebody needed anything, one or two people would try to assist. If the issue could not be resolved through chat, we would immediately start a Zoom call. We had no issues. Every time we made modifications to the code, we promptly let everyone in the group know about them. We only permitted one person to work simultaneously on the backend programming because the errors would be difficult to correct, especially if everyone is pushing and pulling the code through Github. Overall, we have not yet encountered any problems.

7. Github Repository Address

Address to our Github Repository: <https://github.com/AliceLiu17/CSC322-Ebay>