Online Retail Store (FLIPCART)

Team 159
Aditya Dahiya(2021228)|| Adish Jain(2021227)

Scope & Objectives -

With the technological advancements the world has encountered and the growth of internet users worldwide, the scope for e-commerce activities is expanding day by day. With the onset of the COVID-19 Pandemic, several people have resorted to online shopping to fulfill their short-term as well as long-term requirements.

With this project, we intend to make the process of searching, selecting, and purchasing a product feasible virtually, while, at the same time, making it easier for vendors and merchants to sell their inventory.

The front end of the online retailing store is the user interface that the customer and the seller can interact with, once they enter their corresponding login details when they visit the website. It is responsible for displaying product information, handling user input, and facilitating navigation through the site. Using the front end, customers can browse the products, add items to their cart, and also proceed to check out. They can also create an account, view order history, and update their personal information. The front end will also include features like the search bar, and recommendations.

Meanwhile, the sellers can also manage their products, inventory, orders, and delivery agents working under them. This includes a dashboard where the sellers can update product information, and update delivery agents. They can also add new products, update prices and manage their inventory levels.

Stakeholders Identified -

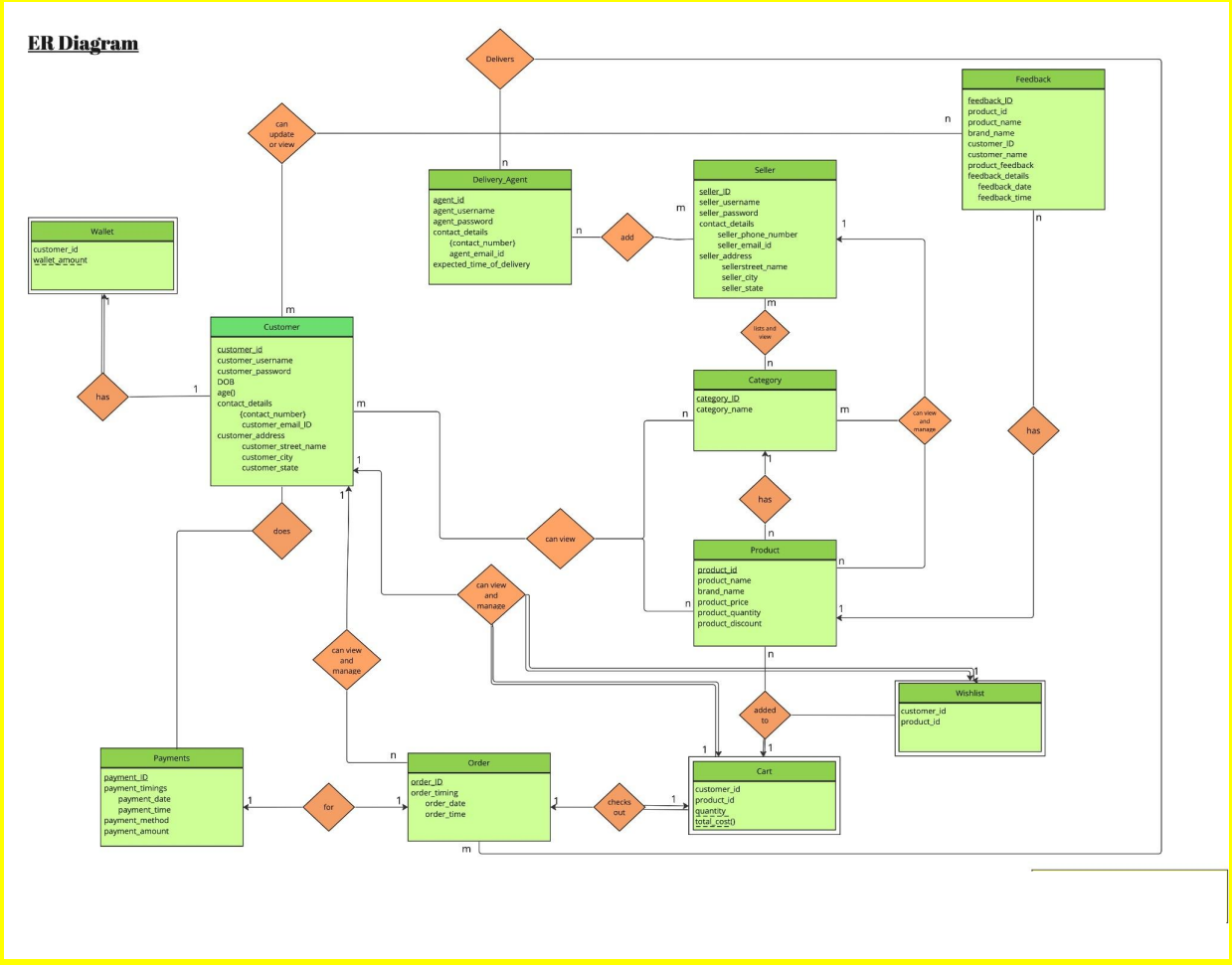
1. **Customers** - the customers can browse categories and products. Each customer has a cart, wallet, and wishlist.
2. **Sellers** - the sellers can add new categories and products. They can update the database such as the products, their categories, inventories, etc whenever required.
3. **Delivery agent -** The delivery agents can view the orders assigned to them and complete their delivery accordingly.

Assumptions and Constraints -

1. **Customer to Product :** these 2 entities are related by the "can view" (m:n) relationship which means it is assumed that m number of customers can view n number of products.
2. **Customer to Category** : these 2 entities are related by the "can view" (m:n) relationship which means it is assumed that m number of customers can view n number of products.
3. **Customer to Cart and Wishlist** : Customer has a "can view and manage" (1:1) relationship with entities cart and wishlist which means it is assumed that each customer has only one cart and one wishlist.
4. **Customer to Wallet** : it is assumed that each customer has only one wallet i.e. it is a 1:1 relation.

5. **Customer to Order :** Each customer "can view and manage" n number of orders i.e. 1:n relation
6. **Customer to Feedback :** n Customers can add m number of product feedback which means that it is assumed that there can be multiple feedback added by any number of customers.
7. **Seller to Category :** m number of sellers can "list" n number of categories which means its an m:n relation. It is assumed that categories can be shared among different sellers. This is the reason why multiple sellers can add multiple categories. They are also connected through the "view and manage" relation(1:n) which assumes a seller can view and manage the category which he/she has created.
8. **Category to Product :** it is assumed that each category can have n number of products but each product can belong to only one category. there is a 1:n relation.
9. **Cart to Product :** Each cart can have n number of products. there is a 1:n relation.
10. **Order to Payments :** there is 1:1 relation which means it is assumed each customer can pay only once for one order, there is no option for partial payments.
11. **Seller to Product :** they are connected through view relation which assumes a seller can view n number of products at a time.
12. **Seller to Delivery_Agent** : n number of sellers can add m number of delivery agents.

ER Diagram -



Relational diagram -

Constraints -

1. Each Customer can only have **one account** from a contact number which must have a **unique** name.
2. Products can't have a discount which is more than 80%.
3. Each product can belong to only one category.
4. Each order ID must be **unique** on a global scale.
5. All entries must be **NOT NULL** except, "items in wishlist" and "items in cart".
6. Only one coupon can be added to an order.

Entities, Attributes & Schemas -

1. **Customer Table -**

Customer_id	INT Primary key NOT NULL
customer_name	Varchar NOT NULL
Customer_password	Varchar NOT NULL UNIQUE
DOB	Date NOT NULL
Customer_number	Varchar UNIQUE NOT NULL
customer_number2	Varchar UNIQUE NOT NULL
customer_email_ID	Varchar UNIQUE NOT NULL
customer_city	Varchar NOT NULL
Customer_state	Varchar NOT NULL
age	INT Derived from DOB

2. **Seller Table** -

seller_id	INT Primary Key NOT NULL
seller_username	Varchar UNIQUE NOT NULL
seller_password	Varchar UNIQUE NOT NULL
seller_phone_number	Varchar UNIQUE NOT NULL
seller_phone_number2	Varchar UNIQUE
seller_email_ID	Varchar UNIQUE NOT NULL
seller_city	Varchar NOT NULL
seller_state	Varchar NOT NULL

3. **Category -**

category_id	INT Primary key NOT NULL
category_name	Varchar NOT NULL UNIQUE

-

4. **Product -**

product_id	INT Primary key NOT NULL
product_name	Varchar NOT NULL UNIQUE
category_id	INT Foreign key
brand_name	Varchar NOT NULL
product_price	INT NOT NULL
product_quantity	INT NOT NULL
product_discount	INT

5. **Delivery Agent** -

agent_id	INT Primary key NOT NULL
agent_username	Varchar NOT NULL UNIQUE
Agent_password	Varchar NOT NULL UNIQUE
contact_number	Varchar NOT NULL UNIQUE
contact_number2	Varchar UNIQUE
agent_email_id	Varchar UNIQUE

expected_time_of_delivery	Varchar NOT NULL

6. **Orders -**

order_id	INT Primary key NOT NULL
order_date	Date NOT NULL
order_time	Time NOT NULL
product_id	INT NOT NULL Foreign Key
customer_id	INT NOT NULL Foreign key
seller_id	INT NOT NULL Foreign key
agent_id	INT NOT NULL Foreign key

7. **Payments -**

payment_id	INT Primary key NOT NULL
order_id	INT NOT NULL Foreign Key
payment_date	Date NOT NULL
payment_time	Time NOT NULL
payment_method	Varchar NOT NULL

payment_amount	INT NOT NULL

8. Cart

customer_id	INT Foreign Key NOT NULL
product_id	INT Foreign key
quantity	INT

9. Wishlist

customer_id	INT Foreign key NOT NULL
Product_id	INT Foreign key

10. Wallet

customer_id	INT Foreign key NOT NULL
wallet_amount	INT NOT NULL

11. Feedback

feedback_id	INT Primary key NOT NULL
product_id	INT Foreign key NOT NULL
product_name	Varchar Foreign key NOT NULL
brand_name	Varchar Foreign key NOT NULL
customer_id	INT Foreign key NOT NULL
customer_name	Varchar Foreign key NOT NULL
product_feedback	Varchar NOT NULL
feedback_date	Date NOT NULL
feedback_time	Time NOT NULL

12. Lists -

category_id	INT NOT NULL Foreign key
seller_id	INT Foreign key NOT NULL

13. Adds -

agent_id	INT Foreign key NOT NULL
selelr_id	INT Foreign key NOT NULL

Weak Entity -

For our application, there is no guest cart i.e. every Cart needs to be associated with a customer, thus cart is a weak entity. Similarly, each Wallet and Wishlist is also a weak entity and are associated with a customer_id.

SQL QUERIES -

1. **Cancel order -** to cancel the order it is ensured that the order is undelivered by using the expected_time_of_delivery of the Order table. The expected delivery time should be greater than zero to cancel the order. the order details are then deleted from the order table and Payments table. This is done by using a nested query. the inventory is updated in the Products table using the JOIN command.

```
-- --check if it is undelivered--
Select order_id FROM Order_ where agent_id in (SELECT agent_id from Delivery_Agent where
expected_time_of_delivery > 0) AND Order_id = 274;

-- update inventory --
UPDATE Product A
   INNER JOIN Order_ B
   ON A.product_id = B.product_id and B.order_id = 274
SET A.product_quantity = 1 + A.product_quantity;

-- delete entry from order and payment database --
Delete from Payment where order_id in (select order_id from order_ where order_id = 274);
delete from Order_ where order_id = 274;
```

2. **Show Cart Summary -** Entries from the Product and Customer table are used to show the cart summary. Total_price_of_cart is derived from product price and quantity.

```
Select I.customer_id ,P.product_name as "Product Name", p.brand_name as "Brand",
p.product_price as "Price", i.quantity, p.product_price * i.quantity as "total_price_of_cart"
from Product p, Cart I
where I.product_id = P.product_id order by I.customer_id;
```

3. **Checking orders not delivered yet** - Entries from table order_ are used to display the order not delivered yet.

```
Select order_id as "Order_ID which hasn't been delivered yet"
FROM Order_
where agent_id in (SELECT agent_id from Delivery_Agent where
expected_time_of_delivery > 0) ORDER BY Order_ID;
```

4. **Showing feedback of each customer** - INNER JOIN is used on Feedback and Customer along with INNER JOIN on Customer and Products to list all feedback given by the particular customer.

```
SELECT f.customer_id
   , c.customer_name
   , p.product_name
   , f.product_feedback
FROM feedback f
INNER JOIN customer c
   on f.customer_id = c.customer_id
INNER JOIN product p
   on f.product_id = p.product_id
ORDER BY f.customer_id;
```

5. **Listing out products of a certain category** - a nested query is used where all DISTINCT orders where at least one product in a particular category say Grocery is ordered are listed using the Category, Product, and Orders Table

```
SELECT p.product_name, c.category_name
from product p INNER JOIN category c on c.category_id = p.category_id;
```

6. **Search all orders where products are ordered from a particular category** - a nested query is used where all DISTINCT orders where at least one product in a particular category say Grocery is ordered are listed using the Category, Product, and Orders Table

```
Select *
from order_
where order_id in(select DISTINCT x.order_id from order_ x
                  where x.product_id in(select  product_id from Category
                                        where category_name = 'Grocery'));
```

7. **checking whether a user can check out his/her cart** - a nested query is used where all DISTINCT orders where at least one product in a particular category say Grocery is ordered are listed using the Category, Product, and Orders Table

```
-- Here, customer 130 is finding out the total price of his cart
Select I.customer_id ,P.product_name as "Product Name", p.brand_name as "Brand",
p.product_price as "Price", i.quantity, p.product_price * i.quantity as "total_price_of_cart"
from Product p, Cart I
where I.product_id = P.product_id and I.customer_id = 133;

-- Here, customer 130 also checks the total amount of money in wallet
Select c.customer_id, w.wallet_amount from customer c
inner join wallet w on c.customer_id = w.customer_id and c.customer_id = 133;

-- Clearly, he wouldn't be able to check out this. So, we increase the wallet amount by 212460
rupees
Update wallet set wallet_amount = 220000 where customer_id = 133;
```

8. **update product cost by a certain percentage** - the product price in the Products table is updated with a certain percentage using the product_price column of the Products Table.

```
Update Product
set product_price = product_price*1.2
where product_id = 123;
```

9. **Inserting a new customer** -

```
insert into Customer (customer_id, customer_name, DOB, customer_number, customer_number2,
customer_email_ID, customer_city, customer_state)
values (1001, 'Jerry', '2001-06-17', '2173546035', '3181383736', 'jerry@ustream.tv', 'Agartala',
'Tripura');
```

Triggers -

1. **Delete_Customer -** if a customer is deleted from the database, its corresponding Cart, wishlist, and Wallet are also deleted. After creating the trigger, we test it by deleting a customer first and then trying to print out their wallet details.

```
DROP TRIGGER IF EXISTS Delete_customer
delimiter $$
Create trigger Delete_customer
After delete ON  Customer
FOR EACH ROW
BEGIN
DELETE from Wishlist where customer_id = old.customer_id;
DELETE from Cart where customer_id = old.customer_id;
DELETE from Wallet where customer_id = old.customer_id;
END; $$

– testing the trigger
Delete From Customer
where customer_id=444;
Select * from Wallet where customer_id=444;
```

2. **Verify_discounts** - if the discount of any product is updated to a value >80, it is set to 80%. We test the trigger by trying to update a product_discount>80, and check the results.

```
DROP trigger IF EXISTS verify_discount
delimiter //
create trigger verify_discount
before insert on Product
for each row
if new.product_discount>80 then set new.product_discount=80;
end if;//

-- trying to insert a new product with dicount >80
insert into Product (product_id, product_name, category_id, brand_name, product_price,
product_quantity, product_discount) values (1001, 'XYZ', 144, 'Walter Group', 9986, 81, 85);
select * from Product where product_id = 1001
```

OLAP QUERIES :

(1) **Finds the number of orders delivered by each delivery agent in a particular month**

```
SELECT d.agent_name, COUNT(*) AS order_count
FROM Delivery_Agent d
JOIN Order_ o ON d.agent_ID = o.agent_id
WHERE MONTH(o.order_date) = 3
GROUP BY d.agent_name WITH ROLLUP;
```

(2) **Total Orders delivered by an agent**:

```
SELECT agent_id, count(order_id) as 'Total Orders'
FROM Order_
GROUP BY agent_id with ROLLUP;
```

(3) **Retrieve the total revenue and number of orders for each seller:**

```
SELECT Seller.seller_name, SUM(Product.product_price * Product.product_quantity) AS
total_revenue, COUNT(Order_.order_ID) AS number_of_orders
FROM Product
INNER JOIN Order_ ON Product.product_id = Order_.product_ID

INNER JOIN Seller ON Order_.seller_ID = Seller.seller_ID
GROUP BY Seller.seller_name WITH ROLLUP;
```

(4) **Retrieves the total sales of a Product:**

```
SELECT Category.category_name,
SUM(Product.product_price * Product.product_quantity) AS total_sales
FROM Product
INNER JOIN Category ON Product.category_id = Category.category_ID
INNER JOIN Order_ ON Product.product_id = Order_.product_ID
WHERE Order_.order_date
GROUP BY Category.category_name WITH ROLLUP;
```

Embedded Queries -

```
1. SELECT customer_state ,COUNT(*) FROM customer GROUP BY
   customer_state;
2. SELECT C.category_name, COUNT(*) as num_products_sold
   FROM Order_ O
   JOIN Product P ON O.product_ID = P.product_ID
   JOIN Category C ON P.category_id = C.category_ID
   GROUP BY C.category_name
   ORDER BY num_products_sold DESC
   LIMIT 5
```

DB Transaction -

```
-- T1
START TRANSACTION;
select * from Customer where customer_id = 1010;
update Customer set customer_city='DELHI' where customer_id=1010;
COMMIT;

-- T2
start transaction;
update customer set customer_number= 98765432 where customer_id= 1010;
COMMIT;

-- T3
start transaction;
insert into Customer
(customer_id,customer_username,customer_password,DOB,customer_number,customer_number2,custo
mer_email_id,customer_city,customer_state)
values('1010','manshaa23','abcdef','2003-06-23','9876526','826573647','mk@gmail.com','Delhi','India');
COMMIT;
```

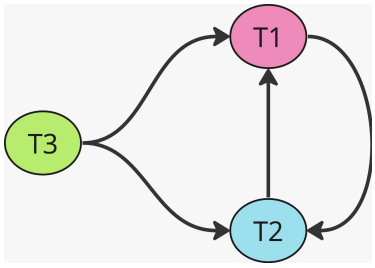
NON CONFLICT SERIALISABLE SCHEDULE :

STEP	T1	T2	T3
1.	START		
2.		START	
3.			START
4.			write(Customer)
5.			COMMIT
6.	read(Customer)		
7.		write(Customer)	
8.		COMMIT	
9.	write(COMMIT)		
10.	COMMIT		

Operation Table :

STEP	TRANSACTIONS	OPERATIONS	DATA VALUE
1.	T1	START TRANSCTION;	-
2.	T2	START TRANSACTION;	-
3.	T3	START TRANSACTION;	-
4.	T3	insert into Customer (customer_id,customer_username,customer_password,DOB,customer_number,customer_number2,customer_email_id,customer_city,customer_state) values('1010','manshaa23','abcdef','2003-06-23','9876526','826573647','mk@gmail.com','Delhi','India');	All
5.	T3	COMMIT;	-
6.	T1	select * from Customer where customer_id = 1010;	All
7.	T2	update customer set customer_number= 98765432 where customer_id= 1010;	customer_number
8.	T2	COMMIT;	-
9.	T1	update Customer set customer_city='DELHI' where customer_id=1010;	customer_city
10.	T1	COMMIT;	-

Precedence Graph :



As we can observe, G(V',E') where V'={T1,T2} there exists a cycle in the subgraph.
Thus, the schedule is non conflict seriliasable.

Conflict Serialisable Schedule :

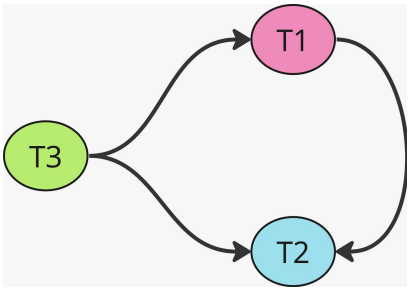
STEPS	T1	T2	T3
1.	START		
2.		START	
3.			START
4.			write(Customer)
5.			COMMIT
6.	read(Customer)		
7.	write(Customer)		
8.	COMMIT		
9.		write(Customer)	
10.		COMMIT	

Operations Table -

STEPS	TRANSACTIONS	OPERATIONS	DATA VALUE
1.	T1	START TRANSACTION;	-
2.	T2	START TRANSACTION	-
3.	T3	START TRANSACTION	-
4.	T3	insert into Customer (customer_id,customer_username,customer_password,DOB,customer_number,customer_number2,customer_email_id,customer_city,customer_state) values('1010','manshaa23','abcdef','2003-06-23','9876526','826573647','mk@gmail.com','Delhi','India');	All
5.	T3	COMMIT;	-
6.	T1	select * from Customer where customer_id = 1010;	All
7.	T1	update Customer set customer_city='DELHI' where customer_id=1010;	Customer_city
8.	T1	COMMIT;	-

9.	T2	update customer set customer_number= 98765432 where customer_id= 1010;	customer_number
10.	T2	COMMIT;	-

Precedence Graph :



As observed, the graph is acyclic.
Thus, the schedule is serialisable.

User Guide:

Firstly, we are presented with a screen which says, Submission for the final deadline of DBMS and we are given an option to continue or to exit.
On further continuing, we are given 3 options which is, to enter as a customer, a seller or as a delivery agent.

Customer Features:

1. We have implemented a login / signup page for the customers.
2. While signing up, we are presented with all the details that the customer needs to fill.
3. For logging in, we take inputs of the customer_id, customer_username and the customer's password. The customer_id shall be used everytime we try to achieve a particular task related to the customer.
4. If the customer's username and password matches with the data in our database, we will enter the customer menu.
5. The customer menu has about 21 options where one option is to exit the menu.
6. **Change Personal Details**: It change personal details whether it's the username, password, date of birth, primary phone number, secondary phone number, email address, city or state.
7. **View Categories**: It allows us to view all the distinct categories available in FLIPKART database.
8. **View Products**: It allows us to view all the products available in FLIPKART database.
9. **View Cart** : It allows us to view the cart of the particular customer who has logged in.
10. **Add Products to Cart**: It allows us to add a specific product into the customer's cart based on the product_id which the customer has to provide.
11. **Add Money to Wallet:** It allows us to add money to the wallet of the customer.
12. **Check wallet Balance:** It allows the customer to view the remaining balance in his/her wallet
13. **Remove Product from Cart**: It allows the customer to remove a particular product from the cart.
14. **View Orders:** It allows the customer to view all the orders he/she has in his/her orders list.
15. **Checkout**: It allows the customer to checkout all the products from the cart. It also handles cases where the customer must have the required amount of money in his/her wallet for the checkout to happen and the amount of products are also consequently reduced in the inventory.
16. **View Delivery Status:** It allows the customer to see how many days it will take for the order to be delivered.
17. **Empty Cart**: It allows the customer to empty his/her cart.
18. **View Product Reviews**: It allows the customer to view all the product reviews.
19. **View your product reviews**: It allows the customer to view the product reviews which he/she has made.
20. **Add Product Reviews**: It allows the customer to make product reviews.

21. **Remove Product Reviews**: It allows the customer to remove product reviews which he/she has made.
22. **View Wishlist**: It allows the customer to view the wishlist that he/she has created.
23. **Add Product to Wishlist**: It allows the customer to add products inside his/her wishlist.
24. **Remove Product from Wishlist:** It allows the customer to remove products from his/her wishlist.
25. **Clear Wishlist:** It allows the customer to clear his/her wishlist.

Seller Featuers: On continuing to enter as a seller, the following options :

1. **Login / Signup :** we have implemented a login/signup page for the seller
2. To login, the seller need to fill in their seller_id, seller_username, seller_password.
3. To Signup, the seller has to input all his details to register his account on the website.
4. If he chooses to login he is further presented with the seller features, if he chooses to signup, after creating his account he is required to login and is presented with the features.
5. The features include :
    - **Search category :** the user can search all the categories available on the website.
    - **Search product** : the user can search all the products available on the website.
    - **Add Category:** as a seller, a user can add any category to the database which will be available to all other users on the website.
    - **Add product** : as a seller, a user can add any product under any existing category which will be available for all the users to view.
    - **Assign delivery agent:** the user can assign a delivery agent to any existing order of his products.
    - **Show all previous order :** the user can look at all hsi previous order which have been delivered.
    - **Show all running orders** : the user can look at his running orders which have not ben delivered yet.
    - **Total revenue** : the user can calculate his overall revenue earned through te website through this option.
6. Lastly, the user is also given the option to exit at any point.

Delivery Agent Featuers: on continuing as a delivery agent the user is presented with the following options :

1. **Login / Signup** - the user is presented with the option to either login if his account already exists or signup if he wants to register a new account.

2. If the user chooses to login, the user is required to enter the agent_id, agent_username, agent_password. If the details match with the database, he is presented with the delivery agent features.
3. If the user chooses to signup, he is required to enter all his details and after his account us registered, he is required to login again. Once he has logged in, he is presented with the agent features -
    - **Show all running order**s - the user can look at his running orders which are not delivered yet.
    - **Show all previous orders** - the user can look at his previous orders, which he has already delivered.
    - **Update delivery status** - the user can update the delivery status of all his running orders.