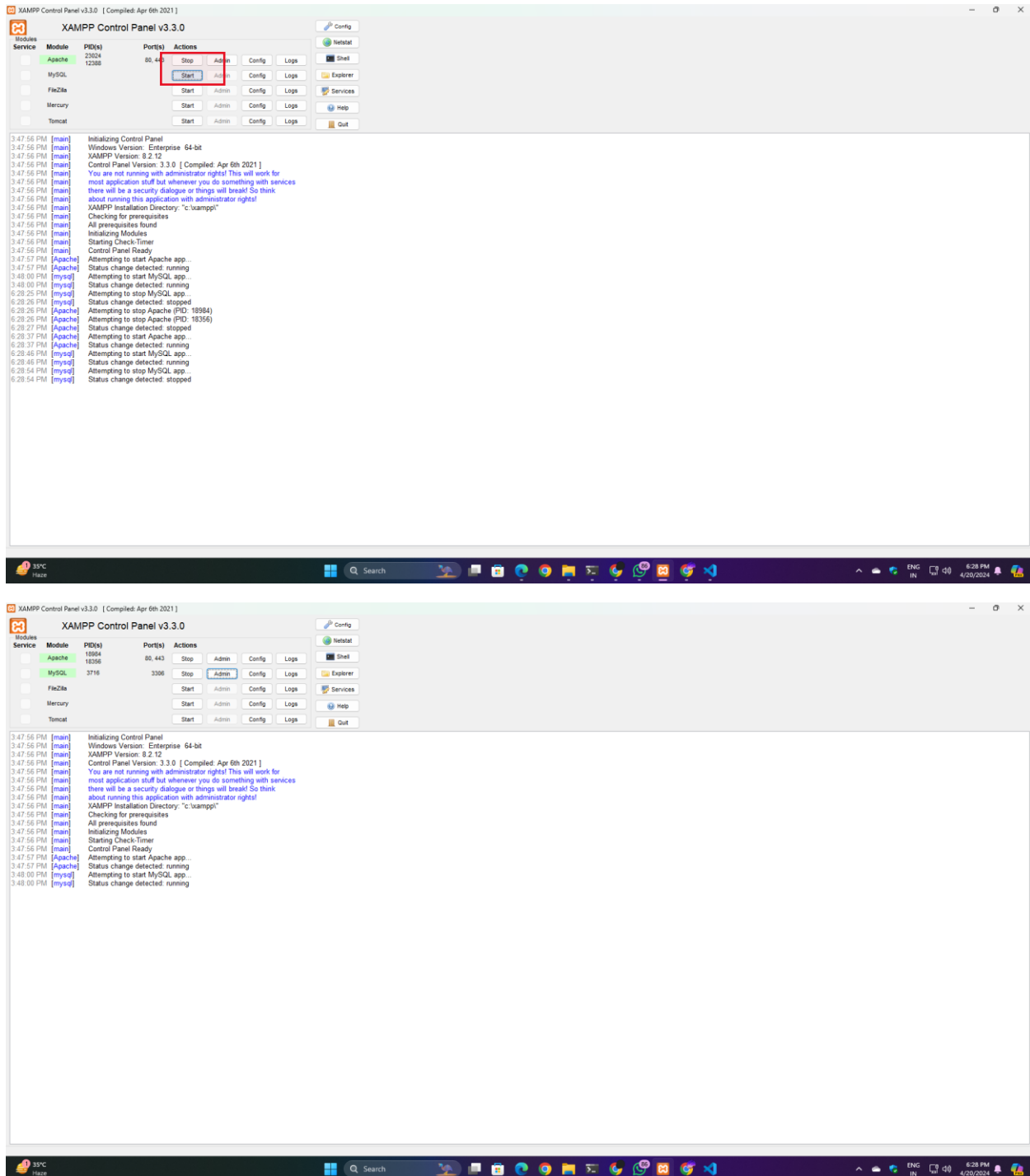
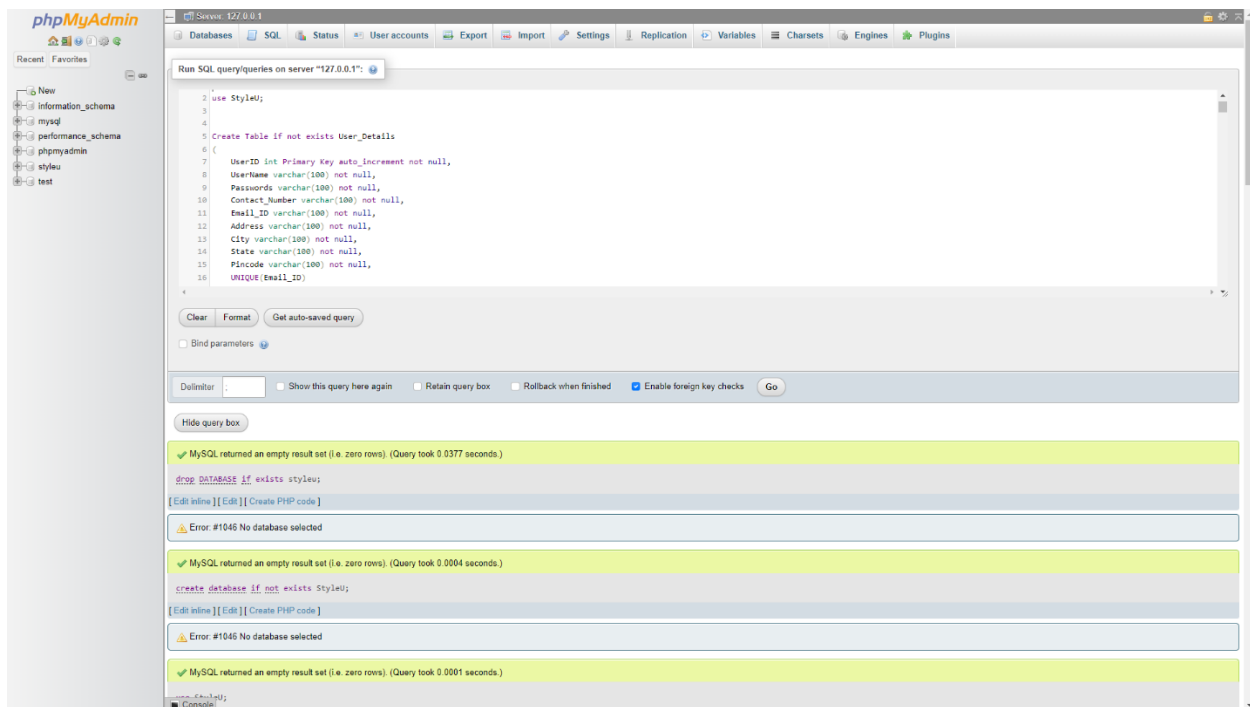


Style U: Online Clothing Store

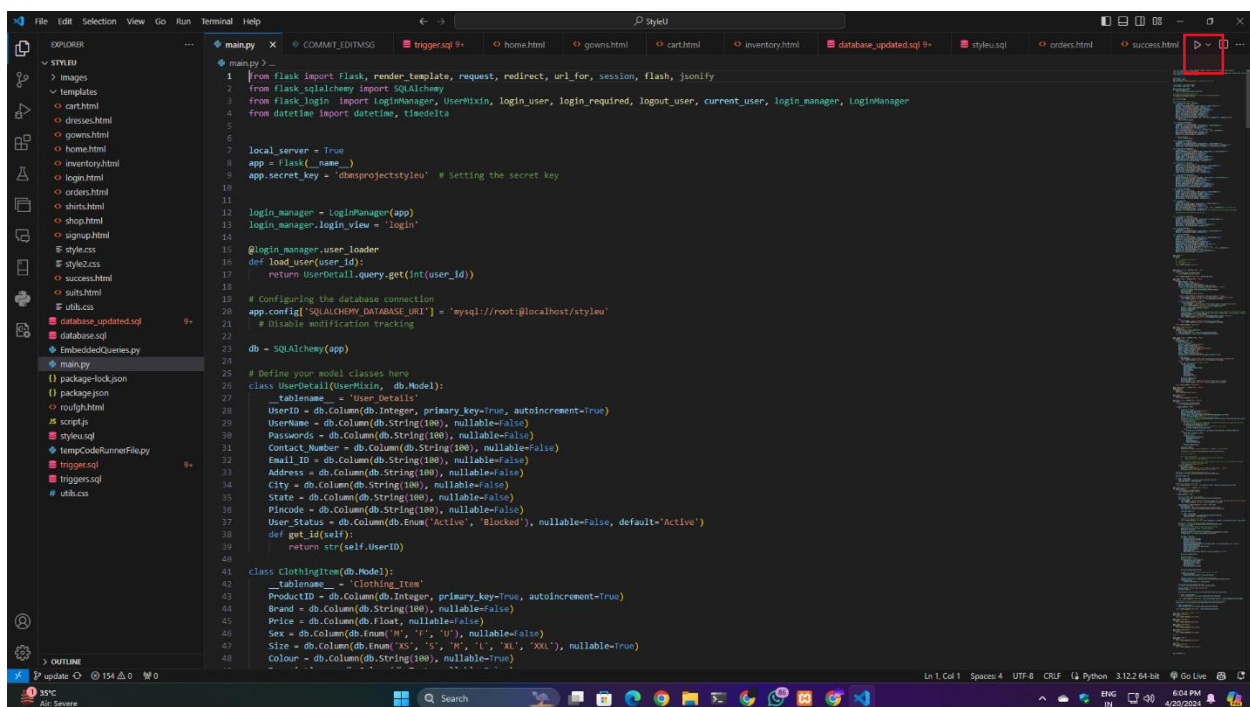
User Guide



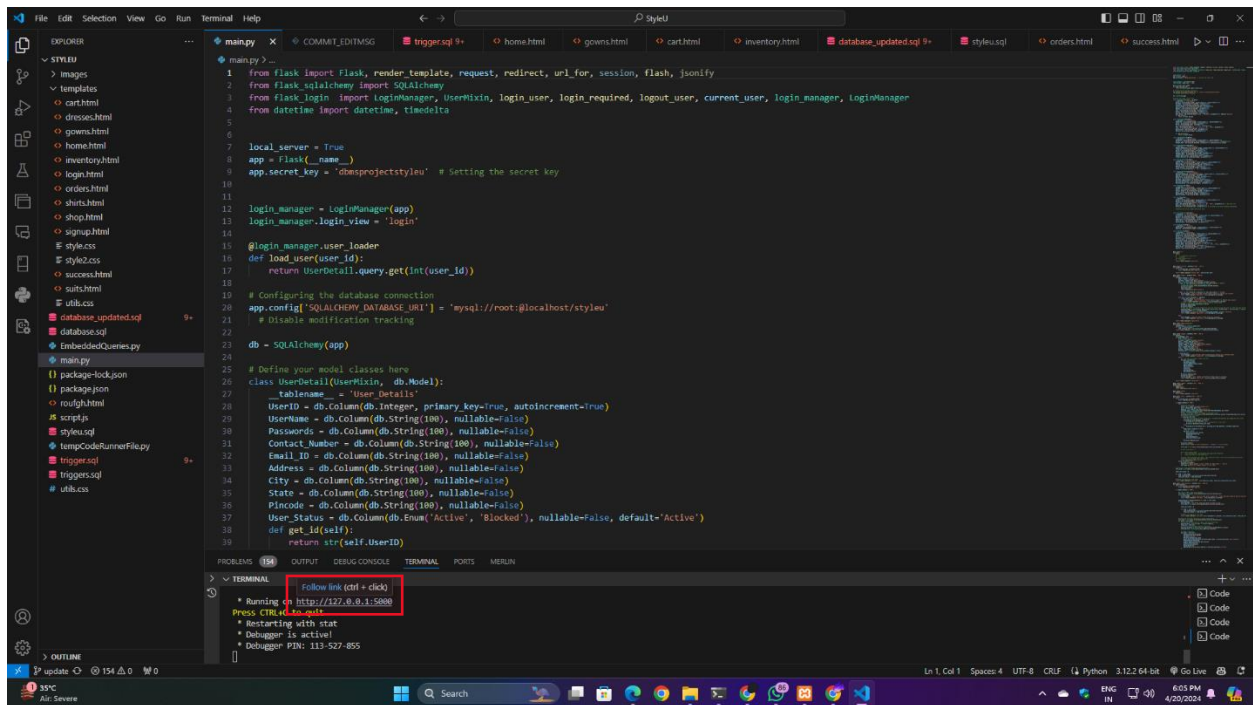
Start up xamp and click on admin to host local php server



Copy and paste the SQL queries on the locally hosted server

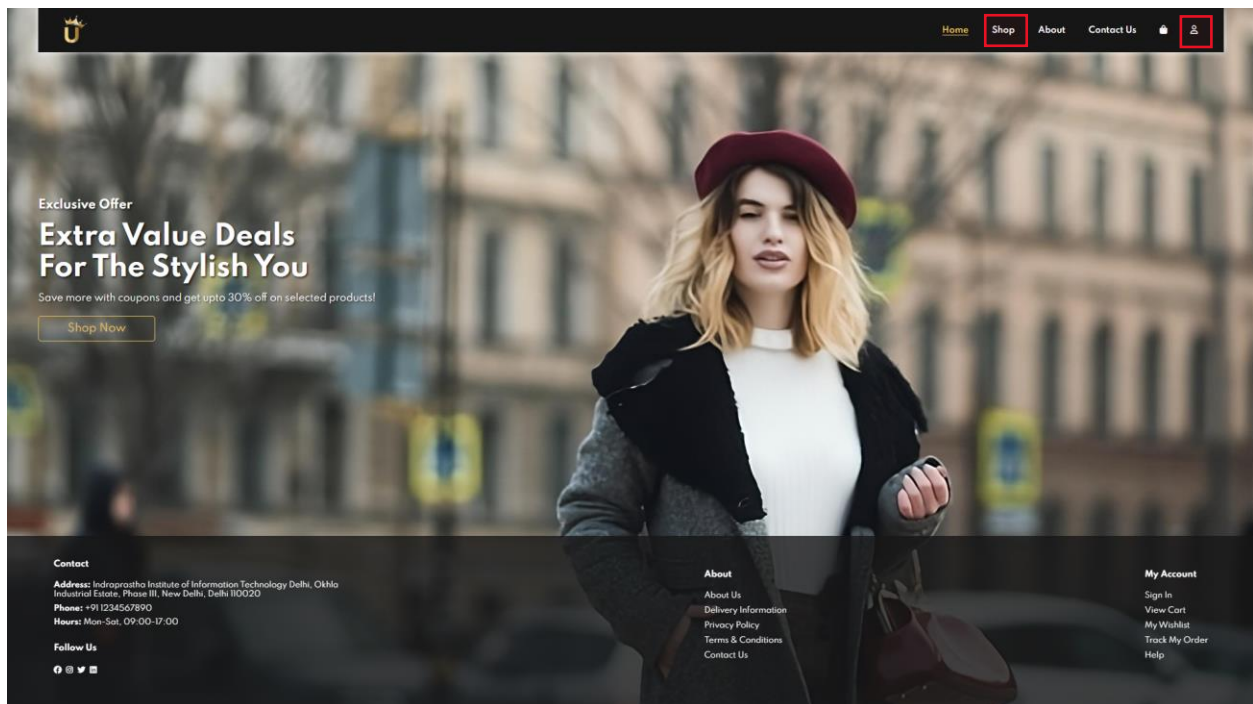


Hit run to start the Program

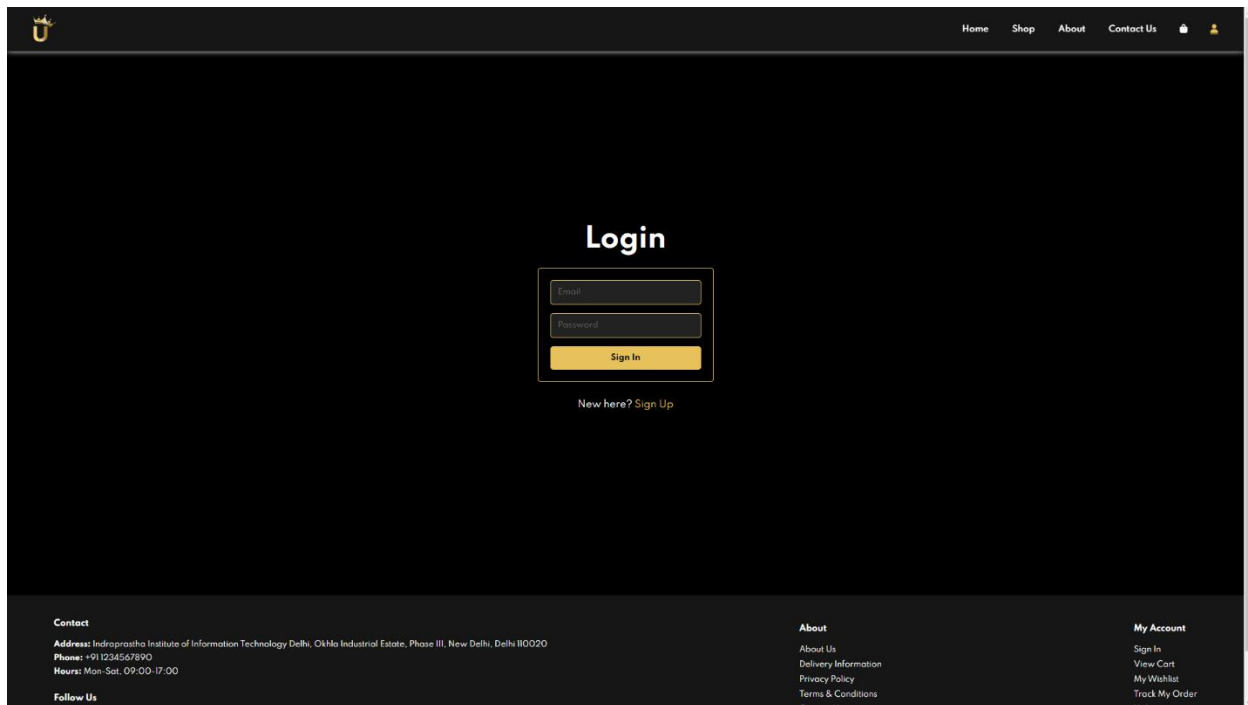


Follow the link to the front-end website

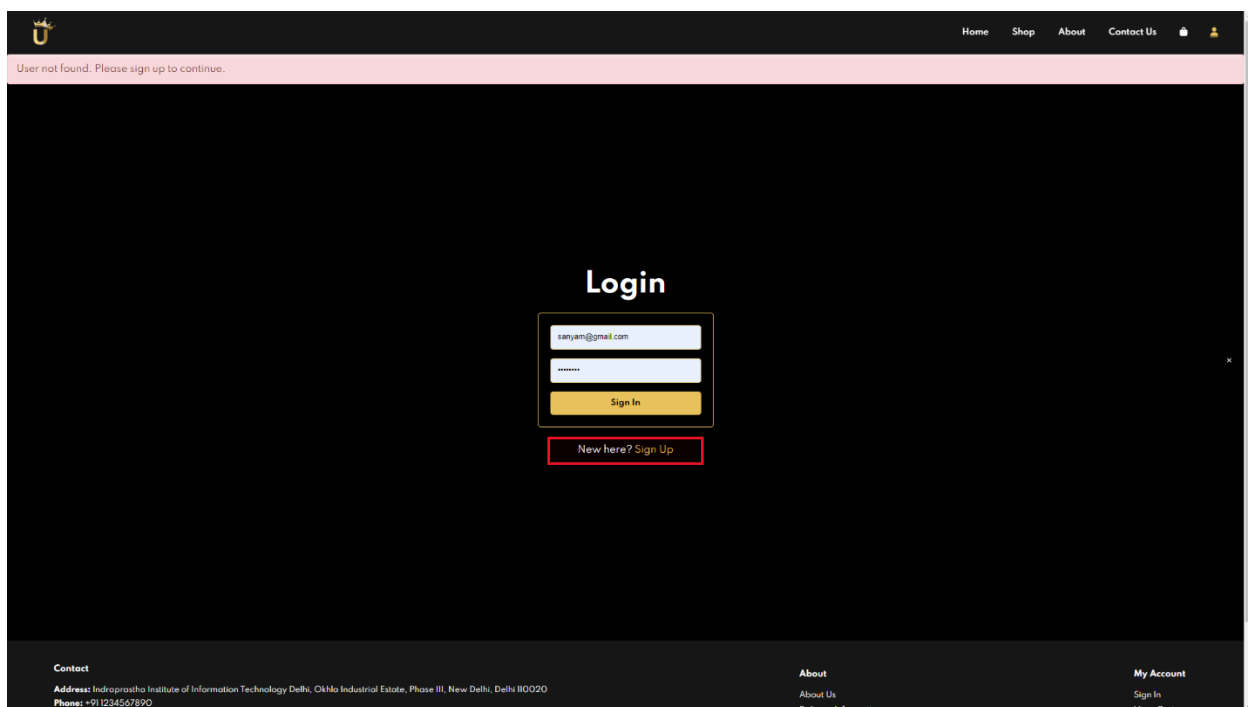
Following the link opens the home page of the website



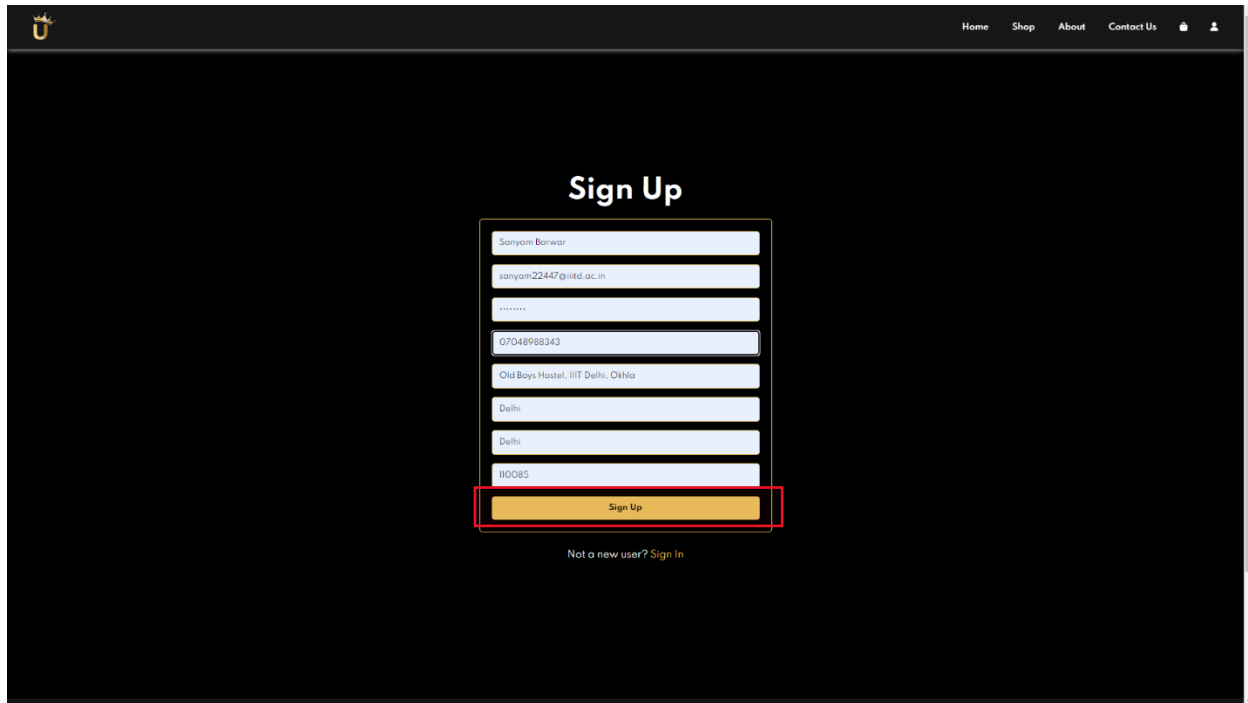
User can see products by clicking on **Shop** without logging in but, to add to cart, they need to log in first



User needs to be registered to Sign In.



If the user does not have an account they need to Sign Up.



The screenshot shows a 'Sign Up' form on a dark background. The form is a vertical stack of input fields with a yellow 'Sign Up' button at the bottom. The fields are pre-filled with the following information: Name: Sanjam Banerjee, Email: sanjam22447@iitd.ac.in, Password: (masked with dots), Phone: 07048988343, Address: Old Boys Hostel, IIT Delhi, Okhla, City: Delhi, State: Delhi, and Pin: 110085. A red rectangle highlights the 'Sign Up' button. Below the form, there is a link: 'Not a new user? Sign In'.

Sign Up

Sanjam Banerjee

sanjam22447@iitd.ac.in

.....

07048988343

Old Boys Hostel, IIT Delhi, Okhla

Delhi

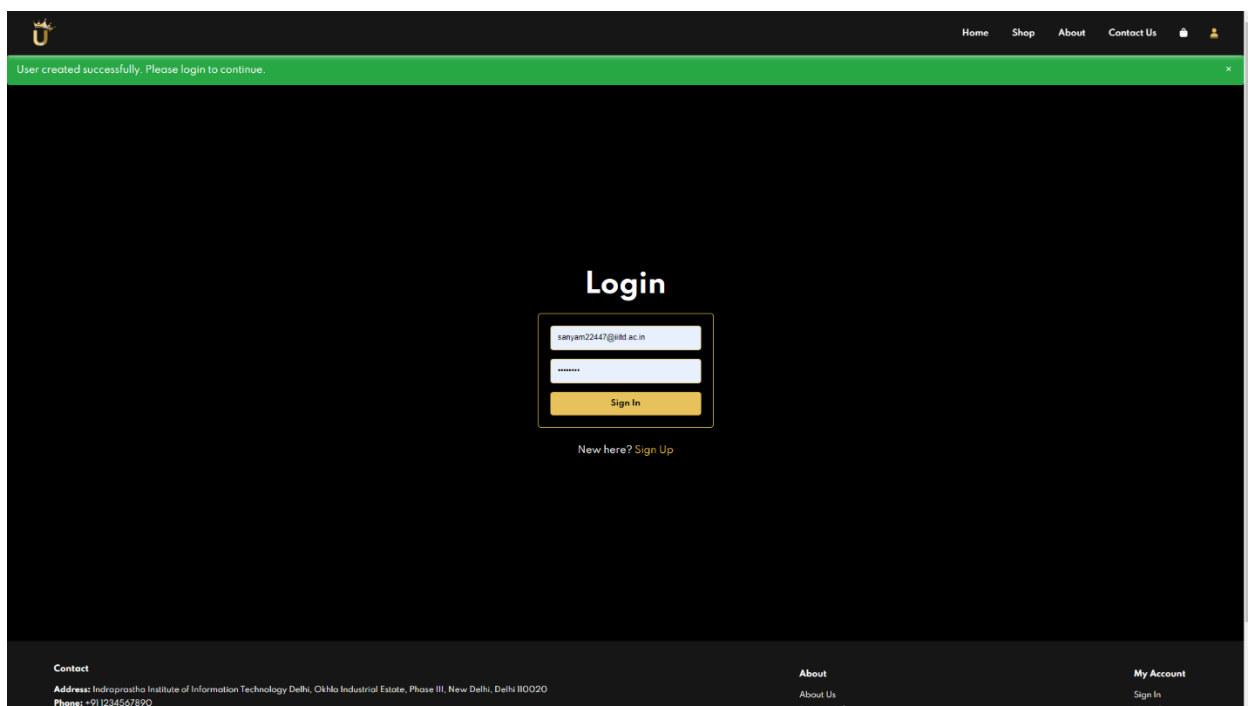
Delhi

110085

Sign Up

Not a new user? [Sign In](#)

- User cannot sign up using an already existing email. It prompts the user if the email is already registered.
- If the user is already registered, they can go to the login page using the link below the form.
- User needs to fill all the required information to sign up.
- After all the required information is filled, the **Sign Up** button redirects the user to the login page where the user can Sign In using the credentials they filled.



The screenshot shows a 'Login' page on a dark background. At the top, a green banner displays the message: 'User created successfully. Please login to continue.' Below this, the 'Login' form is centered, featuring two input fields (Email and Password) and a yellow 'Sign In' button. The email field is pre-filled with 'sanjam22447@iitd.ac.in'. Below the form is a link: 'New here? Sign Up'.

Login

sanjam22447@iitd.ac.in

.....

Sign In

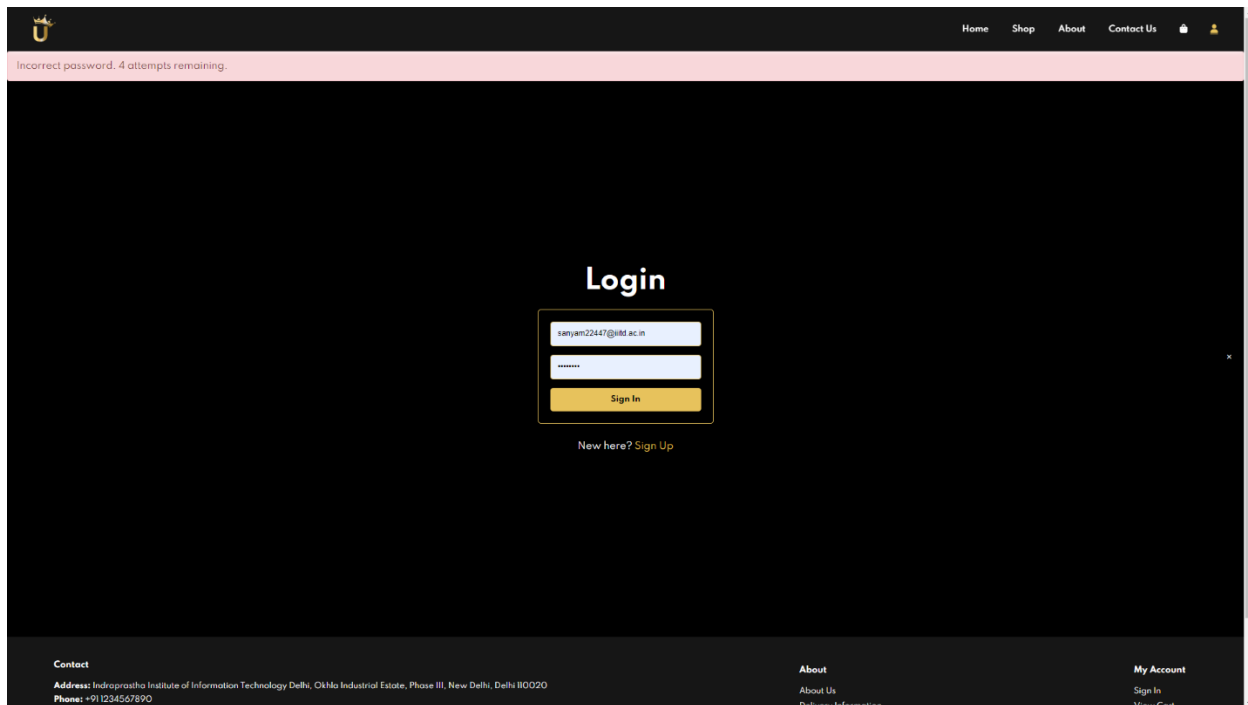
New here? [Sign Up](#)

Message: User created successfully. Please login to continue.

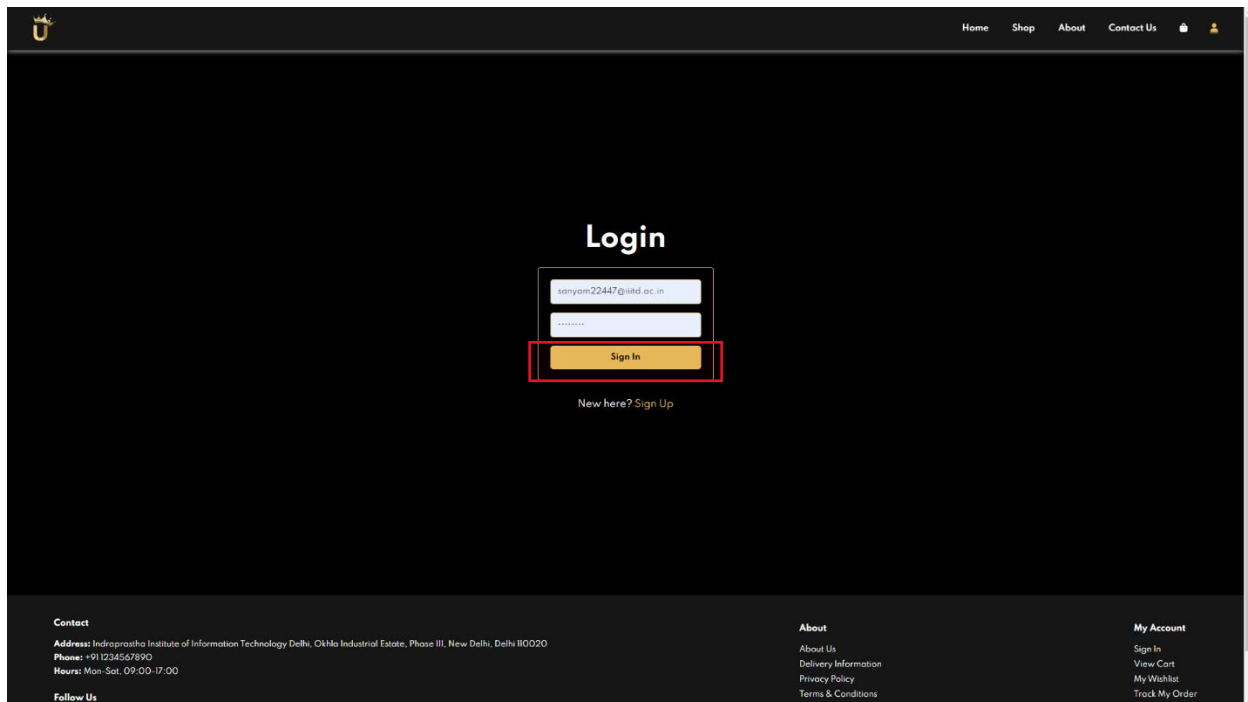
Contact
Address: Indraprastha Institute of Information Technology Delhi, Okhla Industrial Estate, Phase III, New Delhi, Delhi 110020
Phone: +91 234567890

About
[About Us](#)
[Delivery Information](#)

My Account
[Sign In](#)
[View Cart](#)

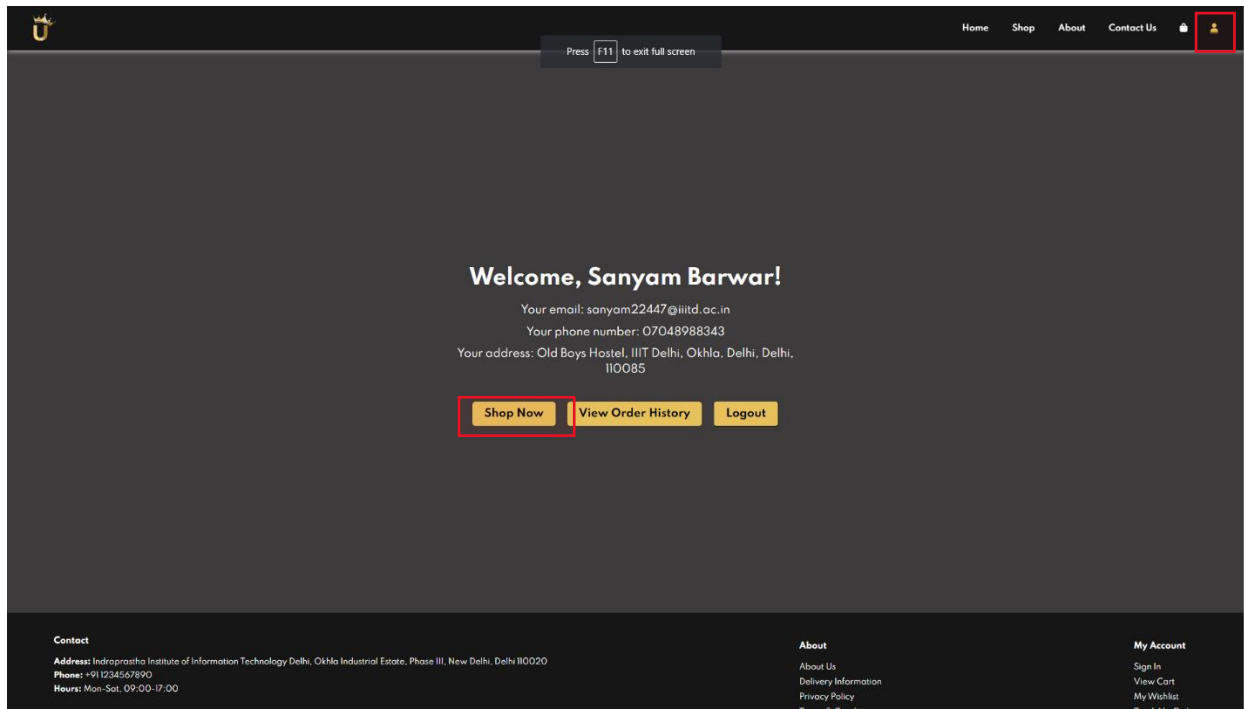


- User needs to sign in using the correct credentials.
- User only gets 5 attempt to enter the correct password, else their account will be blocked. This is a feature included so that someone without the user's knowledge does not sign in using their credentials.
- The attempts reset once a successful sign in is made or a day has passed.

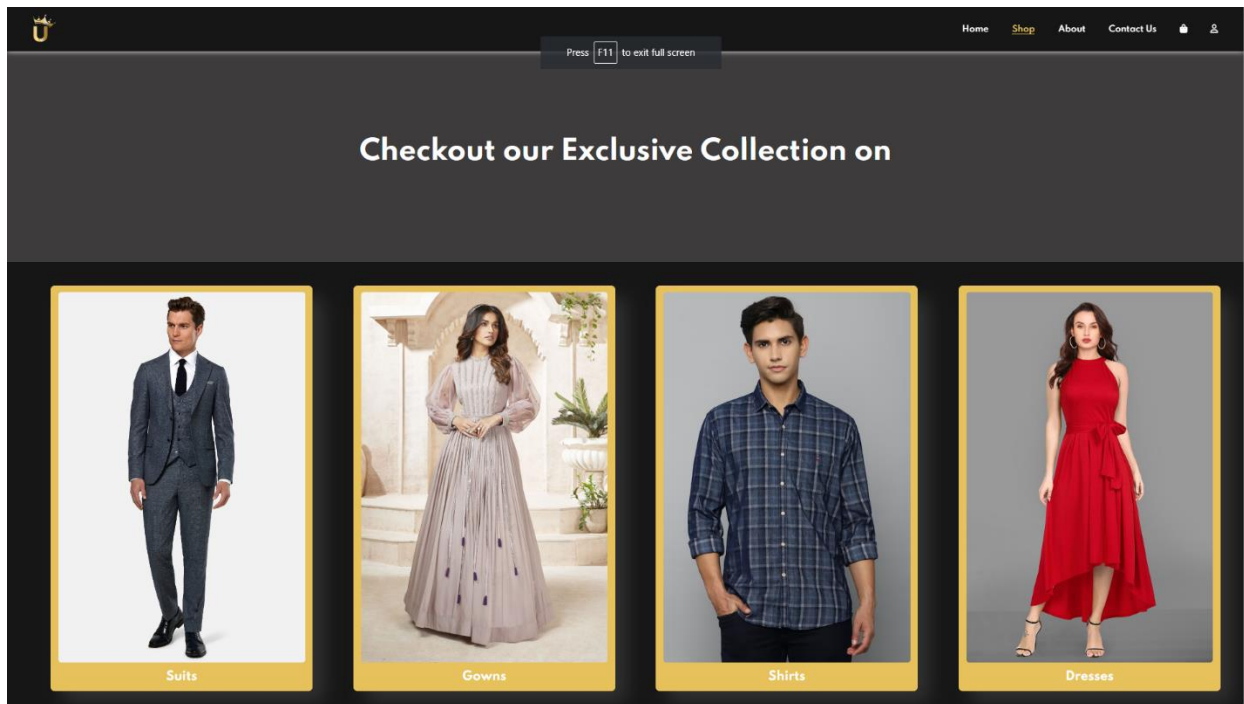


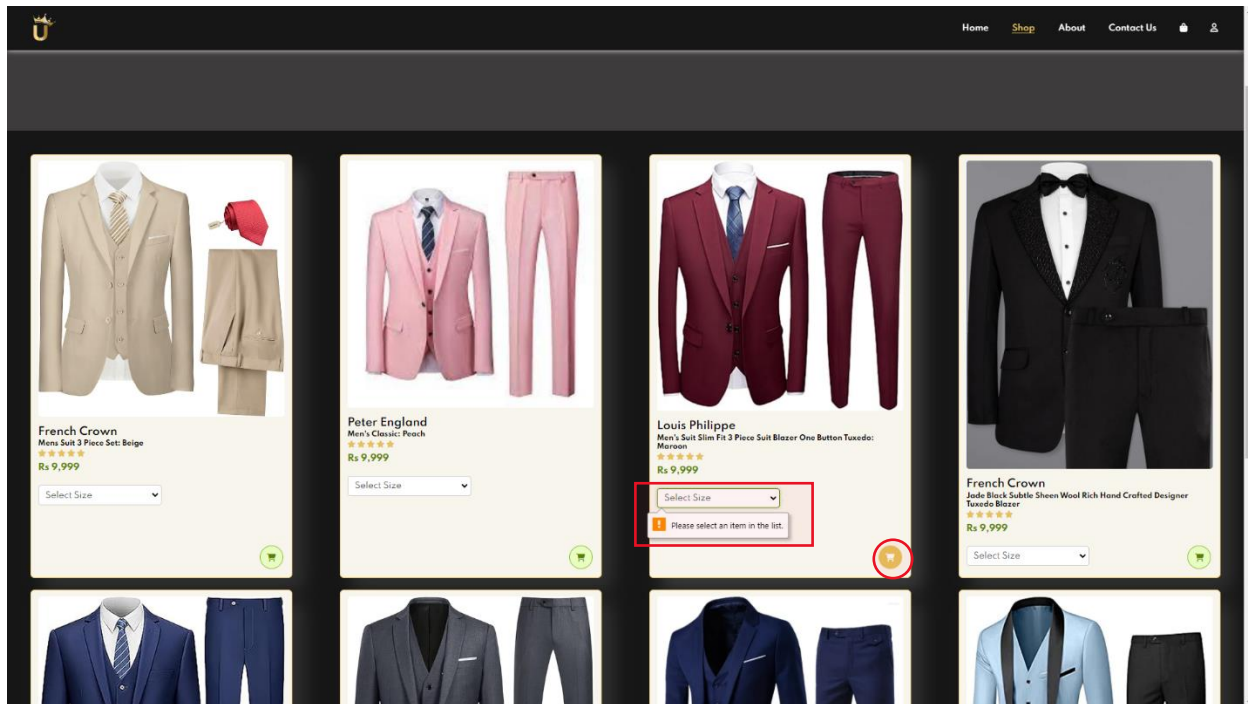
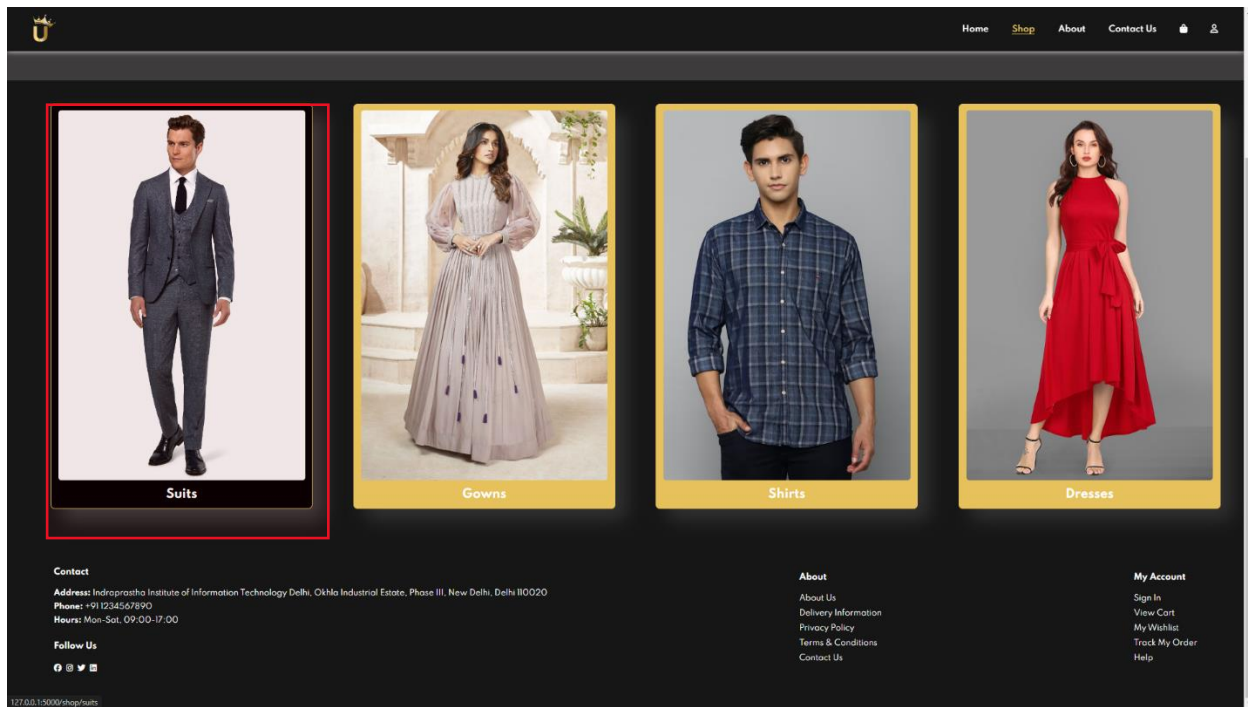
A successful sign in leads to Profile page.

- If the user is authenticated, the profile icon on the top also opens this page.
- The user can choose to shop, view order history, or log out.

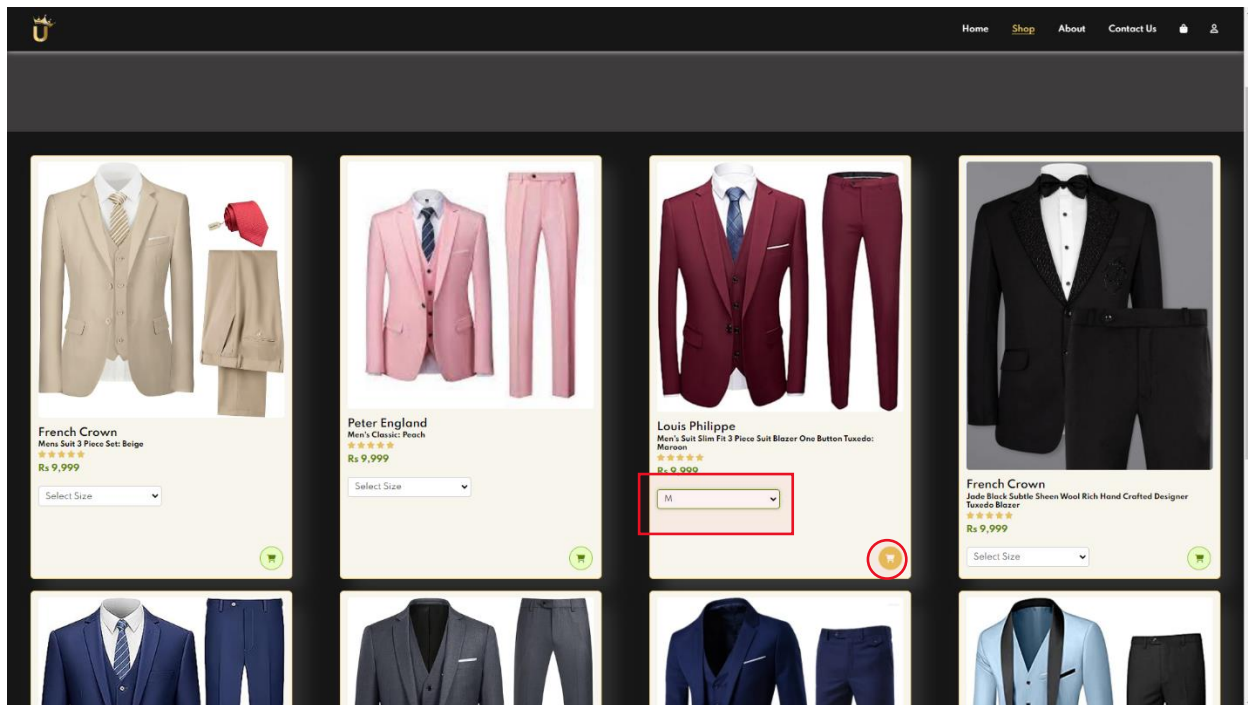


Clicking on the shop button opens up the shop page where the user can shop for the products based on their desired category.

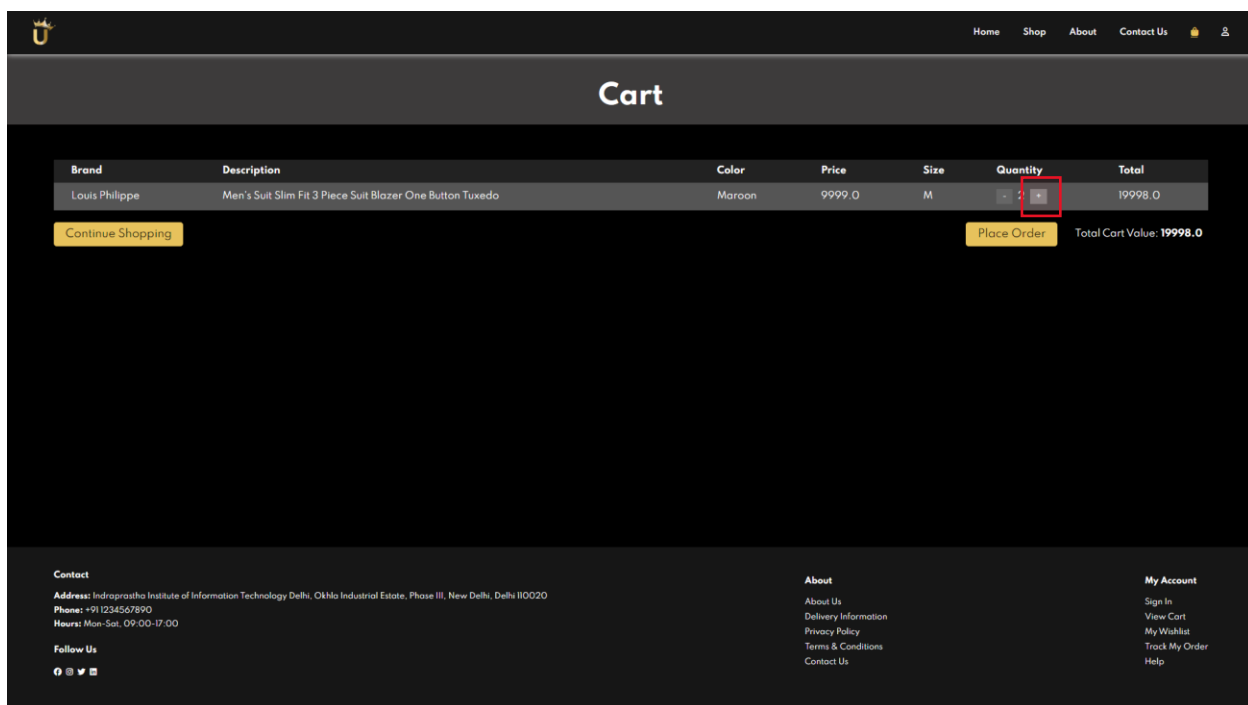




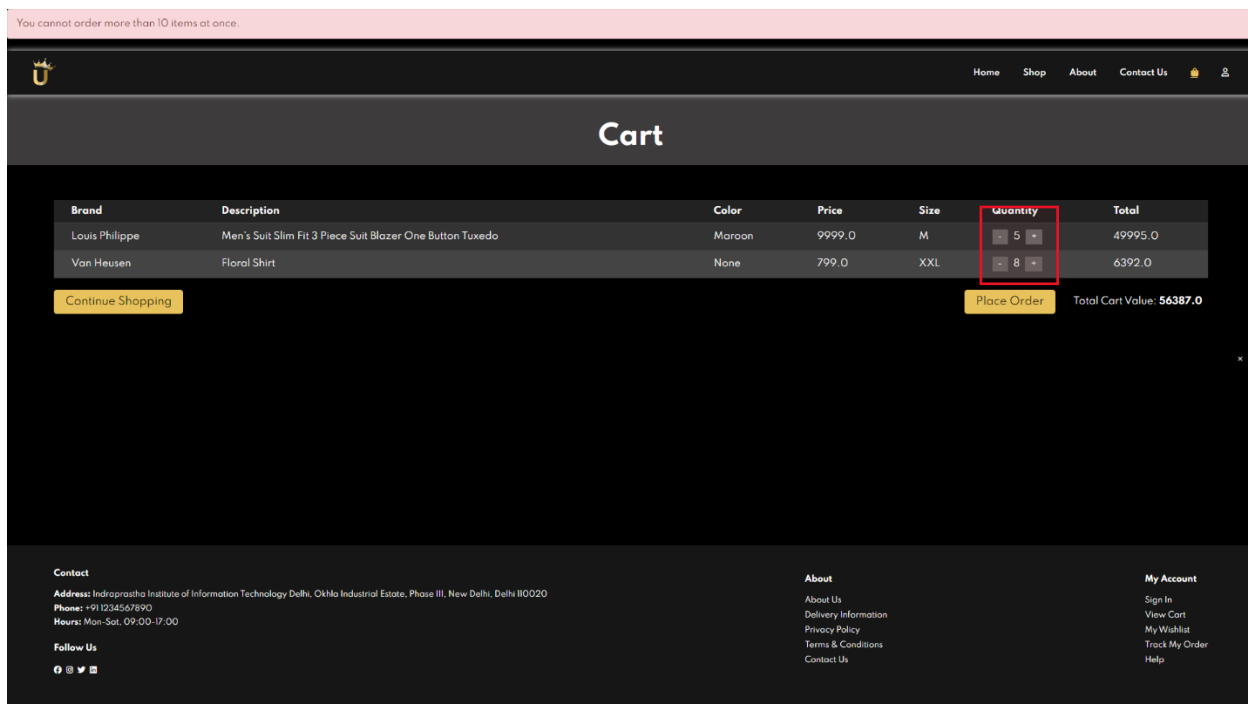
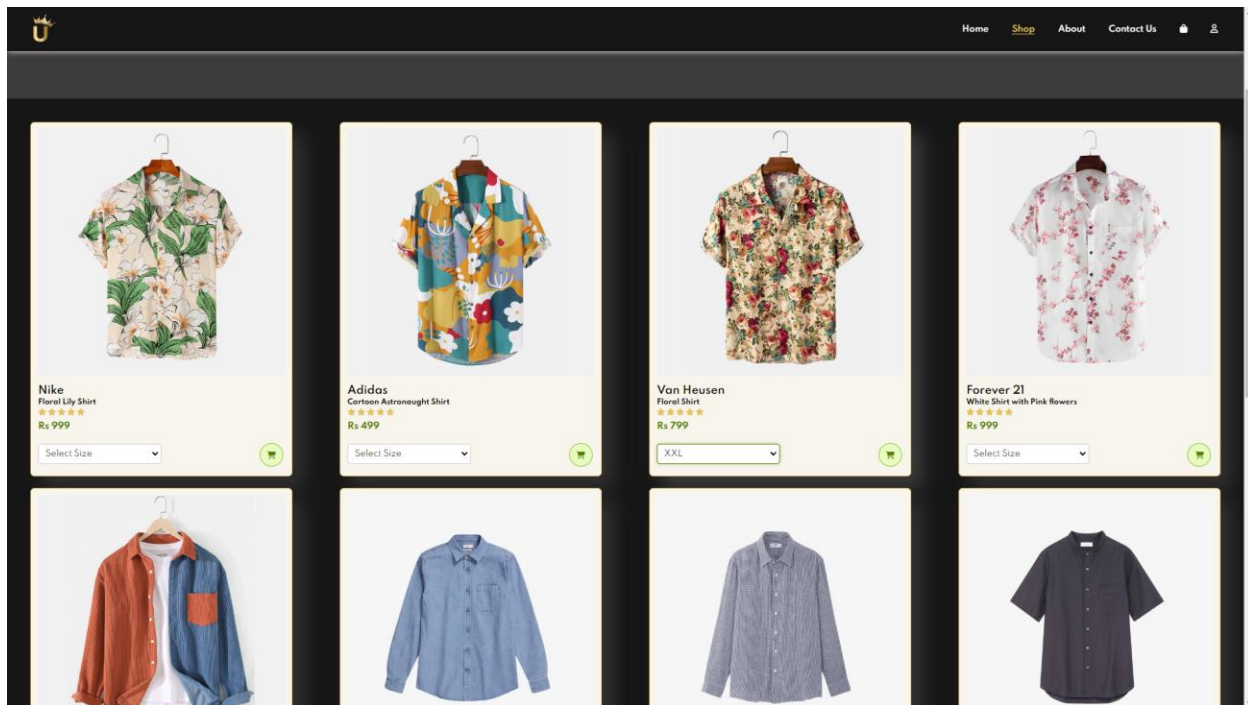
User needs select size before adding the product to cart



After the size is selected by the user, clicking on the cart icon/button leads them to cart page.



- The user can increase or decrease the quantity of the products they want to buy.
- User can choose to continue shopping or place order with the current items in the cart.



- The user cannot place an order with total quantity items of over 10 in the cart.
- An order cannot be placed with an empty cart.
- In both the cases the user is prompted with an error message.

Backend:-

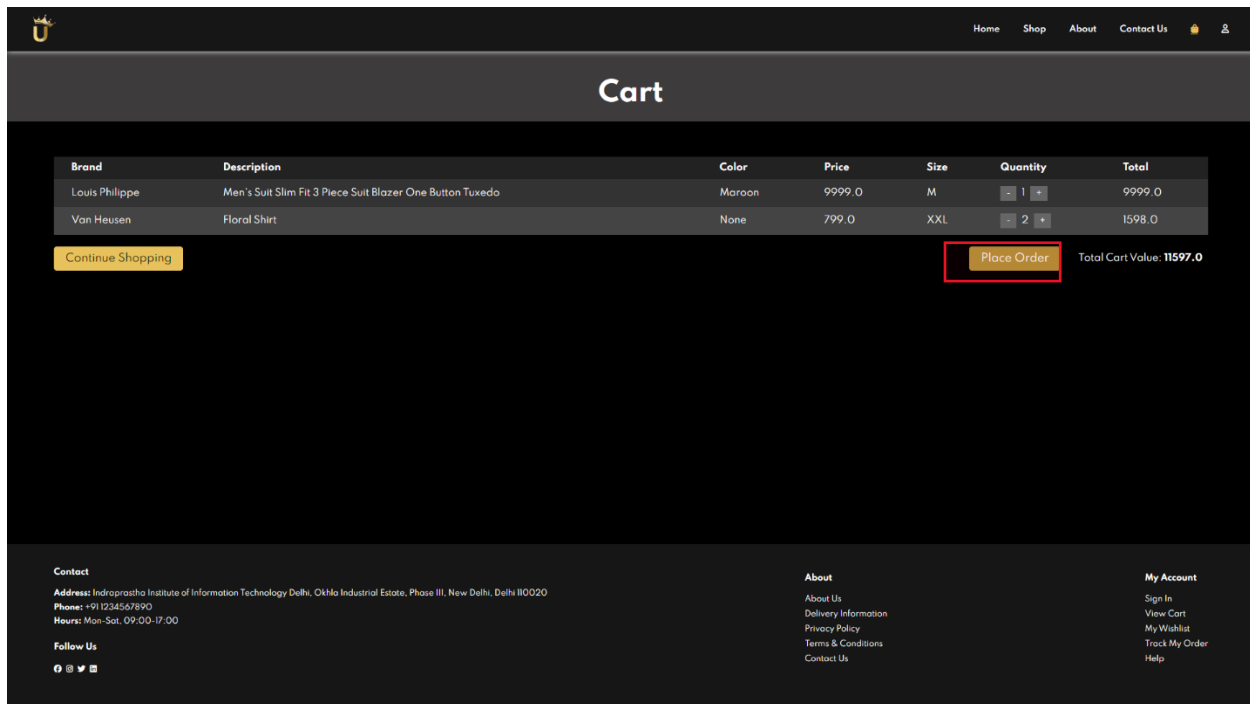
If a POST request is made:

- Retrieves form data such as product_id, size, and quantity.
- Fetches the ClothingItem details based on the product_id.
- Checks if the product is already in the user's cart (Cart table).
- If the product is in the cart, increments the quantity and updates the total value.
- If not, it creates a new entry in the cart with the appropriate details.
- Commits changes to the database and displays a success message.
- Redirects back to the cart page.

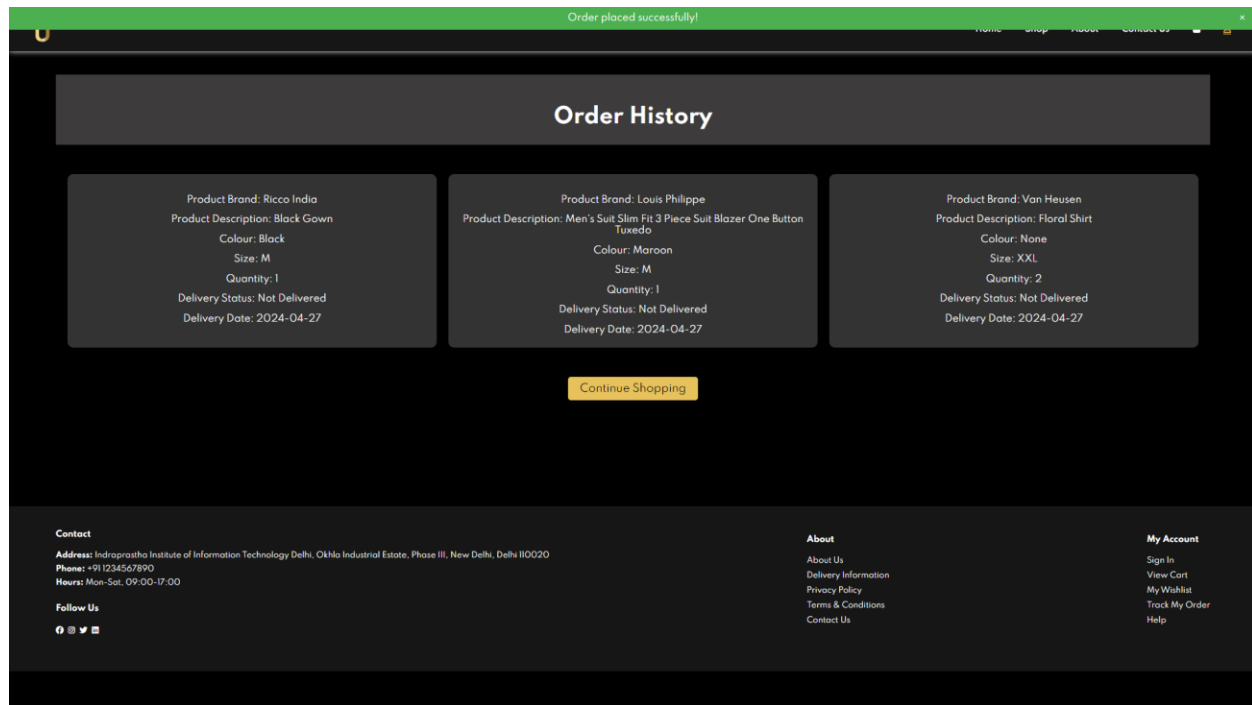
If it's a GET request:

- Fetches all items in the user's cart.
- Calculates the total cart value.
- Renders the cart.html template with the cart items and total cart value.

Note:- The POST request is made when the an item is added to cart from shop page, and GET request is made when the cart is accessed using the cart icon.



- After the user has finalized the products and quantity they want to buy, they can place an order.
- If an order is placed successfully, the cart is emptied.



Backend:-

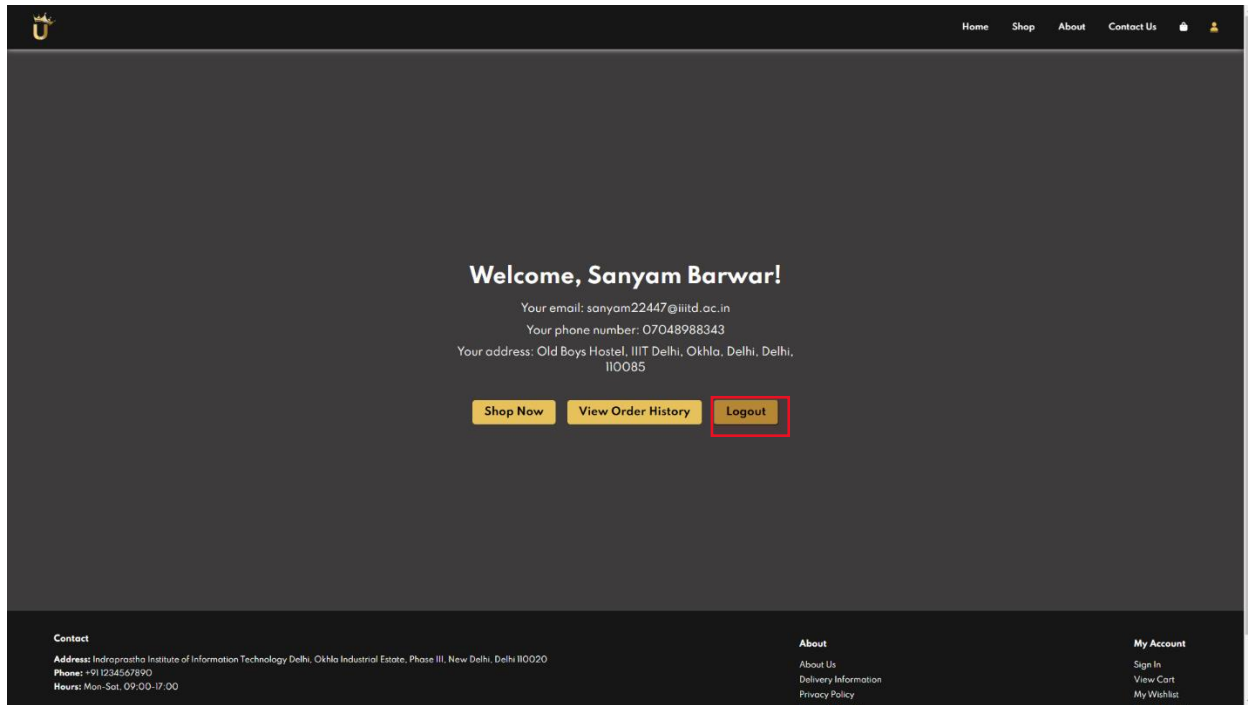
If a POST request is made (meaning a form was submitted):

- Checks if the cart is empty. If so, it shows an error message.
- Checks if the total quantity of items in the cart is more than 10. If so, it shows an error message.
- If everything is valid, it proceeds to:
 - Calculates the delivery date as 7 days from the current date.
 - Finds an available delivery personnel.
 - Creates a new order in the `OrderUser` table with details such as order value, user ID, product details, delivery date, etc.
 - Adds a record to the `PurchaseHistory` table for the user's purchase history.
 - Updates the order ID for the assigned delivery personnel.
 - Reduces the quantity of the ordered product from the inventory.
 - Clears the user's cart after placing the order.
 - Displays a success message.
 - Fetches and displays the updated order history for the user.

If it's a GET request:

- Fetches the user's order history from the `OrderUser` table.
- Associates each order with its corresponding `ClothingItem` details.
- Renders the orders.html template to display the order history.

Note:- The POST request is made when the **Place Order** button is clicked on the cart page. And GET request is made when order history page is viewed through profile page.

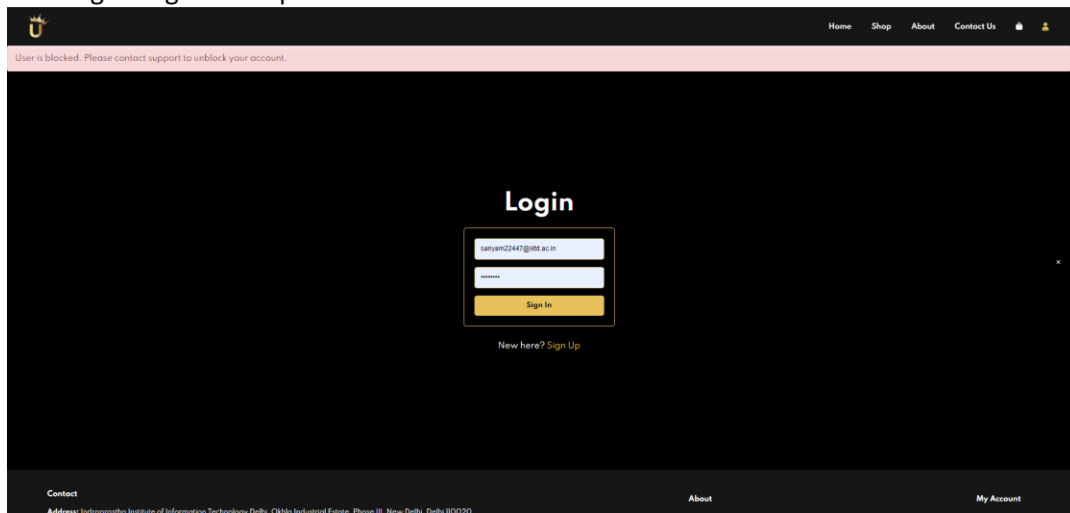


The user can logout if they have successfully ended their shopping experience.

Triggers

1. ``Block_User_After_Five_Attempts``: This trigger as it says blocks a user after five or more failed login attempts within a day. It updates the user status to 'Blocked' in the ``User_Details`` table if the conditions are met.

- Event: This trigger is activated after a new record is inserted into the ``Login_Attempts`` table.
- Count Attempts: It calculates the number of failed login attempts for a specific user within the last 24 hours.
- Check Attempts: If the count of failed attempts is equal to or exceeds five, it indicates multiple unsuccessful login tries.
- Update User Status: In such cases, the trigger updates the user's status to 'Blocked' in the ``User_Details`` table, indicating that the account is temporarily suspended due to security concerns.
- Prevent Unauthorized Access: By setting the user status to 'Blocked', the trigger prevents further login attempts from the blocked user, enhancing security.
- Implementation: The trigger is implemented using MySQL syntax, ensuring that it runs automatically after every insertion into the ``Login_Attempts`` table, thus monitoring and reacting to login attempts in real-time.



2. ``Update_Inventory_On_Low_Stock``: This trigger updates the inventory when an item is low on stock. It adds 50 quantity to the item if it meets necessary condition.

- Event: This trigger fires after a new record is inserted into the ``Order_User`` table.
- Retrieve Stock Count: It fetches the current stock count for the product associated with the newly inserted order.
- Check Stock Level: If the stock count is below 10, it indicates low stock.
- Update Inventory: In case of low stock, the trigger updates the inventory by adding 30 units to the existing quantity for the corresponding product.
- Prevent Stock Depletion: By replenishing the inventory with an additional 30 units, the trigger helps prevent stock depletion and ensures product availability.
- Implementation: Using MySQL syntax, this trigger automatically monitors the inventory levels after each order insertion, maintaining stock levels and preventing shortages.

Embedded Queries

1. Inventory:

```
@app.route('/admin/inventory')
def inventory():
    inventory_data = Inventory.query.all()
    for item in inventory_data:
        item.clothing_item = ClothingItem.query.get(item.ProductID)
    return render_template('inventory.html', inventory=inventory_data)
```

- i. `Inventory.query.all()` is a SQLAlchemy query that retrieves all rows from the `Inventory` table.
- ii. The loop iterates through each item in the `inventory_data`.
- iii. Inside the loop, `ClothingItem.query.get(item.ProductID)` retrieves the corresponding `ClothingItem` object for each `Inventory` item based on the `ProductID`.
- iv. The `clothing_item` attribute is then added to each `Inventory` item, containing the corresponding `ClothingItem` object.
- v. Finally, the `inventory_data` is passed to the `inventory.html` template for rendering.

Inventory						
Product ID	Brand	Description	Quantity	Color	Price	
1	French Crown	Men's Slim Fit 3 Piece Suit	30	Beige	9999.0	
2	Peter England	Men's Classic: Peach	30	Peach	9999.0	
3	Louis Philippe	Men's Suit Slim Fit 3 Piece Suit Blazer One Button Tuxedo	29	Maroon	9999.0	
4	French Crown	Jade Black Subtle Sheen Wool Rich Hand Crafted Designer Tuxedo Blazer	30	Black	9999.0	
5	YSMLOIU	Slim Fit 3 Piece Wedding Suit for Men	30	Blue	9999.0	
6	YouthUp	Men's 3 Piece Suit Slim Fit with Single Row Jacket Suits and Vest	30	Grey Black	9999.0	
7	unbranded	Men's Suits Leisure Suit For Men Four Seasons Business Groom Wedding Dress Formal Fashion Slim-fit Two-piece	30	Dark Blue	9999.0	
8	TOPGH	Men's 3 PCS Suit Notch Lapel Solid Wedding Work Slim Tuxedo Fit	30	Blue Black	9999.0	
9	Lhuiller	Stunning off shoulder lace wedding dress long train	30	White	80999.0	
10	Ricco India	Black Gown	20	Black	100000.0	
11	LimeRoad	Women Gorgeous Gowns Flower Printed Sky	29	Cyan	9999.0	
12	unbranded	Lace Middle Sleeves V-neck Lace-up Floor Length Ball Gown Dress	30	Sky Blue	9999.0	
13	unbranded	Pastel Green Net High Notted Gown	30	Pastel Green	9999.0	
14	Nike	Floral Lily Shirt	30	None	999.0	
15	Adidas	Cartoon Astronaut Shirt	30	None	499.0	
16	Van Heusen	Floral Shirt	27	None	799.0	
17	Forever 21	White Shirt with Pink flowers	30	White	999.0	
18	Jian Jing	Orange Blue Shirt	30	Orange Blue	1599.0	
19	H&M	Denim Full Sleeves Shirt	30	Blue	1999.0	
20	GAP	Full sleeves shirt with check pattern	30	Grey	999.0	
21	Khadi	Simple Shirt	30	Greyish Black	999.0	
22	Zara	Black T-shirt	30	Black	2599.0	
23	H&M	Black Dress	30	Black	1999.0	

2. Cart:

```
257 @app.route('/cart', methods=['GET', 'POST'])
258 def cart():
259     if not current_user.is_authenticated:
260         return redirect(url_for('login'))
261
262     if request.method == 'POST':
263         try:
264             # Retrieve form data
265             product_id = request.form.get('product_id')
266             size = request.form.get('size')
267             quantity = int(request.form.get('quantity'))
268             clothing_item = ClothingItem.query.filter_by(ProductID=product_id).first()
269             price = clothing_item.get_price()
270             # Check if the same product is already in the user's cart
271             existing_cart_item = Cart.query.filter_by(UserID=current_user.UserID, ProductID=product_id).first()
272
273             if existing_cart_item:
274                 # If the same product is found in the cart, increment the quantity and update the cart value
275                 existing_cart_item.Quantity += quantity
276                 if (existing_cart_item.Quantity < 1):
277                     # delete the item from the cart if the quantity is less than 1
278                     db.session.delete(existing_cart_item)
279                 else:
280                     existing_cart_item.Value_Cart = existing_cart_item.Quantity * clothing_item.Price
281             else:
282                 total_value = quantity * price
283                 # Add item to cart
284                 new_cart = Cart(
285                     UserID=current_user.UserID,
286                     ProductID=product_id,
287                     Size=size,
288                     Quantity=quantity,
289                     Value_Cart=total_value
290                 )
291                 db.session.add(new_cart)
292
293             db.session.commit()
294             flash('Product added to cart successfully!', 'success') # Flash message
295             cart_items = Cart.query.filter_by(UserID=current_user.UserID).all()
296             return redirect(url_for('cart'))
297         except Exception as e:
298             db.session.rollback()
299             flash('Error adding product to cart. Please try again later.', 'error')
300             app.logger.error(f'Error adding product to cart: {e}')
301
302     # Fetch cart items associated with the current user
303     cart_items = Cart.query.filter_by(UserID=current_user.UserID).all()
304
305     total_cart_value = 0
306
307     for item in cart_items:
308         item.clothing_item = ClothingItem.query.get(item.ProductID)
309         total_cart_value += item.Value_Cart
310
311     # Render cart page with cart items
312     return render_template('cart.html', cart_items=cart_items, total_cart_value=total_cart_value)
```

- a. SQL Querying: The lines `ClothingItem.query.filter_by(ProductID=product_id).first()` and `Cart.query.filter_by(UserID=current_user.UserID).all()` execute SQL queries directly within the Python code to fetch data from the database. These queries are equivalent to SQL statements like `SELECT * FROM Clothing_Item WHERE ProductID = <product_id>;` and `SELECT * FROM Cart WHERE UserID = <current_user.UserID>;`.
- b. SQL Transactions: The code involves database transactions using SQLAlchemy's ORM methods such as `db.session.commit()` and `db.session.rollback()`, which correspond to SQL `COMMIT` and `ROLLBACK` commands.
- c. SQL Logic: The code contains logic for handling cart items, such as incrementing quantities and updating cart values, which involves interacting with the database using SQL queries.