CS387 Project Deliverable 3 - Design documents
E-Commerce Recommender System

Bhaskar Gupta	Niraj Mahajan	Shivam Goel	Tathagat Verma
180050022	180050069	180050098	180050111
180050022@iitb.ac.in	nirajm@iitb.ac.in	180050098@iitb.ac.in	180050111@iitb.ac.in

1. Data model

Node labels	**Properties**
buyer	id, username, password
seller	id, username, password
admin	id, username, password
del_personnel	id, username, password
product	id, title, description, rating, num_rating, img, price, avl_qty
brand	id, name
category	id, name
order	id, status, qty, timestamp

Edge labels	**Node-Node**	**Properties**
cart	buyer-product	qty
also_viewed	product-product	-
also_bought	product-product	-
view_history	buyer-product	timestamp
prod_seller	product-seller	-
prod_brand	product-brand	-
prod_category	product-brand	-
ord_del	order-del_personnel	-
ord_prod	order-prod	-
buyer_order	buyer-order	-
buyer_rating	buyer-product	rating

Node-Node shows which class of nodes is the edge connecting.

The buyer, seller, admin, del_personnel nodes are disjoint and store only distinct information, i.e username and password. Hence they cannot be normalized further. Similarly for brand and category.
The schema presented is edge normalized since all edges have either 0 or 1 weights (values).

We have introduced multiple "order" nodes for each order since we want to allow a different delivery personnel for each product in the order. This cannot be done by directly having an edge from buyer to product. Hence we have: when a user orders multiple items at once, from the cart, then we will create multiple order nodes, 1 for each product. However they will have a common order_id. Each of the order nodes will eventually create an edge with the delivery personnel when the seller (for that product) accepts the order. This is a denormalization step as we are repeating the order_id at multiple nodes. If we had kept a single order node and then multiple edges from that order node to products, then we would have to store the delivery personnel data on an edge. But the delivery personnel is a node, so it is more efficient to have an edge to the delivery personnel node. Hence having multiple order nodes is more efficient despite storing the order_id at multiple nodes. Efficiency is because to see the pending orders of a delivery personnel, if delivery personnel information is stored in an edge, then will have to go over all the orders. But edges from orders to delivery personnel will make it faster.

We have stored rating of a product as the product's property and also stored it in the buyer_rating relationship. Thus this is also a denormalization. We want to store which user gave a rating for which product. The overall average can be computed using a query every time but since this is expensive and we want to obtain product details fast, we store the rating (i.e average of all user ratings) in the product node as well.

In all other relations (i.e other than order and product rating) there is no repetition of data and hence normalized.

Indexes

buyer - id, username, password
seller - id, username, password
del_personnel - id, username, password
product - id
order - id
category - name
brand - name
The id indexes are basic indexes.

Index on the category and brand names help in giving faster search query and filter results. It is reasonable to have these indexes since addition of categories and brands can be less in frequency, but the use case is much more. Same is the reason for username and password indexes. It will make the log in check faster.

Constraints

- product has exactly one seller
- product has exactly one brand

2. Data generation and loading

We have downloaded a subset of data from the Amazon Review Dataset. The original dataset has 22 broad product categories. Since that went upto 12GB and the corresponding size of ratings to 6GB, we used only 2 categories - Books and Electronics. The number of products then is 2.9M(books) and 0.8M(electronics). The overall size of products data is 2GB.

The ratings data is available in a separate csv. The number of ratings for Books is 51M and 21M for Electronics. Size of the review data is 2GB(books) and 0.8GB(electronics). The script for downloading data (data_load.py) produces the cypher query language code (2 files - product_data.cypher and user_data.cypher)for inserting the above into the graph data DB.

Since the rating data set contains user IDs, we have created buyers as per that. The total number of buyers comes out to be 3.6M.
The sellers, admin and delivery_personnel data is generated by us as it is not present in the Amazon dataset. We have kept only 1 admin and 1 delivery personnel in the initial data. Each product is associated with a seller. We have kept only 1 seller initially.

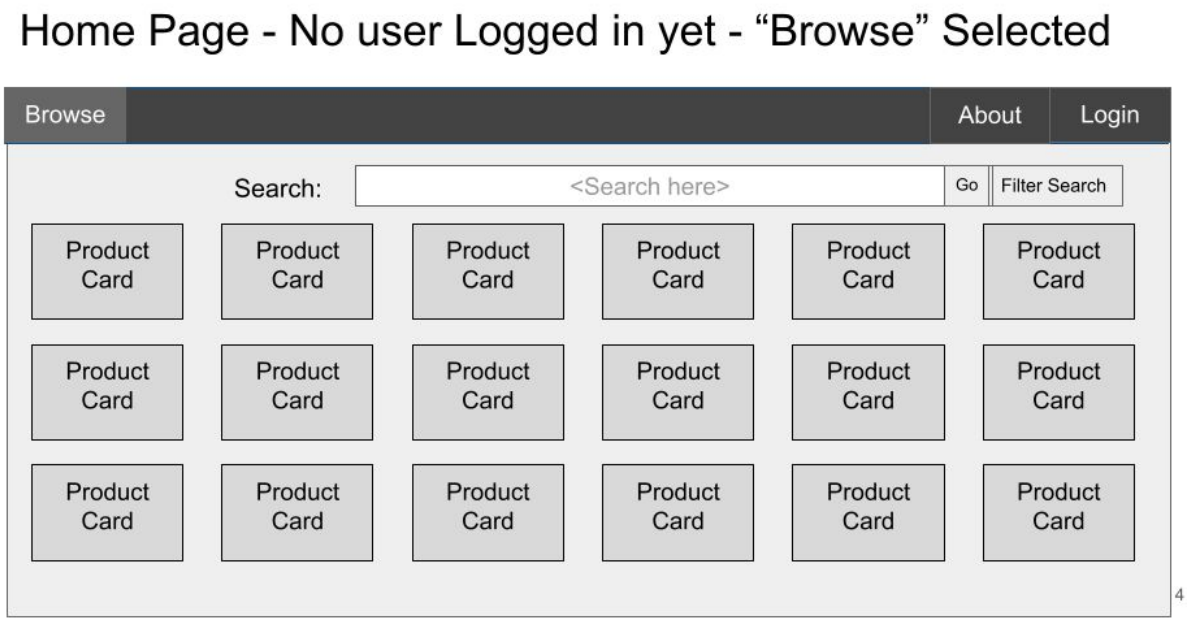
Data loading script to be run as :
$ python3 data_load.py

Files generated and downloaded:
amazon_data/Books.csv
amazon_data/Electronics.csv
amazon_data/meta_Books.json.gz
amazon_data/meta_Electronics.json.gz
product_data.cypher
user_data.cypher

3. Screen design

We will be following a pattern similar to lab 5 - using code “internally” to manage transitions (controllers).
We have summarized the pages and transitions available for every user, by cases - admin, seller, customer, delivery agent.

HOME PAGE (No one has logged in yet)

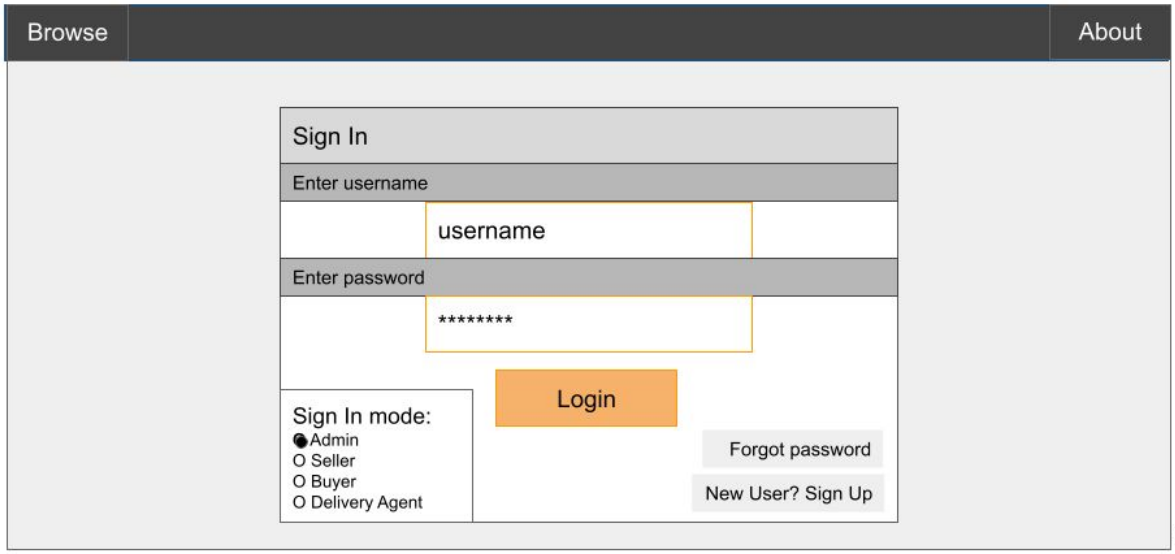


Product Card Expanded

Product Image	Product Title
	Seller Name
	Cost
	Rating
	Category

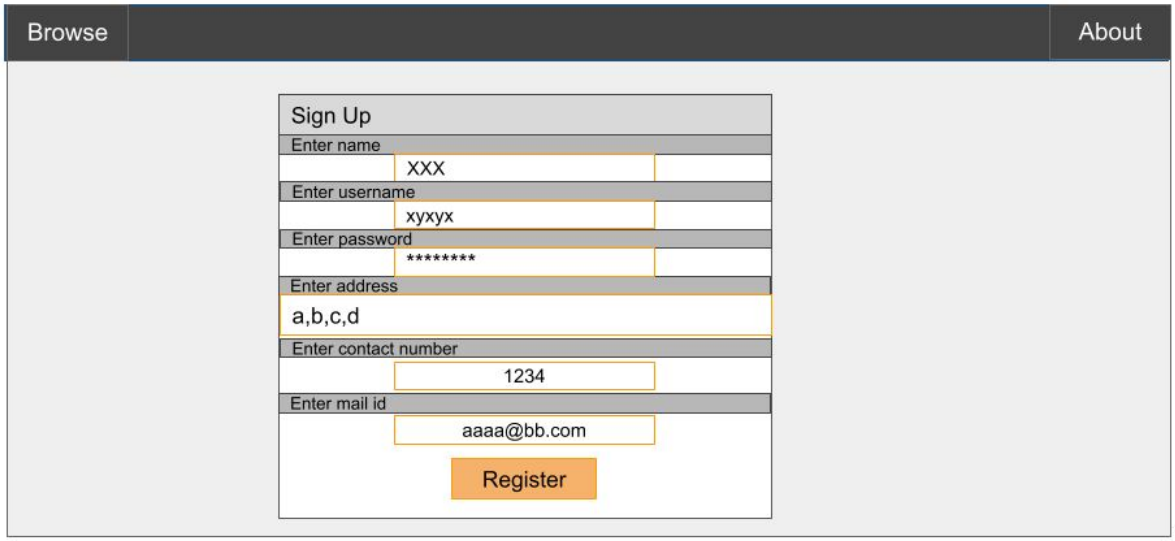
Home Page - No user Logged in yet - “About” Selected

Login Page - On Clicking Login Button



7

Sign Up Page - New Users



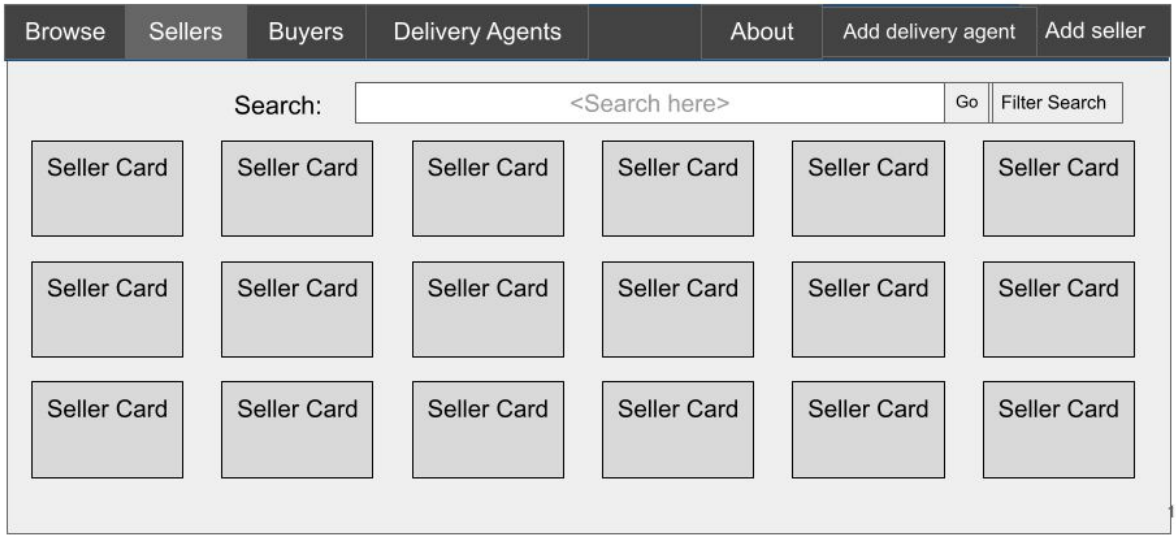
8

Admin Pages

Admin has logged in. All access should be open (except adding stuff to cart). Admin can add new sellers and delivery agents.

Admin Logged In - “Browse” selected by default

Admin Logged In - “Sellers” selected - List Sellers



Seller Card Expanded

Seller Name
Seller Warehouse Address
Contact Number
Mail ID

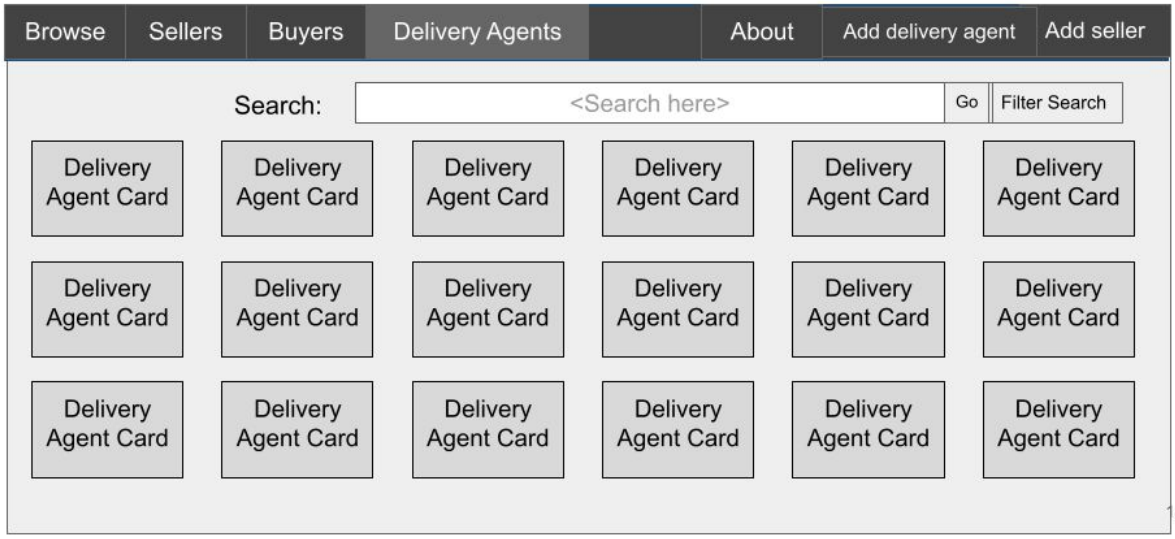
Buyer Card Expanded

Buyer Name
Buyer Address
Contact Number
Mail ID

Admin Logged In - “Buyers” selected - List Buyers

14

Admin Logged In - “Del. Agents” selected - List Del. Agents

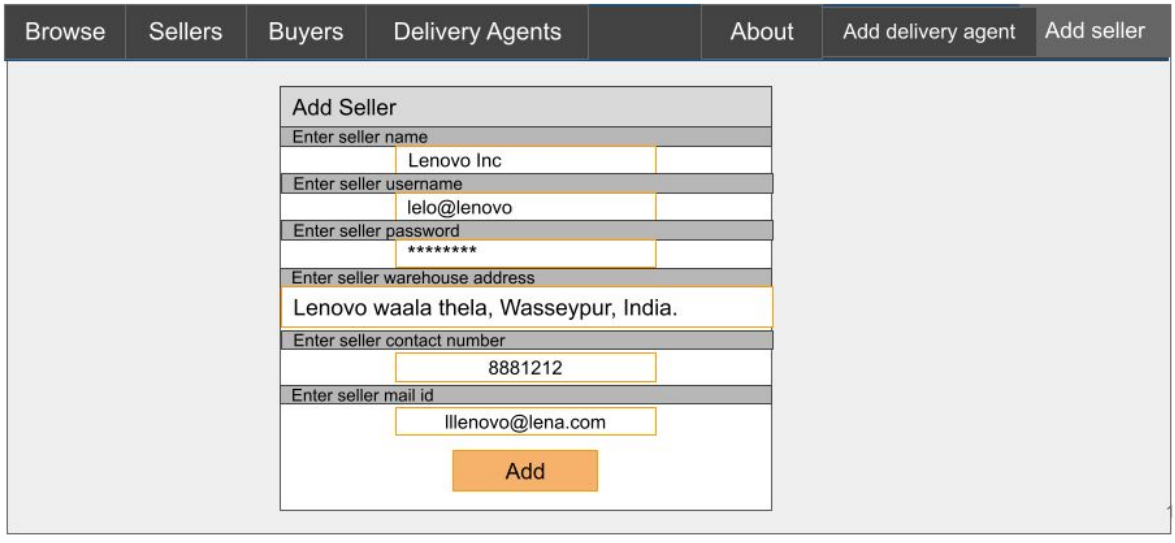


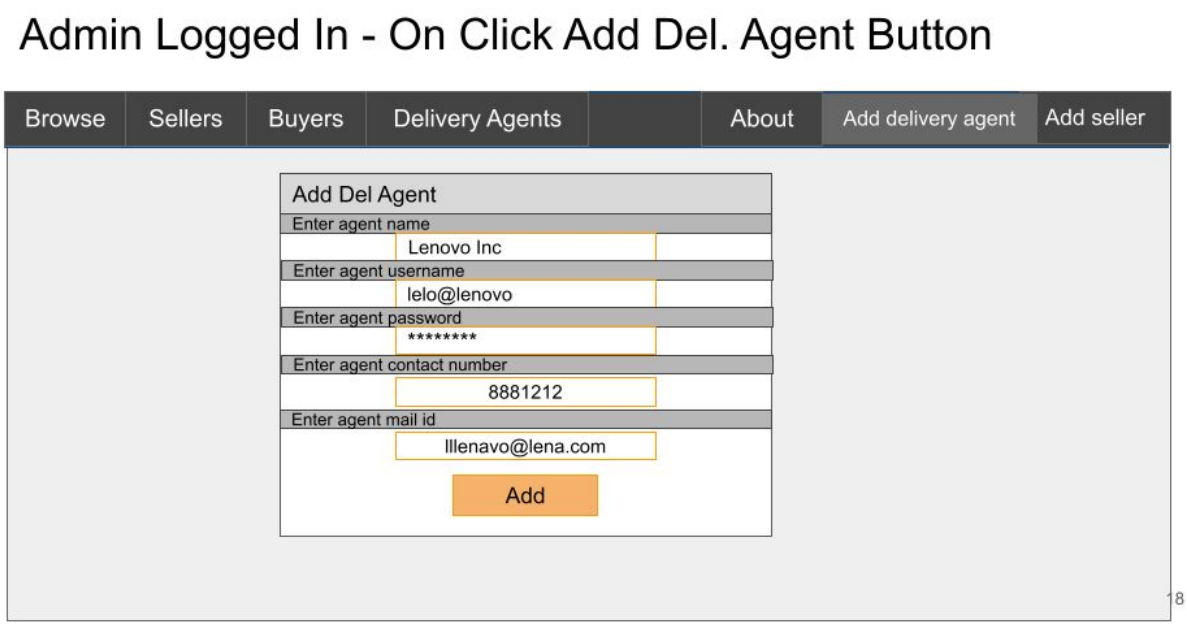
15

Delivery Agent Card Expanded

Agent Name
Contact Number
Mail ID
Number of Orders Delivered
Number of Active Orders

Admin Logged In - On Click Add Seller Button





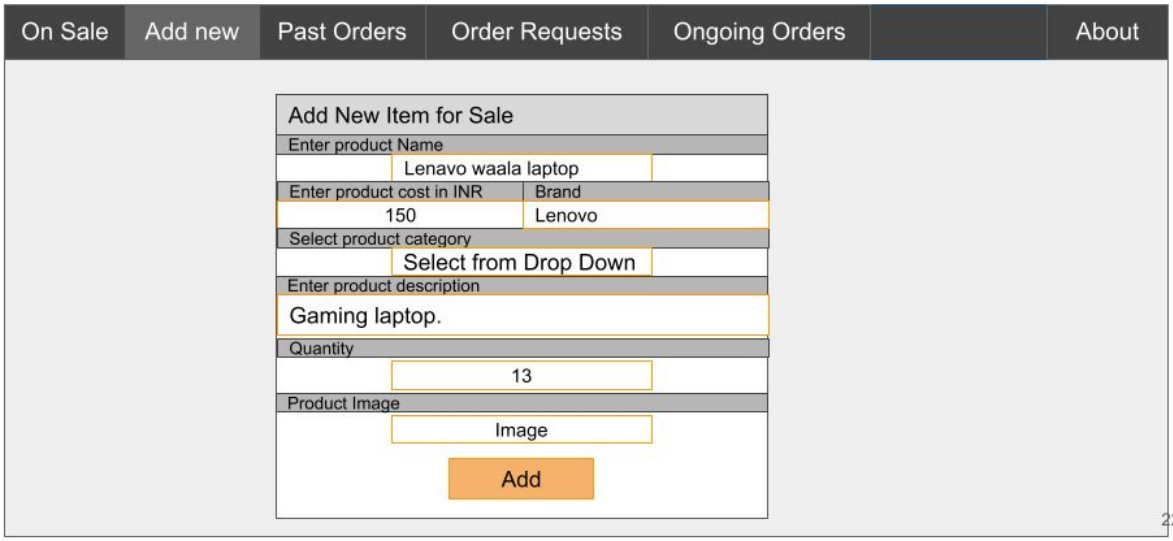
SELLER PAGES

Seller has logged in. He/she should be able to

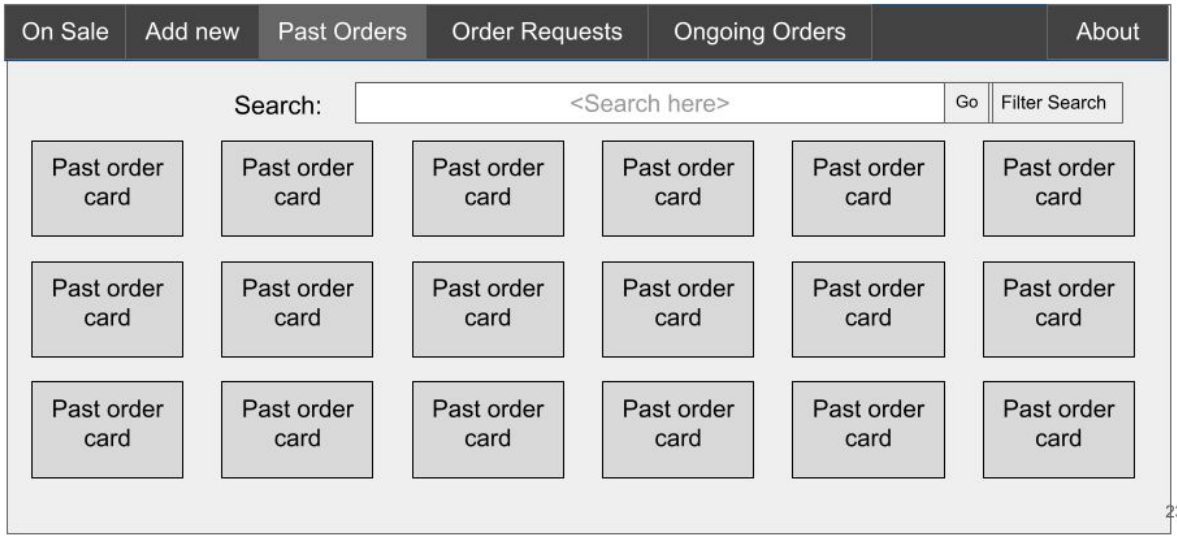
- Add new items
- Restock Items
- View past orders
- View orders pending approval (or in other words, shipping requests)
- View ongoing orders (or, shipped orders)

Seller Logged In - “On sale” selected by default

Seller Logged In - Add new item



Seller Logged In - “Past Orders” selected- list all delivered orders



23

Past Order Card Expanded

Product Image	Order ID	
	Product Title	
	Buyer Name	
	Cost	Quantity
	Category	

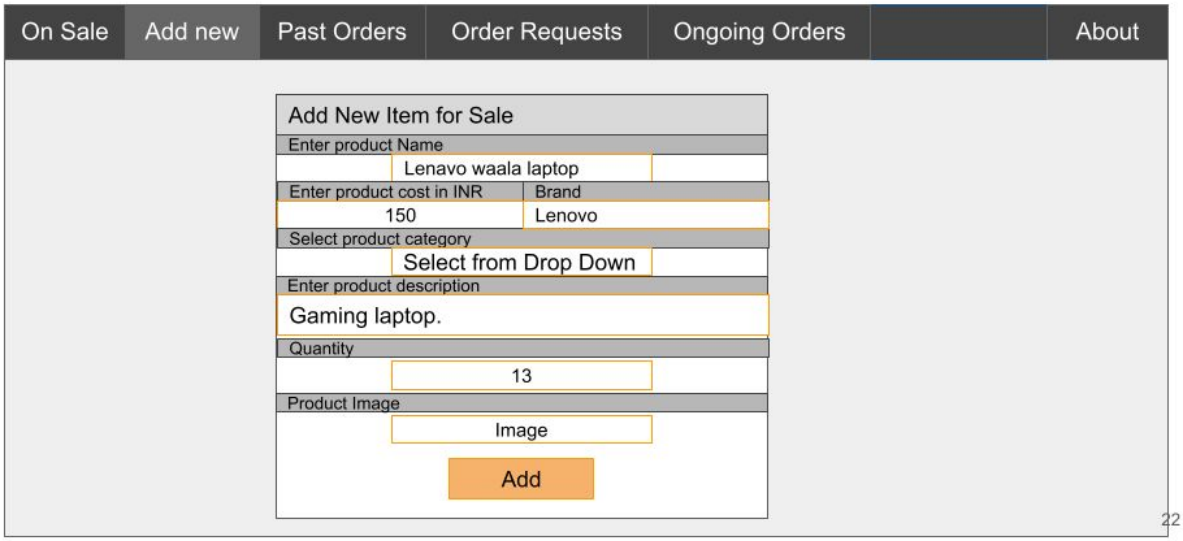
24

Seller Logged In - “Order Requests” selected- list all pending orders

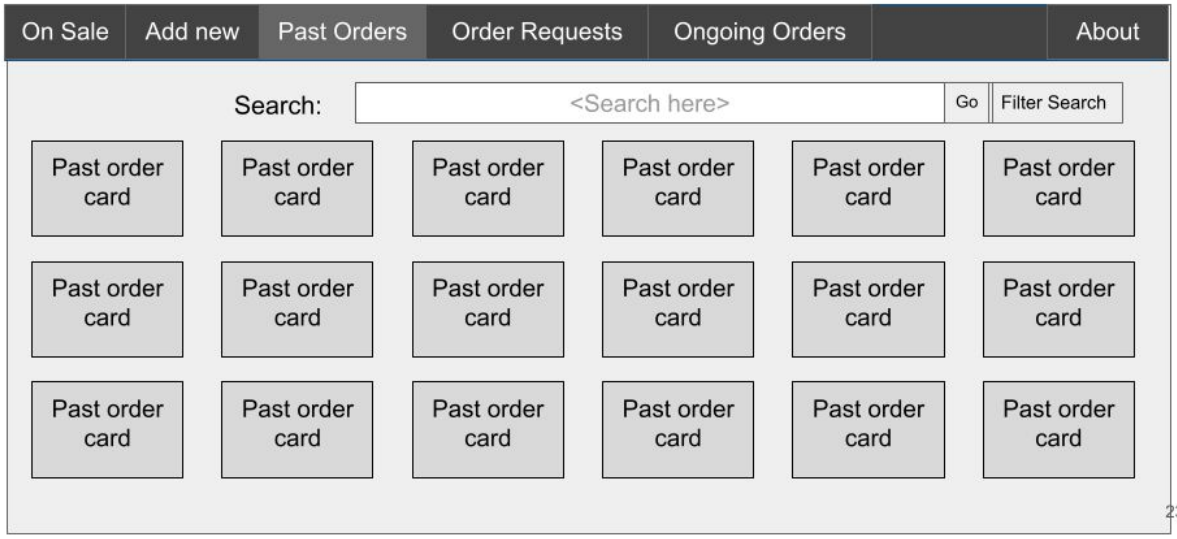
Seller Logged In - “Order Requests” selected- list all pending orders

Seller Sale Card Expanded

Seller Logged In - Add new item



Seller Logged In - “Past Orders” selected- list all delivered orders



Seller Logged In - “Order Requests” selected- list all pending orders

Past Order Card Expanded

Product Image	Order ID	
	Product Title	
	Buyer Name	
	Cost	Quantity
	Category	

Seller Logged In - “Order Requests” selected- list all pending orders

Pending Order Card Expanded

<table>
<tr><td rowspan="5">Product Image</td><td colspan="2">Order ID</td></tr>
<tr><td colspan="2">Product Title</td></tr>
<tr><td colspan="2">Buyer Name</td></tr>
<tr><td>Cost</td><td>Quantity</td></tr>
<tr><td colspan="2">Ship</td></tr>
</table>

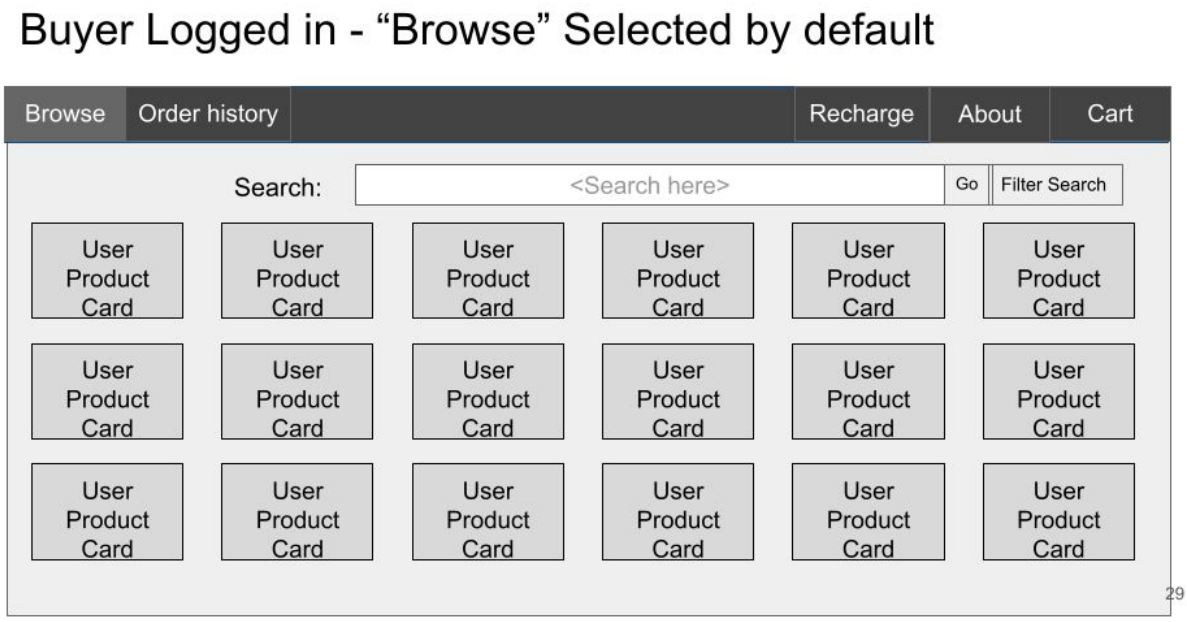
Seller Logged In - “Ongoing Orders” selected- list all shipped orders

UI same as selecting “past orders”. Please refer 3 slides back

BUYER PAGES

Buyer has logged in. He/she should be able to

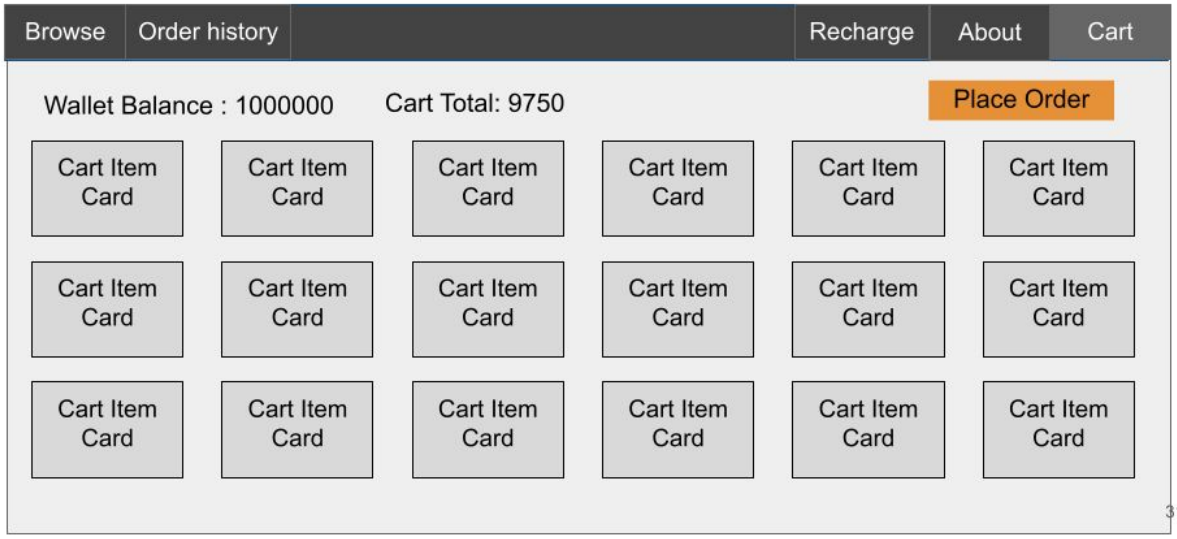
- Browse new items
- Add items to cart
- View and buy from cart
- View past orders and rate them
- Recharge his wallet



User Product Card Expanded

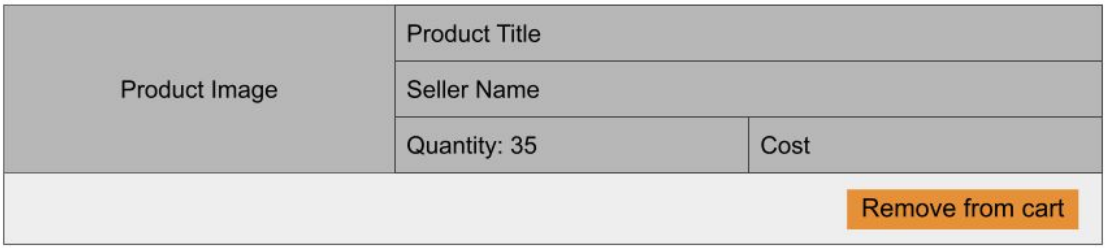
30

Buyer Logged in - “Cart” Selected - List the current cart



31

Cart Item Card Expanded



Buyer Logged in - “Order History” Selected - List the current cart

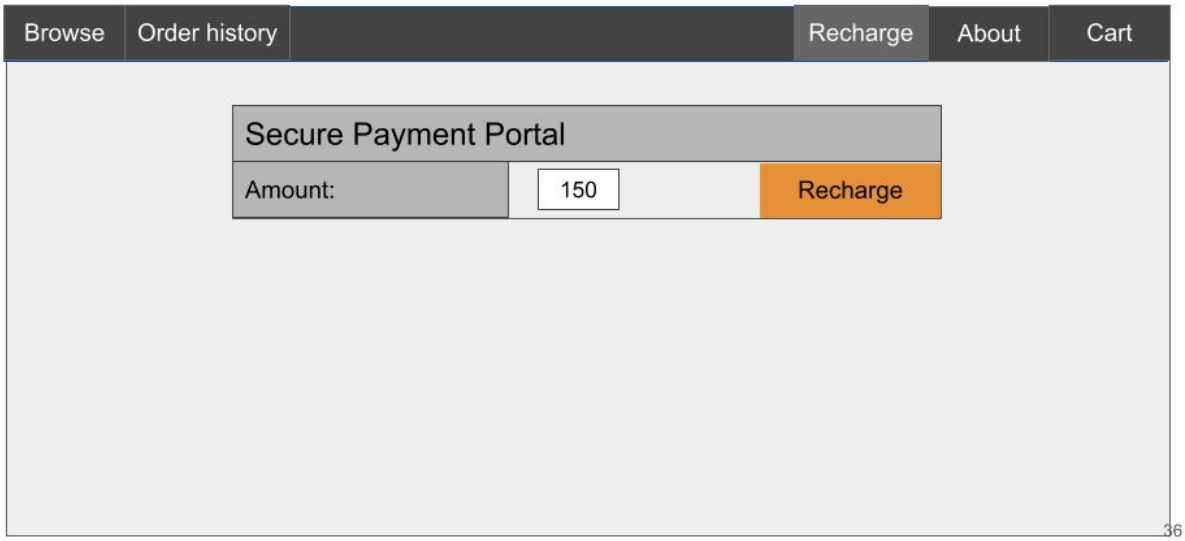
User Order Card Expanded

Order ID
Total Cost
Status
Delivery Agent Name
Delivery Agent Contact Number
Click for more details

User Order Card - “Click for more details” pressed

Order ID		
Total Cost		
Item List:		
Item Name	Quantity	Cost
Rating : ☆☆☆☆☆		
⋮		

Buyer Logged in - “Recharge” Selected - Add money to wallet

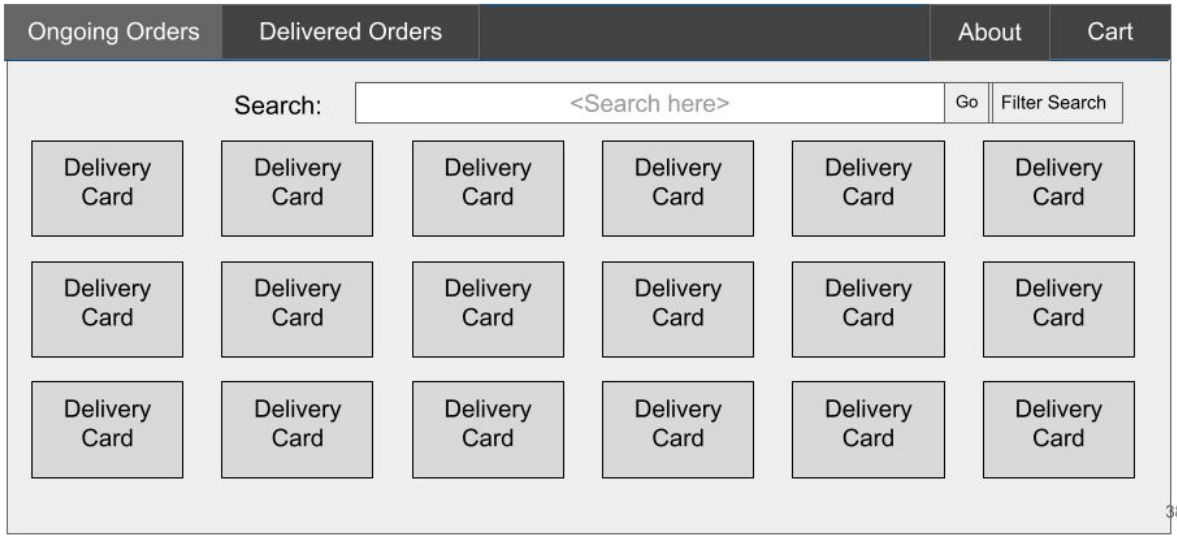


Delivery Agent Pages

Delivery agent has logged in. He/she should be able to

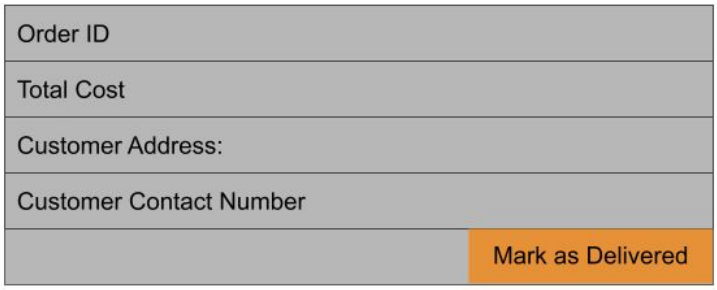
- View his delivered orders history
- View his pending/ongoing orders

Del. Agent Logged in - “Ongoing Orders” Selected - List the orders out for delivery by the agent



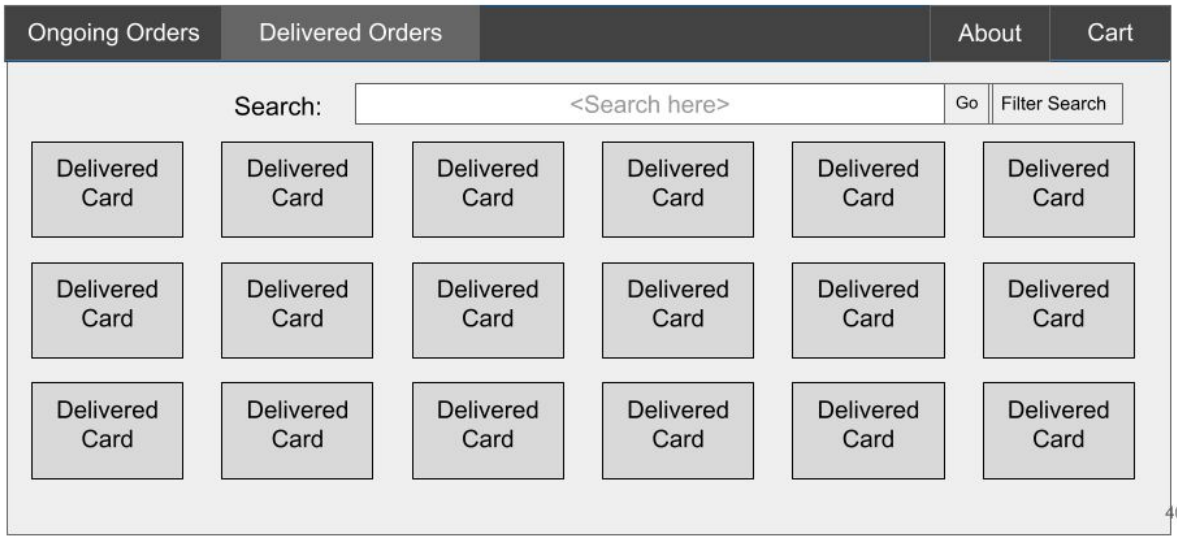
38

Delivery Card Expanded



39

Del. Agent Logged in - “Delivered Orders” Selected - List the orders delivered by the agent



40

Delivered Card Expanded

Order ID
Total Cost
Customer Address:

41

4. Transactions for each use case and its SQL

- **COMMON TO ALL USERS**

1. Login
   For Buyer
   Input: u1=username, p1=password
   Query:
   ```
   MATCH (b:buyer({username:u1, password:p1}))
   RETURN b.id
   ```
   Similarly for other users, i.e admin, seller and delivery personnel (replace buyer with respective user type)

2. Logout
   Query: NA

3. Search Products
   Input : search query string(s)
   Query : MATCH (p:product)
   ```
   WITH apoc.text.levenshteinDistance(s, p.title) as output1
   WITH apoc.text.levenshteinDistance(s, p.description) as output2
   WITH
   ```
   output1*alpha1+output2*alpha2+(5-p.rating)*alpha3+(1/p.numRating)*alpha4 as output
   ```
   ORDER BY output
   RETURN p
   ```

- **USERS WITHOUT LOGIN**

1. View Product
   Input: p1=product id
   Query:
   ```
   MATCH (p:product {id:p1})-[:prod_cat]-(c:category)
   MATCH (p)-[:prod_sell]-(s:seller)
   RETURN p, c, s
   ```

- **BUYERS**

1. View product

Input : b1=buyer_id, p=product node obtained from the previous query
Query:

```
MATCH (p:product {id:p1})-[:prod_cat]-(c:category)
MATCH (p:product {id:p1})-[:prod_brand]-(b:brand)
MATCH (p)-[:prod_sell]-(s:seller)

MATCH (p)-[:also_bought]-(p1:product)
MATCH (p)-[:also_viewed]-(p2:product)

MATCH (p3:product)-[v:view_history]-(:buyer[id:b1])
WHERE time()-v.timestamp<delta
MERGE (p)-[:also_viewed]-(p3)      // updates the also_viewed of product
MERGE (p)-[:view_history{timestamp:time()}]-(:buyer{id:b1})  // update
// view_history of user

RETURN p, c, b, s, p1, p2       // list of p2=also_viewed and p1=also_bought
products
```

2. View Cart
   Input: b1=buyer_id
   Query:

```
MATCH (p:product)-[c:cart]-(b:buyer{id:b1})
RETURN p,c
```

3. Add to Cart
   Input: p1=product id, b1=buyer id, q1=quantity
   Query:

```
MATCH (p:product{id:p1})
MATCH (b:buyer{id:b1})
MERGE (p)-[c:cart]-(b)
ON CREATE SET c.quantity=q1
ON MATCH SET c.quantity=c.quantity+q1
```

4. Remove from cart
   Input: p1=product id, b1=buyer id
   Query:

```
MATCH (p:product{id:p1})-[c:cart]-(b:buyer{id:b1})
DELETE c
```

5. Place order

Input: b1=buyer id, o1=order id
Query:

```
MATCH (b:buyer{id:b1})-[c:cart](-p:product)
MERGE (o:order{timestamp:time(), o_id:o1, status:"requested",
quantity:c.quantity})
MERGE (b)-[:buyer_order]-(o)-[:order_product]-(p)
DELETE c

MATCH (o:order)-[:order_product]-(:buyer[id:b1])
WHERE time()-o.timestamp<delta
MATCH (o)-[:order_product]-(p2:product)
MERGE (p)-[:also_bought]-(p2)      // updates the also_bought of product
```

6. View order history
   Input: b1=buyer id
   Query:

```
MATCH (o:order)-[:order_product]-(:buyer[id:b1])
MATCH (o)-[:order_product]-(p:product)
RETURN o,p
```

7. Give rating
   Input: o1=order id, p1=product id,r1=rating
   Query:

```
MATCH (o:order{o_id:o1})-[:order_product]-(p:product{id:p1})
SET p.rating=(p.rating*p.numRating+r1)/(p.numRating+1)
SET p.numRating=p.numRating+1
```

8. Add money
   Input: b1=buyer id, a1=amount to be added
   Query:

```
MATCH (b:buyer{id:b1})
SET b.money=b.money+a1
```

- **ADMIN**

1. View seller
   Input: s1=seller id
   Query:

```
MATCH (s:seller{id:s1})
```

RETURN s

2. View buyer
   Input: b1=buyer id
   Query:
   MATCH (b:buyer{id:b1})
   RETURN b

3. View delivery personnel
   Input: d1=delivery personnel id
   Query:
   MATCH (d:del_personnel{id:d1})
   RETURN d

4. Add seller
   Input: n1=name, u1=username, p1=password, a1=address, c1=contact
   Query:
   MERGE (s:seller{name:n1, username:u1, password:p1, address:a1, contact:c1})

5. Add delivery personnel
   Similar to point 4

- **SELLER**

1. View product on sale
   Input: s1=seller id
   Query:
   MATCH (p:product)-[:prod_seller]-(s:seller{id:s1})
   MATCH (p)-[:prod_cat]-(c:category)
   RETURN p,c

2. Change quantity
   Input: s1=seller id, p1=product id, q1=new quantity
   Query:
   MATCH (p:product{id:p1})-[:prod_seller]-(s:seller{id:s1})
   SET p.quantity=q1

3. Add product

Input: t1=product title, p1=price, c1=category, b1=brand, d1=description, q1=quantity, i1=image, s1=seller id
Query:

```
MERGE (p:product{title:t1, price:p1, quantity:q1, image:i1})
MERGE (c:category{name:c1})
MERGE (b:brand{name:b1})
MERGE (p)-[:prod_cat]-(c)
MERGE (p)-[:prod:brand]-(b)
MERGE (p)-[:prod_seller]-(s:seller{id:s1})
```

4. View past orders
   Input: s1=seller id
   Query:

```
MATCH (s:seller{id:s1})-[:prod_seller]-(p)
MATCH (o:order{status=”delivered”})-[:order_product]-(p)
RETURN p
```

5. View order requests
   Input: s1=seller id
   Query:

```
MATCH (p:product)-[:prod_seller]-(s:seller{id:s1})
MATCH (p)-[:order_product]-(o:order{status:”requested”})
RETURN p
```

6. Update order requests (also assign delivery personnel)
   Input: s1=seller id, p1=product id, d1=delivery personnel id
   Query:

```
MATCH (p:product{id:p1})-[:prod_seller]-(s:seller{id:s1})
MATCH (p)-[:order_product]-(o:order{status:”requested”})
SET o.status=”shipped”
MATCH (d:del_personnel{id:d1})
MERGE (o)-[:del_order]-(d)
```

7. View ongoing orders
   Input: s1=seller id
   Query:

```
MATCH (p:product)-[:prod_seller]-(s:seller{id:s1})
MATCH (p)-[:order_product]-(o:order{status:”shipped”})
RETURN p
```

- **DELIVERY PERSONNELS**

1. View pending orders
   Input: d1=delivery personnel id
   Query:

```
MATCH (d:del_personnel{id:d1})-[:del_order]-(o:order{status:”shipped”})
MATCH (o)-[:order_product]-(p:product)
RETURN p
```

2. Update pending deliveries
   Input: d1=delivery personnel id, p1=product id
   Query:

```
MATCH (d:del_personnel{id:d1})-[:del_order]-(o:order{status:”shipped”})
MATCH (o)-[:order_product]-(p:product{id:p1})
SET o.status=”delivered”
```

3. View delivered orders
   Input: d1=delivery personnel id
   Query:

```
MATCH (d:del_personnel{id:d1})-[:del_order]-(o:order{status:”delivered”})
MATCH (o)-[:order_product]-(p:product)
RETURN p
```