B. M. S. COLLEGE OF ENGINEERING

(Autonomous Institute, Affiliated to VTU, Belagavi)

Post Box No.: 1908, Bull Temple Road, Bengaluru – 560 019

DEPARTMENT OF MACHINE LEARNING

A project report on

"VALMART (E-commerce Website)"

Submitted in partial fulfilment of the requirements for the award of degree

B.E IN

Artificial Intelligence and Machine Learning

BY

Student Name:	**Adithya G Jayanth**	**Amal Srivatsava**	**Chetna Mundra**
USN:	**1BM21AI024**	**1BM21AI036**	**1BM21AI036**

Under the guidance of

Dr. Sandeep Varma N

Associate Professor

Academic Year: 2022-2023 (Session: October 2022 - February 2023)

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institute, Affiliate to VTU)

Bull Temple Road, Basavanagudi, Bangalore – 560019

Department of Machine Learning

C E R T I F I C A T E

This is to certify that the group project entitled "**ValMart (E-commerce Website)**" is a bona-fide work carried out by **(1BM21AI000) Adithya G Jayanth, (1BM21AI000) Amal Srivatsava, (1BM21AI036) Chetna Mundra** in partial fulfillment for the award of degree of B.E in Artificial Intelligence and Machine Learning from **Visvesvaraya Technological University, Belgaum** during the year **2022-23**. The group project report has been approved as it satisfies the academic requirements in respect of group project prescribed for the Degree.

Signature of the Guide	**Signature of the HoD**
Dr. Sandeep Varma N	**Dr. Gowrishankar**
Associate Professor	Professor & HOD

TABLE of CONTENTS

CHAPTER-1

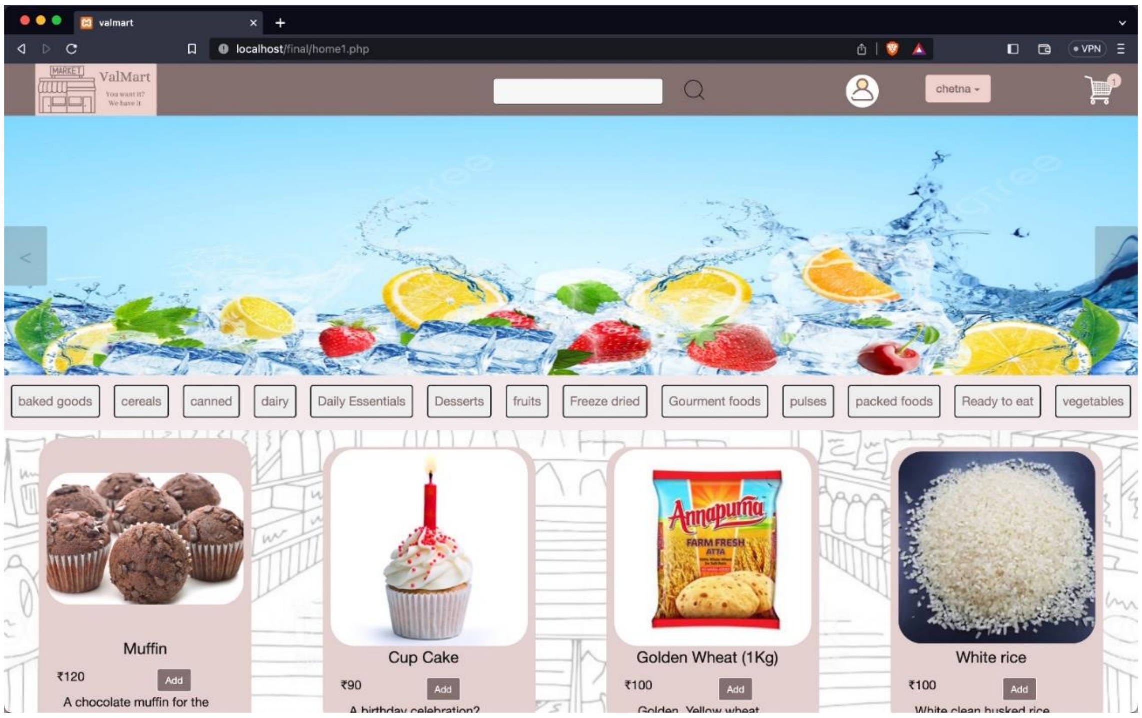
INTRODUCTION

Valmart

ValMart is an e-commerce website designed to offer an online grocery shopping experience with door-to-door delivery services. The website provides a broad range of products, including fresh produce, dairy, pantry essentials, prepared meals, and meal kits. The user-friendly interface enables customers to browse through the store's contents and place orders seamlessly. Users can create accounts to track their orders and purchase history.

ValMart's database management system is built using MySQL with PHP, which helps to manage and organize the data handling required by the application. The website's dynamic pages are created using HTML, CSS, JS, AJAX, and Bootstrap, ensuring a smooth and interactive experience for users.

Delivery personnel can access a list of outgoing deliveries, making it easy to choose and manage deliveries efficiently. Administrators have the capability to modify the contents and quantities of the store inventory, ensuring that the website's product offerings are always up-to-date.



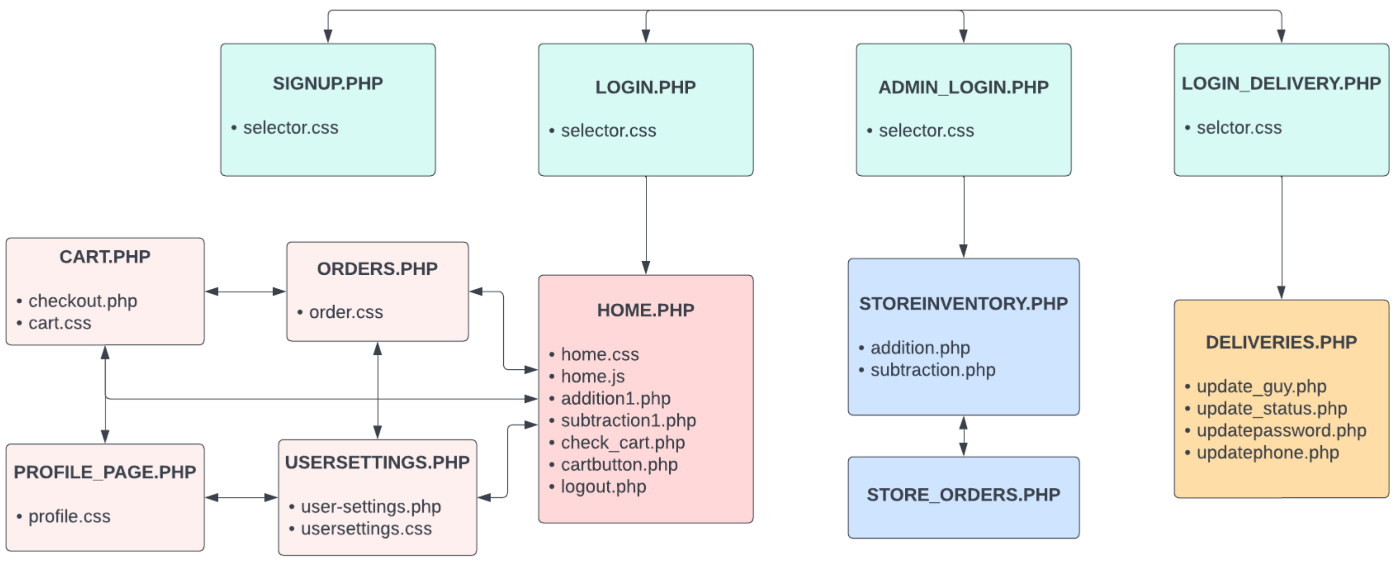
In summary, ValMart is an innovative solution for people who prefer the convenience of online grocery shopping. With its intuitive interface, and reliable delivery services, ValMart is sure to become a popular choice for those who value their time and want to simplify their grocery shopping experience.

CHAPTER-2

1. WEBSITE MODEL

We start with the main or landing page, which is the store page without being logged in. This is useful to new users as they can browse through the contents of the store without actually buying anything. This page is linked to two pages - one for signup and the other to login. One can sign up only as a user but can login either as a user, delivery personnel or administrator. Singing in as a user, allows them to use features of the website such as shopping cart, placing an order, order history, and account management.

Delivery personnel can login to accept and change the delivery status of the deliveries they accept, and their own status to indicate if they are free or mid-delivery. Administrators can login to view information about all orders placed and update the prices and quantities of items in the store inventory. They can also add new products and their images into the database. All 3 classes of users can login from different forms on the same pages. The following diagram shows a brief breakdown of the main pages used by the website and all the supporting parts involved.



2. DESCRIPTION OF PAGES AND SUPPORTING FILES

1. **SIGNUP.PHP**
   This Webpage is designed for customers to sign in to a new account. It checks if email is unique and password is the same and alerts if not. the *form in POST* method and redirects it toLogin.php. It hashes passwords given by the user using the "*hash_password*" function in php and then stores them in the database.

   - **selector.css**
     This is a style sheet used to style signup.php. It has made the web page *fully responsive.*

2. **LOGIN.PHP**
   This webpage is designed for Customers to login using their Email Address and Password. It uses the *form in POST* method and redirects it to Home.php. It uses "*password_verify"* method from php to verify the hashed password.

   - **selector.css**
     This is a style sheet used to style login_delivery.php and login.php. It has made the web page *fully responsive*.

3. **LOGIN_DELIVERY.PHP**
   This webpage is designed for *Delivery executives to login*. Delivery executives can login from their Aadhar number and their password.

   - **selector.css**
     This is a style sheet used to style login_delivery.php and login.php. It has made the web page *fully responsive*

4. **ADMIN_LOGIN.PHP**
   This webpage is designed for *administrators to login*. There is a particular username and password for admin to login which takes them to admin page.

   - **selector.css**
     This is a style sheet used to style admin_login.php. It has made the web page *fully responsive*

     .

5. **HOME.PHP**
   This webpage is the *MAIN* webpage and consists of a navigation bar, side bar for cart, main div and footer.

Navigation bar consists of icon, name, search bar which is fully functional and shows the filtered search results. After the results it again shows all items not shown in the result. If an item is not found, it displays a message.
It has a logout button and a button to show all users accessible web pages like user settings, profile, order history.
Then there is a cart button which has a small badge over it to show how many items are currently in the cart. It updates every time an item is added or removed from the cart using ajax request.

There are an *array of images* that highlight the importance of our webpage displaying important information and highlights of the store.

There is a *category bar* below it which shows all the categories of items that are displayed on the main div. When clicked on any of the categories, it shows all the items that belong to the same category and then shows all other items.

There is an *item div* which shows all the items in the database with fully functional '+' and '-' buttons that show quantity in the cart and updates every time using ajax request.

The sidebar opens up when the button cart in the navigation bar is clicked. It pushes the main page to the side. It shows all the items that are currently in cart, shows total amount and has a button to checkout which uses the *form POST* method and proceeds to fill the order. Cart gets updated every time an item is added or removed from the cart using ajax request and the page doesn't need to refresh itself.

There is a footer which opens up when clicked on the button and it slides up to show the contact information related to our store and website.

- **home.css**
  This is a style sheet used to style home.php and it helps the website to be *fully responsive.*

- **home.js**
  There are many functions in this.

  Function updateCart- It creates a new XMLHttpRequest object and sets the onreadystatechange function to handle the response from the server.
  When the ready state changes to XMLHttpRequest.DONE and the status is 200 (OK), it gets the new updated element for which ID is given from the DOM and sets its inner HTML to the response text. If the status is not 200, it logs an error message to the console.
  Then it opens a new POST request to the server using the action URL, sets the Content-Type header to application/x-www-form-urlencoded, and sends the required parameter in the request body.o change the quantity of the item in cart inside the item display. It also calls 2 more functions those are-

Function updatesidebar- it uses ajax to update the cart without refreshing page using similar method as updateCart

Function updatecartbutton- it uses ajax to update the cart without refreshing the page using a similar method as updateCart.

- **addition1.php**
  This is a supporting file that changes and replaces the div inside each item that is responsible for showing *quantity* and + and - button. It uses *Ajax requests* to only display that div and not refresh the whole page to show changes. This file is used whenever a '+' button is clicked inside item div or inside the cart. It updates the database and increases that particular item by 1 in the cart.

- **subtraction1.php**
  This is a supporting file that changes and replaces the div inside each item that is responsible for showing *quantity* and + and - button. It uses *Ajax requests* to only display that div and not refresh the whole page to show changes. This file is used whenever a '-' button is clicked inside item div or inside the cart. It updates the database and decreases that particular item by 1 in the cart.

- **check_cart.php**
  This is a supporting file that uses *Ajax requests* to update the cart whenever a new item is added or an item is removed from the cart. It replaces the *cart table* without refreshing the page from the database whenever '+' or '-' button is clicked.

- **cartbutton.php**
  This is a supporting file that uses *Ajax requests* to update the *cart button* which shows the quantity of items in the cart whenever a new item is added or an item is removed from the cart. It replaces the cart button without refreshing the page from the database whenever '+' or '-' button is clicked.

- **logout.php**
  This is a supporting file that logs out the user and changes session when clicked on the logout button.

6. <u>ORDERS.PHP</u>

This page is for showing the *order history* of the user and showing the billing information along with that. It uses details and summary a tag to do so.

- **orders.css**
  It is a stylesheet for orders.php and helps to make the page *fully responsive.*

7. PROFILE_PAGE.PHP

This page gives *easy access* to the home page, billing page, user settings page and order history page. Users can review anything from here easily.

- **profile.css**
  It is a stylesheet for profile_page.php and helps to make the page *fully responsive.*

8. CART.PHP

This page is shown only where the user wants to *create a bill and confirm delivery*. He can even directly access it from profile_page.php. The user can't make changes to quantity here, as it shows the final bill. If there are no items in the cart, it displays an alert to add items first.

- **checkout.php**
  This is a supporting file that calls procedure *order_user* in the database and uses the form in the POST method and checks out the items in the cart.

- **cart.css**
  It is a stylesheet for cart.php and helps to make the page *fully responsive.* It uses bootstrap tables.

9. USERSETTINGS.PHP

This page allows the user to change his/her information that has been uploaded in the web page such as phone number, address, email, name, password etc. It uses a *form with POST* request and redirects it to user-settings.php to update it in the database.

- **user-settings.php**
  It takes in the form input from usersettings.php and *updates user information* in the database and redirects to usersettings.php.

10. ADMIN.PHP

This page is for the *store owner* to update the quantity of all items and keep track of these items. they can add or remove items from the database directly. This page uses *Bootstrap tables and is fully responsive.*

- **addition.php**
  This page uses *Ajax requests* and changes the div of the item in admin.php wherever a '+' button is clicked. It adds that item in store inventory in the database by 1 and reflects changes that is its quantity in admin.php without refreshing the page.

- **subtraction.php**
  This page uses *Ajax requests* and changes the div of the item in admin.php wherever a '-' button is clicked. It decreases that item in store inventory in the database by 1 and reflects changes that is its quantity in admin.php without refreshing the page.

11. STORE_ORDERS.PHP

This webpage shows all the orders that have been placed by all users in 1 table. Using summary and details tag, when clicked on a particular order in its row, it spans down to show the *bill of order, customer details and delivery personnel details*. User ID, Customer ID, Delivery personnel ID, Date of order, Date of Delivery, Status, Total amount is shown in the table row.

In bill, item picture, item name, its price, quantity and amount is shown.
In customer information, it shows customer name, phone number, email and Address.
In delivery personnel information, it shows Name, phone number, license and Aadhar Number.
This page uses *Bootstrap tables and grids and is fully responsive.*

12. DELIVERIES.PHP

This webpage is designed exclusively for *Delivery personnel.*
There is a top bar that has a button on the left to open the side panel to change his n*umber or password.* Then the page welcomes him with his name and a logout button on the right.
There is then a set status button which Delivery Personnel can change to *active or inactive* status according to his/her convenience.

When his status is active, A table shows the deliveries that are pending, and he can choose a delivery from that list. The table shows customer name, customer phone number, and location. After he chooses to deliver some order, he can change the delivery status from order confirmed to delivery confirmed and then from delivery confirmed to dispatched and then from dispatched to delivered.

If he is active on Delivery, the Set status button is set to "*busy*" and the button is disabled.

Irrespective of the status, All the orders that the Delivery Personnel has delivered, are shown in the form of a table with location, name of customer, phone number of customer, and date of delivery.

This page uses *Bootstrap tables and is fully responsive.*

- **update_guy.php**

This is a subfile to update the delivery personnel status from *active to inactive* whenever the delivery guy presses a button using the form by *POST* method and reflects the changes in the database.

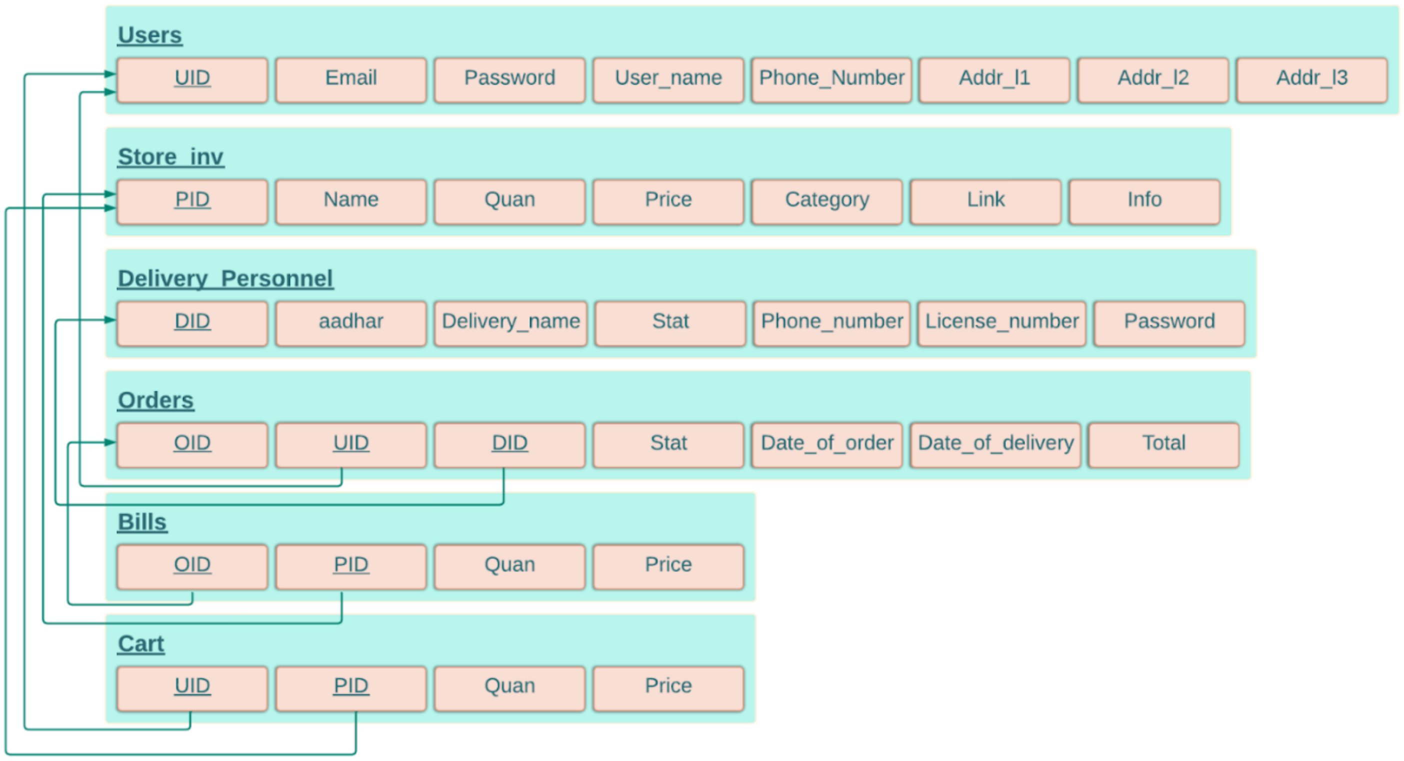
- **update_status.php**
  This is a subfile to update the status of the Ongoing delivery.
  Personnel can choose from the list of deliveries to see which they want to deliver and in the backend it calls the function *assign_order* and changes the status of delivery guy to busy.
  It is also for changing order status from delivery confirmed to *dispatch* and from dispatch to *delivered* in which it calls the procedure *complete_order* in the database.

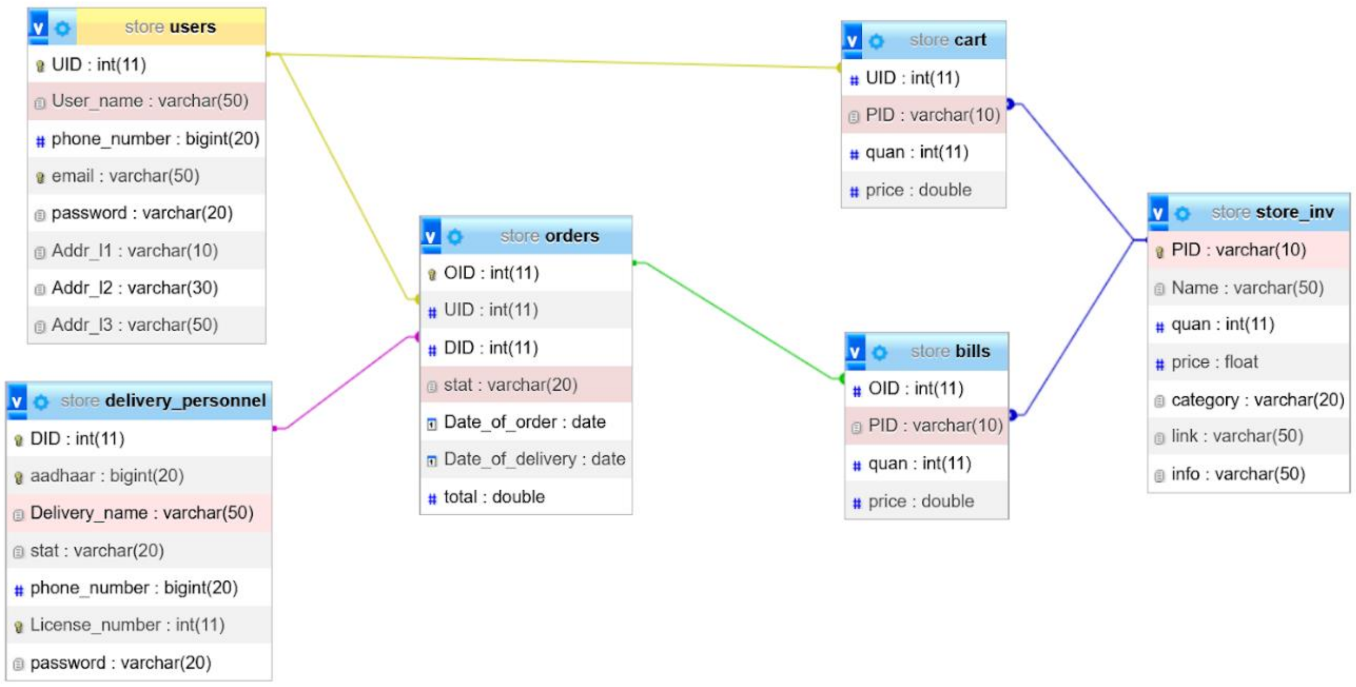
- **updatepassword.php**
  This file is for handling the update password request using the *form in POST* method. it displays error messages and handles changes in the backend as required.

- **updatephone.php**
  This file is for handling the update phone number request using the *form in POST* method. it displays error messages and handles changes in the backend as required.

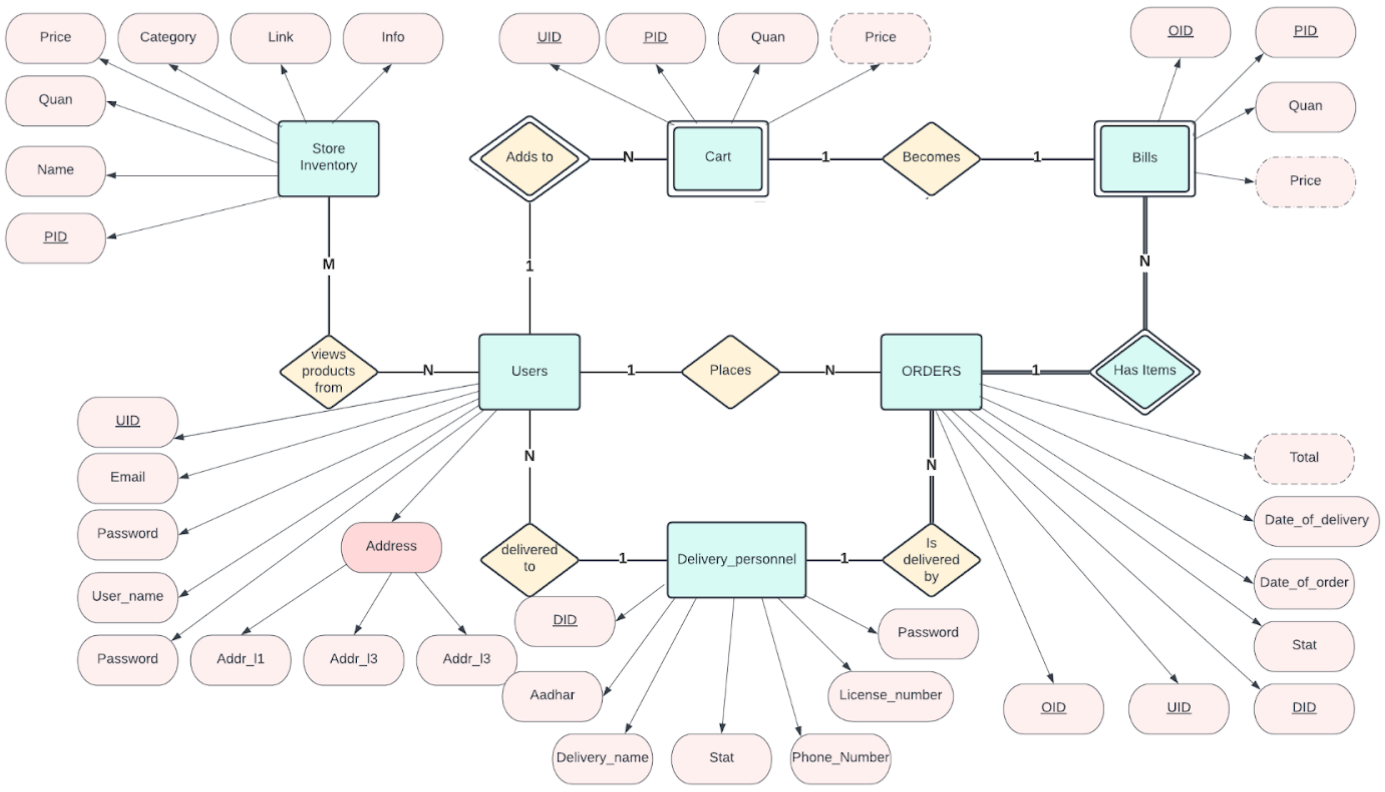
3. DATABASE DESIGN

1. SCHEMA





2. ER DIAGRAM



3. TABLES

Store_inv

First, we start off with a base relation for the product catalog. This will be called store inventory or store_inv for short. This relation will contain a unique product identifier - “PID” or product ID. It will contain the name of the product, the quantity of the products present in the inventory, its price, the category which it falls under, and some supporting information about the product. Further, to ease the interface between PHP and the database, it contains another column with the local device file address of the image corresponding to that product.

Users

This table will contain information about the users. Users can create an account and login using any valid email address. The table contains password, user_name, their phone numbers and three lines of address for delivery. In addition for efficient query processing, the table will contain a User ID column or UID. This will be an abstraction and will only be used on the backend. This will be the primary key and will be set to auto-increment for creating new users.

Delivery_Personnel

This table contains information about Delivery Personnel. They will be required to sign in using their Aadhaar as a safety measure. It contains a column for password, their current status, phone number, Driving license number and a Driver ID or DID. As in the users table, DID will be an abstraction.

Orders

Tabel to keep track of the stakeholders participating in a given order and some supporting information. Contains Order ID to uniquely identify an order, the DID of the person delivering the order, the UID of the person placing the order, the dates of order and delivery and the status and cost of the order.

Bills

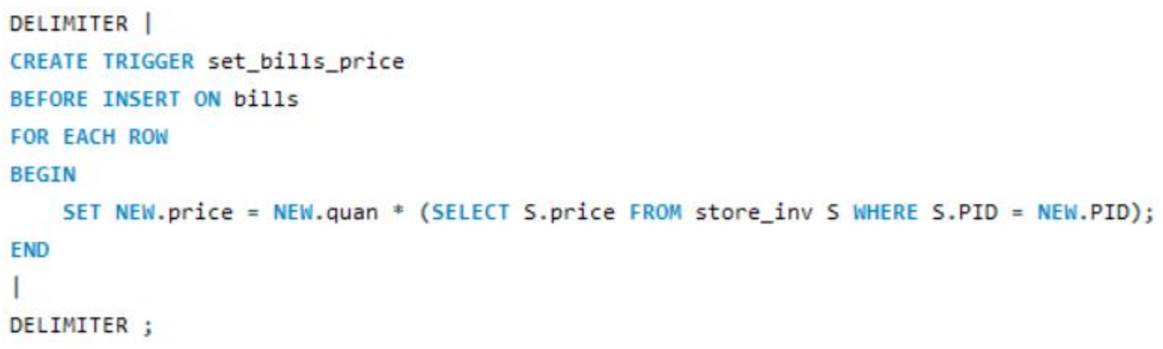
To maintain a record of previously placed orders. Consists of the OID, PIDs of the products that were ordered in the order, their respective quantities and prices.

Cart

Contains details of users current shopping cart. Consists of the UID of the user, the PID of the products in the users cart, and their respective quantities and prices.

4. TRIGGERS

- **Set_bills_price**



```
DELIMITER |
CREATE TRIGGER set_bills_price
BEFORE INSERT ON bills
FOR EACH ROW
BEGIN
    SET NEW.price = NEW.quan * (SELECT S.price FROM store_inv S WHERE S.PID = NEW.PID);
END
|
DELIMITER ;
```

While inserting into the bills table, we only insert the OID, PID and quan. Instead of having to check the price of the corresponding item each time and having to calculate it manually, we write a trigger to do the same for us. Thus, upon insertion of a new row containing the OID, PID and quantity, we calculate the price of the item by using the price from the store_inv table. We multiply the newly inserted quantity with its corresponding price from the store_inv table. This also saves storage space as we don't have to redundantly store the cost of the individual items a second time in a new column

- **Set _cart_price**

This trigger is similar to the above trigger. We only insert the UID, PID and quantity. The price of the items is calculated by the trigger.

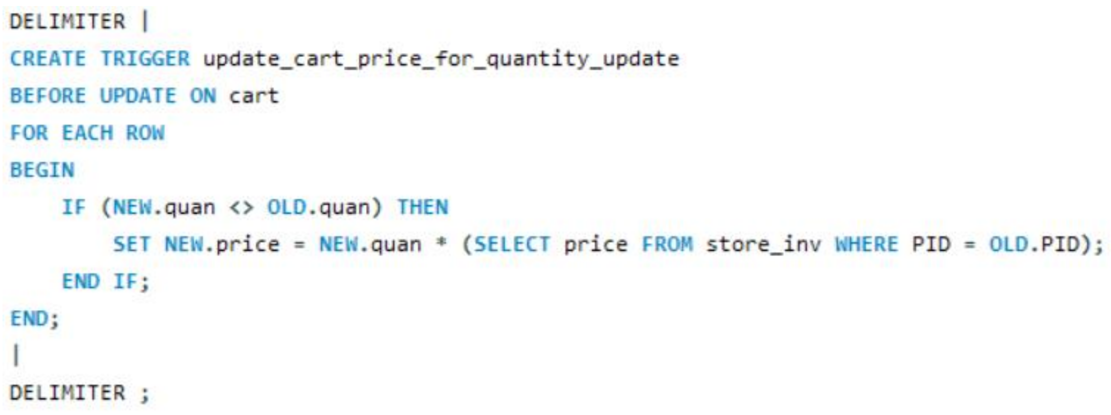
```
DELIMITER |
CREATE TRIGGER set_cart_price
BEFORE INSERT ON cart
FOR EACH ROW
BEGIN
    SET NEW.price = NEW.quan * (SELECT S.price FROM store_inv S WHERE S.PID = NEW.PID);
END
|
DELIMITER ;
```

- **Update_cart_price_for_inventory_update**

Since the database is a dynamic one, information and prices change everyday. We want these changes to reflect in the carts of users. This trigger does exactly that. Everytime the DBA changes the price of an item in the store_inv, the corresponding prices for that item in the cart of all users are recalculated and updated.

```
DELIMITER |
CREATE TRIGGER update_cart_price_for_inventory_update
AFTER UPDATE ON store_inv
FOR EACH ROW
BEGIN
    IF (NEW.price <> OLD.price) THEN
        UPDATE cart SET cart.price = cart.quan * NEW.price WHERE NEW.PID = cart.PID;
    END IF;
END;
|
DELIMITER ;
```

- **Update_cart_price_for_quantity_update**

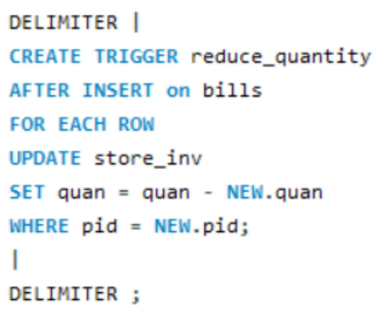


```
DELIMITER |
CREATE TRIGGER update_cart_price_for_quantity_update
BEFORE UPDATE ON cart
FOR EACH ROW
BEGIN
    IF (NEW.quan <> OLD.quan) THEN
        SET NEW.price = NEW.quan * (SELECT price FROM store_inv WHERE PID = OLD.PID);
    END IF;
END;
|
DELIMITER ;
```

This trigger is similar to the above one. This changes the price as and when a user changes the quantity of items in their cart.

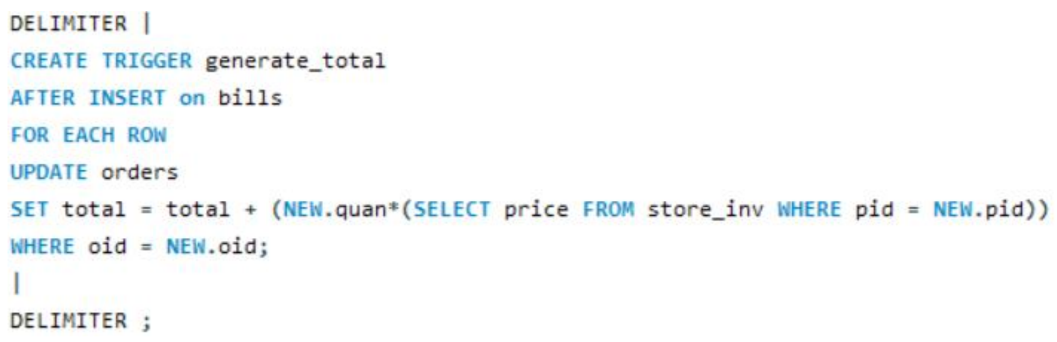
- **Reduce_quantity**

This trigger reduces the quantity of items in the inventory after a successful order has been placed. I.e., after the contents of a user's cart have been moved into bills. This trigger however will fail if the check quantity constraint in the store_inv table fails. This failure is handled by a transaction in the following section.



```
DELIMITER |
CREATE TRIGGER reduce_quantity
AFTER INSERT on bills
FOR EACH ROW
UPDATE store_inv
SET quan = quan - NEW.quan
WHERE pid = NEW.pid;
|
DELIMITER ;
```

- **Generate total**



```
DELIMITER |
CREATE TRIGGER generate_total
AFTER INSERT on bills
FOR EACH ROW
UPDATE orders
SET total = total + (NEW.quan*(SELECT price FROM store_inv WHERE pid = NEW.pid))
WHERE oid = NEW.oid;
|
DELIMITER ;
```

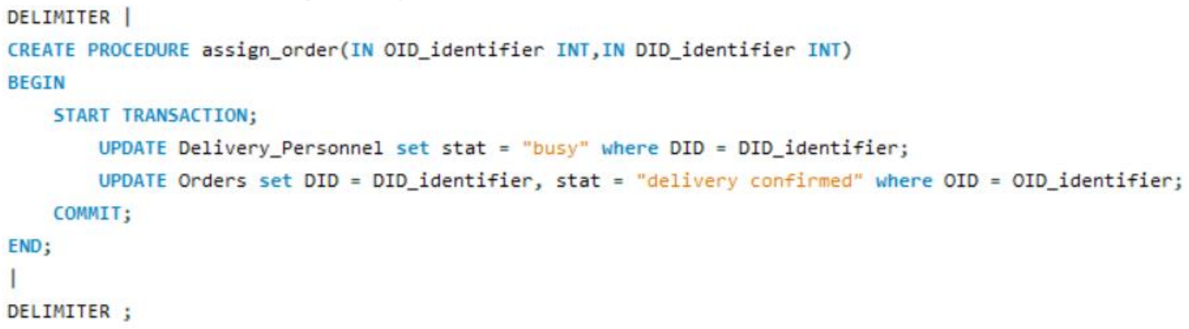
This trigger calculates the total price for each item that exists in a confirmed order - i.e., whenever data is inserted into the bills relation.

5. PROCEDURES

There are some operations that require the simultaneous and proper insertion and update of multiple tables at once. Should any one operation fail, all of them must be reverted back. This is achieved by the use of transactions. The transactions used are given below.

- **Assign_order**

The assign order transaction is used to allot an order to a given Delivery Personnel. It takes in the Order ID and the DID as parameters. Upon successful execution, it changes the order and DID status in the Orders table and the status of the Delivery Personnel in the corresponding table.

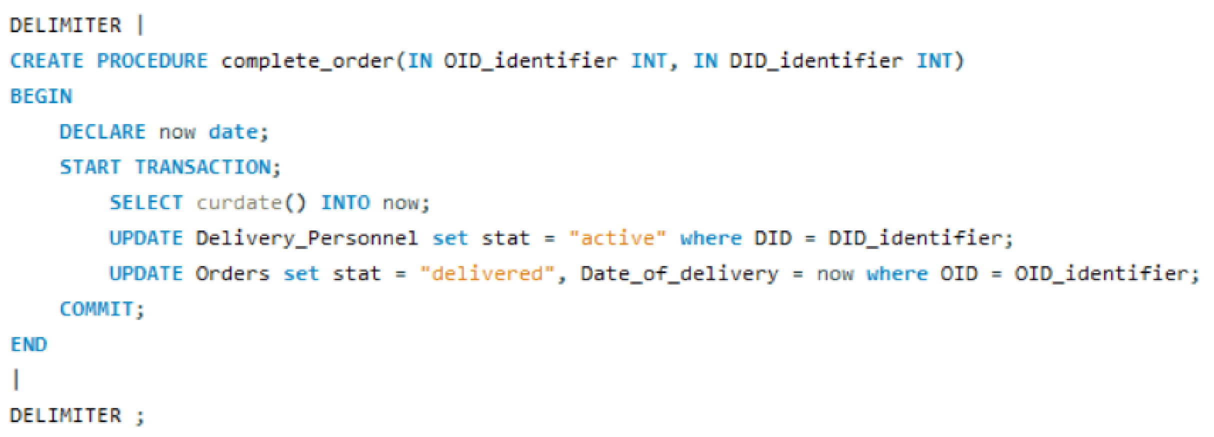


```
DELIMITER |
CREATE PROCEDURE assign_order(IN OID_identifier INT,IN DID_identifier INT)
BEGIN
    START TRANSACTION;
        UPDATE Delivery_Personnel set stat = "busy" where DID = DID_identifier;
        UPDATE Orders set DID = DID_identifier, stat = "delivery confirmed" where OID = OID_identifier;
    COMMIT;
END;
|
DELIMITER ;
```

To call the procedure use “call assign_order(UID,DID);”

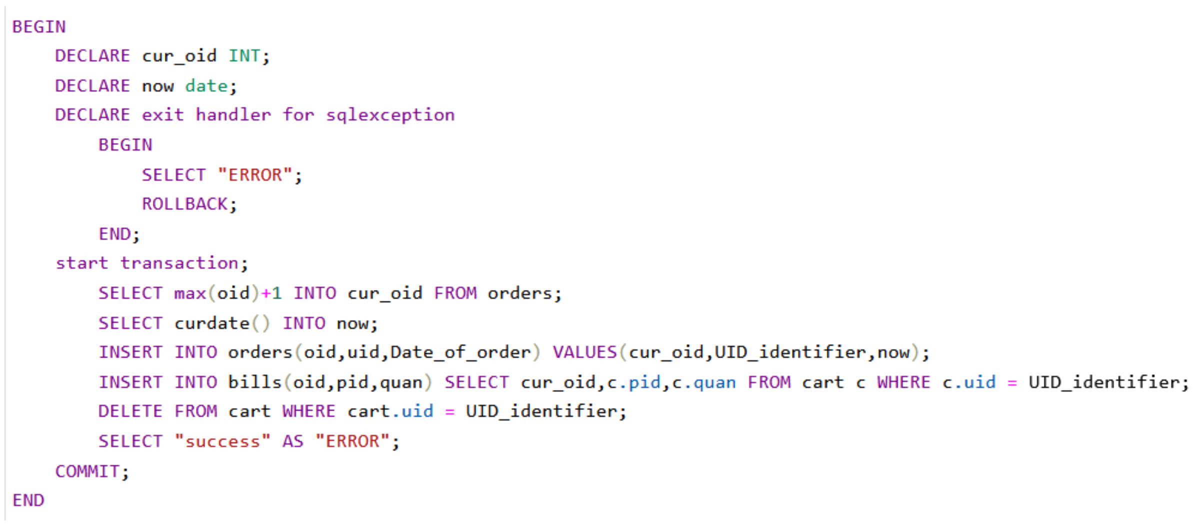
- **Complete_order**

This transaction changes the status of the delivery personnel and the corresponding order. It also updates the date of delivery in the orders table. It takes the OID and DID as input parameters.



```
DELIMITER |
CREATE PROCEDURE complete_order(IN OID_identifier INT, IN DID_identifier INT)
BEGIN
    DECLARE now date;
    START TRANSACTION;
        SELECT curdate() INTO now;
        UPDATE Delivery_Personnel set stat = "active" where DID = DID_identifier;
        UPDATE Orders set stat = "delivered", Date_of_delivery = now where OID = OID_identifier;
    COMMIT;
END
|
DELIMITER ;
```

- **Order_user**



```
BEGIN
    DECLARE cur_oid INT;
    DECLARE now date;
    DECLARE exit handler for sqlexception
        BEGIN
            SELECT "ERROR";
            ROLLBACK;
        END;
    start transaction;
        SELECT max(oid)+1 INTO cur_oid FROM orders;
        SELECT curdate() INTO now;
        INSERT INTO orders(oid,uid,Date_of_order) VALUES(cur_oid,UID_identifier,now);
        INSERT INTO bills(oid,pid,quan) SELECT cur_oid,c.pid,c.quan FROM cart c WHERE c.uid = UID_identifier;
        DELETE FROM cart WHERE cart.uid = UID_identifier;
        SELECT "success" AS "ERROR";
    COMMIT;
END
```

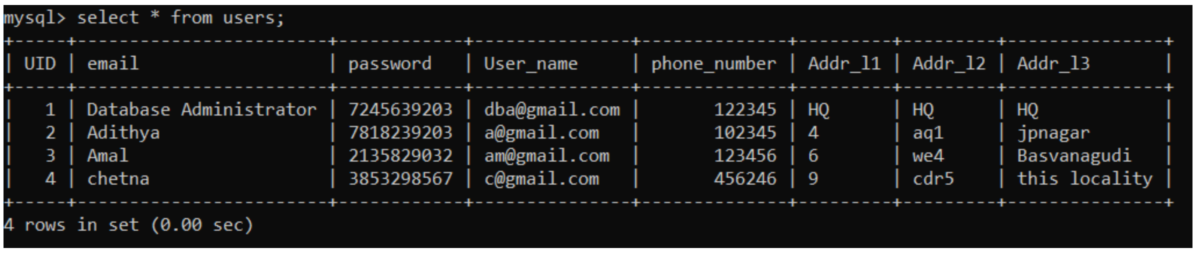
This is the most important transaction of the database. This is used to place an order. It takes in the UID as an input parameter. First, it creates a new order in the orders table and sets the date of order. Then it moves all the contents of the given users cart into the bills table. This sets off the above discussed triggers. Finally the same is erased from the cart table. If at any point the **CHECK_quan** constraint is violated, the entire transaction is rolled back. This is done so as to eliminate the possibility of junk entries in the orders table.

Note: The transactions have been executed as a part of stored procedures.

6. Sample Database Operations (standalone)

This section shows an example

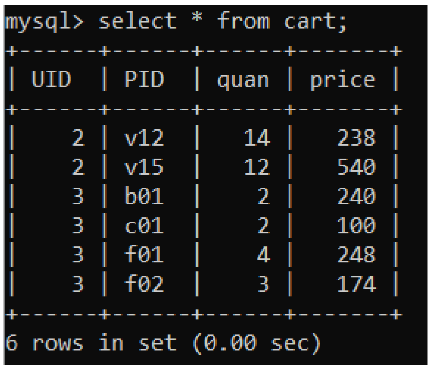
Sample state of Users:



```
mysql> select * from users;
+-----+-------------------------+------------+---------------+--------------+---------+---------+---------------+
| UID | email                   | password   | User_name     | phone_number | Addr_l1 | Addr_l2 | Addr_l3       |
+-----+-------------------------+------------+---------------+--------------+---------+---------+---------------+
|   1 | Database Administrator  | 7245639203 | dba@gmail.com |       122345 | HQ      | HQ      | HQ            |
|   2 | Adithya                 | 7818239203 | a@gmail.com   |       102345 | 4       | aq1     | jpnagar       |
|   3 | Amal                    | 2135829032 | am@gmail.com  |       123456 | 6       | we4     | Basvanagudi   |
|   4 | chetna                  | 3853298567 | c@gmail.com   |       456246 | 9       | cdr5    | this locality |
+-----+-------------------------+------------+---------------+--------------+---------+---------+---------------+
4 rows in set (0.00 sec)
```

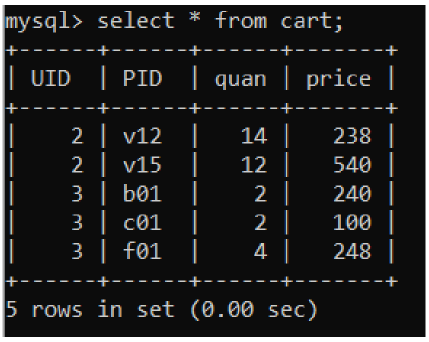
Output 1:

Inserting into cart.
Triggers fired: Set_cart_price



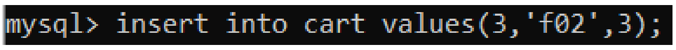
```
mysql> select * from cart;
+------+------+------+-------+
| UID  | PID  | quan | price |
+------+------+------+-------+
|    2 | v12  |   14 |   238 |
|    2 | v15  |   12 |   540 |
|    3 | b01  |    2 |   240 |
|    3 | c01  |    2 |   100 |
|    3 | f01  |    4 |   248 |
|    3 | f02  |    3 |   174 |
+------+------+------+-------+
6 rows in set (0.00 sec)
```

Fig 1.1



```
mysql> select * from cart;
+------+------+------+-------+
| UID  | PID  | quan | price |
+------+------+------+-------+
|    2 | v12  |   14 |   238 |
|    2 | v15  |   12 |   540 |
|    3 | b01  |    2 |   240 |
|    3 | c01  |    2 |   100 |
|    3 | f01  |    4 |   248 |
+------+------+------+-------+
5 rows in set (0.00 sec)
```

Fig 1.2



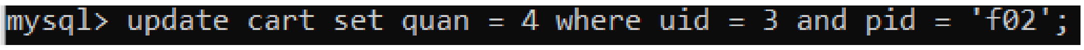
```
mysql> insert into cart values(3,'f02',3);
```

Fig 1.3

Figure 1.1 shows a snapshot of the cart table. Figure 1.3 shows a query used to insert a new row on the cart table. This triggers the set_cart_price trigger. Thus, although we have only inserted UID,PID and quan, the price is automatically filled.

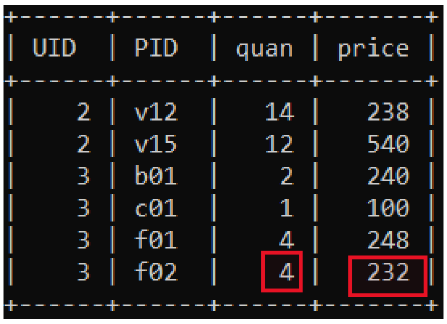
Output 2:

Updating a quantity in the cart.
Trigger fired: Update_cart_price_for_quantity_update



```
mysql> update cart set quan = 4 where uid = 3 and pid = 'f02';
```

Figure 1.2 shows the cart table before the above query is executed. Since a quantity value has changed, the trigger Update_cart_price_for_quantity_update fires. The output is seen in figure 2.1 where the price of UID = 3 and PID = f02 has changed from 174 to 232.

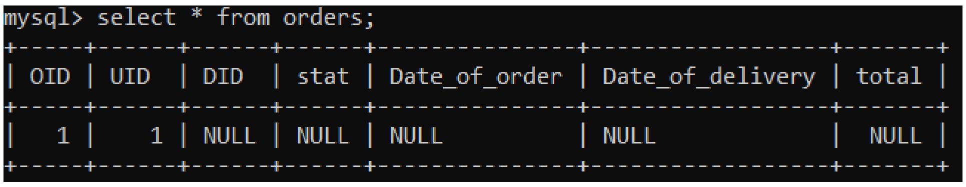


```
+------+------+------+-------+
| UID  | PID  | quan | price |
+------+------+------+-------+
|    2 | v12  |   14 |   238 |
|    2 | v15  |   12 |   540 |
|    3 | b01  |    2 |   240 |
|    3 | c01  |    1 |   100 |
|    3 | f01  |    4 |   248 |
|    3 | f02  |    4 |   232 |
+------+------+------+-------+
```

Fig 2.1

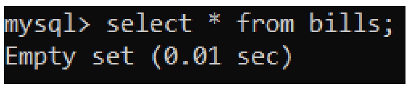
Output 3:

Placing an order
Triggers fired: set_bills_price, reduce_quantity, generate_total
Transaction(through stored procedure): Order_user
Participating relations: Orders, bills, cart, store_inv



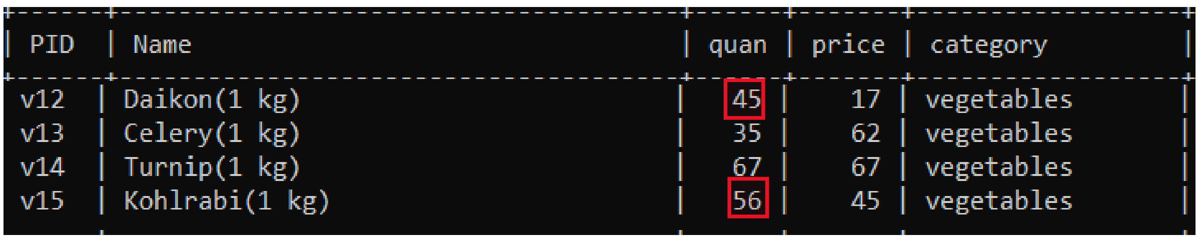
```
mysql> select * from orders;
+-----+------+------+------+---------------+------------------+-------+
| OID | UID  | DID  | stat | Date_of_order | Date_of_delivery | total |
+-----+------+------+------+---------------+------------------+-------+
|   1 |    1 | NULL | NULL | NULL          | NULL             |  NULL |
+-----+------+------+------+---------------+------------------+-------+
```

Fig 3.1



```
mysql> select * from bills;
Empty set (0.01 sec)
```

Fig 3.2



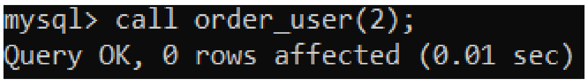
```
+------+-------------------------------------------+------+-------+------------------+
| PID  | Name                                      | quan | price | category         |
+------+-------------------------------------------+------+-------+------------------+
 v12   | Daikon(1 kg)                              |  45  |    17 | vegetables       |
 v13   | Celery(1 kg)                              |  35  |    62 | vegetables       |
 v14   | Turnip(1 kg)                              |  67  |    67 | vegetables       |
 v15   | Kohlrabi(1 kg)                            |  56  |    45 | vegetables       |
```

Fig 3.3

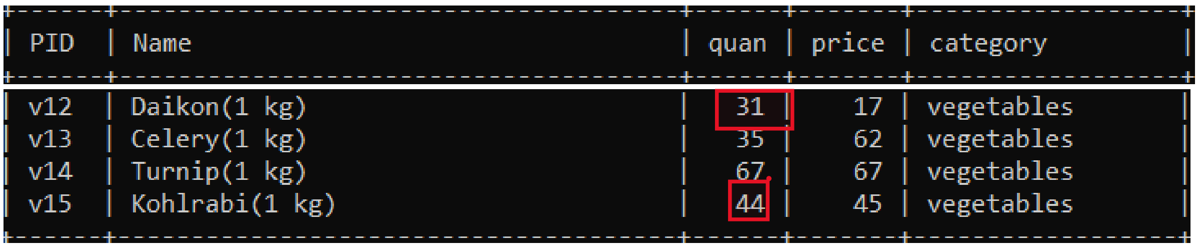
Figures 3.1, 3.2 and 3.3 show respectively the orders, bills, and store_inv relations at a given time. The store_inv relation has been cut short in the image to fit into the page and is only displaying the required details. The present state of the cart relation is shown in Figure 2.1.
Here, we place an order for the user with UID = 2.
This is achieved by calling the stored procedure order_user.
We pass in a parameter as UID = 2. So we use "CALL order_user(2).
The changes that are to occur are:

1. Reduction of quantity of corresponding items in the store_inv. (Fig 3.4)
2. Creation of a new entry in the orders table. (Fig 3.5)
3. Moving contents of cart to bills. (Fig 3.6)
4. Deletion of corresponding cart contents. (Fig 3.7)

Thus, we execute:

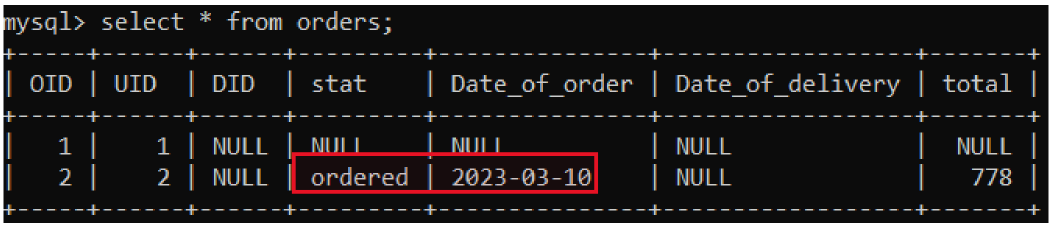


```
mysql> call order_user(2);
Query OK, 0 rows affected (0.01 sec)
```



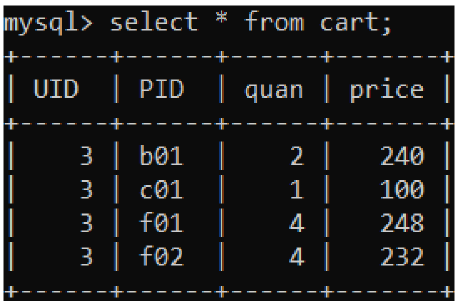
```
+------+---------------------------------------+------+-------+--------------------+
| PID  | Name                                  | quan | price | category           |
+------+---------------------------------------+------+-------+--------------------+
| v12  | Daikon(1 kg)                          |  31  |    17 | vegetables         |
| v13  | Celery(1 kg)                          |  35  |    62 | vegetables         |
| v14  | Turnip(1 kg)                          |  67  |    67 | vegetables         |
| v15  | Kohlrabi(1 kg)                        |  44  |    45 | vegetables         |
+------+---------------------------------------+------+-------+--------------------+
```

Fig 3.4



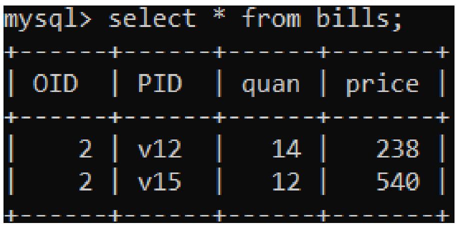
```
mysql> select * from orders;
+-----+-----+------+---------+---------------+------------------+-------+
| OID | UID | DID  | stat    | Date_of_order | Date_of_delivery | total |
+-----+-----+------+---------+---------------+------------------+-------+
|   1 |   1 | NULL | NULL    | NULL          | NULL             |  NULL |
|   2 |   2 | NULL | ordered | 2023-03-10    | NULL             |   778 |
+-----+-----+------+---------+---------------+------------------+-------+
```

Fig 3.5



```
mysql> select * from cart;
+------+------+------+-------+
| UID  | PID  | quan | price |
+------+------+------+-------+
|    3 | b01  |    2 |   240 |
|    3 | c01  |    1 |   100 |
|    3 | f01  |    4 |   248 |
|    3 | f02  |    4 |   232 |
+------+------+------+-------+
```

Fig 3.6



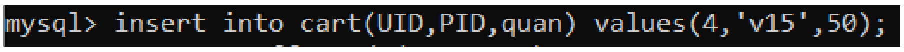
```
mysql> select * from bills;
+------+------+------+-------+
| OID  | PID  | quan | price |
+------+------+------+-------+
|    2 | v12  |   14 |   238 |
|    2 | v15  |   12 |   540 |
+------+------+------+-------+
```

Fig 3.7

As we can see, a new order has been created. The items of user 2 have been moved from the cart to bills corresponding to this new order. The corresponding quantity changes have also occurred in the store_inv. The total amount has been generated for the order and the price for each of the items has been updated in bills. Now we will see a case where the transaction fails.

Output 4:
Attempting to order more items than in inventory

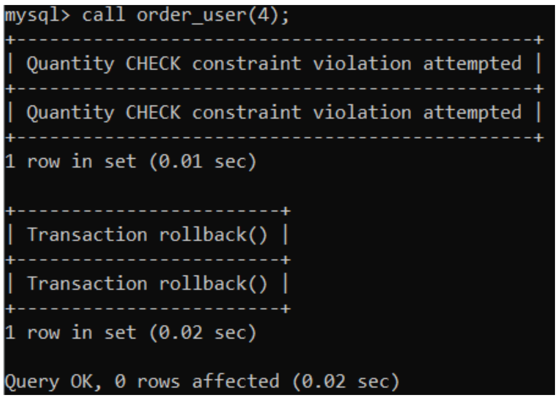
We will attempt to order 50 units of PID ‘v15’. The store_inv is currently in the state given by figure 3.4. Clearly we are ordering more than available.



```
mysql> insert into cart(UID,PID,quan) values(4,'v15',50);
```

The cart will then be given by Figure 4.1 as shown. To place this order, we call order_user(4). The output is shown in Figure 4.2 The transaction fails because one of the
Operations in it violated a check constraint because of a trigger that fired on updating the store_inv table.

Further there is no addition of junk rows to the order table and the database is restored to the state it was in before the query was called. Thus the orders table still looks like the image in figure 3.5

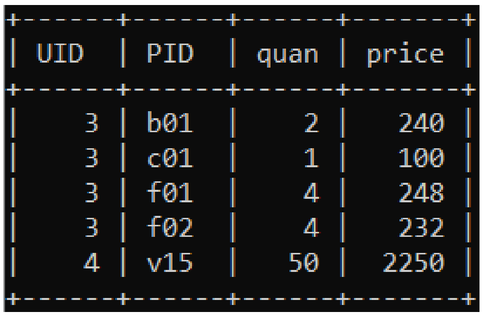


```
mysql> call order_user(4);
+-------------------------------------------+
| Quantity CHECK constraint violation attempted |
+-------------------------------------------+
| Quantity CHECK constraint violation attempted |
+-------------------------------------------+
1 row in set (0.01 sec)

+------------------------+
| Transaction rollback() |
+------------------------+
| Transaction rollback() |
+------------------------+
1 row in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)
```

Fig 4.1

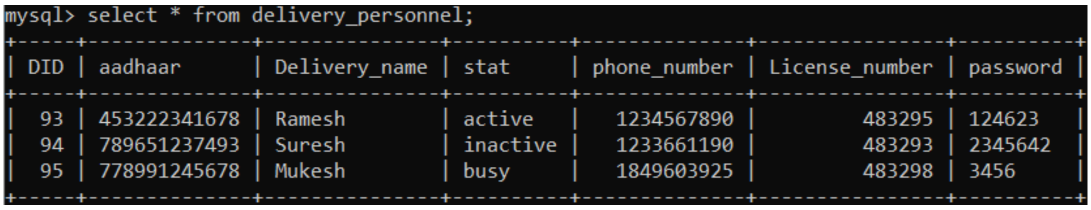


```
+------+------+------+-------+
| UID  | PID  | quan | price |
+------+------+------+-------+
|    3 | b01  |    2 |   240 |
|    3 | c01  |    1 |   100 |
|    3 | f01  |    4 |   248 |
|    3 | f02  |    4 |   232 |
|    4 | v15  |   50 |  2250 |
+------+------+------+-------+
```

Fig 4.2

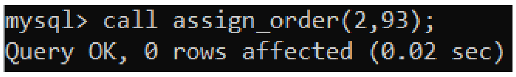
Output 5:
Assigning an order to a delivery personnel.
Stored procedure used: Assign_order

To do so we call the stored procedure assign_order. We pass in the OID and DID. In the orders table, this changes the DID to the value we pass, changes the status to delivery confirmed and in the delivery personnel table, it changes the status to busy. Like the previous case, to avoid junk values in the tables, this has been implemented by transactions through stored procedures. Initially the orders table is in state given by figure 3.5. And the Delivery Personnel table is in a state given by figure 5.1 as below. We will assign the delivery to the only active and non-busy driver : DID - 93. We execute CALL assign_order(2,93).



```
mysql> select * from delivery_personnel;
+-----+--------------+---------------+----------+--------------+----------------+----------+
| DID | aadhaar      | Delivery_name | stat     | phone_number | License_number | password |
+-----+--------------+---------------+----------+--------------+----------------+----------+
|  93 | 453222341678 | Ramesh        | active   |   1234567890 |         483295 | 124623   |
|  94 | 789651237493 | Suresh        | inactive |   1233661190 |         483293 | 2345642  |
|  95 | 778991245678 | Mukesh        | busy     |   1849603925 |         483298 | 3456     |
+-----+--------------+---------------+----------+--------------+----------------+----------+
```

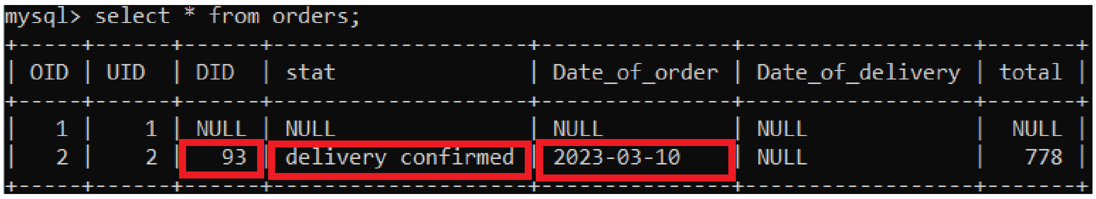
Fig 5.1



```
mysql> call assign_order(2,93);
Query OK, 0 rows affected (0.02 sec)
```

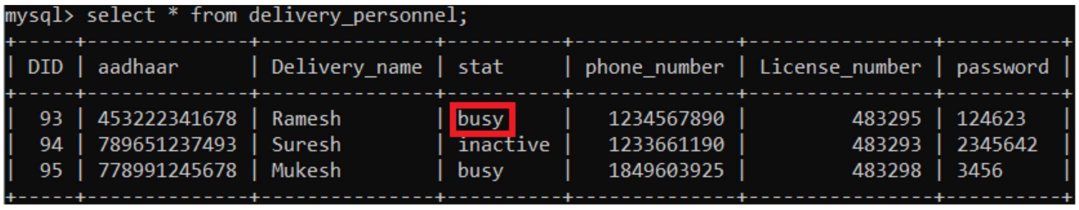
Fig 5.2

The new state of the orders and delivery personnel tables are shown in figures 5.3 and 5.4. The changes have been represented using red rectangles.



```
mysql> select * from orders;
+-----+-----+------+--------------------+---------------+------------------+-------+
| OID | UID | DID  | stat               | Date_of_order | Date_of_delivery | total |
+-----+-----+------+--------------------+---------------+------------------+-------+
|   1 |   1 | NULL | NULL               | NULL          | NULL             |  NULL |
|   2 |   2 |   93 | delivery confirmed | 2023-03-10    | NULL             |   778 |
+-----+-----+------+--------------------+---------------+------------------+-------+
```

Fig 5.3 Orders Table

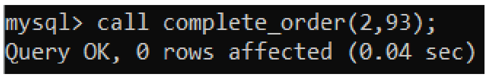


```
mysql> select * from delivery_personnel;
+-----+--------------+---------------+----------+--------------+----------------+----------+
| DID | aadhaar      | Delivery_name | stat     | phone_number | License_number | password |
+-----+--------------+---------------+----------+--------------+----------------+----------+
|  93 | 453222341678 | Ramesh        | busy     |   1234567890 |         483295 | 124623   |
|  94 | 789651237493 | Suresh        | inactive |   1233661190 |         483293 | 2345642  |
|  95 | 778991245678 | Mukesh        | busy     |   1849603925 |         483298 | 3456     |
+-----+--------------+---------------+----------+--------------+----------------+----------+
```

Fig 5.4 Delivery personnel table

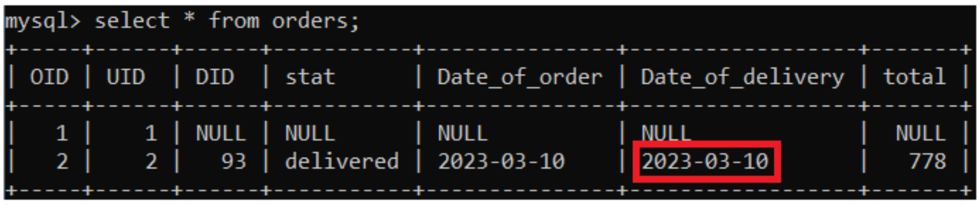
Output 6:
Completing an order
Stored procedure used: complete_order

This is the last step of a given order. It changes back the status of the Delivery Personnel to active and fills in the date of delivery corresponding to the order in the orders table. It also changes the state of the order to ‘delivered’. Initially the above mentioned tables are in state as in Figure 5.3 and 5.4. After completion they are in 6.2 and 6.3. Like the previous case, the transaction is necessary to ensure that either both the updates are done or none are.



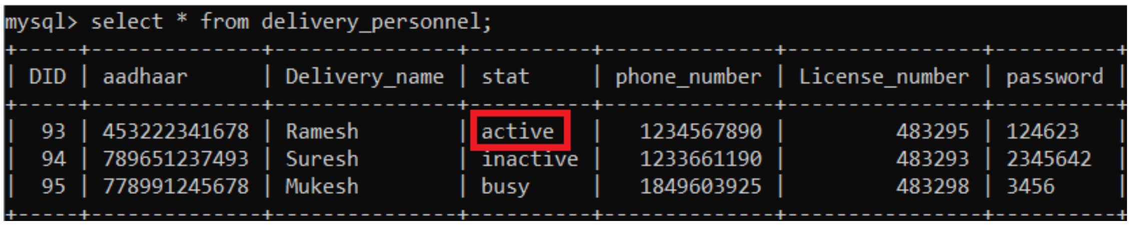
```
mysql> call complete_order(2,93);
Query OK, 0 rows affected (0.04 sec)
```

Fig 6.1



```
mysql> select * from orders;
+-----+-----+-----+-----------+---------------+------------------+-------+
| OID | UID | DID | stat      | Date_of_order | Date_of_delivery | total |
+-----+-----+-----+-----------+---------------+------------------+-------+
|   1 |   1 | NULL| NULL      | NULL          | NULL             |  NULL |
|   2 |   2 |  93 | delivered | 2023-03-10    | 2023-03-10       |   778 |
+-----+-----+-----+-----------+---------------+------------------+-------+
```

Fig 6.2 Orders table

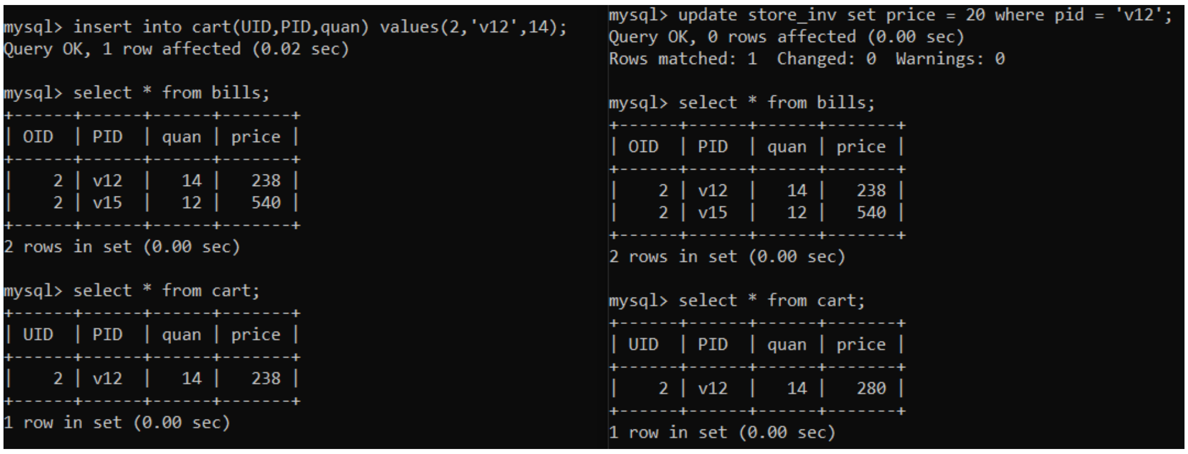


```
mysql> select * from delivery_personnel;
+-----+--------------+---------------+----------+--------------+----------------+----------+
| DID | aadhaar      | Delivery_name | stat     | phone_number | License_number | password |
+-----+--------------+---------------+----------+--------------+----------------+----------+
|  93 | 453222341678 | Ramesh        | active   |   1234567890 |         483295 | 124623   |
|  94 | 789651237493 | Suresh        | inactive |   1233661190 |         483293 | 2345642  |
|  95 | 778991245678 | Mukesh        | busy     |   1849603925 |         483298 | 3456     |
+-----+--------------+---------------+----------+--------------+----------------+----------+
```

Fig 6.3 Delivery Personnel Table

<u>Output 7:</u>
Updating the price of an item in store_inv.
Trigger fired: update_cart_price_for_inventory_update

Whenever the price of an item changes, the change must reflect in the shopping carts of the user. At the same time, the previous orders and bills must remain unaffected. Since the order and bills are only updated by the order_user transaction - stored procedure, our second problem is already solved. We use the trigger to solve the first problem.



```
mysql> insert into cart(UID,PID,quan) values(2,'v12',14);
Query OK, 1 row affected (0.02 sec)

mysql> select * from bills;
+------+-----+------+-------+
| OID  | PID | quan | price |
+------+-----+------+-------+
|    2 | v12 |   14 |   238 |
|    2 | v15 |   12 |   540 |
+------+-----+------+-------+
2 rows in set (0.00 sec)

mysql> select * from cart;
+------+-----+------+-------+
| UID  | PID | quan | price |
+------+-----+------+-------+
|    2 | v12 |   14 |   238 |
+------+-----+------+-------+
1 row in set (0.00 sec)
```

```
mysql> update store_inv set price = 20 where pid = 'v12';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> select * from bills;
+------+-----+------+-------+
| OID  | PID | quan | price |
+------+-----+------+-------+
|    2 | v12 |   14 |   238 |
|    2 | v15 |   12 |   540 |
+------+-----+------+-------+
2 rows in set (0.00 sec)

mysql> select * from cart;
+------+-----+------+-------+
| UID  | PID | quan | price |
+------+-----+------+-------+
|    2 | v12 |   14 |   280 |
+------+-----+------+-------+
1 row in set (0.00 sec)
```

We see that the prices have been unaltered in the bills table, but have reflected the change in the cart table.

CHAPTER-3

Tools Used

1. HTML

- **<html>**: The root element that defines an HTML document.
- **<head>**: The element that contains metadata about the document.
- **<title>**: The element that defines the title of the document.
- **<body>**: The element that contains the visible content of the document.
- **<div>**: The element that creates a container to group other elements.
- **<p>**: The element that defines a paragraph of text.
- **<img>**: The element that inserts an image into the document.
- **<a>**: The element that creates a hyperlink to another web page or resource.
- **<ul>**: The element that creates an unordered list.
- **<li>**: The element that creates a list item in an ordered or unordered list.
- **<table>**: The element that creates a table to organize data.
- **<tr>**: The element that creates a row in a table.
- **<td>**: The element that creates a cell in a table.
- **<form>**: The element that creates a form to collect user input.
- **<input>**: The element that creates an input field for a form.
- **<select>**: The element that creates a drop-down list for a form.
- **<button>**: The element that creates a clickable button.
- **<textarea**>: The element that creates a multi-line input field for a form.
- **<label>**: The element that creates a label for a form element.
- **<br>**: The element that creates a line break in the document.
- **<summary>**: The element that defines a visible heading for the content in a **<details>** element, which can be clicked to toggle the display of the content.
- **<details>**: The element that creates a disclosure widget containing additional information that can be toggled on or off by clicking on the summary element.

2. CSS

- **color**: property used to define the color of text or elements.
- **font-size:** property used to define the size of text.
- **margin and padding:** properties used to create space around elements.
- **display**: property used to control the layout of elements.
- **float and clear:** properties used to control the placement of elements in relation to other elements.
- **background:** property used to define the background color or image of an element.
- **border:** property used to add borders to elements.
- **position:** property used to position elements on the page.
- **text-align:** property used to control the alignment of text.

- **transition:** property used to create smooth transitions between CSS property values.
- **box-sizing**: property used to control how the total width and height of an element is calculated.
- **line-height:** property used to control the spacing between lines of text.
- **text-decoration:** property used to add or remove underlines, overlines, and strikethroughs from text.
- **text-transform:** property used to control the capitalization of text.
- **box-shadow:** property used to add shadows to elements.
- transform and transition: properties used to create advanced animations and effects.
- **flex and grid:** properties used to create flexible and responsive layouts.
- **overflow**: property used to control what happens when an element's content is too large to fit inside its container.
- **z-index:** property used to control the stacking order of elements.
- **cursor:** property used to change the appearance of the mouse pointer when hovering over an element.
- **opacity**: property used to control the transparency of an element.
- **list-style**: property used to add style to list elements.
- **background-position**: property used to control the position of a background image within an element.
- **border-radius**: property used to create rounded corners on elements.
- **text-shadow**: property used to add shadows to text.
- **text-overflow**: property used to control what happens when text overflows its container.
- **word-wrap**: property used to control how long words are broken in text.
- **outline:** property used to add an outline around an element.
- **user-select**: property used to control if text can be selected by the user.
- **filter:** property used to apply visual effects to elements, such as blurring or changing the color

3. PHP

- **mysqli_connect**: A function used to establish a connection to a MySQL database using PHP.
- **die**: A function used to stop the execution of a script and print an error message.
- **mysqli_connect_error**: A function used to return the last error message from the MySQL server.
- **mysqli_query**: A function used to execute a SQL query on a MySQL database.
- **mysqli_num_rows**: A function used to get the number of rows returned by a MySQL query.
- **mysqli_fetch_assoc**: A function used to fetch a single row from a MySQL result set as an associative array.
- **header**:A function is used in PHP to send HTTP headers to a client or a server, including content-type, location, cache-control, and more.

- **hash_password:**Hashing a password involves converting the password into a fixed-length string of characters using a one-way hash function.
- **Verify_password:** verifying a password involves comparing the hashed version of the user's input to the stored hashed password to determine if it is correct.

4. Javascript

- **querySelector:** to select elements from the DOM
- **createElement:** to create a new HTML element
- **appendChild**: to add the close button element to the modal content
- **addEventListener:** to handle user events such as clicks and key presses
- **preventDefault:** to prevent the default behavior of an element when it is clicked
- **style:** to modify the CSS style of an element
- **key:** to check the key that was pressed on the keyboard

5. Tools used by Ajax Script:

In our Website we use ajax to create a new XMLHttpRequest object and set the onreadystatechange function to handle the response from the server.

When the ready state changes to XMLHttpRequest.DONE and the status is 200 (OK), it gets the new updated element for which ID is given from the DOM and sets its inner HTML to the response text. If the status is not 200, it logs an error message to the console.

Then it opens a new POST request to the server using the action URL, sets the Content-Type header to application/x-www-form-urlencoded, and sends the required parameter in the request body.

- **XMLHttpRequest**: This is a built-in JavaScript object used to send HTTP or HTTPS requests to a server and receive responses.
- **function**: This is a keyword used to define a function in JavaScript.
- **if statement**: This is a control statement used to execute a block of code if a certain condition is true.
- **document.getElementById**: This is a built-in method in JavaScript used to retrieve a reference to an HTML element based on its ID.
- **innerHTML**: This is a property in JavaScript used to get or set the HTML content of an element.
- **console.error**: This is a method in JavaScript used to display an error message in the browser console.
- **open**: This is a method in XMLHttpRequest used to set the HTTP method, URL, and other optional parameters for the request.

- **setRequestHeader**: This is a method in XMLHttpRequest used to set the value of an HTTP request header.
- **send**: This is a method in XMLHttpRequest used to send the HTTP request to the server.
- **application/x-www-form-urlencoded**: This is a MIME type used to encode form data in the HTTP request.

6. Bootstrap

- **Bootstrap buttons** provide pre-defined styling for clickable elements, including colors, sizes, and shapes.
- **Bootstrap button groups** allow multiple buttons to be grouped together and styled as a unit.
- **Bootstrap grid containers** create a flexible grid layout with responsive breakpoints for arranging content in rows and columns.
- **Tables** with Bootstrap classes include predefined styles for tables, such as striped rows, hover effects, and responsive design.

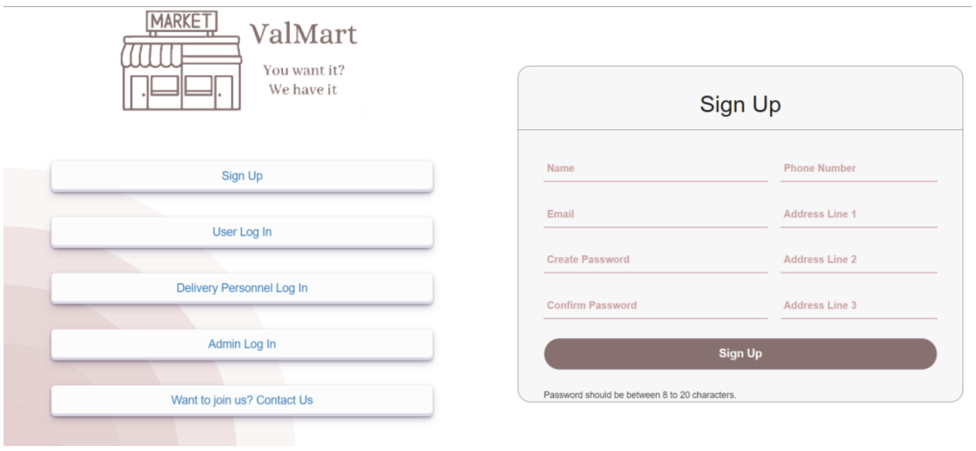
7. Google Fonts & icons

- **Google Fonts** is a library of free, open-source fonts that can be easily integrated into HTML documents, allowing website designers to use a wide range of typefaces without worrying about licensing issues or downloading and hosting the font files themselves.
- **Google Icons** is a library of customizable vector icons that can be easily integrated into HTML documents, allowing website designers to add visually appealing and functional icons to their designs without the need for specialized design software or graphic design skills.

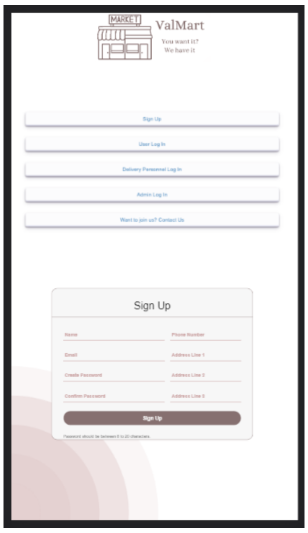
CHAPTER-4

Module wise Screenshots

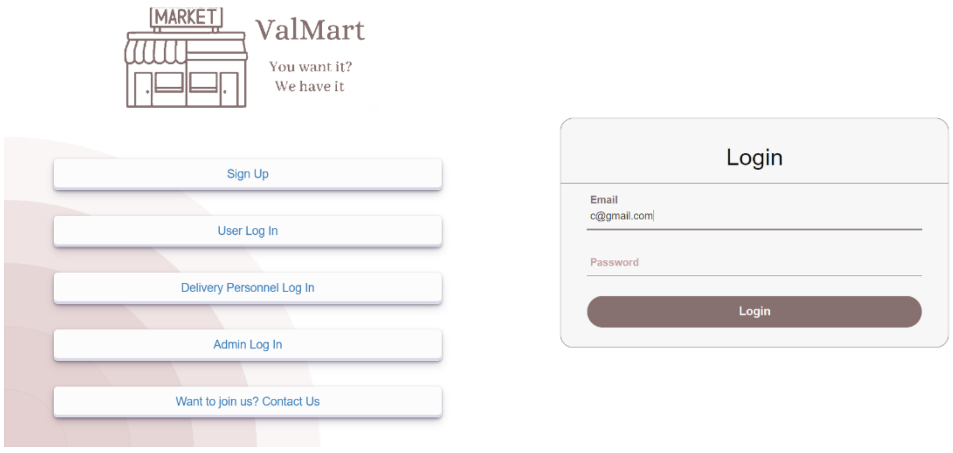
1. SIGNUP.PHP



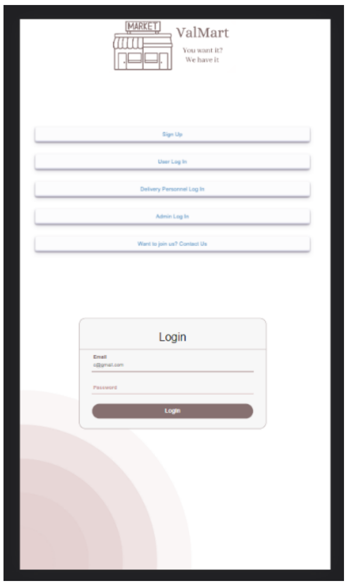
Password should be between 8 to 20 characters.



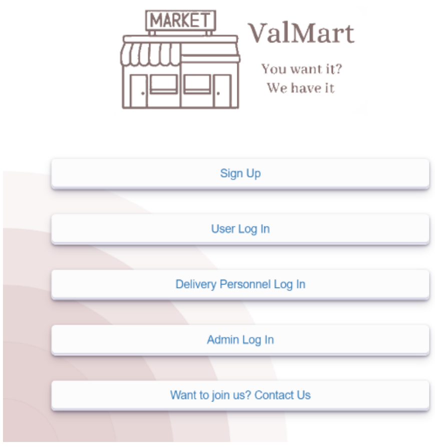
2. LOGIN.PHP

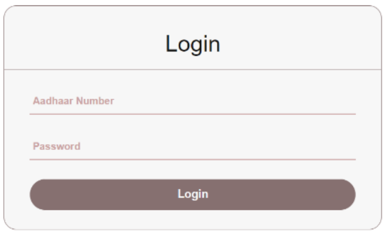


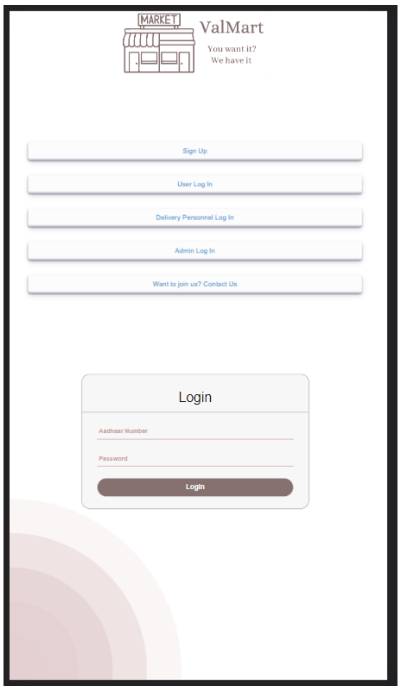
Email
c@gmail.com



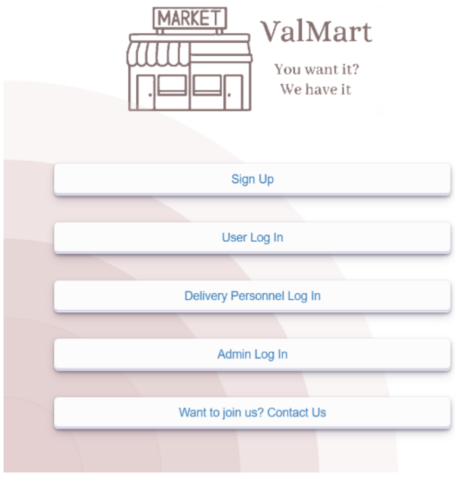
3. LOGIN_DELIVERY.PHP

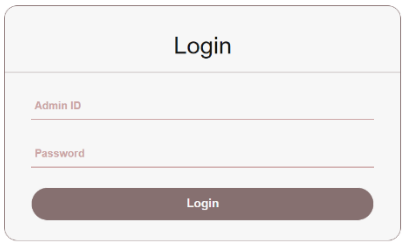


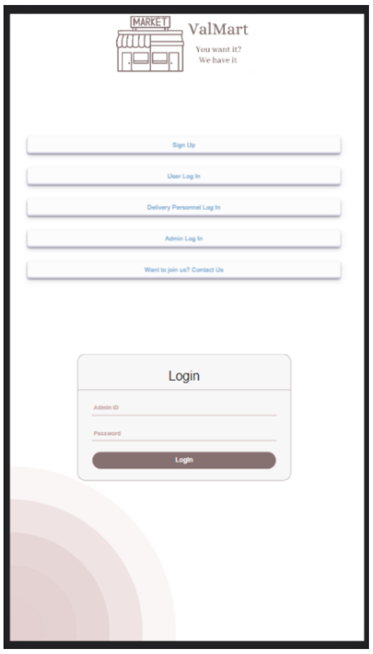




4. ADMIN_LOGIN.PHP



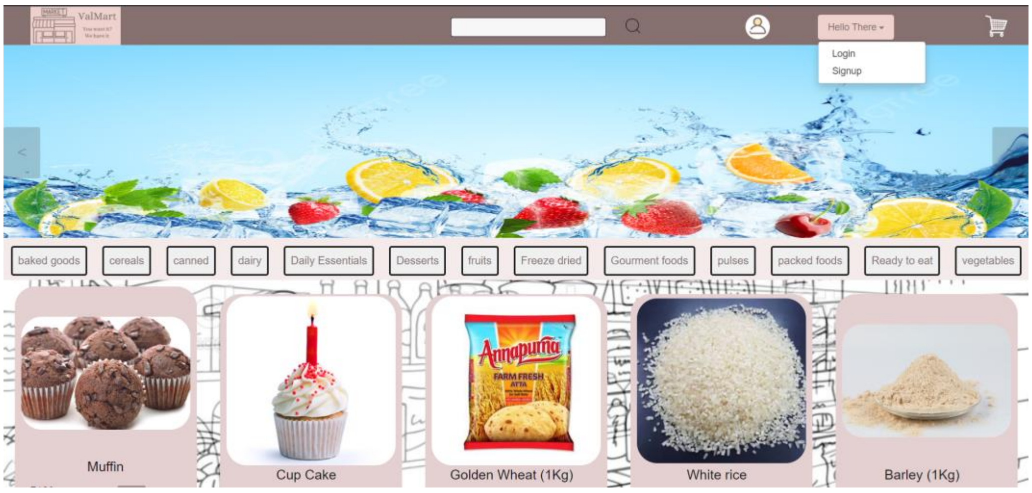




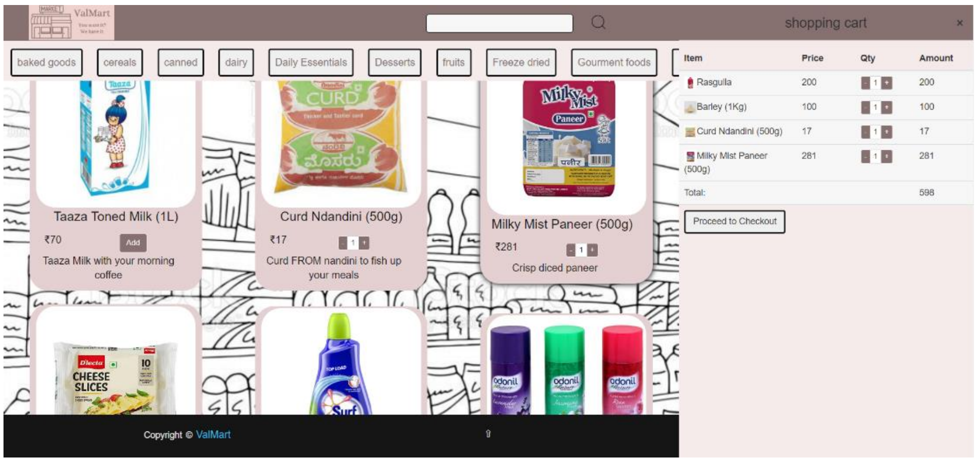
.

5. HOME.PHP

Before logging in



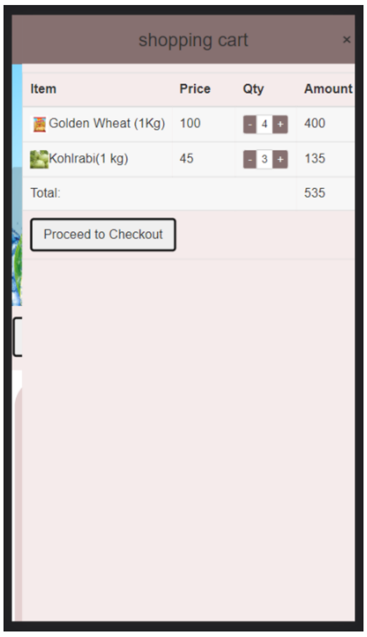
After logging in



shopping cart

Item	Price	Qty	Amount
Rasgulla	200	1	200
Barley (1Kg)	100	1	100
Curd Ndandini (500g)	17	1	17
Milky Mist Paneer (500g)	281	1	281
Total:			598

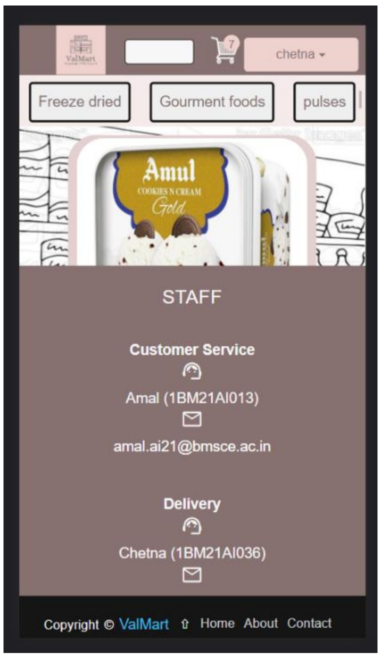
Proceed to Checkout



shopping cart

Item	Price	Qty	Amount
Golden Wheat (1Kg)	100	4	400
Kohlrabi(1 kg)	45	3	135
Total:			535

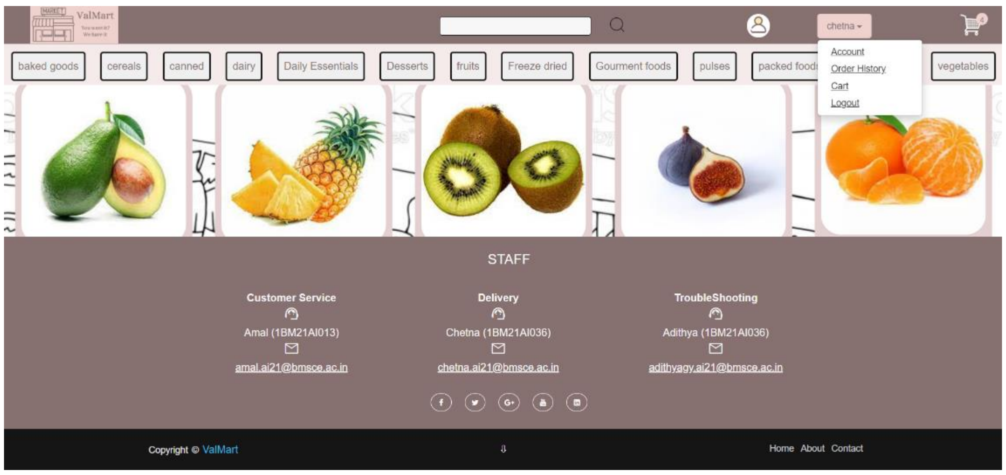
Proceed to Checkout



STAFF

Customer Service
Amal (1BM21AI013)
amal.ai21@bmsce.ac.in

Delivery
Chetna (1BM21AI036)



STAFF

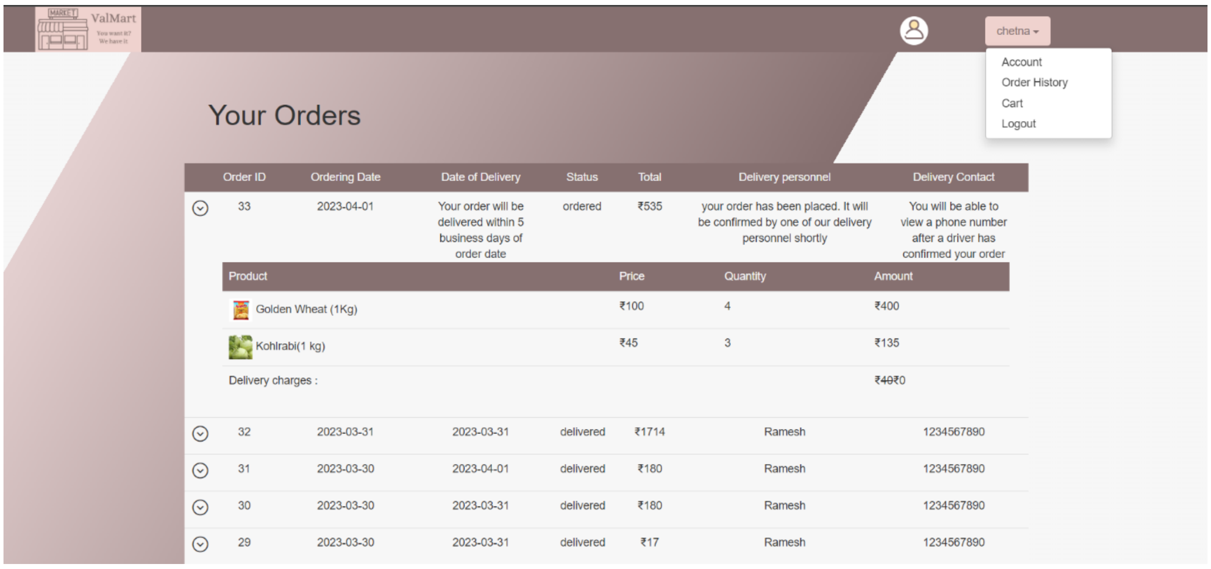
Customer Service
Amal (1BM21AI013)
amal.ai21@bmsce.ac.in

Delivery
Chetna (1BM21AI036)
chetna.ai21@bmsce.ac.in

TroubleShooting
Adithya (1BM21AI036)
adithyagy.ai21@bmsce.ac.in

6. ORDERS.PHP

The drop down menu is there in every page in navigation bar



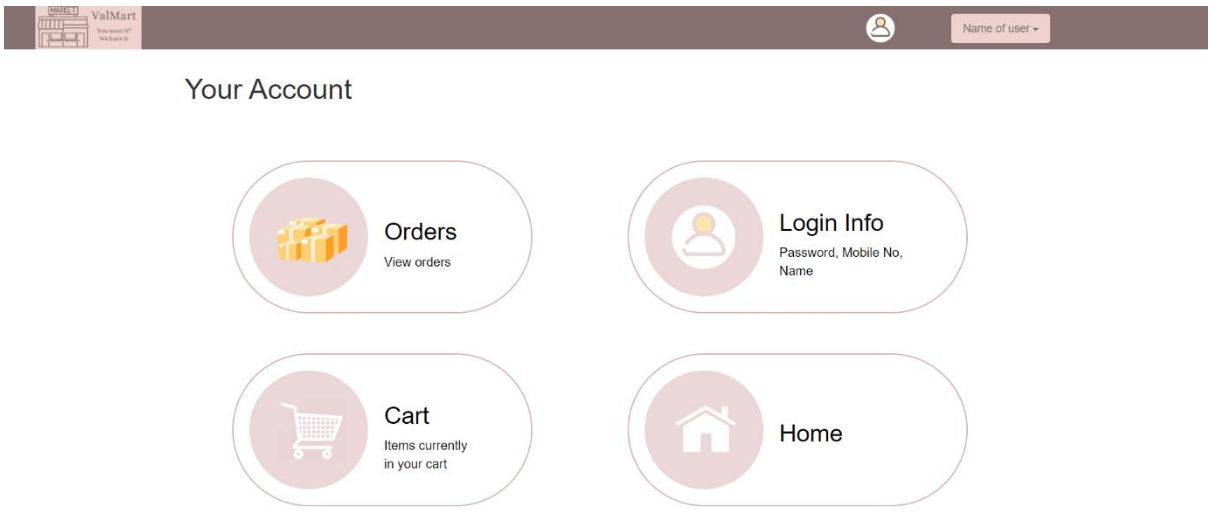
Your Orders

	Order ID	Ordering Date	Date of Delivery	Status	Total	Delivery personnel	Delivery Contact
	33	2023-04-01	Your order will be delivered within 5 business days of order date	ordered	₹535	your order has been placed. It will be confirmed by one of our delivery personnel shortly	You will be able to view a phone number after a driver has confirmed your order

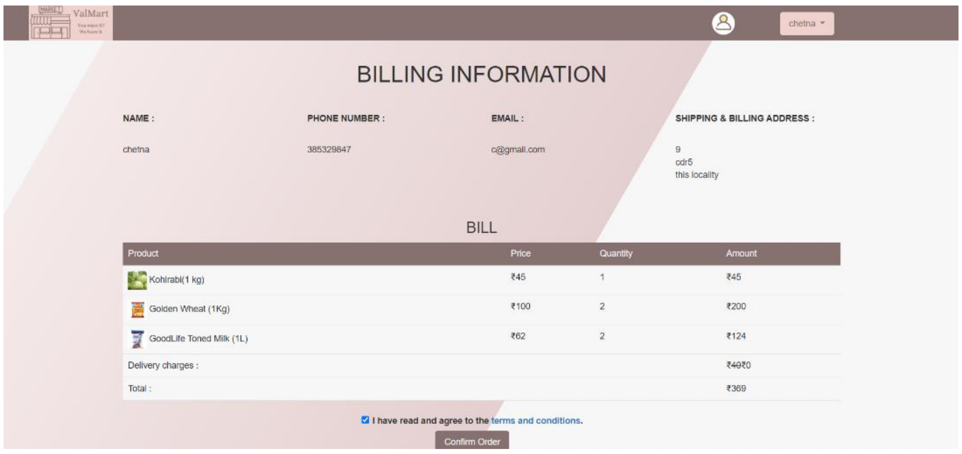
Product	Price	Quantity	Amount
Golden Wheat (1Kg)	₹100	4	₹400
Kohlrabi(1 kg)	₹45	3	₹135
Delivery charges :			~~₹40~~₹0

	Order ID	Ordering Date	Date of Delivery	Status	Total	Delivery personnel	Delivery Contact
	32	2023-03-31	2023-03-31	delivered	₹1714	Ramesh	1234567890
	31	2023-03-30	2023-04-01	delivered	₹180	Ramesh	1234567890
	30	2023-03-30	2023-03-31	delivered	₹180	Ramesh	1234567890
	29	2023-03-30	2023-03-31	delivered	₹17	Ramesh	1234567890

7. PROFILE_PAGE.PHP



8. CART.PHP



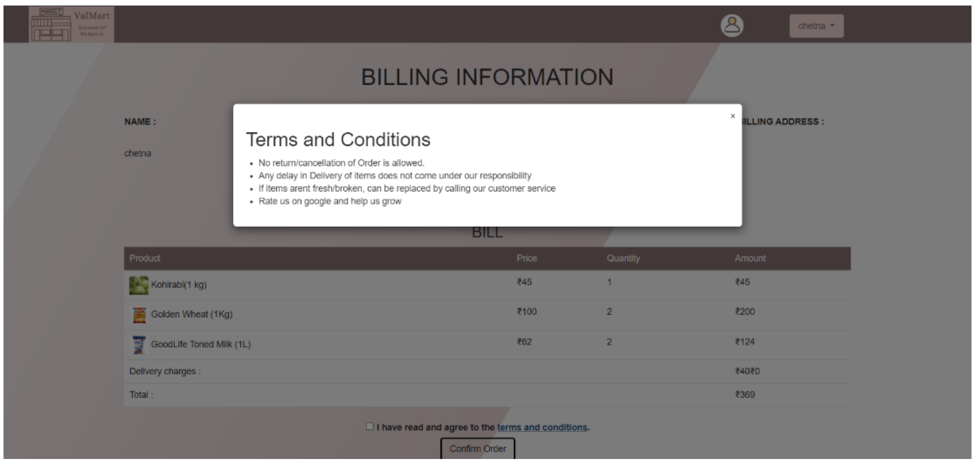
BILLING INFORMATION

NAME :	PHONE NUMBER :	EMAIL :	SHIPPING & BILLING ADDRESS :
chetna	385329847	c@gmail.com	9 cdr5 this locality

BILL

Product	Price	Quantity	Amount
Kohlrabi(1 kg)	₹45	1	₹45
Golden Wheat (1Kg)	₹100	2	₹200
GoodLife Toned Milk (1L)	₹62	2	₹124
Delivery charges :			~~₹40~~₹0
Total :			₹369

☑ I have read and agree to the terms and conditions.



BILLING INFORMATION

NAME :

chetna

Terms and Conditions

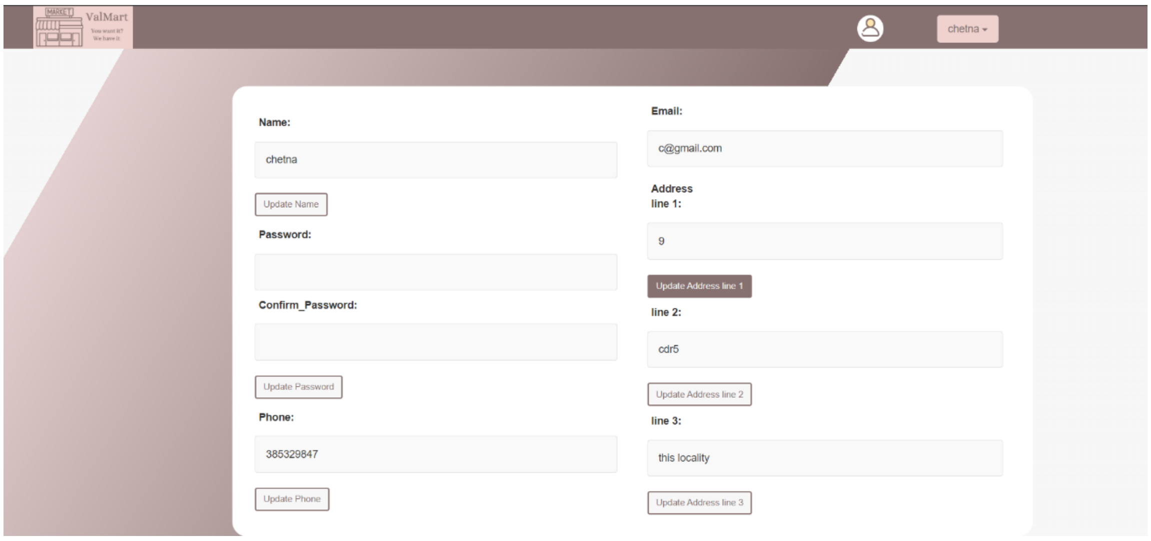
- No return/cancellation of Order is allowed.
- Any delay in Delivery of items does not come under our responsibility
- If items arent fresh/broken, can be replaced by calling our customer service
- Rate us on google and help us grow

BILL

Product	Price	Quantity	Amount
Kohlrabi(1 kg)	₹45	1	₹45
Golden Wheat (1Kg)	₹100	2	₹200
GoodLife Toned Milk (1L)	₹62	2	₹124
Delivery charges :			~~₹40~~₹0
Total :			₹369

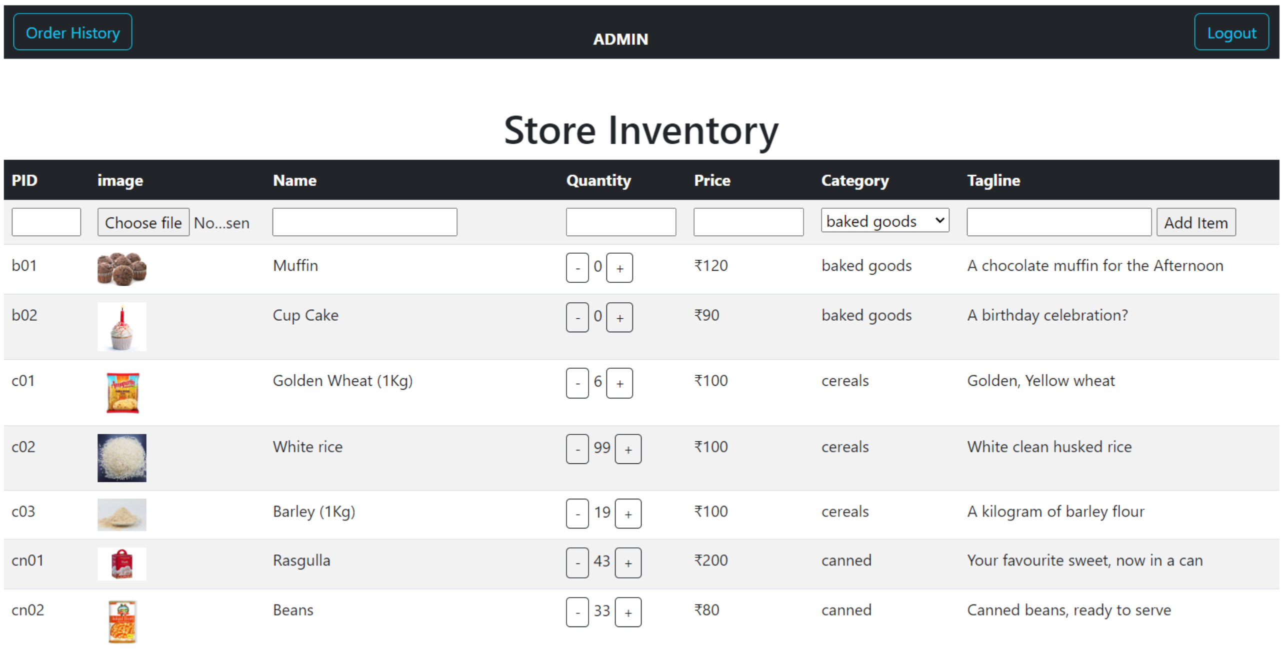
☐ I have read and agree to the terms and conditions.

9. USERSETTINGS.PHP



Name: chetna	Email: c@gmail.com
Password:	Address line 1: 9
Confirm_Password:	line 2: cdr5
Phone: 385329847	line 3: this locality

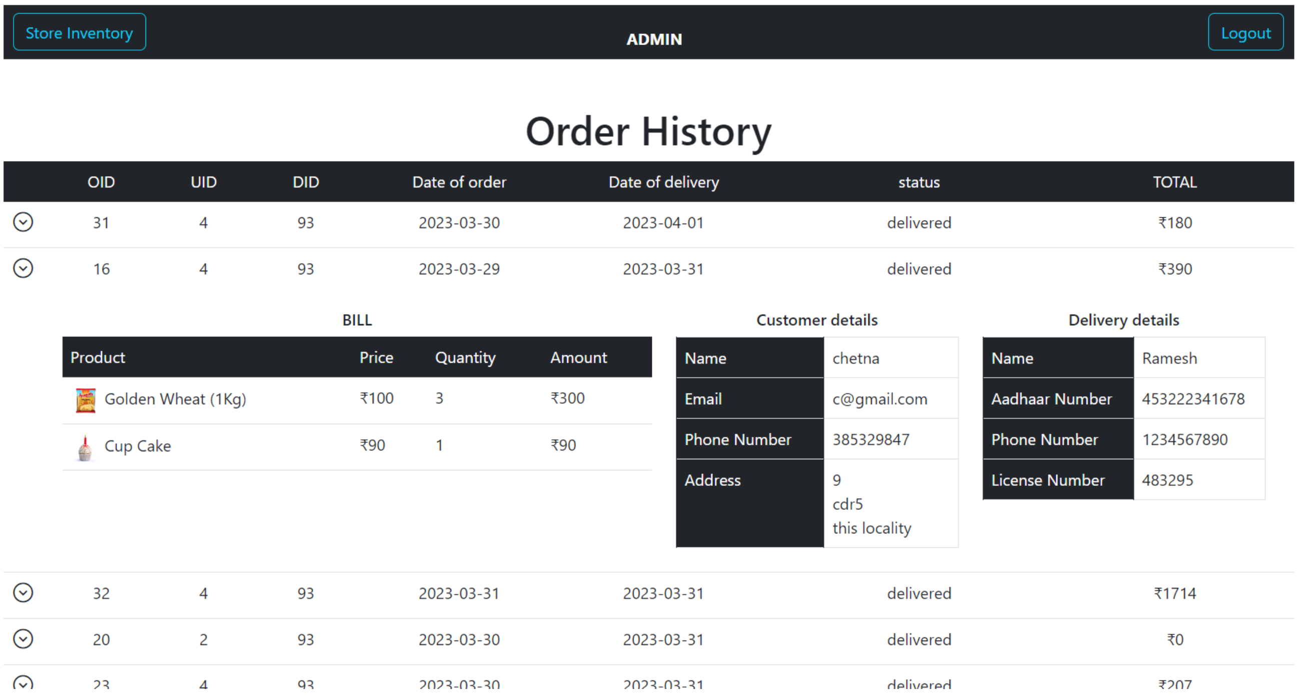
10. ADMIN.PHP



Store Inventory

PID	image	Name	Quantity	Price	Category	Tagline
b01		Muffin	0	₹120	baked goods	A chocolate muffin for the Afternoon
b02		Cup Cake	0	₹90	baked goods	A birthday celebration?
c01		Golden Wheat (1Kg)	6	₹100	cereals	Golden, Yellow wheat
c02		White rice	99	₹100	cereals	White clean husked rice
c03		Barley (1Kg)	19	₹100	cereals	A kilogram of barley flour
cn01		Rasgulla	43	₹200	canned	Your favourite sweet, now in a can
cn02		Beans	33	₹80	canned	Canned beans, ready to serve

11. STORE_ORDERS.PHP



Order History

OID	UID	DID	Date of order	Date of delivery	status	TOTAL
31	4	93	2023-03-30	2023-04-01	delivered	₹180
16	4	93	2023-03-29	2023-03-31	delivered	₹390

BILL

Product	Price	Quantity	Amount
Golden Wheat (1Kg)	₹100	3	₹300
Cup Cake	₹90	1	₹90

Customer details

Name	chetna
Email	c@gmail.com
Phone Number	385329847
Address	9 cdr5 this locality

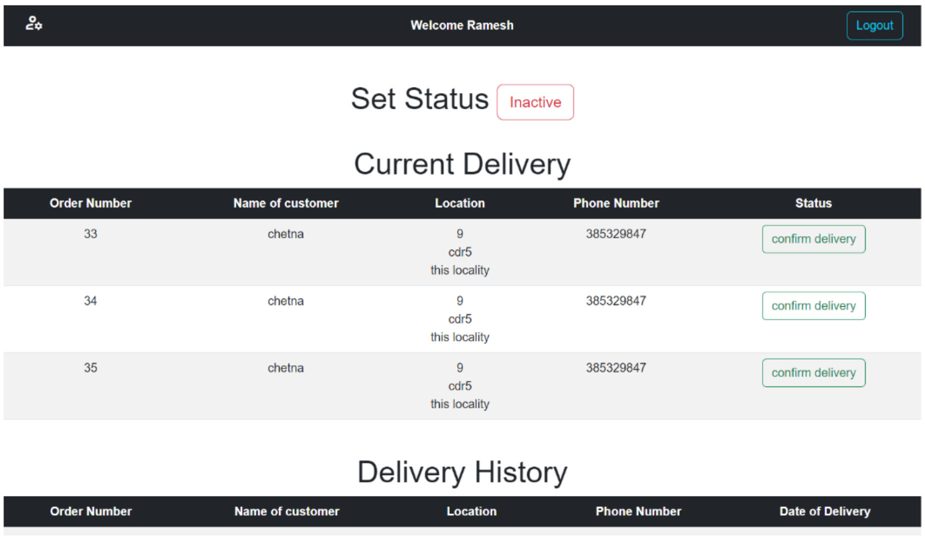
Delivery details

Name	Ramesh
Aadhaar Number	453222341678
Phone Number	1234567890
License Number	483295

OID	UID	DID	Date of order	Date of delivery	status	TOTAL
32	4	93	2023-03-31	2023-03-31	delivered	₹1714
20	2	93	2023-03-30	2023-03-31	delivered	₹0
23	4	93	2023-03-30	2023-03-31	delivered	₹207

12. DELIVERIES.PHP

When status is active



Welcome Ramesh — Logout

Set Status Inactive

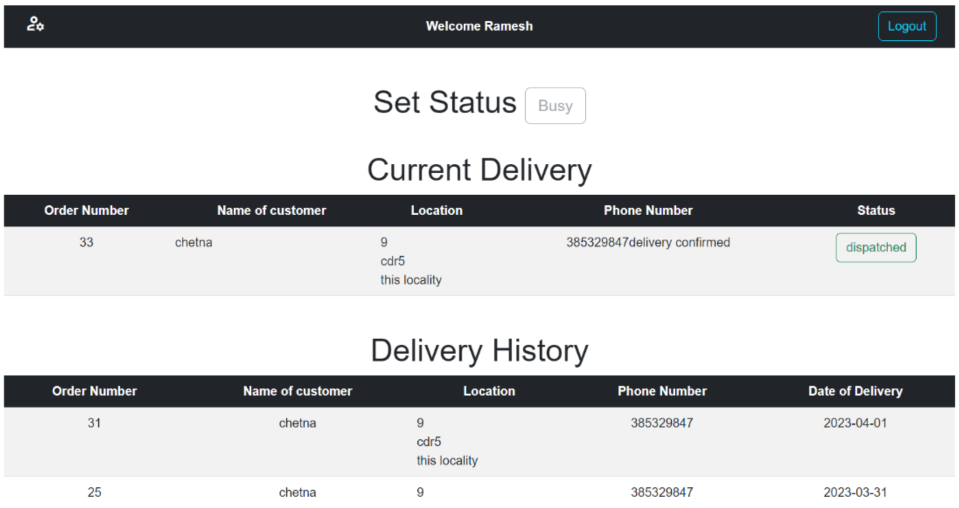
Current Delivery

Order Number	Name of customer	Location	Phone Number	Status
33	chetna	9 cdr5 this locality	385329847	confirm delivery
34	chetna	9 cdr5 this locality	385329847	confirm delivery
35	chetna	9 cdr5 this locality	385329847	confirm delivery

Delivery History

Order Number	Name of customer	Location	Phone Number	Date of Delivery

When status is busy



Welcome Ramesh — Logout

Set Status Busy

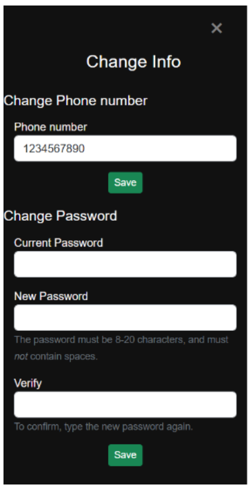
Current Delivery

Order Number	Name of customer	Location	Phone Number	Status
33	chetna	9 cdr5 this locality	385329847delivery confirmed	dispatched

Delivery History

Order Number	Name of customer	Location	Phone Number	Date of Delivery
31	chetna	9 cdr5 this locality	385329847	2023-04-01
25	chetna	9	385329847	2023-03-31

When status is inactive and also if settings bar is open



×

Change Info

Change Phone number

Phone number: 1234567890

Save

Change Password

Current Password:

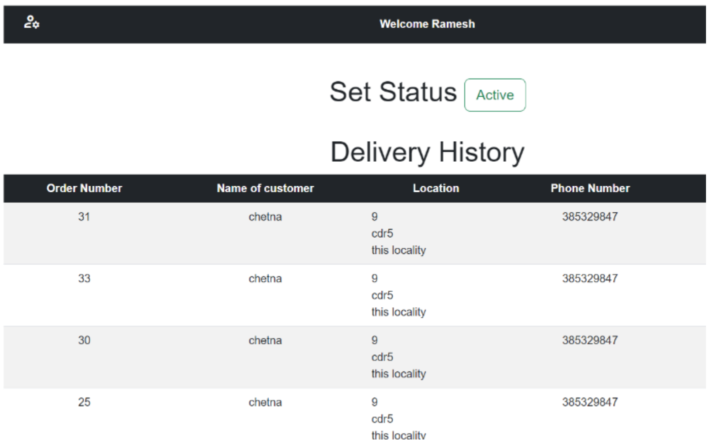
New Password:

The password must be 8-20 characters, and must not contain spaces.

Verify:

To confirm, type the new password again.

Save



Welcome Ramesh

Set Status Active

Delivery History

Order Number	Name of customer	Location	Phone Number
31	chetna	9 cdr5 this locality	385329847
33	chetna	9 cdr5 this locality	385329847
30	chetna	9 cdr5 this locality	385329847
25	chetna	9 cdr5 this locality	385329847

CHAPTER-5

CONCLUSION

This report illustrates the implementation of a website with effective usage of front end and back-end tools to deploy an ecommerce website. The use of PHP and XAMPP provides a robust server-side environment for the application, while HTML, CSS, and JS allow for the creation of visually appealing and highly functional user interfaces. Additionally, the use of AJAX allows for the creation of dynamic and responsive web pages, while SCSS provides a powerful tool for creating and maintaining a consistent and customizable visual style.

During the development of this project, we have learned basic structuring using html, responsive design using CSS and bootstrap, interactive website design using JavaScript, data retrieval and data manipulation operations on a database using SQL and PHP.
We learned ajax to make fast and dynamic pages along with other tools to provide the user with the best interface and experience up to our capabilities.

REFERENCES

- https://www.w3schools.com/js/js_ajax_http_send.asp
- https://www.w3schools.com/js/js_ajax_php.asp
- https://www.w3schools.com/bootstrap5/bootstrap_grid_basic.php
- https://www.w3schools.com/bootstrap5/bootstrap_tables.php
- https://www.w3schools.com/bootstrap5/bootstrap_buttons.php
- https://www.w3schools.com/php/php_ref_mysqli.asp