COEN 6312 – MODEL DRIVEN SOFTWARE ENGINEERING

GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

DELIVERABLE 3

E-COMMERCE SYSTEM

Submitted to:

Dr. Wahab Hamou-Lhadj

Submitted by:

Karanjot Singh Kochar (40161109)

Navdeep Singh (40161110)

Gurveen Kaur Bhaweja (40164885)

Sri Harsha Vellaturi (40185295)

Srinath Ananthula(40174198)

Parul Maini (40181624)

CONTENTS

LIST OF FIGURES

24. Update the availability of the product by seller.
25. State diagram for Remove products.
26. Remove products from the list of products offered by the seller.
27. State diagram for Update order status task performed by the seller.
28. Shipment status of the order provided by the seller.
29. State diagram for Create order.
30. Create order and inform the seller about the order.
31. State diagram for Cancel order task performed by the buyer.
32. Buyer can cancel the order only if the status is “IN PROCESSING”.
33. State diagram for Add rating task performed by the buyer.
34. Buyer can rate products after delivery

1. State diagrams for buyer class

- **Buyer-Sign up:**

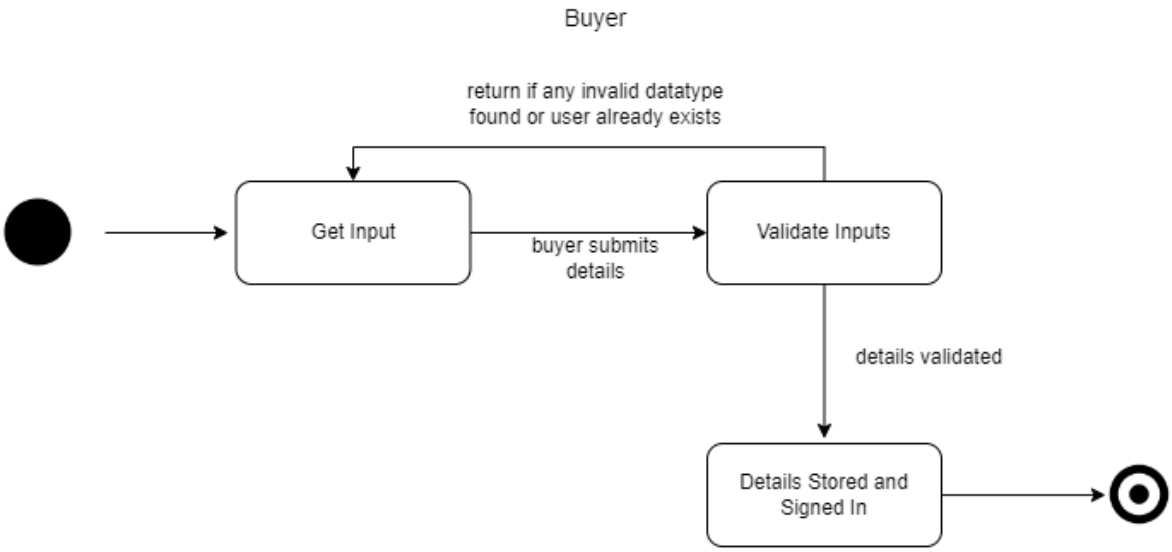

Fig 1 – State diagram to perform Sign-up task by the buyer.

In this state, the object has three states namely Get Input, Validate Inputs, and Details Stored and Signed In. The first state is to get details from the buyer. Here the buyer enters the details for creating an account on the website. In the second state, the entered details by the buyer are validated as per the given input datatype and the system checks whether the user is creating a new account or not. This is done by checking whether the email address exists in the database or not. If the input datatype field is validated and the user is new then it goes to the third stage else it goes back to the first stage. In the third stage, the buyer details are stored and Signed In to the website successfully.

```
432
433     @staticmethod
434     def signup(email_id, password, phone_number, name, address):
435         check=Buyer.check_for_existance(email_id, phone_number)
436         if check:
437             add=[]
438             ad
439             b=  (method) save: () -> None                hone_number, name, add)
440             b.save()
441             c=Cart(email_id, add[0])
442             c.save()
443             o=OrderHistory(email_id)
444             o.save()
445             print('Buyer Signed up successfully')
446             return True
447         else:
448             return False
449
450     def update_buyer(self):
451         data=self.get_json()
452         databasemodel.update_buyer(data)
```

Fig 2 – Sign-Up task performed by the buyer.

- **Buyer-Login:**

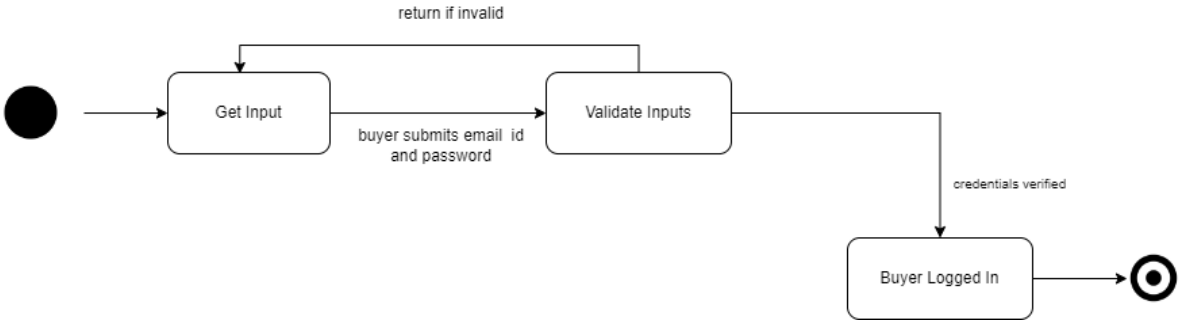

Fig 3 – State diagram for Login task performed by the buyer.

In this the initial state is to get the login credentials from the buyer. When buyer submits the login credentials, they are moved into second state where the login credentials are verified. If the credentials are invalid, then it will again go back to the initial state else the object will move to the final state which is logged in.

```
529
530     @staticmethod
531     def login(email_id, password):
532         data=databasemodel.read('buyers.json')
533         buyers=data['buyers']
534         if len(buyers)==0:
535             print('Buyer does not exist')
536             return False
537 
538         elif len(buyers)>0:
539             for i in buyers:
540                 if i['email_id']==email_id:
541                     print('Buyer exists! Verifying password')
542                     if i['password']==password:
543                         print('Buyer logged in')
544                         return Buyer.find_buyer(email_id)
545                     else:
546                         print('Wrong Password')
547                         return False
548         else:
549             print("Buyer does not exist")
550             return False
551 
```

Fig 4 – Login task performed by the buyer.

- **Buyer-Update details:**

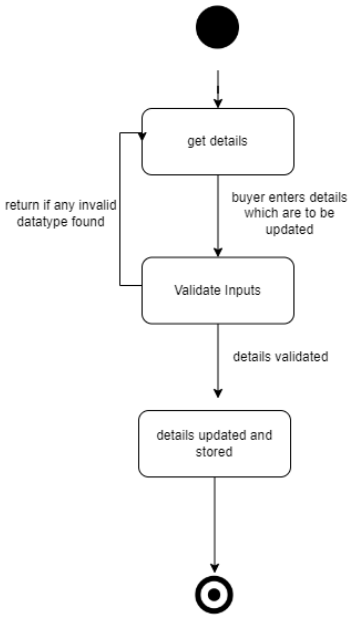

Fig 5 – State diagram for Updating details task performed by the buyer.

In this, the initial state is to get the details which are previously filled by the buyer. The buyer enters the details which need to be updated. When buyer submits the details, they are moved into second state where the details are verified as per the given input datatype. If the details are invalid then it will again go back to the initial state else the object will move to the final state where the details are updated and stored in the database.

```
473     def update_information(self):
474         t=int(input('Which information you want to update\n1. Update Password\n2. Update Phone Number\n3.
475         if t==1:
476             new_password=input('Enter New Password')
477             self.password=new_password
478             self.update_buyer()
479             print('Password Updated successfully')
480             return True
481         elif t==2:
482             new_phone_number=input('Enter new phone number')
483             data=databasemodel.read('buyers.json')
484             buyers=data['buyers']
485             for i in buyers:
486                 if i['phone_number']==new_phone_number:
487                     if i['email_id']==self.email_id:
488                         print('Phone Number already linked with your account')
489                         return False
490                     else:
491                         print('Phone Number already linked with some other account')
492                         return False
493             self.phone_number=new_phone_number
494             self.update_buyer()
495             print('Phone number Updated successfully')
496             return True
497         elif t==3:
498             if len(self.address)<3:
499                 address=input('Add Address')
```

```
            return True
        elif t==3:
            if len(self.address)<3:
                address=input('Add Address')
                for i in self.address:
                    if i==address:
                        print('Address Entered is already linked with this account')
                        return False
                self.address.append(address)
                self.update_buyer()
                print('Address list Updated successfully')
                return True
            else:
                print('No more new addresses can be added')
                return False
        else:
            print('Incorrect Input')
            return False
```

Fig 6 – Update details by the buyer.

- **Buyer- Add to wish list:**

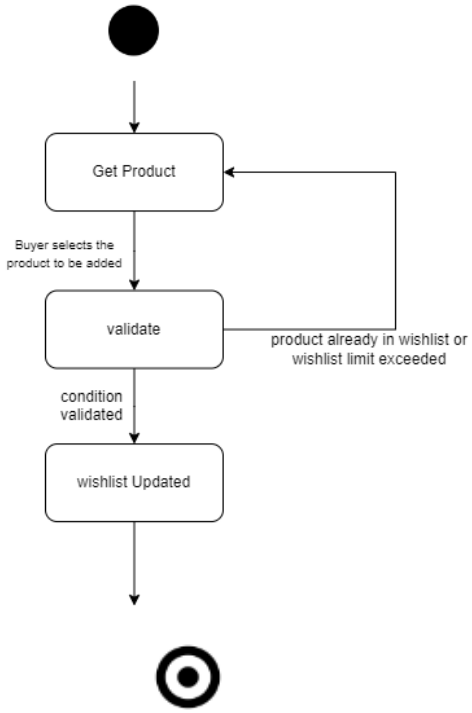

Fig 7 – State diagram for add products to the Wishlist by the buyer.

In the initial state the buyer selects the products which should be added to the Wishlist. When the buyer selects the products to be added into the Wishlist, they are moved into second state where the products are validated. If the products in the Wishlist are already added or Wishlist limit exceeded, then it will go back to the initial state else the object will move to the final state where the Wishlist is update.

```
358
359     def add_to_wishlist(self, p):
360         for i in self.wishlist:
361             if i==p:
362                 print("product is already added in wishlist")
363                 return False
364         else:
365             if len(self.wishlist)<10:
366                 self.wishlist.append(p)
367                 self.update_buyer()
368                 print('Item added in wishlist successfully')
369                 return True
370             else:
371                 print('No more products can be added in the wishlist')
372                 return False
373
```

Fig 8 – Adding products to the Wishlist by the buyer.

2. State diagrams for Cart:

- **Cart - Add Product:**

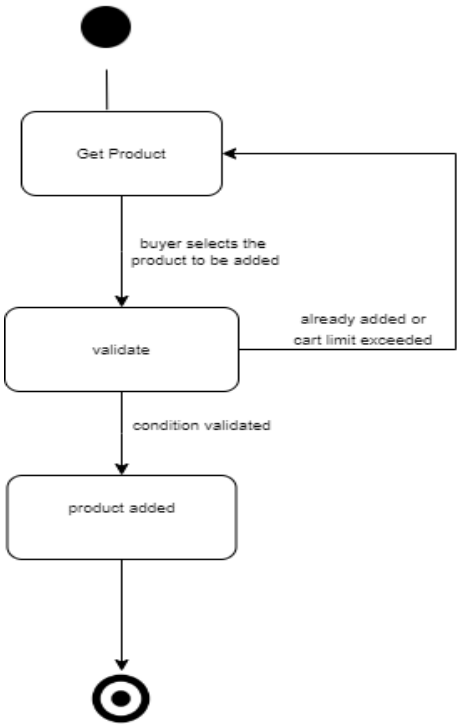

Fig 9 – State diagram for add product task performed by the buyer.

In the initial state the buyer selects the products which are to be added to the cart. When the buyer selects the products to be added in the cart, they are moved into second state where the products are validated. If the products in the cart are already added or the cart limit exceeded, then it will go back to the initial state else the object will move to the final state where all the products are added to the cart.

```
    def add_product(self, p):
        for i in self.products:
            if i==p:
                print('item is already in the cart')
                return False
        if len(self.products)<5:
            a=Product.get_product(p)
            if a.availability=='out of stock':
                print('product is out of stock\n')
            else:
                self.products.append(p)
                self.update_cart()
                print('product added in cart')
                return True
        else:
            print('cart is already full')
            return False
```

Fig 10 – Add products into the cart by the buyer.

- **Cart - Manage address and place order:**

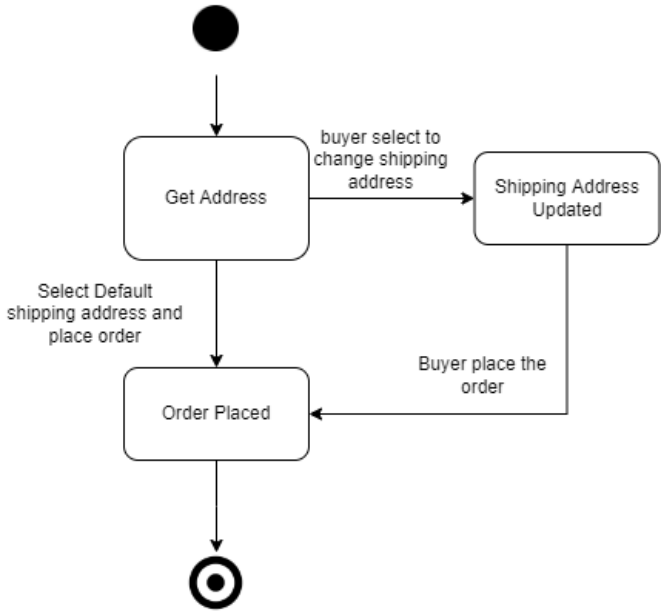

Fig 11 – State diagram for Manage address and Place order task performed by the buyer.

In the initial state the buyer gets the list of address linked to the account. If the buyer wants to change shipping address, this can be done in the second stage where the buyer can update the shipping address and proceed to the final state where the order is placed. If the buyer wants the shipment to be shipped to the default address, then transition occurs directly from initial state to final state.

```
    def manage_address(self):
        print(f"Current shipping address is: {self.shipping_address}")
        t=int(input('1. keep this shipping address as it is\n 2. Change shipping address'))
        if t==1:
            print('Shipping address is not changed')
            return True
        elif t==2:
            print('Select new Shipping address\n')
            data=databasemodel.read('buyers.json')
            buyers=data['buyers']
            for i in buyers:
                if i['email_id']==self.buyer_email:
                    k=1
                    for a in i['address']:
                        print(f'{k}. {a}')
                        print('\n')
                        k=k+1
                    sel=int(input('Enter the adress number which you want to set as shipping address'))
                    self.shipping_address=i['address'][sel-1]
                    self.update_cart()
                    print('Shipping address is updated')
                    print(f"New shipping address is: {self.shipping_address}")
                    return True
        else:
            print('wrong input')
            return False
```

```
        def place_order(self):
            if len(self.products)>=1:
                print("your cart has these products:\n")
                data=databasemodel.read('products.json')
                products=data['products']
                k=1
                for i in self.products:
                    for p in products:
                        if p['product_id']==i:
                            print(f'{k}. ')
                            pprint(p)
                            print('\n')
                            k=k+1
                t=input("\ndo you want to continue (yes or no)\n")
                if t=='yes':
                    products_ordered=[]
                    for i in self.products:
                        products_ordered.append(i)
                    buyer_email=self.buyer_email
                    status=[]
                    for i in self.products:
                        a={'product': i,
                            'status':'in processing'}
                        status.append(a)
                    now=datetime.now()
                    n=now.strftime("%d/%m/%Y %H:%M:%S")
                    a=n.split(' ')
                    b=a[1].split(':')
                    c=b[0]+'-'+b[1]+'-'+b[2]
                    order_id=self.buyer_email+'-'+a[0]+'@'+c
```

```
                for i in self.products:
                    a={'product': i,
                       'status':'in processing'}
                    status.append(a)
                now=datetime.now()
                n=now.strftime("%d/%m/%Y %H:%M:%S")
                a=n.split(' ')
                b=a[1].split(':')
                c=b[0]+'-'+b[1]+'-'+b[2]
                order_id=self.buyer_email+'-'+a[0]+'@'+c
                order_placed=Order.create_order(order_id, products_ordered, status, buyer_email)
                a=OrderHistory.get_order_history(buyer_email)
                a.orders.append(order_placed['order_id'])
                a.update_order_history()
                self.products.clear()
                self.update_cart()
                print('Order Placed Successfully')
                return True
            else:
                print('\nOrder not placed\n')
        else:
            print('The cart is empty, NO order can be placed')
```

Fig 12- Manage address and place order tasks performed by the buyer.

- **Cart-remove products:**

Fig 13 – State diagram for remove products task.

In this state the buyer first selects the products and then remove them from the cart.

```
32      def remove_product(self):
33          if len(self.products)>0:
34              print('Which item you want to remove from cart')
35              k=1
36              data= databasemodel.read('products.json')
37              products=data['products']
38              for i in self.products:
39                  for p in products:
40                      if p['product_id']==i:
41                          print(f'{k}..')
42                          pprint(p)
43                          print('\n')
44                          k=k+1
45              t=int(input('Enter the number for the item you want to remove\n'))
46              self.products.pop(t-1)
47              self.update_cart()
48              print('Item removed successfully')
49              return True
50          else:
51              print("There are no products in the cart\n")
```

Fig 14 – Remove products from the cart by the buyer.

3. State diagrams for Seller class:

- **Seller-Sign up:**

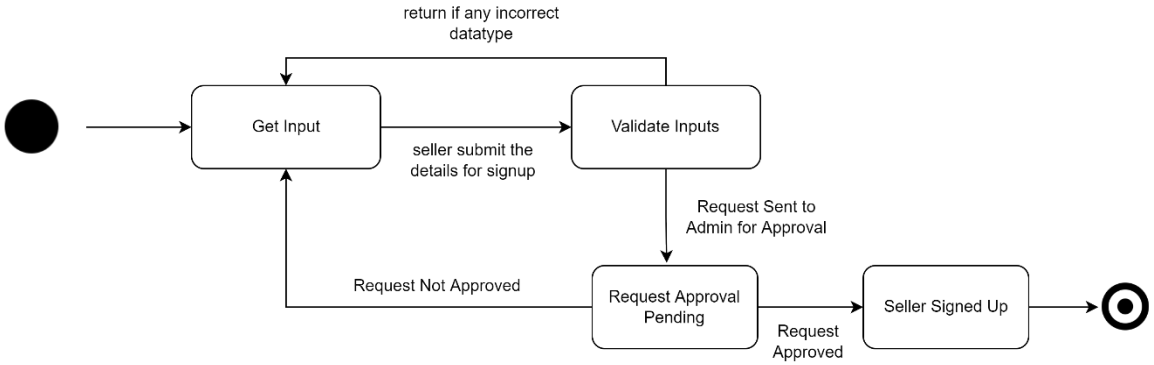

Fig 15 – State diagram for sign-Up task performed by the Seller.

In this state, the object mainly has 4 states namely Get input, validate inputs, request approval pending, and Seller signed up. In the first state, seller needs to sign up to the website using email address, phone number company name, business address, years of experience, account number and password. In the second state, these details are now validated. If the details do exist on the website or if there is any datatype issue, the signup request will be returned to signup page displaying a message “Details already exists or wrong details”. After Validation of the submitted details, now at the third state, signup request is sent to Admin for approval. After cross checking all the details, if the user details successfully pass the conditions, user account will be approved, and Seller is now signed up. Else, if the details submitted are wrong then the signup request is not approved.

```python
    @staticmethod
    def request_signup(email_id, password, phone_number, company_name, business_address, years_of_experienc
        data=databasemodel.read('sellers.json')
        sellers=data['sellers']
        if len(sellers)>0:
            for i in sellers:
                if i['email_id']==email_id or i['phone_number']==phone_number or i['company_name']==company
                    print('Seller already existed! Request Denied')
                    return False
        a={'email_id':email_id,
            'password':password,
            'phone_number':phone_number,
            'company_name':company_name,
            'business_address':business_address,
            'years_of_experience':years_of_experience,
            'account_number':account_number,
            'institution_number':institution_number,
            'transit_number':transit_number}
        admin=Admin.get_object()
        admin.seller_requests.append(a)
        admin.update_admin()
        print("sign up request received")
        return True
```

```python
    @staticmethod
    def approve_request(email_id, password, phone_number, company_name, business_address, years_of_experier
        s=Seller(email_id, password, phone_number, company_name, business_address, years_of_experience, acc
        s.save()
        print('seller registered successfully')
        return True
```

Fig 16 – Sign-up task performed by the seller.

- **Seller-login:**

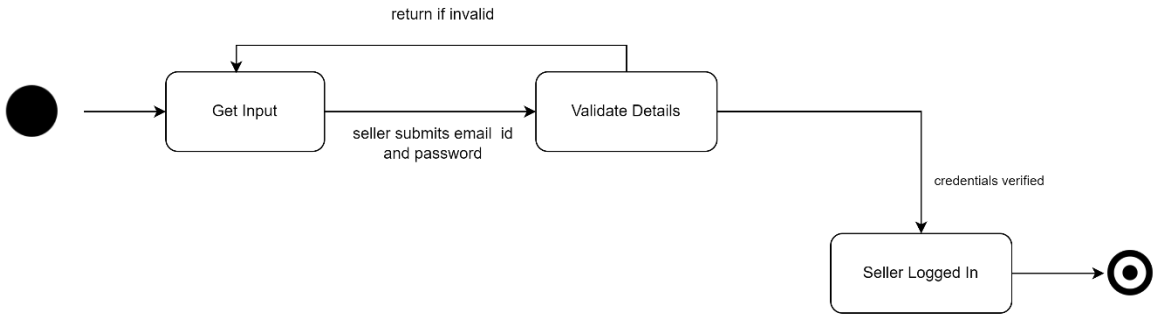

Fig 17- State diagram for Login task performed by the seller.

In this the initial state, seller needs to provide the inputs and then is should be approved by the admin of the website. Once the seller is approved, user can now login into the account. When seller submits the login credentials, they are moved into second state where the login credentials are verified. If the credentials are invalid, then it will again go back to the initial state else the object will move to the final state which is logged in.

```
320     @staticmethod
321     def login(email_id, password):
322         data=databasemodel.read('sellers.json')
323         sellers=data['sellers']
324         if len(sellers)==0:
325             print('Seller does not exist')
326             return
327 
328         elif len(sellers)>0:
329             for i in sellers:
330                 if i['email_id']==email_id:
331                     print('Seller exists! Verifying password')
332                     if i['password']==password:
333                         print('Seller logged in')
334                         return Seller.find_seller(email_id)
335                     else:
336                         print('Wrong Password')
337                         return
338             print("Seller does not exist")
339             return
340         else:
341             print("Seller does not exist")
342             return
343 
```

Fig 18 – Login task performed by the seller.

- **Seller-update details:**

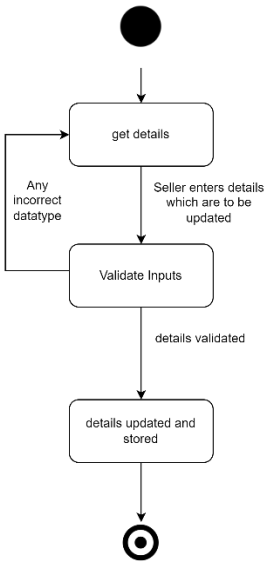

Fig 19 – State diagram for update details by seller.

In this, the initial state is to get the details which are previously filled by the seller. The seller enters the details which need to be updated. When seller submits the details, they are moved into second state where the details are verified as per the given input datatype. If the details are invalid or having any datatype issues, then it will redirect to the initial state. Else, the object will move to the final state where the details are updated and stored in the database.

```
285     def update_details(self):
286         i=int(input("\nEnter 1. To update Password 2. To Update years of experience 3. To update account nu
287         if i==1:
288             password=input("\nEnter new password")
289             self.password=password
290             self.update_seller()
291             print("\nPassword Updated\n")
292             return
293         elif i==2:
294             exp=input("\nEnter updated value for years of experience\n")
295             self.years_of_experience=exp
296             self.update_seller()
297             print("\nyears of experience updated\n")
298             return
299         elif i==3:
300             acc_number=input("\nEnter new account number\n")
301             self.account_number=acc_number
302             self.update_seller()
303             print("\nAccount number updated")
304             return
305         elif i==4:
306             inst_number=input("\nEnter new Institution number\n")
307             self.institution_number=inst_number
308             self.update_seller()
309             print("\nInstitution number updated\n")
310             return
311         elif i==5:
312             tran_number=input("\nEnter new Transit number\n")
313             self.transit_number=tran_number
314             self.update_seller()
315             print("\nTransit number Updated\n")
316             return
317         else:
318             print("\nInvalid Input\n")
319             return
```

Fig 20 – Update the details by the seller.

- **Seller- add products:**

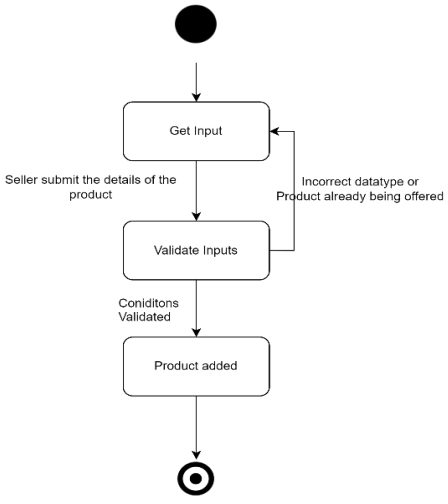

Fig 21 – State diagram for add products task performed by the seller.

This state diagram validates and then adds new products to the list. This is done in three states. In first state, seller will give the input to add a product into the list. Later, it is validated based on the datatype requirements and the product existence. If ever, the product ID is matching with the existing list, the seller will be redirected to the products page with a comment “cannot add the product, as the product ID matched with the existing products” or if there is any datatype issue, the user is redirected to products page. Else, in the third stage, product is successfully added to the list.

```
149     def add_product(self):
150         p_brand=input('\nenter the product brand\n')
151         p_name=input('\nenter product name\n')
152         p_seller=input('\nenter your company name\n')
153         p_category=input('\nEnter product category (choose from electronics, clothing, accessories, food it
154         p_description=input('\nEnter some product description\n')
155         p_price=float(input('\nEnter the price of product\n'))
156         p_availability=input('\nEnter product availability i.e. Whether it is "in stock", "will be out soon
157         p_size=input('\nEnter product size/weight whatever is applicable\n')
158         p_id= p_brand + p_name + p_seller + p_size
159         p_id=p_id.upper()
160         for i in self.products_offered:
161             if i==p_id:
162                 print('This product is already offered by you')
163                 return False
164         a=Product.create_product(p_brand, p_name,p_seller,p_category, p_description, p_price, p_availabilit
165         if a:
166             self.products_offered.append(a['product_id'])
167             self.update_seller()
168             print('Product added successfully')
169             return True
170         else:
171             print("error adding product")
172             return False
```

Fig 22 - Add products to the list of products offered by the seller.

- **Seller- update products:**

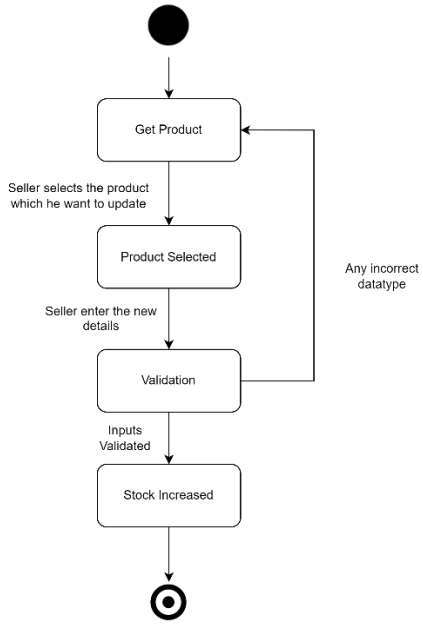

Fig 23 – State diagram for update products by the seller.

In this state diagram, seller can update the product availability on the website. In first stage, product list is retrieved. In second state, seller will be selecting a particular product that needs to be updated. In the third state, seller will enter the new details of the products that needs to be displayed on

the website. If the input has any datatype issue, user is redirected to the products page and the details are not updated. Else, if the input validated successfully, product value is updated accordingly.

```
240     def update_product(self):
241         if len(self.products_offered)>0:
242             print('\nselect the product which you want to update\n')
243             k=1
244             data=databasemodel.read('products.json')
245             products=data['products']
246             for i in self.products_offered:
247                 for p in products:
248                     if i==p['product_id']:
249                         print(f'{k}.....')
250                         pprint(p)
251                         print('\n')
252                         k=k+1
253             t=int(input('Enter the specific number'))
254             if t<=len(self.products_offered):
255                 for z in products:
256                     if z['product_id']==self.products_offered[t-1]:
257                         pprint(z)
258                         print('\n')
259                 l=int(input('\nSelect 1 if you want to update the price or 2 if you want to update the avai
260                 j=input('\nEnter the value: ')
261                 if l==1:
262                     for m in products:
263                         if m['product_id']==self.products_offered[t-1]:
264                             m['price']=j
265                         databasemodel.update_product(m)
266                     print('Update Successful')
267                     return True
268                 elif l==2:
269                     for m in products:
```

```
265                         databasemodel.update_product(m)
266                     print('Update Successful')
267                     return True
268                 elif l==2:
269                     for m in products:
270                         if m['product_id']==self.products_offered[t-1]:
271                             m['availability']=j
272                         databasemodel.update_product(m)
273                     print('Update Successful')
274                     return True
275                 else:
276                     print('Invalid Input')
277                     return False
278             else:
279                 print('Invalid Input')
280                 return False
281         else:
282             print("\nNo Products are being offered by you!\n")
283             return False
```

Fig 24 - Update the availability of the product by seller.

- **Seller - remove products:**

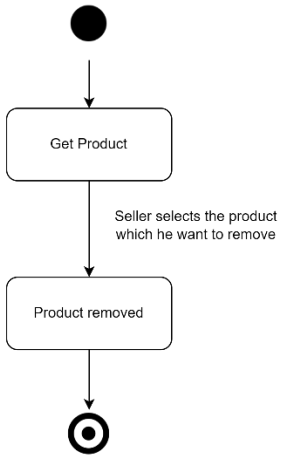

Fig 25 - State diagram for Remove products.

This State Diagram has 2 states, get product list and Products removed. Seller now checks the list of products and based on the availability seller is authorized to remove the products from the list in the final state.

```
193     def remove_product(self):
194         data=databasemodel.read('products.json')
195         products=data['products']
196         if len(self.products_offered)>0:
197             print('select the product which you want to remove')
198             k=1
199             for i in self.products_offered:
200                 for p in products:
201                     if i==p['product_id']:
202                         print(f'{k}.....')
203                         pprint(p)
204                         print('\n')
205                         k=k+1
206             t=int(input('Enter the specific number'))
207             if t<=len(self.products_offered):
208                 p=self.products_offered.pop(t-1)
209                 j=0
210                 for i in products:
211                     if i['product_id']==p:
212                         a=products.pop(j)
213                         databasemodel.delete_product(a)
214                         print('product removed successfully')
215                         return True
216                     else:
217                         j=j+1
218                 b=databasemodel.read('buyers.json')
219                 buyers=b['buyers']
220                 c=databasemodel.read('cart.json')
221                 carts=c['carts']
```

```
215                             return True
216                         else:
217                             j=j+1
218                     b=databasemodel.read('buyers.json')
219                     buyers=b['buyers']
220                     c=databasemodel.read('cart.json')
221                     carts=c['carts']
222                     for i in buyers:
223                         for j in i['wishlist']:
224                             if j==p:
225                                 i['wishlist'].remove(j)
226                                 databasemodel.update_buyer(i)
227                     for k in carts:
228                         for l in k['products']:
229                             if l==p:
230                                 k['products'].remove(l)
231                                 databasemodel.update_cart(k)
232                     self.update_seller()
233                 else:
234                     print('Invalid input')
235                     return False
236             else:
237                 print("No Products are being offered by you!")
238                 return False
239
```

Fig 26 – Remove products from the list of products offered by the seller.

- **Seller: Update Order status**

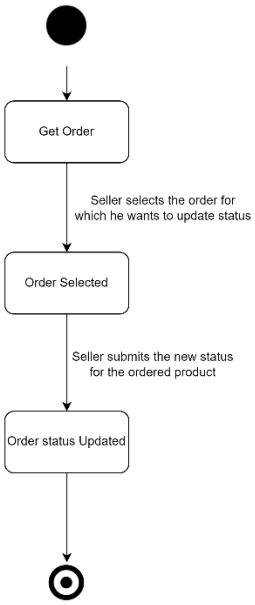

Fig 27 – State diagram for Update order status task performed by the seller.

Order status state diagram passes through three states, get orders, order selected, and order status updated. In the first state, user will be getting requests from the buyers to ship the products. After the confirmation of the order, in the second state, single order is selected from the list of orders available. Then seller will process the shipment and updates the shipping information to the buyer in the third state.

```
290     def order_status(self):
291         if len(self.orders)>0:
292             print("Which Order's status you want to update?")
293             k=1
294             for i in self.orders:
295                 print(f'{k}.  {i}\n')
296                 k=k+1
297             t=int(input('Enter the specific number'))
298             p_id=self.orders[t-1]
299             o=Order.get_order(p_id)
300             d=databasemodel.read('products.json')
301             products=d['products']
302             for i in o.status:
303                 if i['status']=='in processing' or i['status']=='shipped':
304                     p=Product.get_product(i['product'])
305                     if p.seller_name==self.company_name:
306                         for y in products:
307                             if y['product_id']==p.product_id:
308                                 pprint(y)
309                         print(f"The current status is {i['status']}\n")
310                         s=input('Enter the updated status or press 0 if you do not want to update')
311                         if s=='0':
312                             continue
313                         else:
314                             i['status']=s
315                             o.update_order()
316         else:
317             print("There are no orders for you yet!\n")
318             return
319
```

Fig 28 – Shippment status of the order provided by the seller.

4. State diagrams for Order Class

- **Order- Create Order**

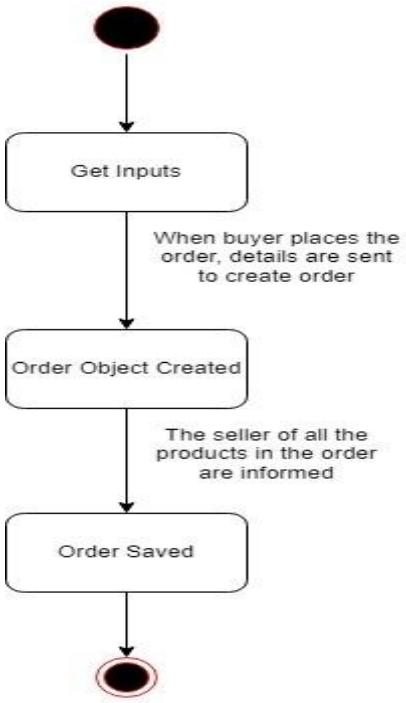

Fig 29 – State diagram for Create order.

The first state is to get inputs, which will be sent when the buyer places the order from the cart. Then the Order Object is created which is the second state. After that, the sellers of all the products under this order are linked with this order and then finally the order is saved.

```
11      @staticmethod
12      def create_order(order_id, products_ordered, status, buyer_email):
13          o=Order(order_id, products_ordered, status, buyer_email)
14          data=databasemodel.read('products.json')
15          products=data['products']
16          data1=databasemodel.read('sellers.json')
17          sellers=dat   (variable) products: Any
18          for p in o.products:
19              for z in products:
20                  if z['product_id']==p:
21                      for i in sellers:
22                          if i['company_name']==z['seller_name']:
23                              i['orders'].append(o.order_id)
24                              databasemodel.update_seller(i)
25          return o.save()
```

Fig 30 – Create order and inform the seller about the order.

- **Cancel Order**

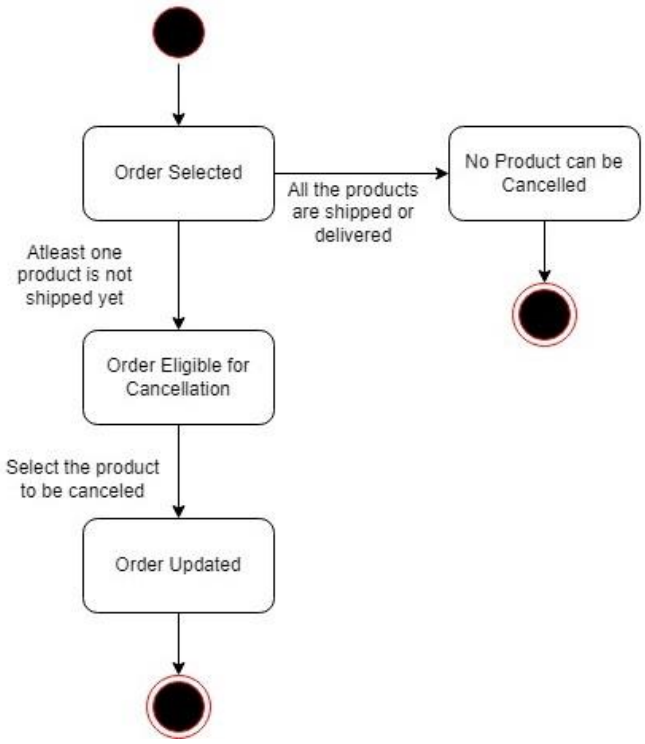

Fig 31 – State diagram for Cancel order task performed by the buyer.

The buyer selects the order under which the buyer wants to cancel some product. The first state we have is that the order is selected. Now for that order, it is checked whether the prodcuts are shipped

or not because buyer can only cancel the ordered product before their shipping. If there is atleast one product under that order which is not shipped then we have second state as Order Eligible for cancellation else the second state would be order not eligible for cancellation. If it is eligible then the buyer selects the product which he wants to cancel (from the ones which are not shipped yet), then that product is cancelled and the order is updated which is the final state.

```
59      def cancel_order(self):
60          cancelation_allowed=[]
61          for i in self.status:
62              if i['status']=='in processing':
63                  cancelation_allowed.append(i['product'])
64
65          if len(cancelation_allowed)==0:
66              print('you cannot cancel your order now')
67          else:
68              print('which product you want to cancel')
69              k=1
70              data=databasemodel.read('products.json')
71              products=data['products']
72              for i in cancelation_allowed:
73                  for p in products:
74                      if p['product_id']==i:
75                          print(f'{k}. (Order ID:- {i}) {p}')
76                          print('\n')
77                          k=k+1
78              t=int(input('Enter the specific number'))
79              to_be_cancel=cancelation_allowed[t-1]
80              for i in self.status:
81                  if i['product']==to_be_cancel:
82                      i['status']='order cancelled'
83                      self.update_order()
84
```

Fig 32 – Buyer can cancel the order only if the status is “IN PROCESSING”.

- **Add Rating**

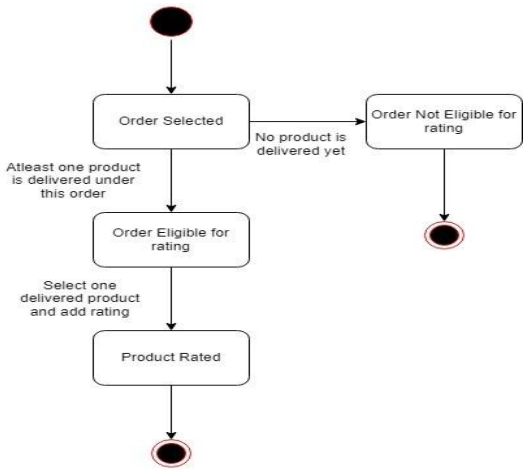

Fig 33 - State diagram for Add rating task performed by the buyer.

Firstly, the buyer selects the order for which he wants to rate products. Then the first state is that the order has been selected. Now, a buyer can rate a product purchased in an order only if the product has been delivered. If no product is delivered, then the second state is that Order is not eligible for rating. If at least one product is delivered, then the second state is that the Order is eligible for rating. Then the buyer selects the product which he wants to rate and give rating which gives us the final state that the product is rated.

```
84
85      def add_rating(self):
86          can_be_rated=[]
87          for i in self.status:
88              if i['status']=='delivered':
89                  can_be_rated.append(i['product'])
90          if len(can_be_rated)==0:
91              print('currently you cannot rate products purchased under this order')
92          else:
93              k=1
94              data=databasemodel.read('products.json')
95              products=data['products']
96              print('select product which you want to rate')
97              for i in can_be_rated:
98                  for p in products:
99                      if p['product_id']==i:
100                         print(f'{k}... {p}')
101                         print('\n')
102                         k=k+1
103             t=int(input('give the specific product number which you want to rate'))
104             to_rate=can_be_rated[t-1]
105             r=int(input('Enter the rating (integer value) on 5 point scale'))
106             for i in products:
107                 if i['product_id']==to_rate:
108                     i['rating'].append(r)
109                     databasemodel.update_product(i)
110                     return True
```

Fig 34 – Buyer can rate products after delivery.