

MOBIFY

Online Mobile Shopping Website

Gavin Da Costa
Khizer Khan
Abhinav Naik

DATABASE DESIGN METHODOLOGY

DESCRIPTION :

MOBIFY is an online mobile shopping website. This system consists of customers, seller and an admin. Customer can view mobiles listed by the admin or the seller, add item to the cart, pay for order, and view their order history. Admin can edit the mobile information, manage customer orders, and view information about all orders, customers, sellers and can add or remove brands.

ENTITIES :

1. **Seller:**

- Description: Represents individuals or organizations that sell products.

2. **Customer:**

- Description: Represents individuals that purchase products.

3. **Product:**

- Description: Represents the mobiles that are sold.

4. **Brand:**

- Description: Represents the brand or manufacturer associated with a product.

5. **Order:**

- Description: Represents an order where a customer purchases one or more products from a seller.

6. **Payment:**

- Description: Represents the financial transaction associated with an order.

ENTITY RELATIONSHIPS

1. **Seller sells Product**

- Description: Describes the fact that a seller sells one or more products.

2. **Brand Manufactures Product**

- Description: Indicates the association between a product and its brand.

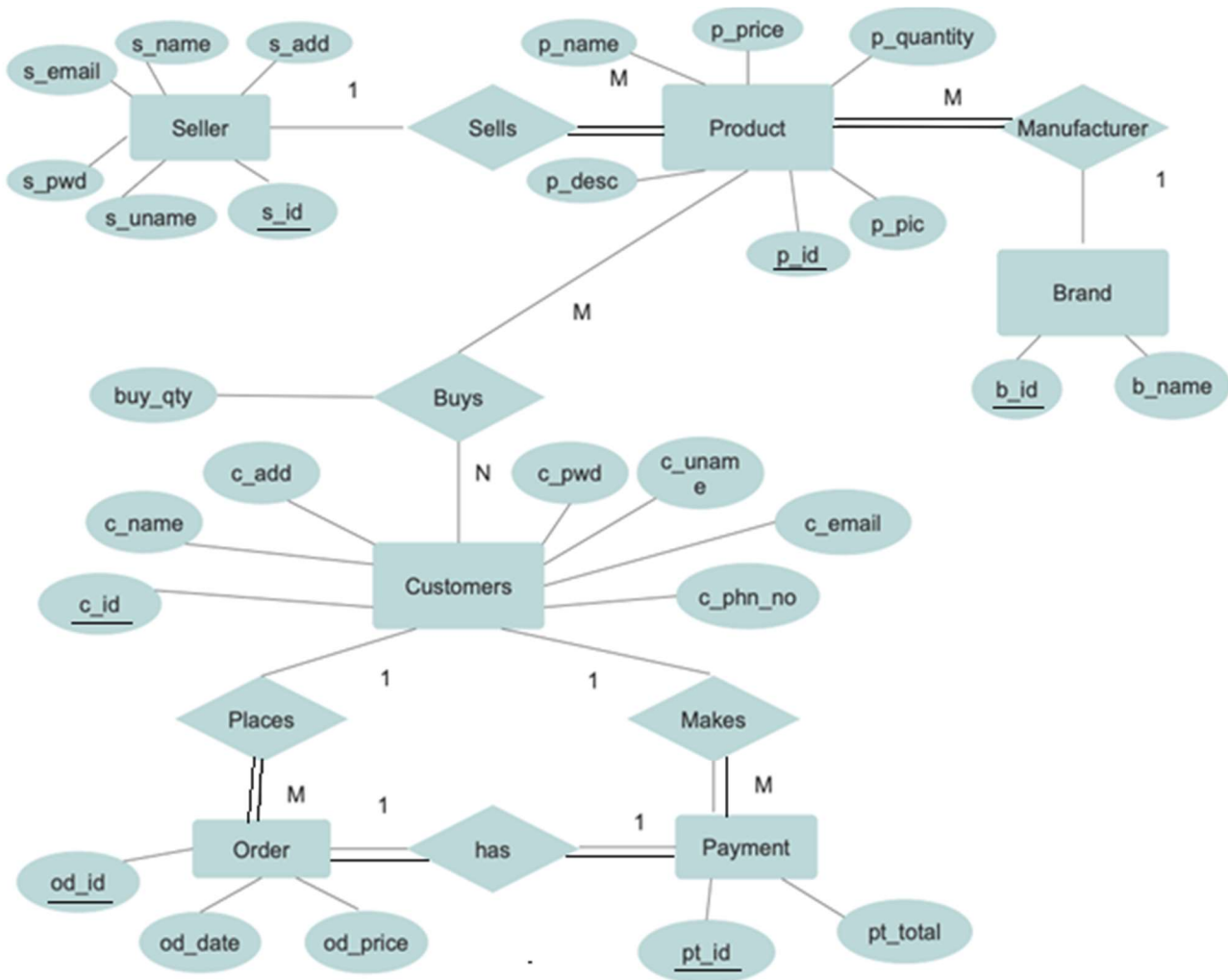
3. **Customer places Order:**

- Description: Depicts the relationship between a customer and an order.

4. **Order has Payment:**

- Description: Shows the relationship between an order and its associated payment.

ENTITY-RELATIONSHIP (ER) DIAGRAM



ENTITY-RELATIONSHIP (ER) DIAGRAM ENTITIES

1. Seller:

- Attributes: ID (s_id), Name(s_name), Email(s_email), Address(s_add), Username(s_uname), Password(s_pwd)

2. Customer:

- Attributes: ID (c_id), Name(c_name), Email(c_email), Address(c_add), Username(c_uname), Password(c_pwd), Phone(c_phno)

3. Product:

- Attributes: ProductID (primary key) (p_id), Name (p_name), Description (p_desc), Price (p_price), ProductPicture(p_pic), product (p_qty).

4. Brand:

- Attributes: BrandID (primary key) (b_id), Name (b_name).

5. Order:

- Attributes: OrderID (primary key) (od_id), Date (od_date), Order Price(od_price).

6. **Payment:**

- Attributes: PaymentID (primary key) (pt_id), Payment Total(pt_total).

ENTITY-RELATIONSHIP (ER) DIAGRAM RELATIONSHIPS

1. **Seller sells Product**

- **Cardinality:** One-to-Many (One seller can sell many products; one product is sold by one seller).
- **Participation of the entities:** The product entity has total participation in the ER diagram while there is partial participation between the Seller entity.

2. **Brand Manufactures Product**

- **Cardinality:** Many-to-One (Many products can belong to one brand; one brand can have many products).
- **Participation of the entities:** The product entity has total participation in the ER diagram while there is partial participation between the Brand entity.

3. **Customer places Order**

- **Cardinality:** One-to-Many (One customer can place many orders; one order is placed by one customer).
- **Participation of the entities:** The Order entity has total participation in the ER diagram while there is partial participation between the Customer entity.

4. **Order has Payment**

- **Cardinality:** One-to-One (One order is associated with one payment; one payment is associated with one order).
- **Participation of the entities:** The Order entity has total participation in the ER diagram and also there is total participation between the Payment entity.

CONVERSION OF THE ER DIAGRAM TO RELATIONAL MAPPING VIA ALGORITHMS

Step 1: Mapping of Regular Entities

- For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
- Choose one of the key attributes of E as the primary key for R.
- If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

Step 2: Mapping of Weak Entity Types

- For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of R.
- Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
- The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W.

Step 3: Mapping of Binary 1:1 Relation Types

- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.
- There are three possible approaches:
- Foreign Key approach: Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S.
- Merged relation option: An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.
- Cross-reference or relationship relation option: The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

Step 4: Mapping of Binary 1:N Relationship Types.

- For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type.
- Include as a foreign key in S the primary key of the relation T that represents the other entity type participating in R.
- Include any simple attributes of the 1:N relation type as attributes of S

Step 5: Mapping of Binary M:N Relationship Types.

- For each regular binary M:N relationship type R, create a new relation S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.
- Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.

Step 6: Mapping of Multivalued attributes.

- For each multivalued attribute A, create a new relation R.
- This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type of relationship type that has A as an attribute.
- The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

SELLER

<u>s_id</u>	s_uname	s_pwd	s_email	s_name	s_add
-------------	---------	-------	---------	--------	-------

CUSTOMER

<u>c_id</u>	c_name	c_add	c_email	c_username	c_pwd	c_phno
-------------	--------	-------	---------	------------	-------	--------

BRAND

<u>b_id</u>	b_name
-------------	--------

PRODUCT

<u>p_id</u>	p_name	p_price	p_desc	p_pic	s_id	b_id	p_qty
-------------	--------	---------	--------	-------	------	------	-------

CART

<u>c_id</u>	<u>p_id</u>	buy_qty
-------------	-------------	---------

ORDER

<u>od_id</u>	od_date	od_price	c_id
--------------	---------	----------	------

PAYMENT

<u>pt_id</u>	pt_total	c_id
--------------	----------	------

ORDER PAYMENT

<u>od_id</u>	<u>pt_id</u>
--------------	--------------

NORMALIZATION

SELLER

<u>s_id</u>	s_uname	s_pwd	s_email	s_name	s_add
-------------	---------	-------	---------	--------	-------

1NF

- s_id (primary key) is atomic because it's a single value.
- s_name, s_add, s_uname, and s_pwd are also atomic since they represent single values.

Therefore, the table is in 1NF.

2NF

- s_id is the primary key, and all other attributes (s_name, s_add, s_uname, and s_pwd) are functionally dependent on the primary key.

Therefore, the table is in 2NF.

3NF

- There are no transitive dependencies.
- Each non-prime attribute (s_name, s_add, s_uname, s_pwd) depends only on the primary key (s_id).

Therefore, the table is in 3NF.

CUSTOMER

<u>c_id</u>	c_name	c_add	c_email	c_username	c_pwd	c_phno
-------------	--------	-------	---------	------------	-------	--------

1NF

- c_id (primary key) is atomic because it's a single value.
- c_name, c_add, c_username, and c_pwd, c_phno and c_email are also atomic since they represent single values.

Therefore, the table is in 1NF.

2NF

- c_id is the primary key, and all other attributes (c_name, c_add, c_username, and c_pwd, c_phno, c_email) are functionally dependent on the primary key(c_id).

Therefore, the table is in 2NF.

3NF

- There are no transitive dependencies.
- Each non-prime attribute (c_name, c_add, c_username, and c_pwd, c_phno, c_email) depends only on the primary key (c_id).

Therefore, the table is in 3NF.

BRAND

<u>b_id</u>	b_name
-------------	--------

1NF

- b_id (primary key) is atomic because it's a single value.
- b_name is also atomic since it represent single value.

Therefore, the table is in 1NF.

2NF

- b_id is the primary key, and attribute b_name is functionally dependent on the primary key.

Therefore, the table is in 2NF.

3NF

- There are no transitive dependencies.
- Non-prime attribute b_name depends only on the primary key (b_id).

Therefore, the table is in 3NF.

PRODUCT

<u>p_id</u>	p_name	p_price	p_desc	p_pic	s_id	b_id	p_qty
-------------	--------	---------	--------	-------	------	------	-------

1NF

- p_id (primary key) is atomic because it's a single value.
- p_name, p_price, p_desc, s_id, b_id, and p_qty are also atomic since they represent single values.

Therefore, the table is in 1NF.

2NF

- p_id is the primary key, and all other attributes (p_name, p_price, p_desc, s_id, b_id, and p_qty) are functionally dependent on the primary key(p_id).

Therefore, the table is in 2NF.

3NF

- There are no transitive dependencies.
- Each non-prime attribute (p_name, p_price, p_desc, s_id, b_id, and p_qty) depends only on the primary key (p_id).

Therefore, the table is in 3NF.

CART

<u>c_id</u>	<u>p_id</u>	buy_qty
-------------	-------------	---------

1NF

- (c_id, p_id) (primary key) is atomic because it's a single value.
- buy_qty is also atomic since it represent single value.

Therefore, the table is in 1NF.

2NF

- b_id is the primary key, and attribute buy_qty is functionally dependent on the primary key. (c_id, p_id)

Therefore, the table is in 2NF.

3NF

- There are no transitive dependencies.
- Non-prime attribute buy_qty depends only on the primary key (c_id, p_id).

Therefore, the table is in 3NF.

ORDER

<u>od_id</u>	od_date	od_price	c_id
--------------	---------	----------	------

1NF

- od_id (primary key) is atomic because it's a single value.
- od_date, od_price, c_id are also atomic since they represent single values.

Therefore, the table is in 1NF.

2NF

- p_id is the primary key, and all other attributes (od_date, od_price, c_id) are functionally dependent on the primary key(od_id).

Therefore, the table is in 2NF.

3NF

- There are no transitive dependencies.
- Each non-prime attribute (od_date, od_price, c_id) depends only on the primary key (od_id).

Therefore, the table is in 3NF.

ORDER PAYMENT

<u>od_id</u>	<u>pt_id</u>
--------------	--------------

1NF

- (od_id, pt_id) (primary key) is atomic because it's a single value.

Therefore, the table is in 1NF.

2NF

- There is no attribute that is not a primary key.

Therefore, the table is in 2NF.

3NF

- There is no attribute that is not a primary key.

Therefore, the table is in 3NF.

PAYMENT

<u>pt_id</u>	pt_total	c_id
--------------	----------	------

1NF

- pt_id (primary key) is atomic because it's a single value.
- pt_total, c_id are also atomic since they represent single values.

Therefore, the table is in 1NF.

2NF

- p_id is the primary key, and all other attributes (pt_total, c_id) are functionally dependent on the primary key(pt_id).

Therefore, the table is in 2NF.

3NF

- There are no transitive dependencies.
- Each non-prime attribute (pt_total, c_id) depends only on the primary key (pt_id).

Therefore, the table is in 3NF.

DATA DEFINITION LANGUAGE QUERIES

```
CREATE TABLE `brand` (  
    `b_id` int(11) NOT NULL,  
    `b_name` varchar(100) NOT NULL  
);
```

```
CREATE TABLE `seller` (  
    `s_id` int(11) NOT NULL,  
    `s_name` varchar(50) NOT NULL,  
    `s_add` varchar(250) NOT NULL,  
    `s_uname` varchar(50) NOT NULL,  
    `s_pwd` varchar(20) NOT NULL,  
    `s_email` varchar(50) NOT NULL  
);
```

```
CREATE TABLE `customer` (  
    `c_id` int(11) NOT NULL,  
    `c_fname` varchar(50) NOT NULL,  
    `c_add` varchar(250) NOT NULL,  
    `c_uname` varchar(50) NOT NULL,  
    `c_pwd` varchar(20) NOT NULL,  
    `c_email` varchar(50) NOT NULL,  
    `c_phno` bigint(20) NOT NULL  
);
```

```
CREATE TABLE `product` (  
    `p_id` int(11) NOT NULL,
```

```
`p_name` varchar(50) NOT NULL,  
`p_price` float NOT NULL,  
`p_desc` varchar(250) NOT NULL,  
`s_id` int(11) DEFAULT NULL,  
`p_pic` varchar(255) NOT NULL,  
`b_id` int(11) NOT NULL,  
`p_qty` int(11) NOT NULL  
);
```

```
CREATE TABLE `cart` (  
  `c_id` int(11) NOT NULL,  
  `p_id` int(11) NOT NULL,  
  `buy_qty` int(11) NOT NULL  
);
```

```
CREATE TABLE `orders` (  
  `od_id` int(11) NOT NULL,  
  `c_id` int(11) NOT NULL,  
  `od_date` date NOT NULL,  
  `od_price` float NOT NULL  
);
```

```
CREATE TABLE `payment` (  
  `pt_id` int(11) NOT NULL,  
  `c_id` int(11) DEFAULT NULL,  
  `pt_total` float DEFAULT NULL,  
  `txn_id` varchar(50) DEFAULT NULL
```

);

CREATE TABLE `orderpayment` (

 `od_id` int(11) NOT NULL,

 `pt_id` int(11) NOT NULL

);

CREATE TABLE `customerproduct` (

 `c_id` int(11) NOT NULL,

 `p_id` int(11) NOT NULL,

 `buy_qty` int(11) NOT NULL,

 `od_id` int(11) NOT NULL,

 `cp_id` int(11) NOT NULL

);

CREATE TABLE `admin` (

 `adm_username` varchar(50) NOT NULL,

 `adm_password` varchar(50) NOT NULL,

 `adm_name` varchar(100) NOT NULL

);

CREATE TABLE `productbackup` (

 `pb_id` int(11) NOT NULL,

 `pb_name` varchar(50) DEFAULT NULL,

 `p_price` float DEFAULT NULL,

 `p_desc` varchar(250) DEFAULT NULL,

 `s_id` int(11) DEFAULT NULL,

```
`p_pic` varchar(255) DEFAULT NULL,  
`b_id` int(11) DEFAULT NULL,  
`p_qty` int(11) DEFAULT NULL  
);
```

```
ALTER TABLE `admin`  
  
ADD PRIMARY KEY (`adm_username`);
```

```
ALTER TABLE `brand`  
  
ADD PRIMARY KEY (`b_id`);
```

```
ALTER TABLE `cart`  
  
ADD PRIMARY KEY (`c_id`,`p_id`),  
  
ADD KEY `p_id` (`p_id`),  
  
ADD KEY `c_id` (`c_id`);
```

```
ALTER TABLE `customer`  
  
ADD PRIMARY KEY (`c_id`),  
  
ADD UNIQUE KEY `c_uname` (`c_uname`),  
  
ADD UNIQUE KEY `c_email` (`c_email`);
```

```
ALTER TABLE `customerproduct`  
  
ADD PRIMARY KEY (`cp_id`);
```

```
ALTER TABLE `orderpayment`  
  
ADD PRIMARY KEY (`od_id`,`pt_id`),  
  
ADD KEY `pt_id` (`pt_id`);
```

```
ALTER TABLE `orders`  
  
ADD PRIMARY KEY (`od_id`),  
  
ADD KEY `c_id` (`c_id`);
```

```
ALTER TABLE `payment`  
  
ADD PRIMARY KEY (`pt_id`),  
  
ADD KEY `c_id` (`c_id`);
```

```
ALTER TABLE `product`  
  
ADD PRIMARY KEY (`p_id`),  
  
ADD KEY `s_id` (`s_id`),  
  
ADD KEY `b_id` (`b_id`);
```

```
ALTER TABLE `productbackup`  
  
ADD PRIMARY KEY (`pb_id`);
```

```
ALTER TABLE `seller`  
  
ADD PRIMARY KEY (`s_id`),  
  
ADD UNIQUE KEY `s_uname` (`s_uname`),  
  
ADD UNIQUE KEY `s_email` (`s_email`),  
  
ADD KEY `s_id` (`s_id`);
```

```
ALTER TABLE `brand`  
  
MODIFY `b_id` int(11) NOT NULL AUTO_INCREMENT;  
  
ALTER TABLE `customer`  
  
MODIFY `c_id` int(11) NOT NULL AUTO_INCREMENT;  
  
ALTER TABLE `customerproduct`
```



```
MODIFY `cp_id` int(11) NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE `orders`
```

```
MODIFY `od_id` int(11) NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE `payment`
```

```
MODIFY `pt_id` int(11) NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE `product`
```

```
MODIFY `p_id` int(11) NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE `seller`
```

```
MODIFY `s_id` int(11) NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE `cart`
```

```
ADD CONSTRAINT `cart_ibfk_1` FOREIGN KEY (`c_id`) REFERENCES `customer` (`c_id`) ON  
DELETE CASCADE,
```

```
ADD CONSTRAINT `cart_ibfk_2` FOREIGN KEY (`p_id`) REFERENCES `product` (`p_id`) ON  
DELETE CASCADE;
```

```
ALTER TABLE `orderpayment`
```

```
ADD CONSTRAINT `orderpayment_ibfk_1` FOREIGN KEY (`od_id`) REFERENCES `orders` (`od_id`)  
ON DELETE CASCADE,
```

```
ADD CONSTRAINT `orderpayment_ibfk_2` FOREIGN KEY (`pt_id`) REFERENCES `payment` (`pt_id`)  
ON DELETE CASCADE;
```

```
ALTER TABLE `orders`
```

```
ADD CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`c_id`) REFERENCES `customer` (`c_id`) ON  
DELETE CASCADE;
```

```
ALTER TABLE `orderstrack`
```

```
ADD CONSTRAINT `orderstrack_ibfk_1` FOREIGN KEY (`c_id`) REFERENCES `customer` (`c_id`),
```

```
ADD CONSTRAINT `orderstrack_ibfk_2` FOREIGN KEY (`s_id`) REFERENCES `seller` (`s_id`),
```

```
ADD CONSTRAINT `ordertrack_ibfk_3` FOREIGN KEY (`od_id`) REFERENCES `orders` (`od_id`);
```

```
ALTER TABLE `payment`
```

```
ADD CONSTRAINT `payment_ibfk_1` FOREIGN KEY (`c_id`) REFERENCES `customer` (`c_id`);
```

```
ALTER TABLE `product`
```

```
ADD CONSTRAINT `product_ibfk_1` FOREIGN KEY (`s_id`) REFERENCES `seller` (`s_id`) ON  
DELETE CASCADE,
```

```
ADD CONSTRAINT `product_ibfk_2` FOREIGN KEY (`b_id`) REFERENCES `brand` (`b_id`) ON  
DELETE CASCADE;
```

VIEW

```
Create VIEW `paymentview` AS
```

```
SELECT `p`.`pt_id` AS `pt_id`, `p`.`txn_id` AS `txn_id`, `op`.`od_id` AS `od_id`, `p`.`pt_total` AS `pt_total`,  
`c`.`c_fname` AS `c_fname`, `c`.`c_undef` AS `c_undef`, `c`.`c_email` AS `c_email`
```

```
FROM ((`payment` `p` join `orderpayment` `op` on(`p`.`pt_id` = `op`.`pt_id`)) join `customer` `c`  
on(`c`.`c_id` = `p`.`c_id`));
```

TRIGGER

```
DELIMITER $$
```

```
CREATE TRIGGER `t1` BEFORE DELETE ON `product` FOR EACH ROW BEGIN
```

```
INSERT into productbackup(pb_id,pb_name,p_desc,p_pic,p_price,p_qty,b_id,s_id)  
VALUES(old.p_id,old.p_name,old.p_desc,old.p_pic,old.p_price,old.p_qty,old.b_id,old.s_id);
```

```
end
```

```
$$
```

```
DELIMITER ;
```

DATA MANIPULATION LANGUAGE QUERIES**USER SECTION :****REGISTER :**

```
SELECT * from `customer` where c_username='$user_name' OR c_email='$user_email';
```

```
INSERT          into          `customer`(c_fname,c_add,c_username,c_pwd,c_email,c_phno)          values
('$full_name','$user_address','$user_name','$user_pass','$user_email','$user_phno');
```

LOGIN :

```
SELECT * from `customer` where c_username='$user_name';
```

PAYMENT :

```
SELECT * FROM cart
```

```
INNER JOIN product ON product.p_id = cart.p_id
```

```
INNER JOIN customer ON customer.c_id = cart.c_id
```

```
WHERE cart.c_id = $c_id;
```

HOME :

```
SELECT * from `product` order by rand() LIMIT 0,9
```

```
SELECT * from `product` where s_id = $seller_id
```

```
SELECT * from `product` where b_id = $brand_id order by rand() LIMIT 0,9
```

```
SELECT * from `seller`
```

```
SELECT * from `brand`
```

```
SELECT * from `product` where p_name like '%$key%'
```

```
SELECT * from product where p_id=$pro_id;
```

```
SELECT * from product
```

```
INNER join seller using (s_id)
```

```
INNER join brand using (b_id)
```

```
where p_id=$pro_id;
```

CART :

```
SELECT * from `cart` where c_id=$customer_id
```

```
DELETE FROM `cart` WHERE c_id=$c_id and p_id=$p_id;
```

```
SELECT * from `cart` where c_id=$customer_id AND p_id=$product_id
```

```
INSERT into `cart` (c_id,p_id,buy_qty) values($customer_id,$product_id,1);
```

```
SELECT *
```

```
FROM cart
```

```
INNER JOIN product ON Product.p_id = cart.p_id
```

```
WHERE c_id=$c_id;
```

```
SELECT * FROM cart
```

```
INNER JOIN product ON product.p_id = cart.p_id
```

```
INNER JOIN customer ON customer.c_id = cart.c_id
```

```
WHERE cart.c_id = $c_id;
```

PAYMENT :

```
SELECT * FROM cart
```

```
INNER JOIN product ON product.p_id = cart.p_id
```

```
INNER JOIN customer ON customer.c_id = cart.c_id
```

```
WHERE cart.c_id = $c_id;
```

ORDERS :

```
UPDATE `cart`
```

```
SET buy_qty = $quantity
```

WHERE c_id=\$c_id and p_id=\$p_id;

SELECT * FROM cart

INNER JOIN product ON product.p_id = cart.p_id

INNER JOIN customer ON customer.c_id = cart.c_id

WHERE cart.c_id = \$c_id;

UPDATE product

SET p_qty = p_qty - \$qty

WHERE p_id = \$p_id;

INSERT into `payment` (c_id,pt_total,txn_id) values (\$c_id,\$totalPrice,\$trans_id');

INSERT into `orders` (c_id,od_price,od_date) values (\$c_id,\$totalPrice,CURRENT_DATE);

INSERT into `orderpayment` (od_id,pt_id) values (\$last_INSERT_id_od,\$last_INSERT_id_pt);

INSERT INTO customerproduct (c_id, p_id, buy_qty, od_id) SELECT c_id, p_id, buy_qty, \$last_INSERT_id_od AS od_id FROM cart;

DELETE FROM cart

WHERE c_id = \$c_id;

SELECT orders.od_id,orders.od_date,orders.od_price,payment.txn_id

FROM orders

INNER JOIN orderpayment on orderpayment.od_id=orders.od_id

INNER JOIN payment on payment.pt_id=orderpayment.pt_id

where orders.c_id=\$c_id

order by od_id desc;

SELECT product.p_pic,product.p_name,product.p_price,customerproduct.buy_qty

FROM `customerproduct`

INNER JOIN product on product.p_id=customerproduct.p_id

WHERE customerproduct.od_id=\$od_id;

ADMIN SECTION :

ADMIN LOGIN :

SELECT * from `admin` where adm_username='\$user_name';

UTILITIES :

ADD BRAND :

SELECT * from `brand` where b_name='\$brand_name';

INSERT into `brand` (b_name) values ('\$brand_name');

INSERT into `product` (p_name,p_price,p_desc,s_id,p_pic,b_id,p_qty) values
('\$mobile_name','\$mobile_price','\$mobile_desc','\$mobile_seller','\$mobile_pic','\$mobile_brand','\$mobile_qty');

SELECT * from `seller` where s_name='\$seller_name';

DELETE FROM `product` WHERE p_id=\$p_id;

DELETE FROM `customer` WHERE c_id=\$c_id;

DELETE FROM `seller` WHERE s_id=\$s_id;

DELETE FROM `brand` WHERE b_id=\$b_id;

```
SELECT * from `productbackup`  
  
INNER join `seller` using (s_id)  
  
INNER join `brand` using (b_id);
```

```
SELECT product.p_pic,product.p_name,product.p_price,customerproduct.buy_qty  
  
FROM `customerproduct`  
  
INNER JOIN product on product.p_id=customerproduct.p_id  
  
WHERE customerproduct.od_id=$od_id;
```

```
SELECT orders.od_id, customer.c_fname,customer.c_email,orders.od_date,orders.od_price  
  
FROM orders  
  
INNER JOIN customer on orders.c_id=customer.c_id order by od_id desc;
```

```
SELECT * from paymentview order by od_id desc;
```

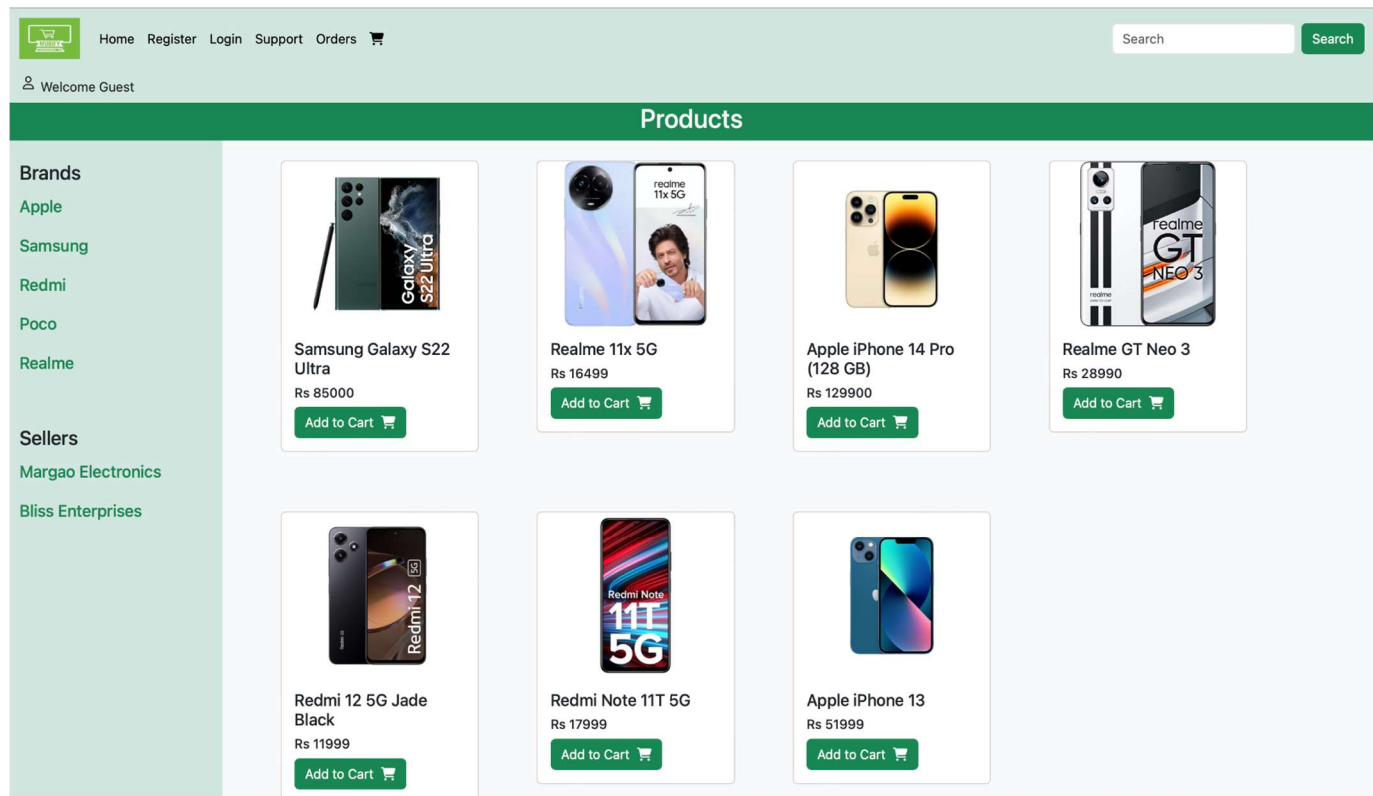
```
SELECT * from `product`  
  
INNER join `seller` using (s_id)  
  
INNER join `brand` using (b_id);
```

```
UPDATE `product`  
  
SET p_qty = $pro_qty  
  
WHERE p_id=$p_id;
```

```
SELECT * from `customer`;
```

FRONTEND EXPLANATION :

HOME PAGE:



The homepage manages user sessions for logging in and out, potentially handles shopping cart functionality (though the specific implementation is missing), and dynamically displays product information based on selected brands and sellers. The modular structure and Bootstrap integration contribute to code organization and improved user interface design. The script follows a responsive design approach, making it suitable for various screen sizes.

Here's an overview of its functionalities :

DYNAMIC PRODUCT DISPLAY:

- The main content area of the webpage is designed to dynamically display product information based on brands and sellers. It also displays the products searched in the search bar.
- The left sidebar includes lists of brands and sellers. The brand and seller information is likely retrieved from `getSellers()` and `getBrands()` functions.
- The functions `getProducts()`, `getUniqueProductSeller()`, and `getUniqueProductBrand()` are called to fetch and display product information.
- The dynamic display suggests that the webpage can adapt its content based on the selected brand or seller, providing a more personalized and filtered view of products.

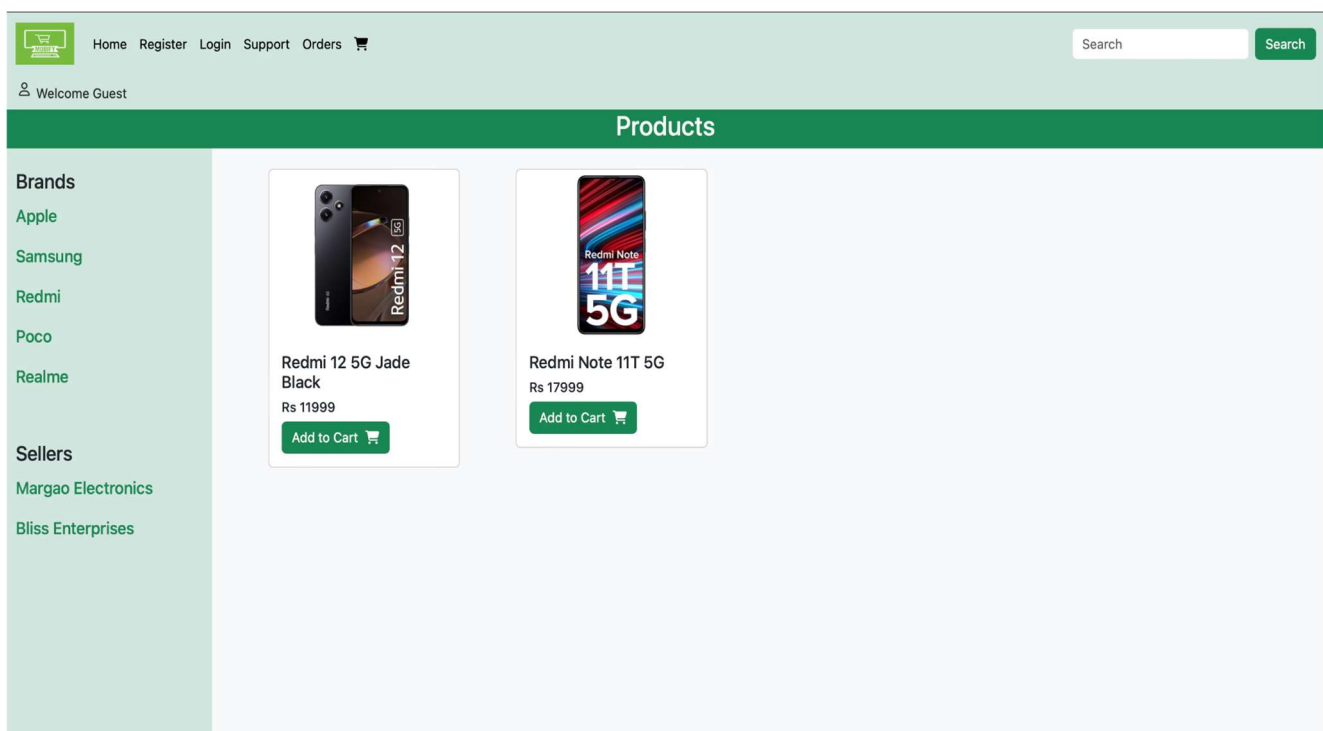
LOGO AND NAVIGATION TOGGLE BUTTON :

- The navigation bar includes a logo on the left side. The logo is displayed as a clickable element which refers to the index page.
- A toggle button is added for collapsing and expanding the navigation links on smaller screens.

NAVIGATION LINKS:

- Home: Provides a link to the home page ("index.php").
- Register and Login: These links are conditionally displayed based on whether the user is currently logged in. If the session variable "username" is not set, meaning the user is not logged in, the links to "Register" and "Login" are displayed. Otherwise, they are omitted.
- Support: Links to the support page ("./support.php").
- Orders: Links to the orders page ("./order.php").
- Cart: Includes a link to the cart page ("./cart.php") with an icon indicating a shopping cart. The number of items in the cart is dynamically displayed by querying the database to count the rows in the "cart" table associated with the current user.

SEARCH FORM:



- A search form is included on the right side of the navbar.
- It allows users to input search queries, and when submitted, the form redirects to "search_product.php" with the search keyword as a parameter.

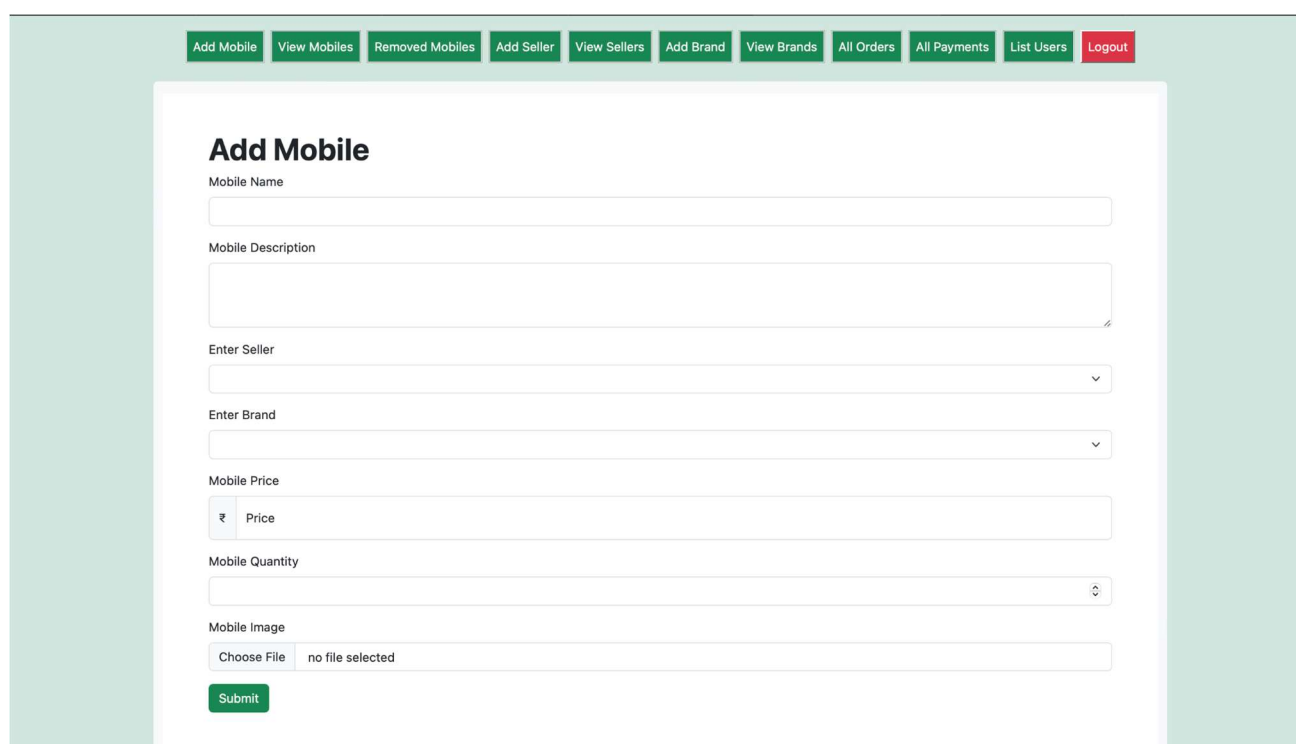
ADMIN PAGE:

This page forms an administrative interface that empowers the administrator of the website to manage , monitor, and update various aspects of the e-commerce platform for effective working of the website.

LOGIN PAGE FOR THE ADMIN:

- The admin login page is a crucial security feature. It presents a login form where authorized administrators can enter their credentials to access the admin panel.

ADDING PRODUCT:



The screenshot displays the 'Add Mobile' form within an admin dashboard. At the top, a navigation bar contains buttons for 'Add Mobile', 'View Mobiles', 'Removed Mobiles', 'Add Seller', 'View Sellers', 'Add Brand', 'View Brands', 'All Orders', 'All Payments', 'List Users', and 'Logout'. The main form area is titled 'Add Mobile' and includes the following fields: 'Mobile Name' (text input), 'Mobile Description' (text area), 'Enter Seller' (dropdown menu), 'Enter Brand' (dropdown menu), 'Mobile Price' (input with a currency symbol '₹' and a 'Price' label), 'Mobile Quantity' (input with a spinner icon), and 'Mobile Image' (file upload area with 'Choose File' and 'no file selected' text). A green 'Submit' button is located at the bottom of the form.

- Admins can use this script to add new products to the website. The script includes a form to input details like product name, category, description, price, and image.
- The admin uses this feature in order to add a product (mobile). To do so the admin has to enter the mobile name , it's description, it's picture and the quantity of mobiles in stock. Select the name of the seller that offers the particular product.

ADDING SELLER:

Add Seller

Name

Email address

Username

Password

Address

[Submit](#)

- This feature facilitates the addition of new sellers to the platform. Admins can use a form to input seller details, including company name, contact information, and other relevant data.

VIEW SELLER :

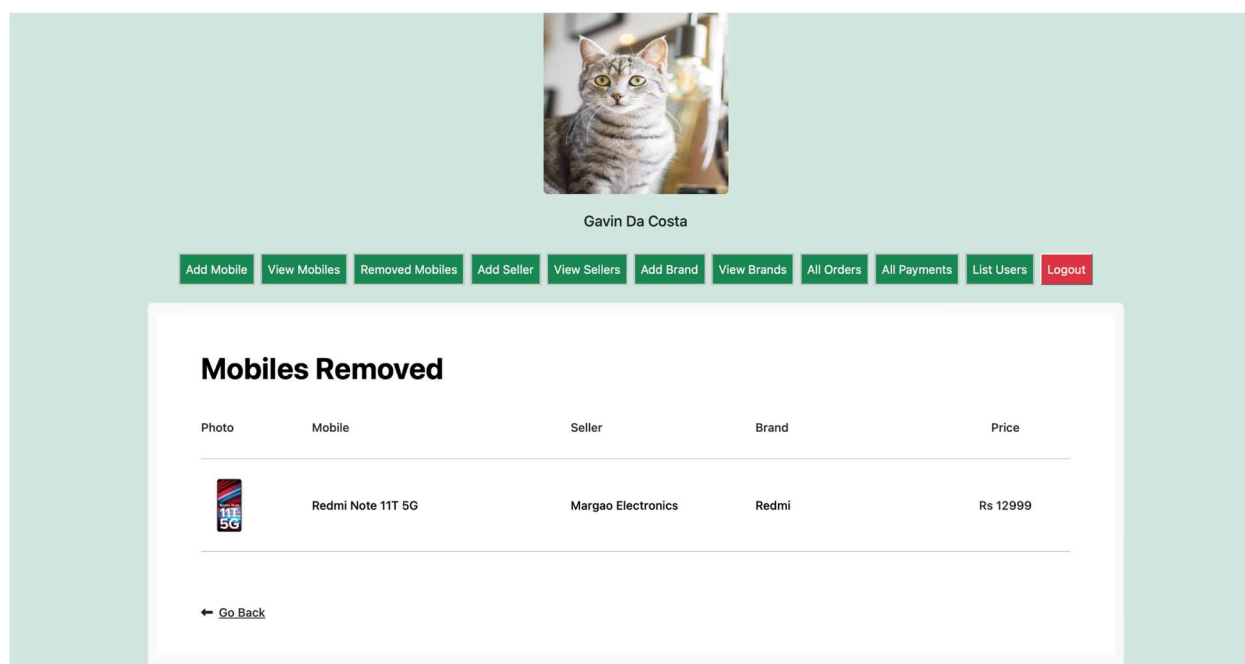
Gavin Da Costa

Sellers

Seller Name	Email	Username	Address	Remove
Margao Electronics	margaoelectronics@gmail.com	melectronics	Mayfair Apartment, Margao, Goa - 403601 (Near Margao Municipality, Behind Canara Bank)	×
Bliss Enterprises	blisspvtltd@gmail.com	bliss123	Fatorpa Goa	×

[← Go Back](#)


- Displays the Name of the seller and its contact information.
- This feature enables the admin to remove a seller from the website.

REMOVE PRODUCT:


Gavin Da Costa

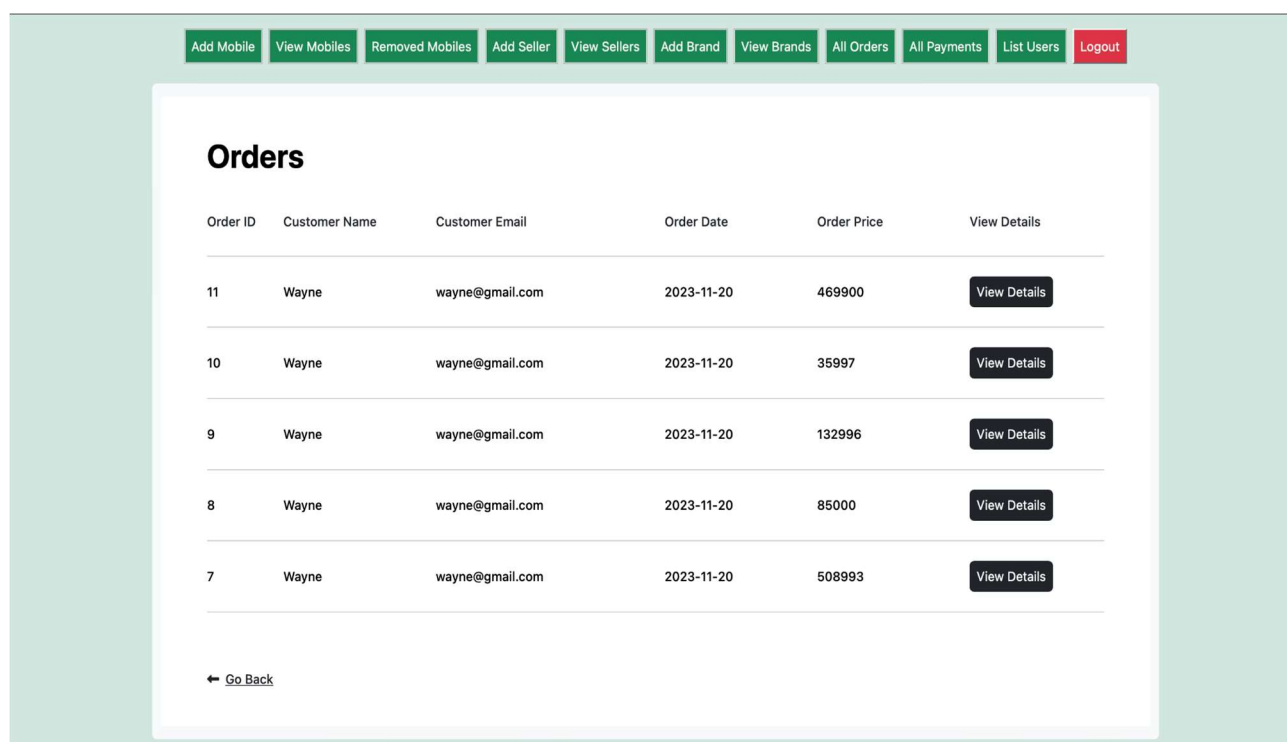
[Add Mobile](#)
[View Mobiles](#)
[Removed Mobiles](#)
[Add Seller](#)
[View Sellers](#)
[Add Brand](#)
[View Brands](#)
[All Orders](#)
[All Payments](#)
[List Users](#)
[Logout](#)

Mobiles Removed

Photo	Mobile	Seller	Brand	Price
	Redmi Note 11T 5G	Margao Electronics	Redmi	Rs 12999

[← Go Back](#)

- It shows the admin the products removed by the him/her from the home page. After the removal of a product. It displays the information of the discarded product.

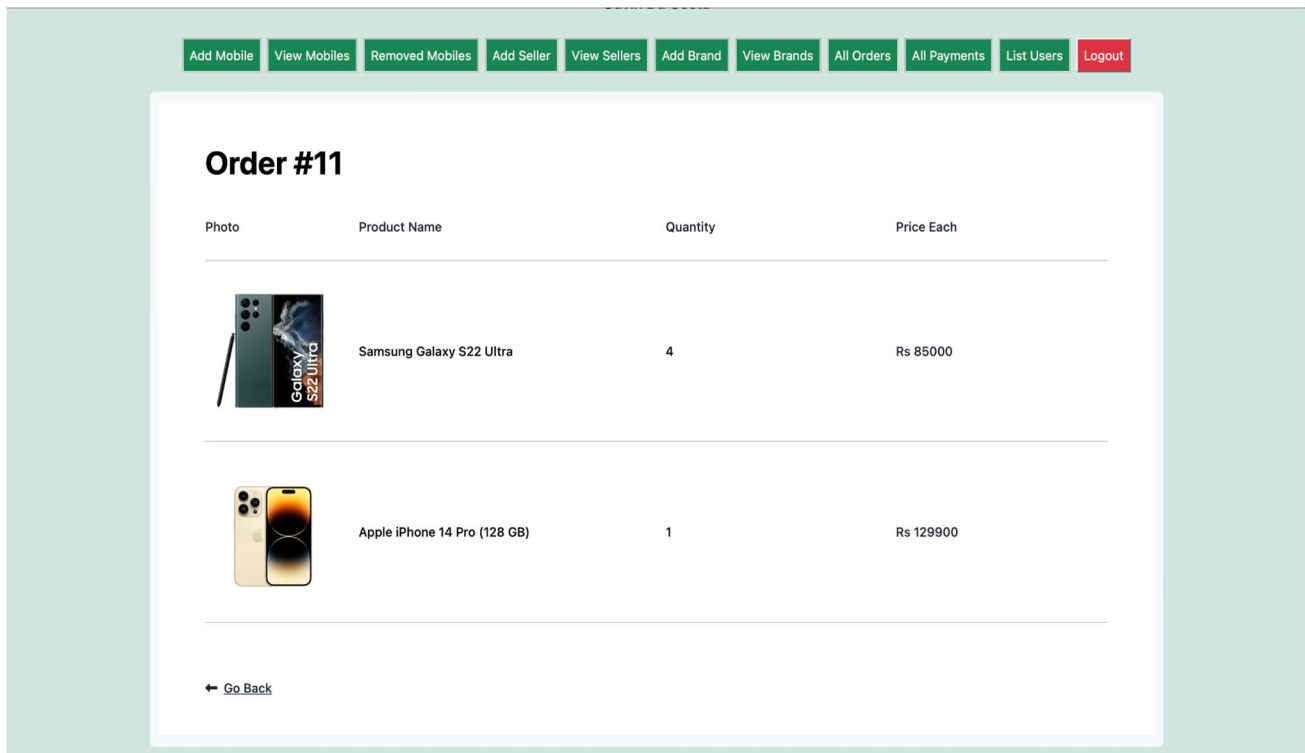
VIEW ORDER & ORDER DETAILS :


[Add Mobile](#)
[View Mobiles](#)
[Removed Mobiles](#)
[Add Seller](#)
[View Sellers](#)
[Add Brand](#)
[View Brands](#)
[All Orders](#)
[All Payments](#)
[List Users](#)
[Logout](#)

Orders

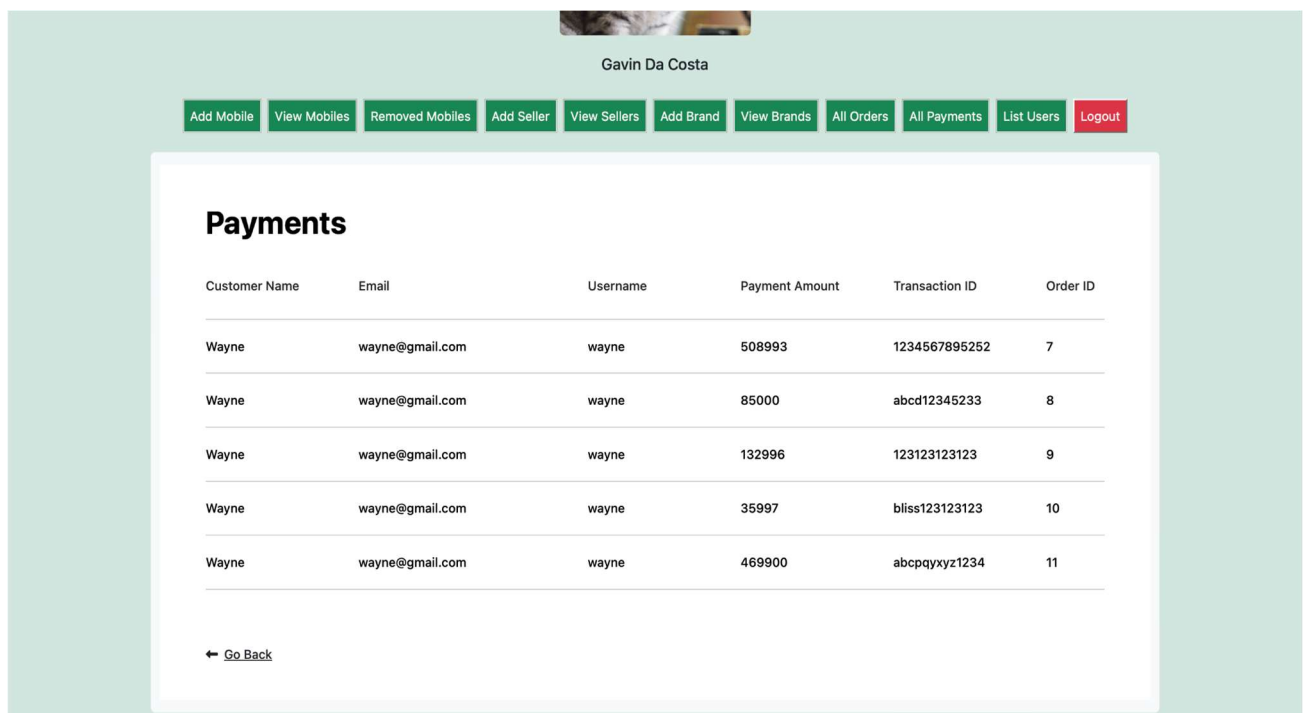
Order ID	Customer Name	Customer Email	Order Date	Order Price	View Details
11	Wayne	wayne@gmail.com	2023-11-20	469900	View Details
10	Wayne	wayne@gmail.com	2023-11-20	35997	View Details
9	Wayne	wayne@gmail.com	2023-11-20	132996	View Details
8	Wayne	wayne@gmail.com	2023-11-20	85000	View Details
7	Wayne	wayne@gmail.com	2023-11-20	508993	View Details

[← Go Back](#)

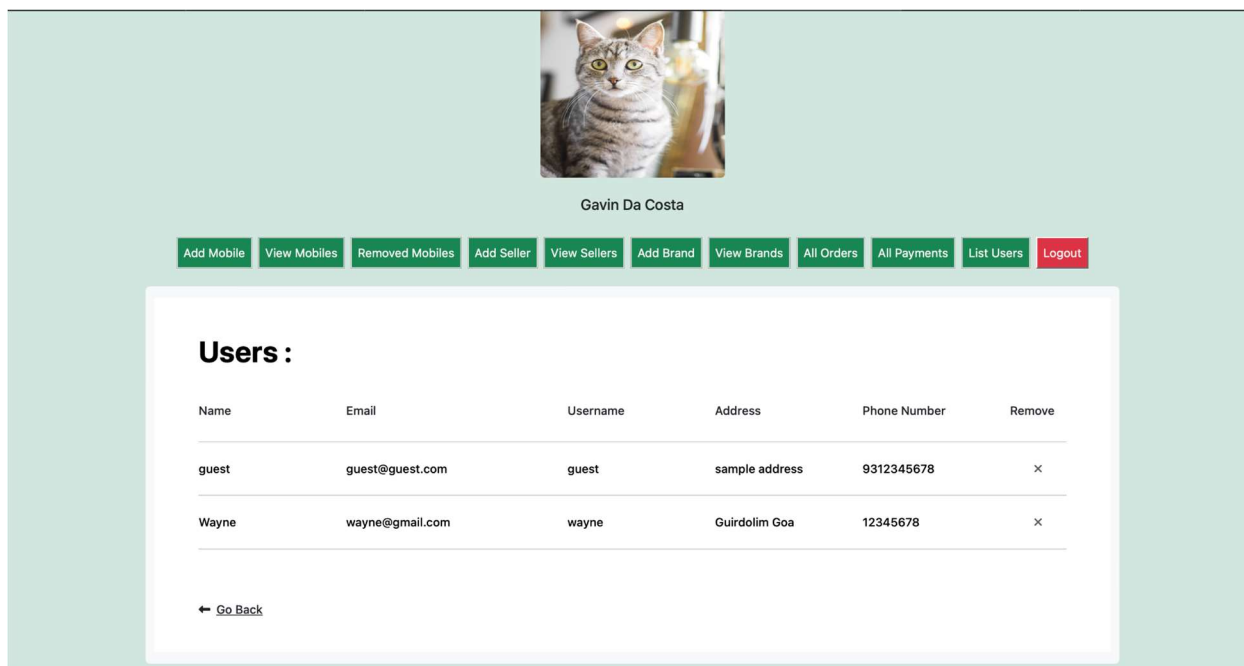


- Displays all the orders that have been made .
- Has an additional feature of viewing the detailed order information

VIEW PAYMENT:



- Admin can use this to review payment transaction history

VIEW USERS :


Gavin Da Costa

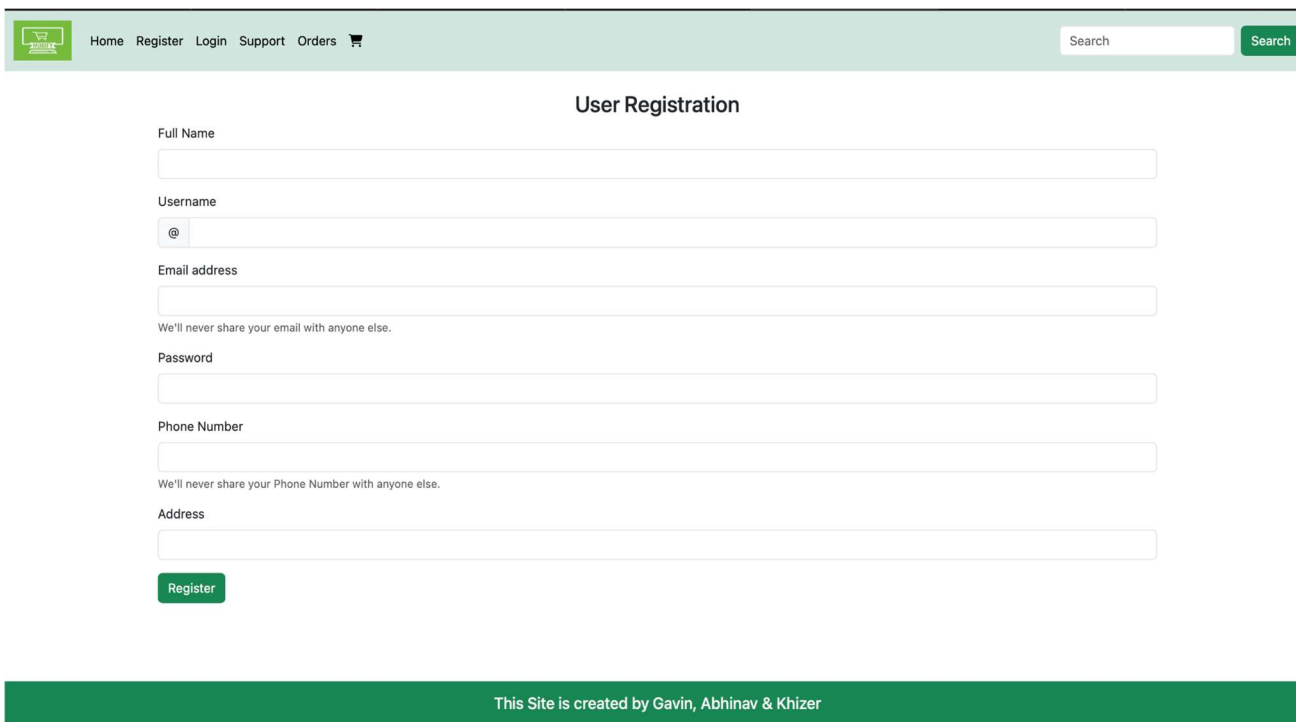
[Add Mobile](#)
[View Mobiles](#)
[Removed Mobiles](#)
[Add Seller](#)
[View Sellers](#)
[Add Brand](#)
[View Brands](#)
[All Orders](#)
[All Payments](#)
[List Users](#)
[Logout](#)

Users :

Name	Email	Username	Address	Phone Number	Remove
guest	guest@guest.com	guest	sample address	9312345678	×
Wayne	wayne@gmail.com	wayne	Guirdolim Goa	12345678	×

[← Go Back](#)

- This function allows admins to view details about users registered on the platform.
- It includes information such as user IDs, names, email addresses, and grants the admin the power to delist a user from the site.

USER REGISTRATION PAGE :


User Registration

Full Name

Username

Email address

We'll never share your email with anyone else.

Password

Phone Number

We'll never share your Phone Number with anyone else.

Address

[Register](#)

This Site is created by Gavin, Abhinav & Khizer

- This page handles user registration by processing form submissions, checking for duplicate users, inserting new users into the database, and providing a user-friendly interface for registration.
- The form includes standard fields for user information, and Bootstrap is used for styling.

USER REGISTRATION HANDLING:

- The script checks if the form has been submitted by checking for the existence of the `$_POST['add_user']` parameter.
- If the form is submitted, it retrieves the user's full name, username, email, password, phone number, and address from the form.
- It then queries the database to check if a user with the same username or email already exists. If a match is found, it displays an alert indicating that the user already exists.
- If no match is found, it inserts the new user's information into the `customer` table, displays a success message, and redirects the user to the `index.php` page.

FORM VALIDATION AND SECURITY:

- The includes a basic form validation by marking certain fields as required.
- The script uses prepared statements in SQL queries, which is good practice to prevent SQL injection.

User Login Page:

The screenshot shows a web application's user login interface. At the top, a light green header bar contains a shopping cart icon, navigation links for Home, Register, Login, Support, and Orders, and a search bar with a green 'Search' button. Below the header, the main content area is titled 'User Login'. It features a form with two input fields: 'Username' (with an '@' icon) and 'Password'. A green 'Login' button is positioned below the password field. A link for 'Register' is provided for users who do not have an account. At the bottom of the page, a solid green footer bar contains the text 'This Site is created by Gavin, Abhinav & Khizer'.

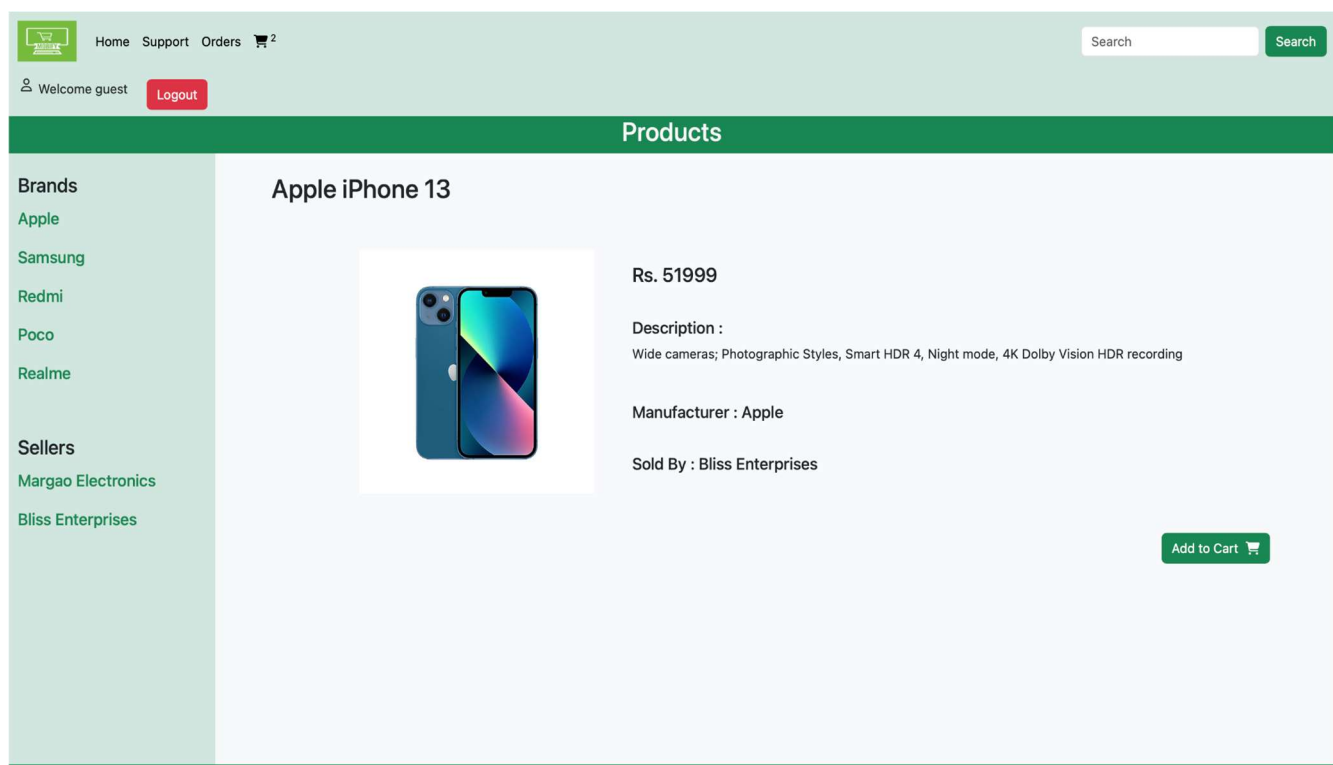
DATABASE CONNECTION:

- The script includes a file (`connect.php`) that presumably contains the database connection details. It uses the `mysqli` extension to connect to a mysql database.

USER VERIFICATION:

- When the form is submitted (`verify_user` button is clicked), the script retrieves the entered username (`cuname`) and password (`upass`) from the form.
- It then performs a select query on the `customer` table to check if a user with the entered username exists in the database.
- If the user does not exist (`$numrows == 0`), it displays an alert saying "user doesn't exist."
- If the user exists, it compares the entered password with the stored password in the database.
- If the passwords match, it displays an alert saying "login successful" and sets up a php session with user information.
- If the passwords do not match, it displays an alert saying "incorrect password."

VIEW PRODUCT DETAILS:



USER SESSIONS:

- The script handles user sessions with the code block checking if the `$_SESSION` parameter is set.
- If the user initiates a logout by accessing the page with the `logout` parameter in the URL, the script starts a session (if not already started), unsets all session variables, destroys the session, and then provides a visual confirmation through a JavaScript alert.
- The session-related operations ensure that the user is logged out, and the session data is cleared.

SESSION HANDLING:

- The script starts a session if it hasn't been started (`session_start()`).
- It checks if a user is logged in by verifying the existence of `$_SESSION['username']`.

ORDER PLACEMENT:

- If a user is logged in and the form with the name `addorder` is submitted, the script proceeds to process the order.
- It retrieves the customer ID (`$c_id`) from the session and fetches product details from the cart for that customer using a SQL query.
- It calculates the total price of the order based on the products in the cart.

PAYMENT RECORD INSERTION:

- It inserts a record into the `payment` table, associating the customer, total price, and transaction ID.

ORDER RECORD INSERTION:

- It inserts a record into the `orders` table, associating the customer, total price, and the current date.

ORDER-PAYMENT ASSOCIATION:

- It inserts a record into the `orderpayment` table, associating the order and payment records.

CUSTOMER-PRODUCT ASSOCIATION:

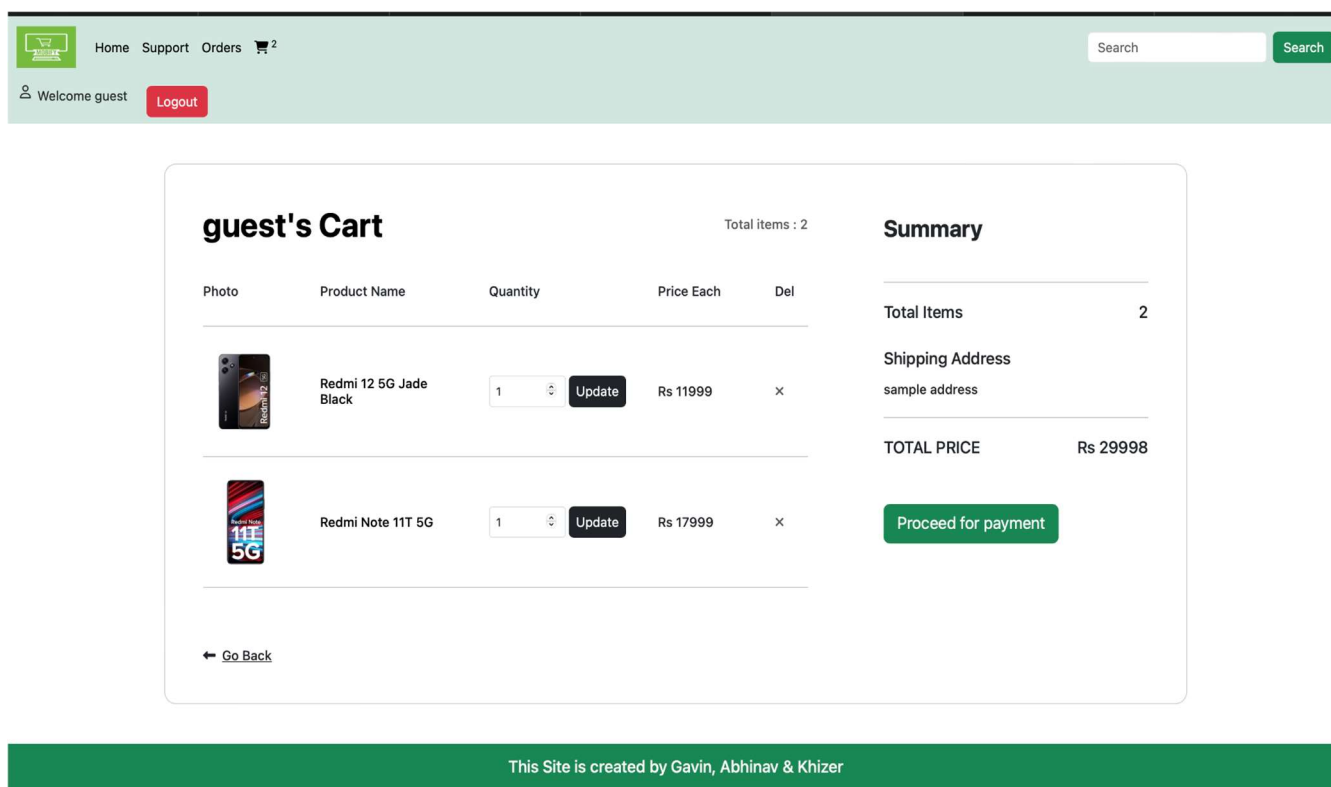
- It inserts records into the `customerproduct` table, associating the customer, product, and order details. This appears to represent the products purchased by the customer in the order.

CART CLEANUP:

- It deletes the products from the cart for the current customer.

ALERT AND REDIRECTION:

- Finally, it displays an alert indicating that the order has been placed successfully and redirects the user to the `index.php` page.

CART:

- Once logged in the user can access the cart.
- This page handles shopping cart actions such as updating quantities and deleting items, and dynamically displays the user's cart contents.

Here's an overview of its functionalities:

LOGOUT HANDLING:

- If the URL parameter **logout** is set, the script starts a session, unsets all session variables, destroys the session, displays a logout success message through a JavaScript alert, and then redirects the user to the **index.php** page.

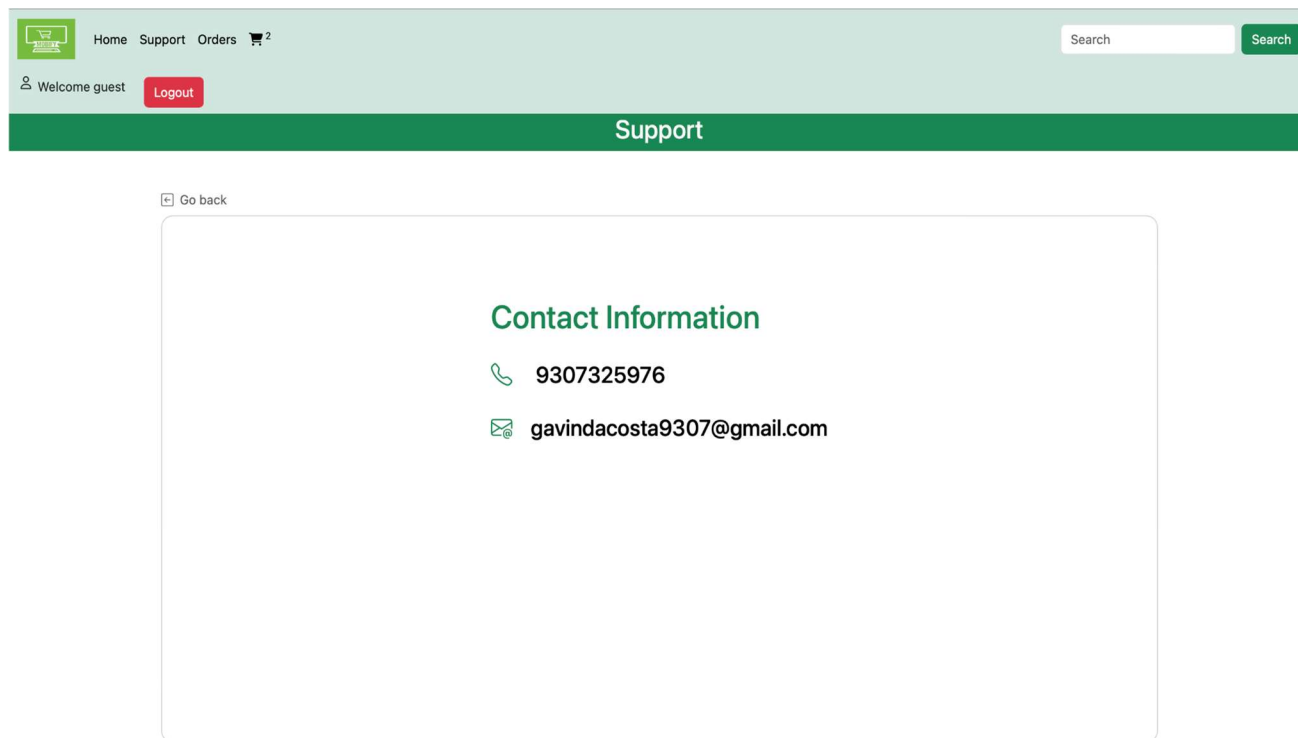
SHOPPING CART ACTIONS:

- If the URL parameter **pro_id** is set, the script checks for the existence of a session and retrieves the quantity and product ID from the form post data.
- It then checks if the requested quantity is within the available stock for the selected product. If yes, it updates the user's cart with the new quantity and redirects to the cart page. If not, it displays an alert indicating the maximum stock available.
- If the URL parameters **delete_item_cid** and **delete_item_pid** are set, the script deletes the corresponding item from the user's cart and redirects to the cart page.

PRODUCT DISPLAY SECTION:

- The product display section includes a loop (`getCartProducts()`) that retrieves and displays information about products in the user's cart. This includes the product photo, name, quantity, price each, and a delete button.
- The loop generates a row for each product in the cart, creating a dynamic and informative display.

SUPPORT PAGE:



- This page creates a support/contact information page.
 - It allows users to logout, handles shopping cart functionality, and provides contact details for support
- Here's an overview of its functionalities.

CONTACT INFORMATION:

- The contact information includes a phone number and an email address.

GO BACK LINK:

- A "go back" link is provided, which, when clicked, triggers the `history.back()` function, allowing the user to navigate back to the previous page.