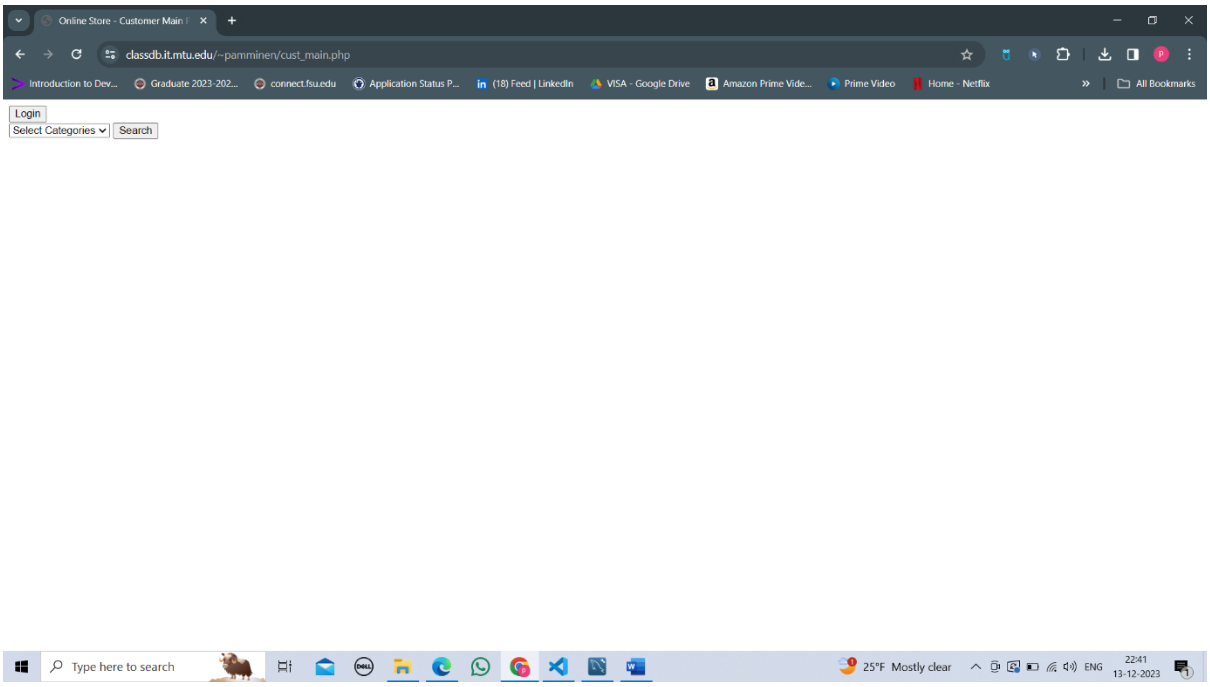
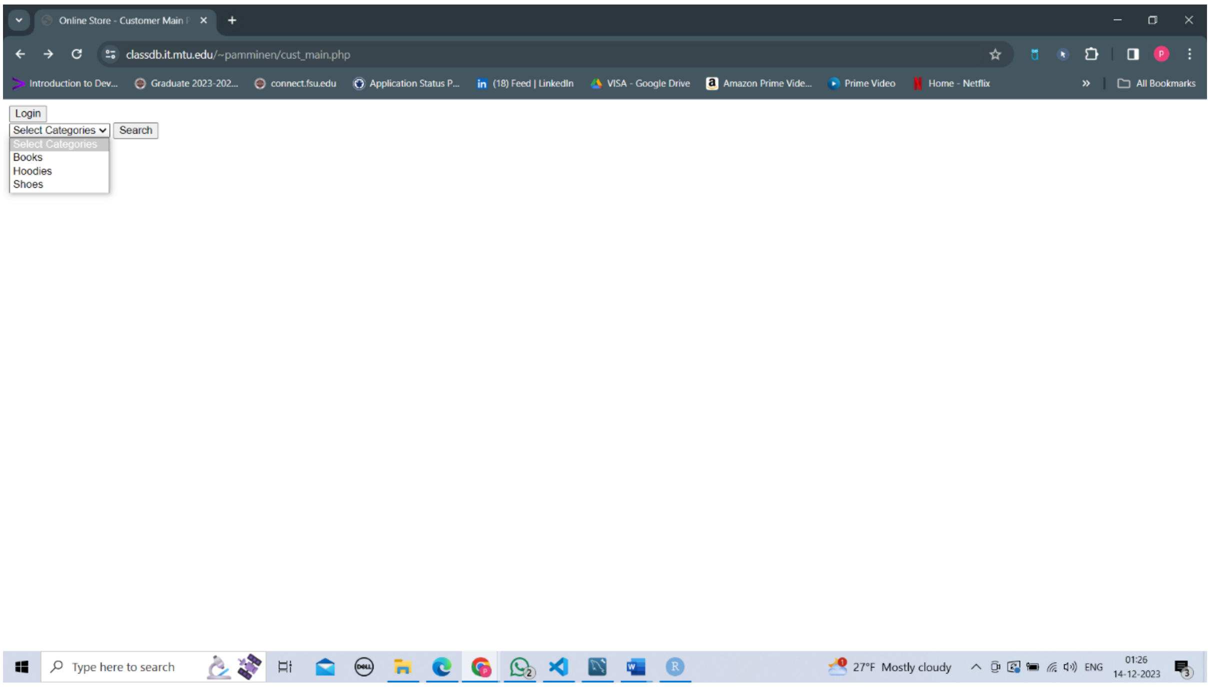
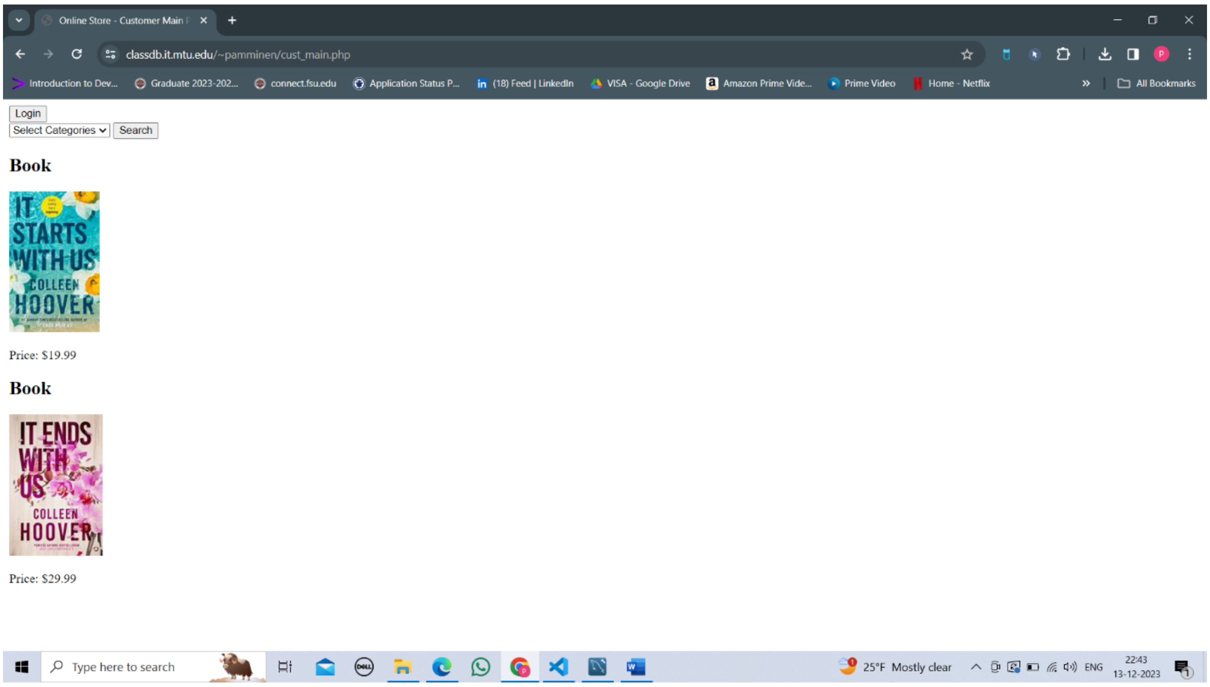
1. Customer main page before login.


Online Store - Customer Main
classdb.it.mtu.edu/~pamminen/cust_main.php
Login
Select Categories
Search


Online Store - Customer Main
classdb.it.mtu.edu/~pamminen/cust_main.php
Login
Select Categories
Search
Select Categories
Books
Hoodies
Shoes

2. Customer can choose category, then search.

I choosed book category and search and it will display 2 images and price



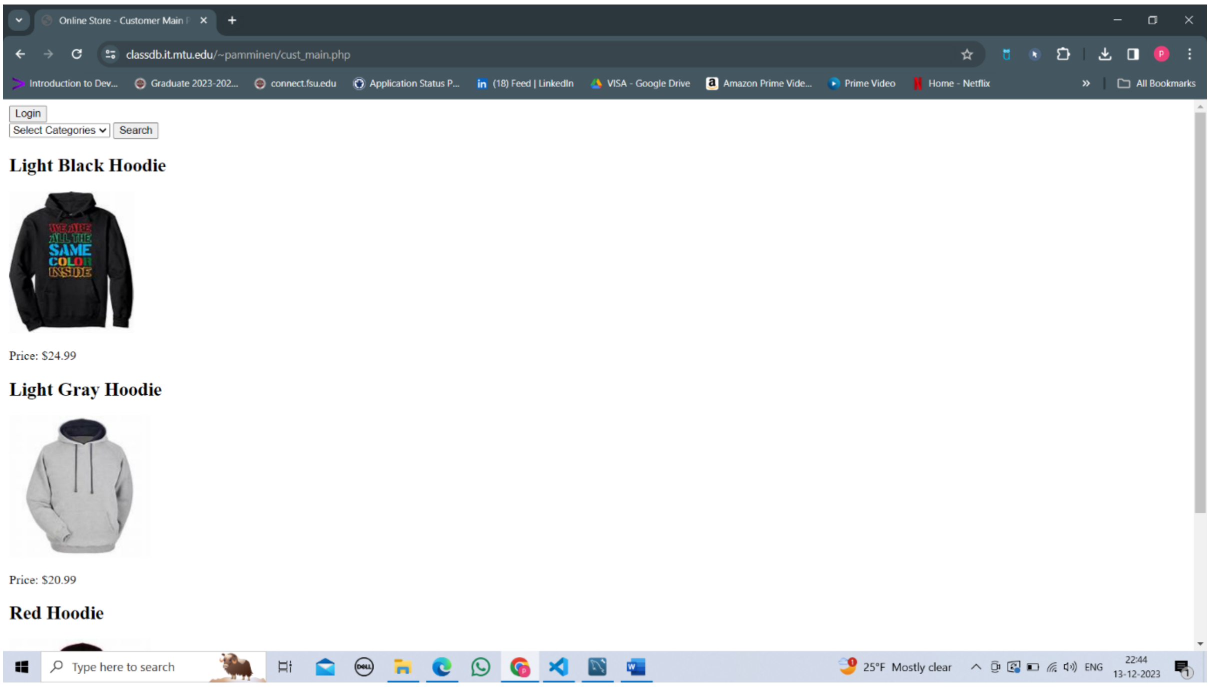
Book

Price: $19.99

Book

Price: $29.99

If I choose hoodie category then search it will display 3 images. And price


Online Store - Customer Main
classdb.it.mtu.edu/~pamminen/cust_main.php
Login
Select Categories
Search
WE ARE ALL THE SAME COLOR INSIDE
Type here to search
25°F Mostly clear
ENG
22:44
13-12-2023

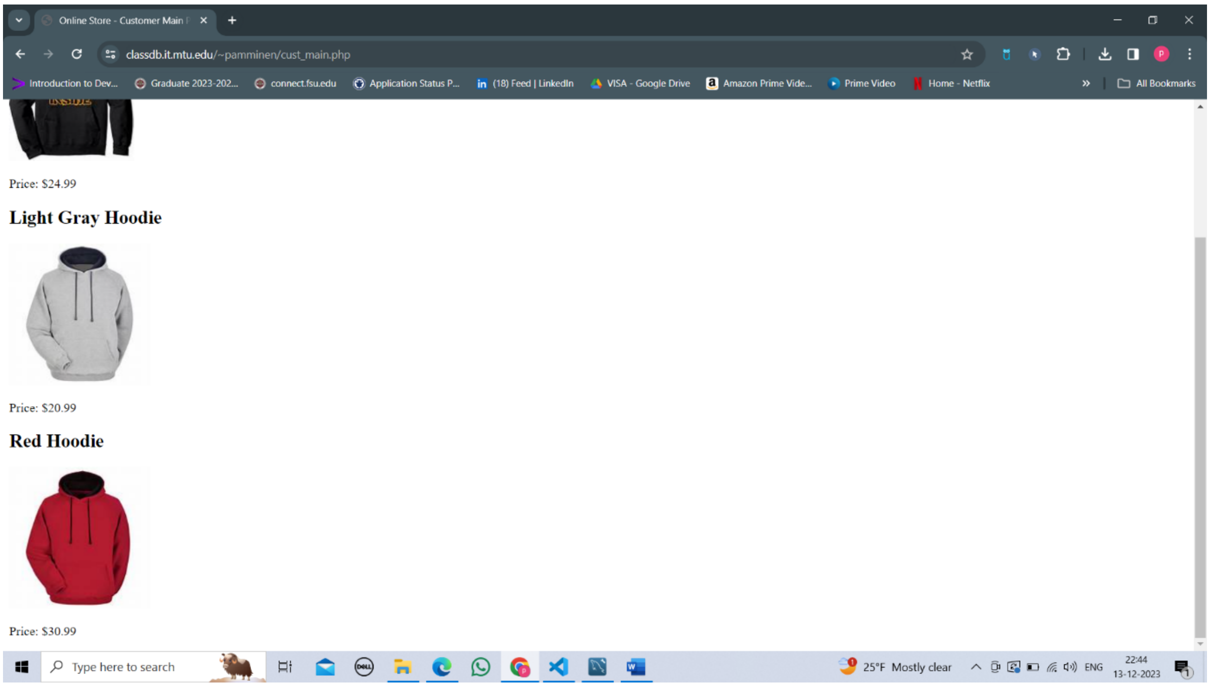
Light Black Hoodie

Price: $24.99

Light Gray Hoodie

Price: $20.99

Red Hoodie



Price: $24.99

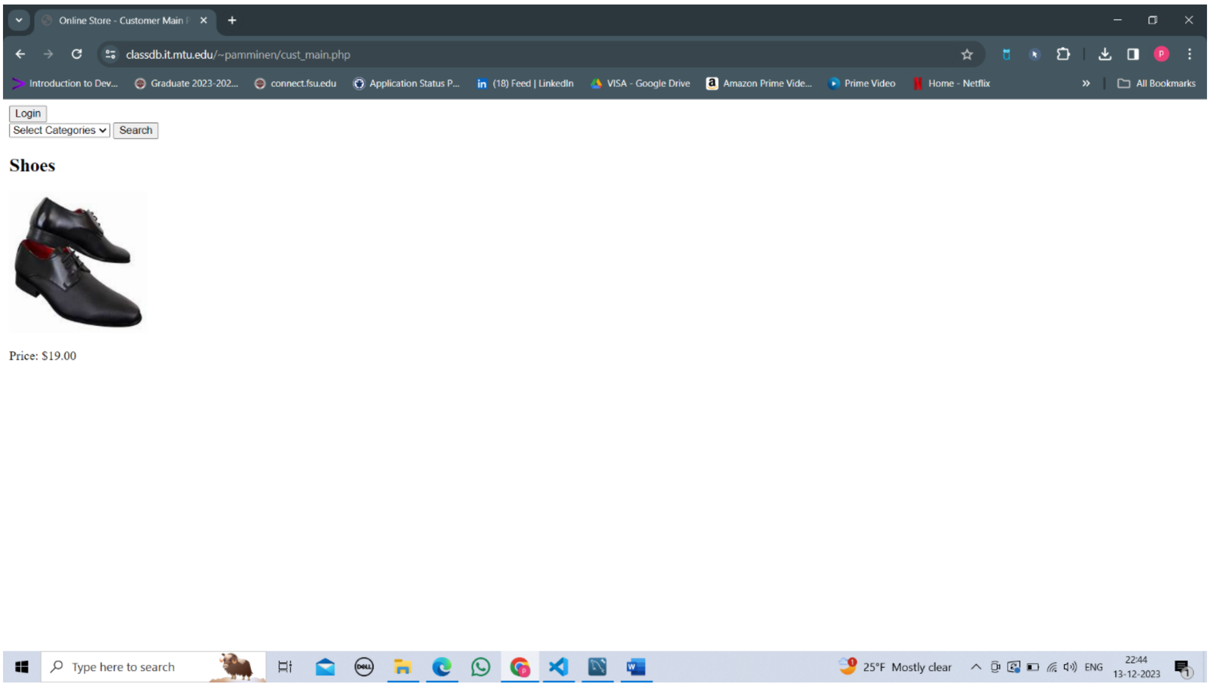
Light Gray Hoodie

Price: $20.99

Red Hoodie

Price: $30.99

If I choose shoes category then search then it will show images and price

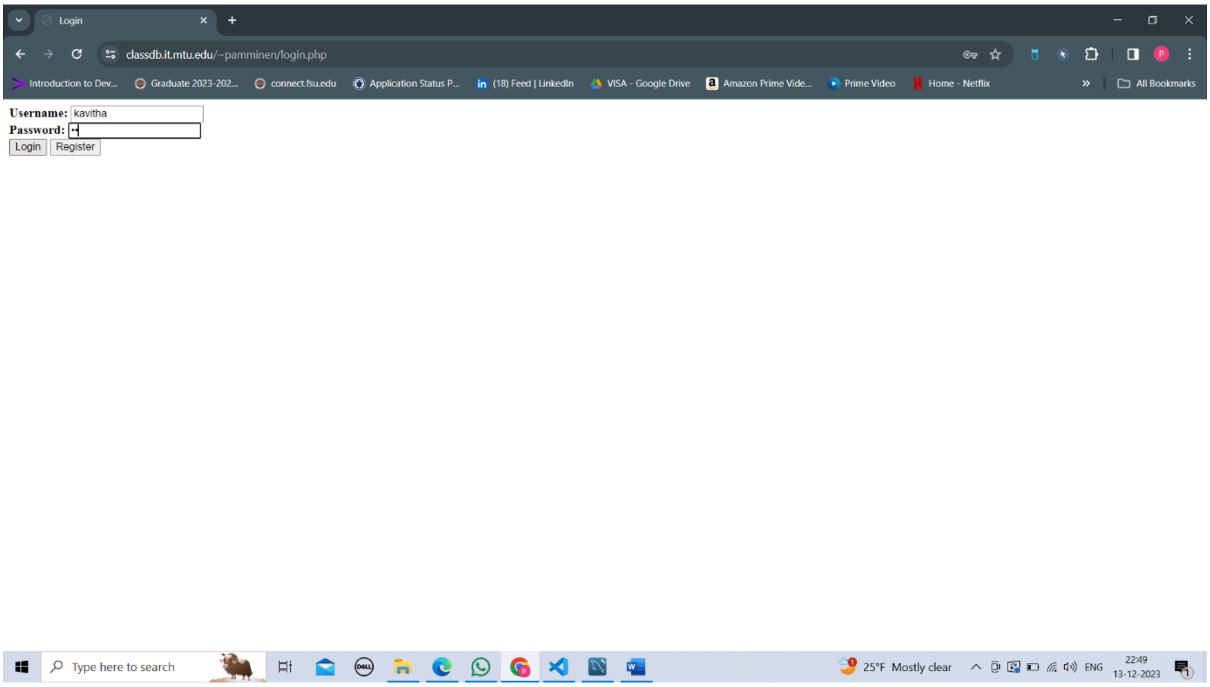


Shoes

Price: $19.00

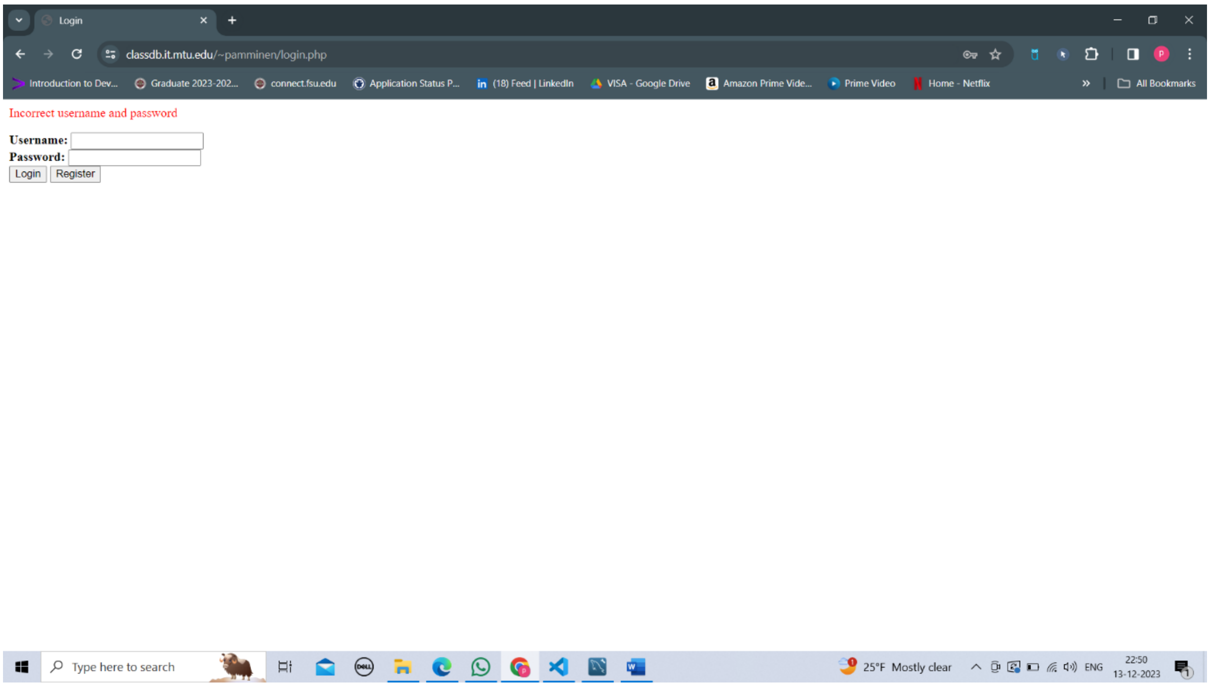
3. Customer clicked the Login button from the main page

If I click valid password



Username: kavitha
Password: ••

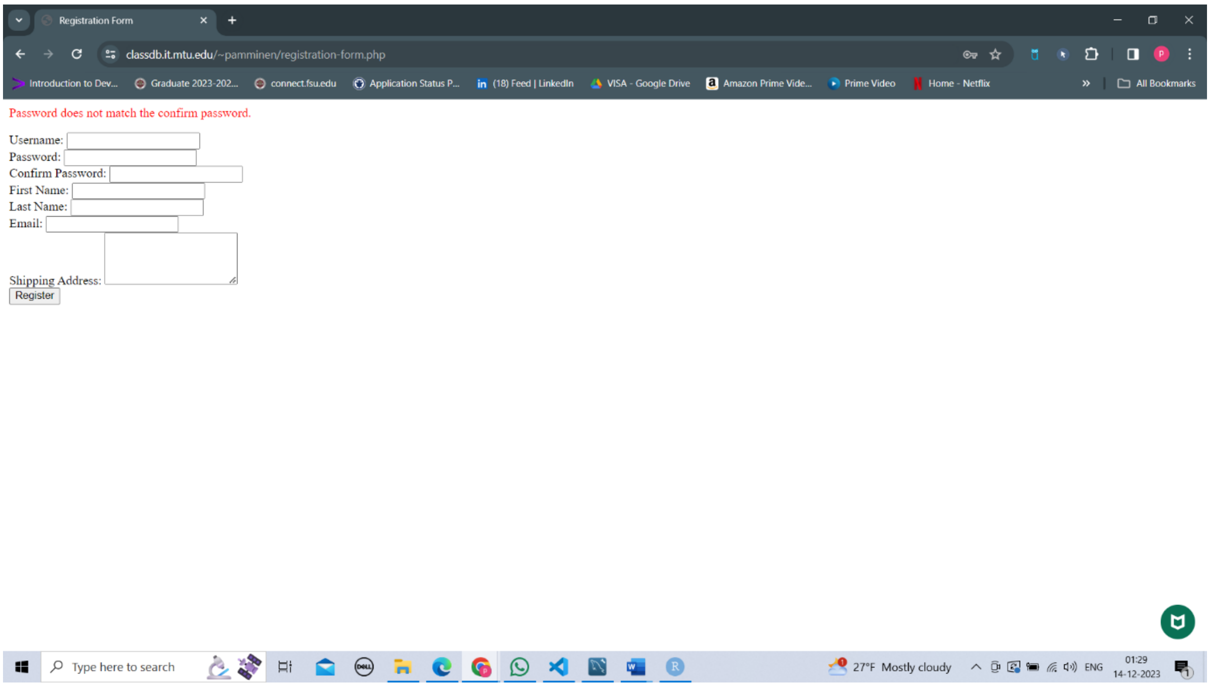
If I click invalid password



Incorrect username and password

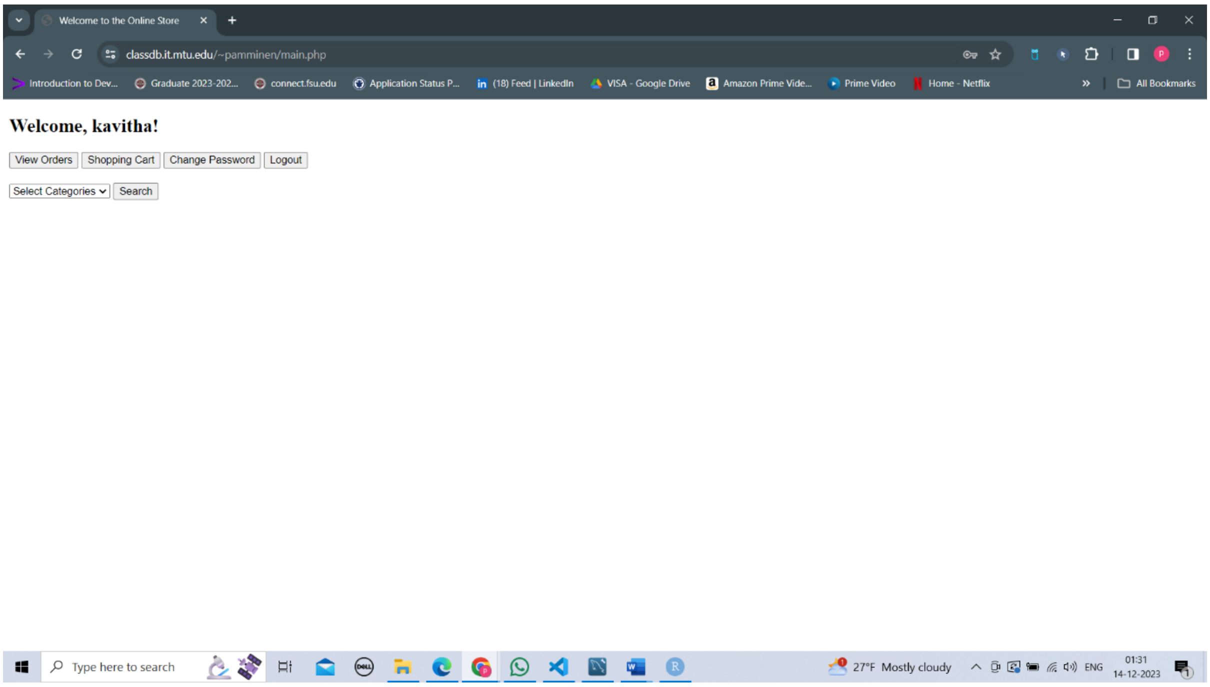
Username:
Password:

When I give mismatch password during registration



Password does not match the confirm password.

Username:
Password:
Confirm Password:
First Name:
Last Name:
Email:
Shipping Address:

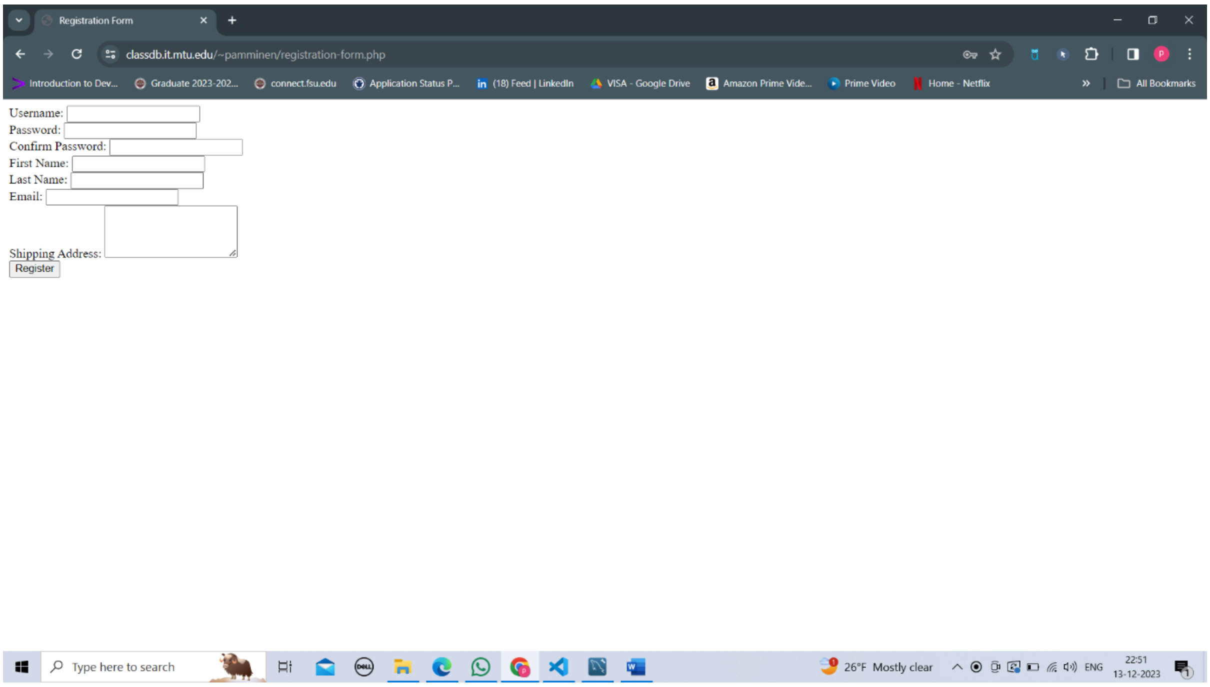


Welcome, kavitha!

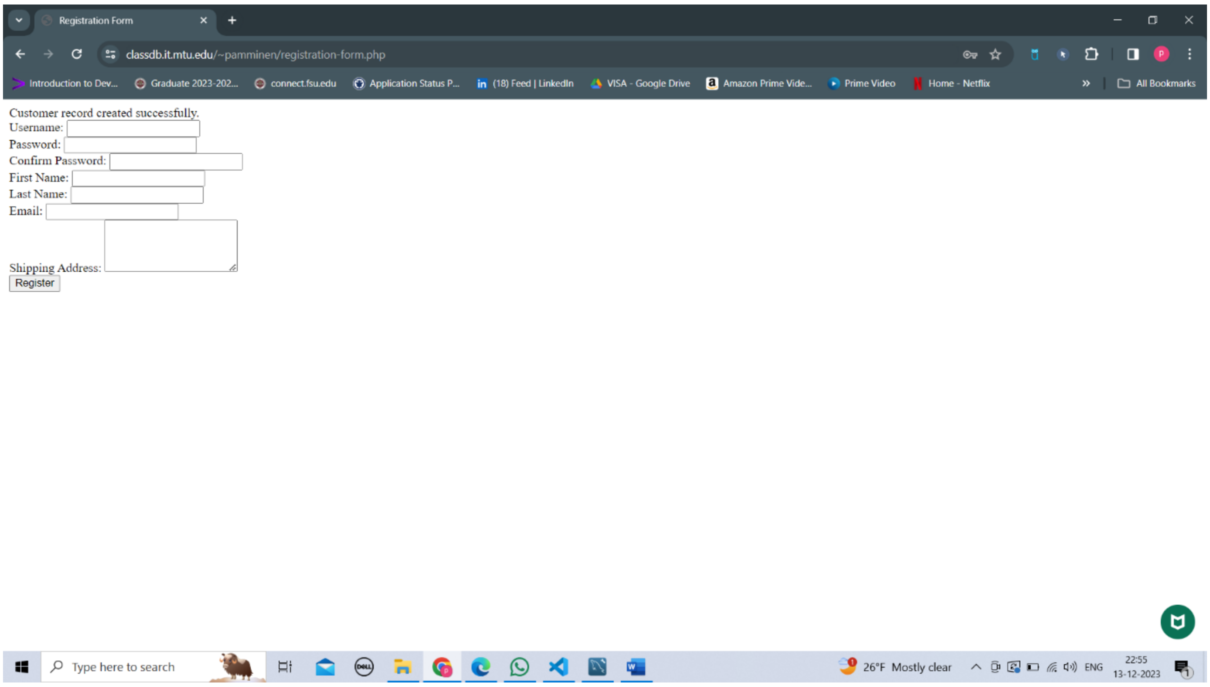
When I give correct credentials it will redirect to login page

4. New user registration

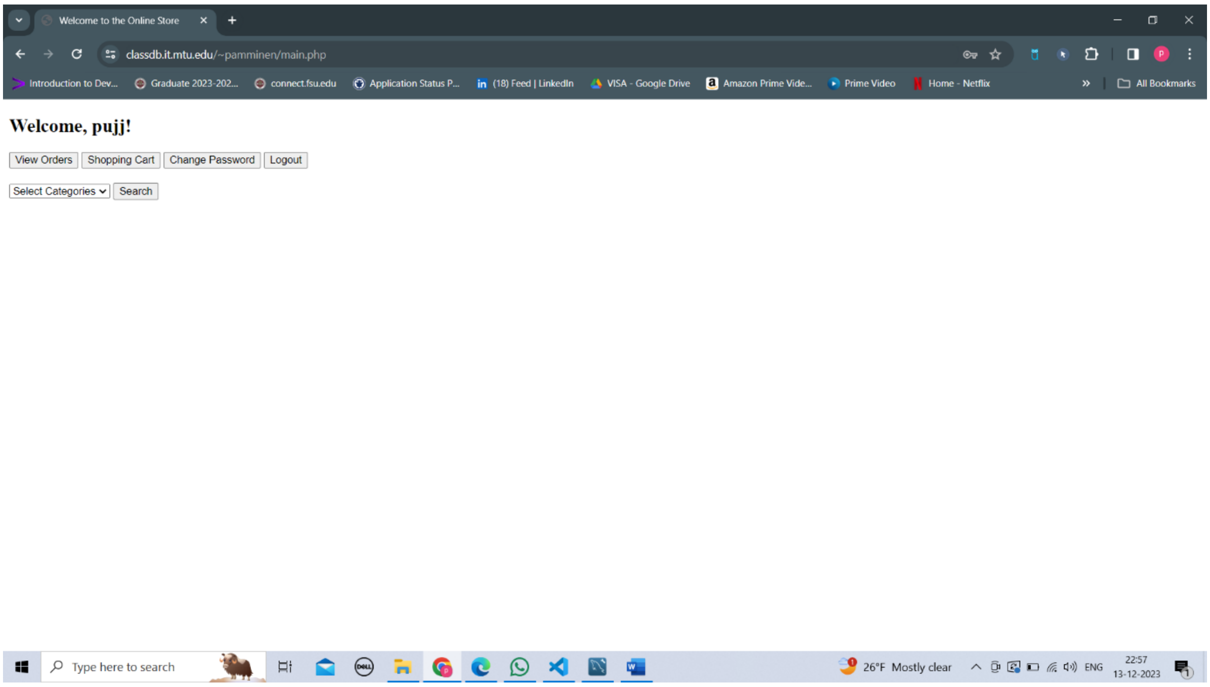
Before registration



After Registration



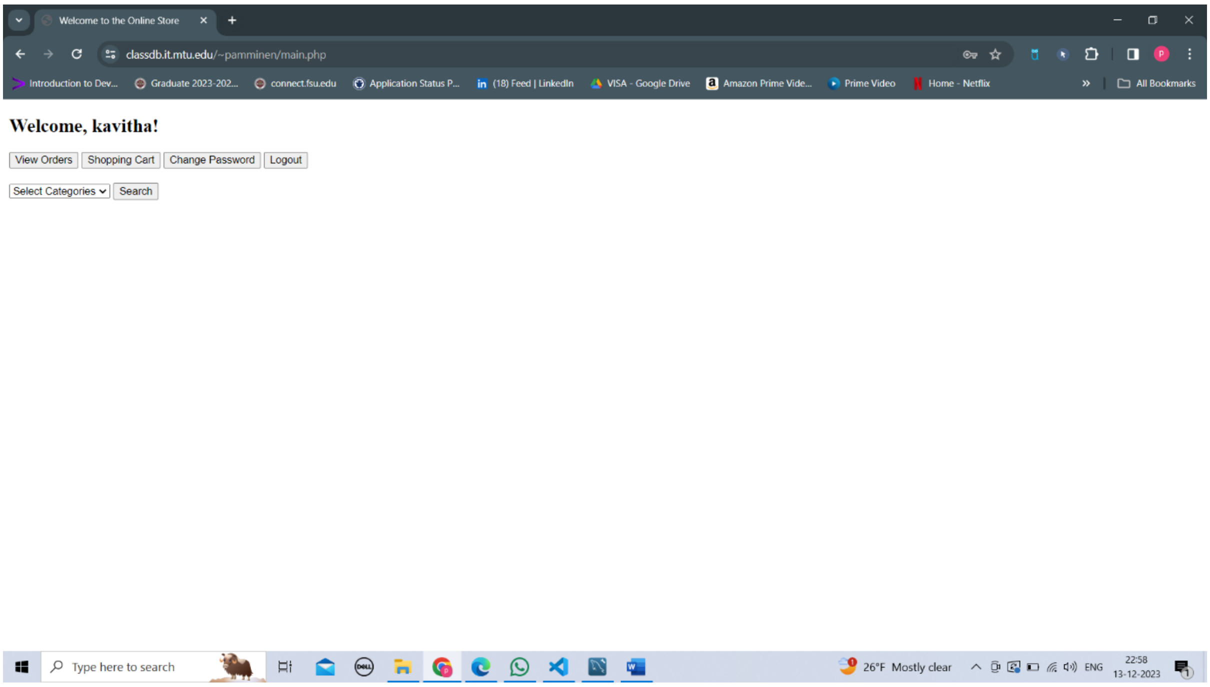
Customer record created successfully.



Welcome, pujj!

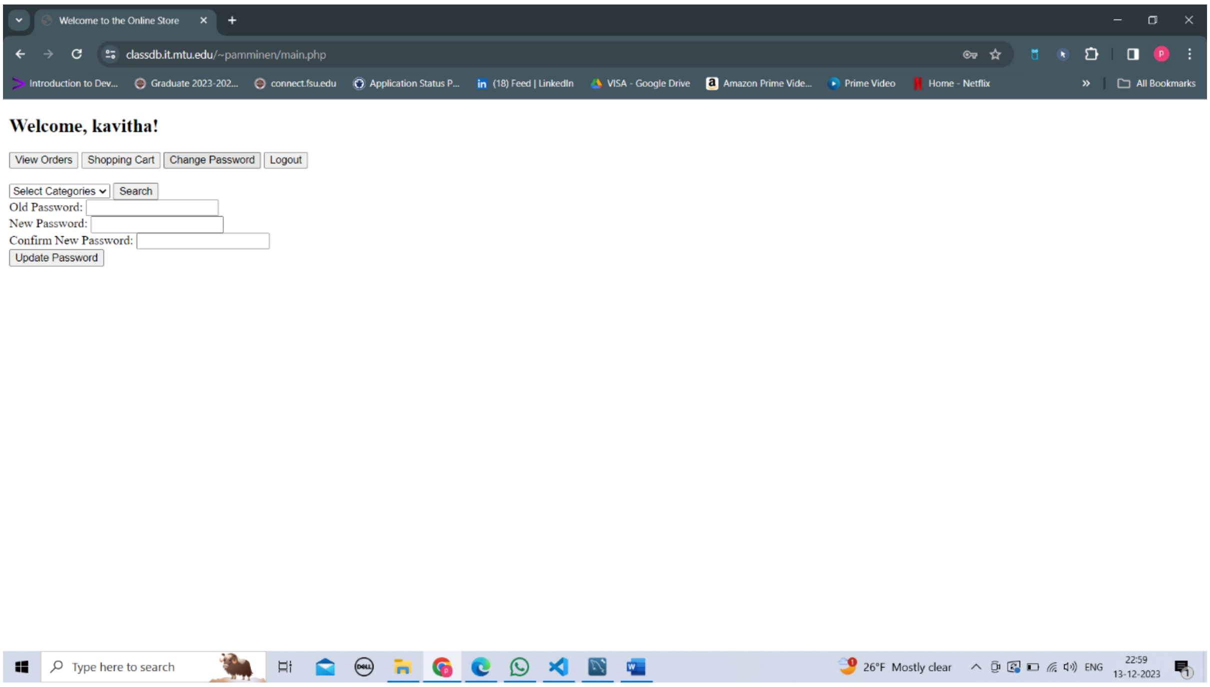
5. Customer main page after login. Show a welcome message and more function buttons

Here I created a new register



Welcome, kavitha!

6. Change password from the main page

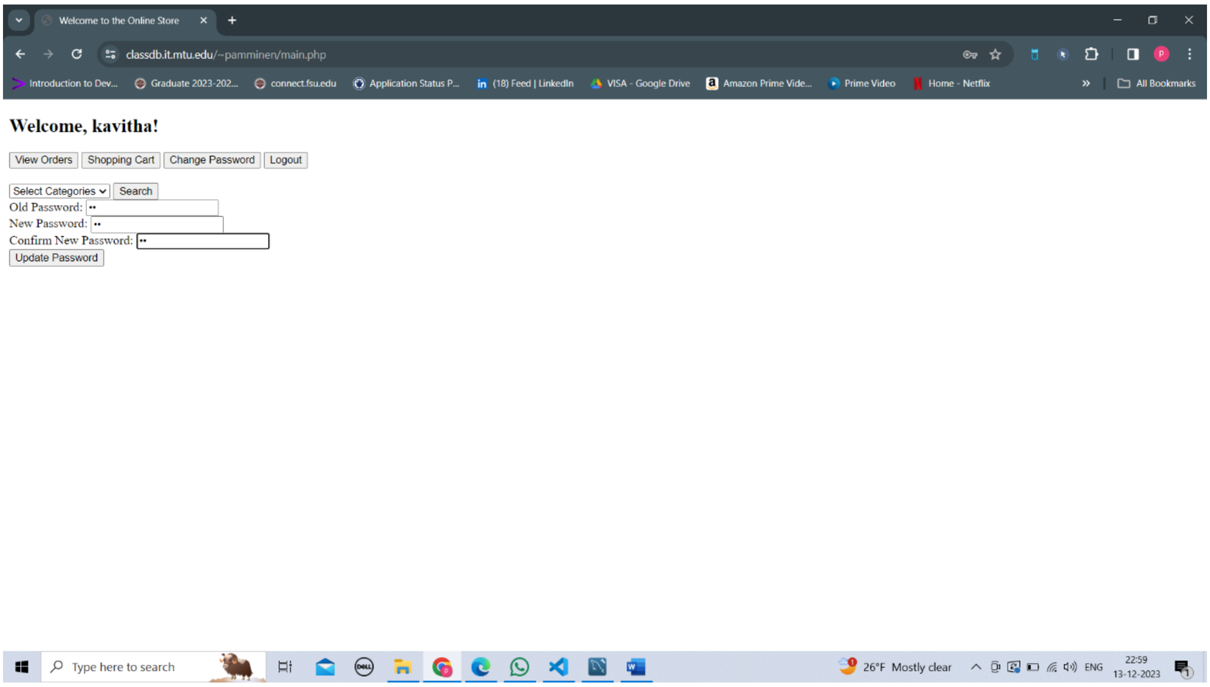


Welcome, kavitha!

Old Password:

New Password:

Confirm New Password:



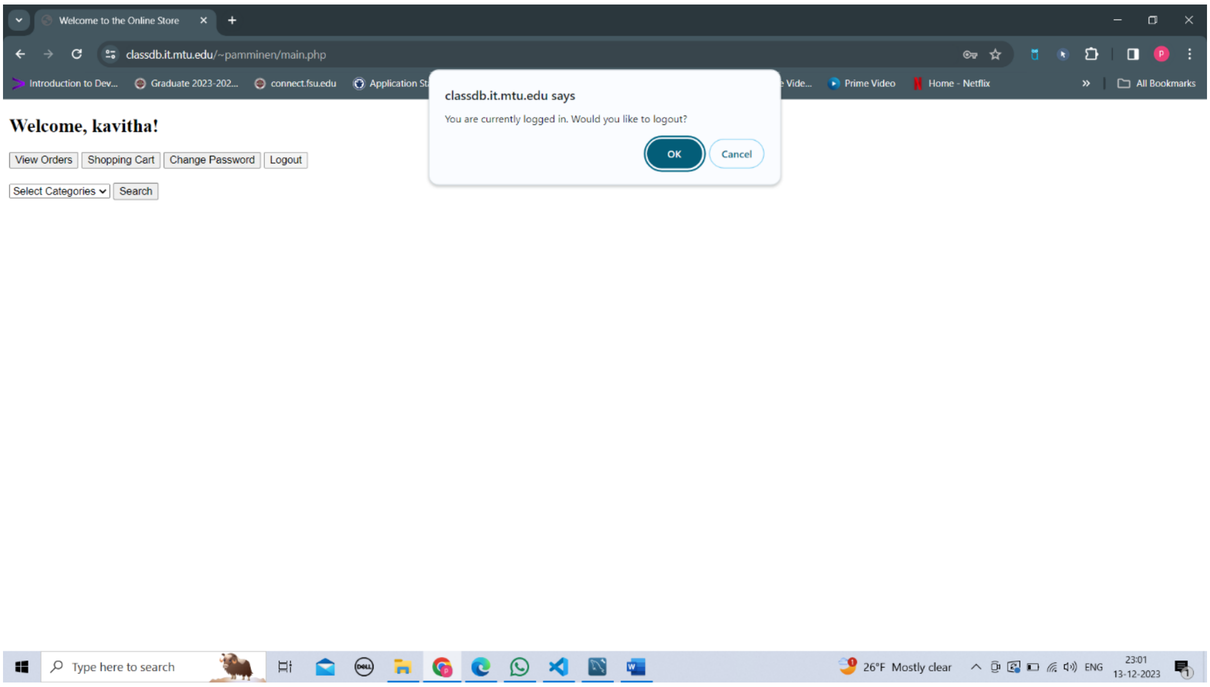
Welcome, kavitha!

Old Password: ••

New Password: ••

Confirm New Password: ••

7. Logout from the main page After user clicked the logout button in the main page, user will be prompted to confirm logout

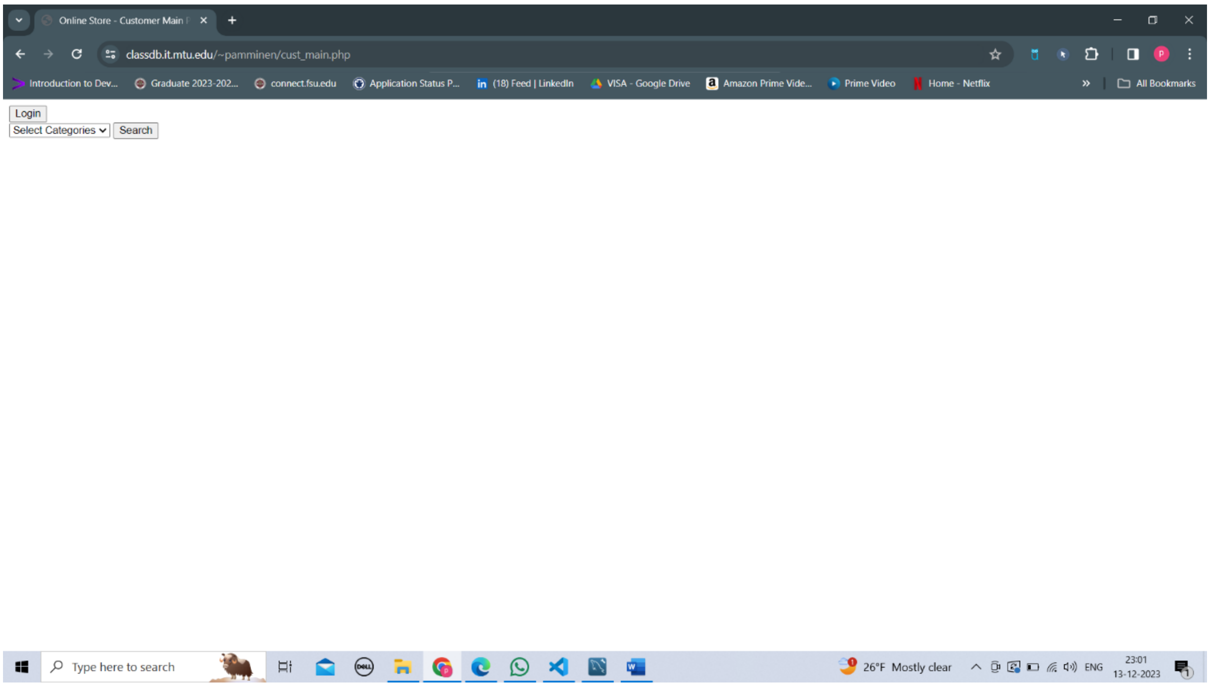


Welcome, kavitha!

classdb.it.mtu.edu says

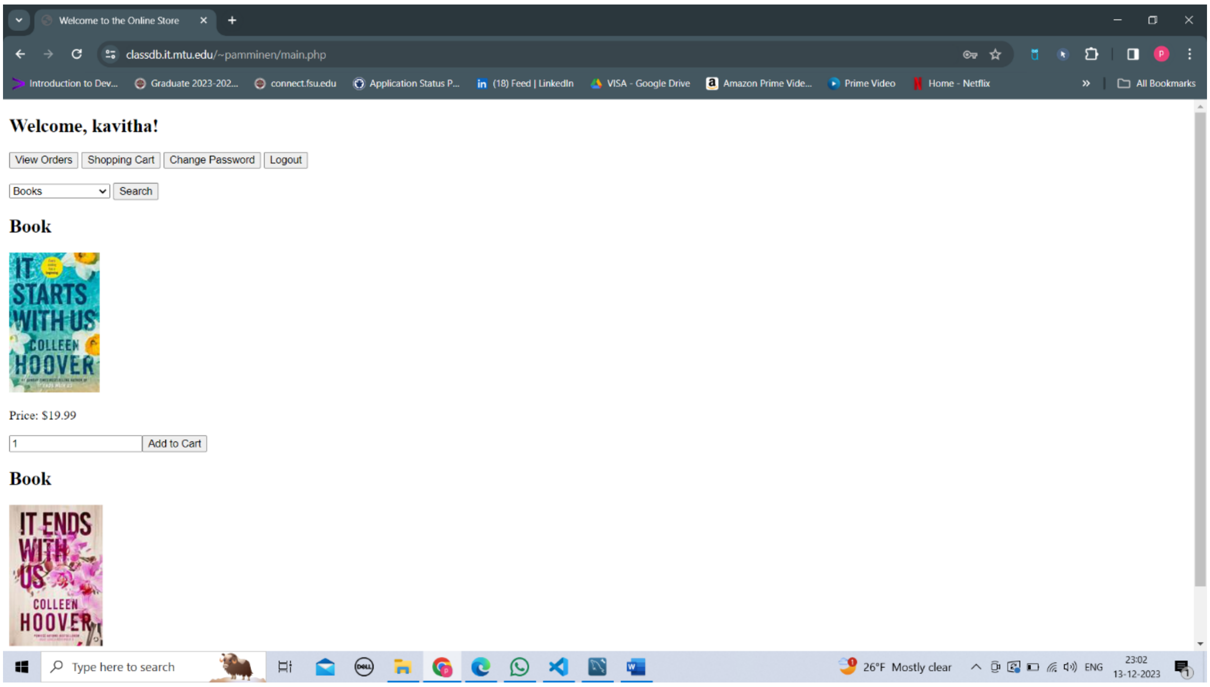
You are currently logged in. Would you like to logout?

After user clicked the logout again, user will be redirect to the main page with only browsing function



Online Store - Customer Main
classdb.it.mtu.edu/~pamminen/cust_main.php
Login
Select Categories
Search

8. Choose a category, click search after login Because customer logged in, therefore can enter quantity and add item to cart

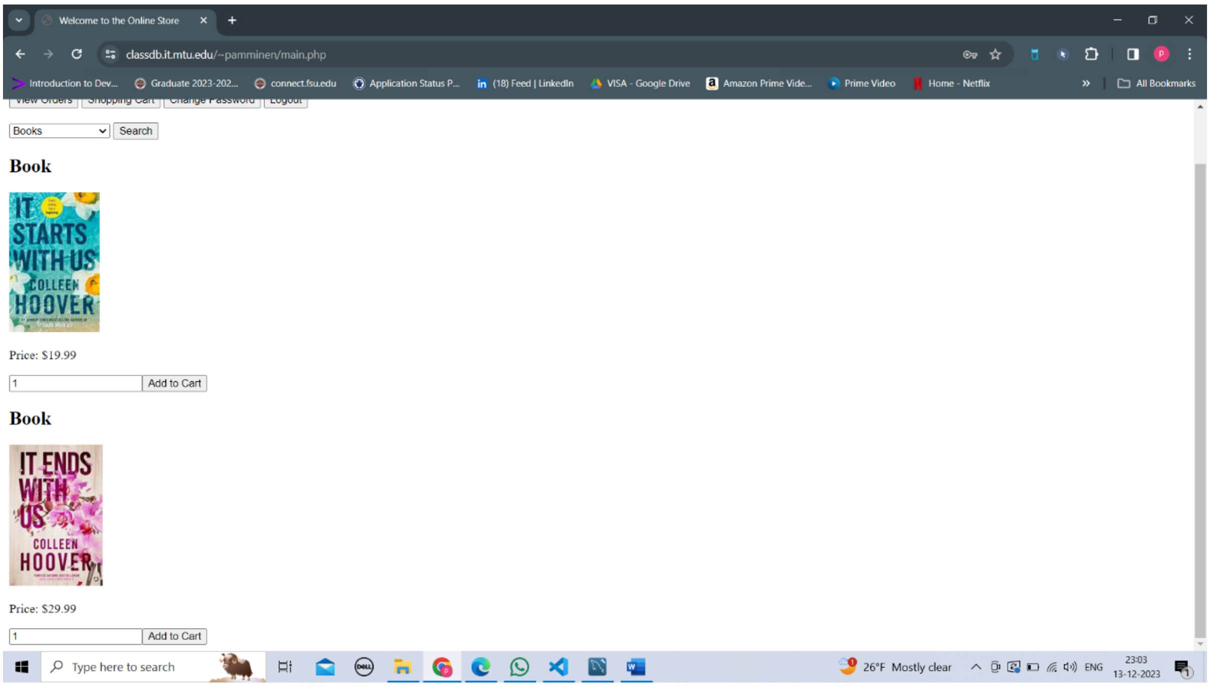


Welcome, kavitha!

Book

Price: $19.99

Book


Welcome to the Online Store
classdb.it.mtu.edu/~pamminen/main.php
Books
Search
1
Add to Cart
1
Add to Cart

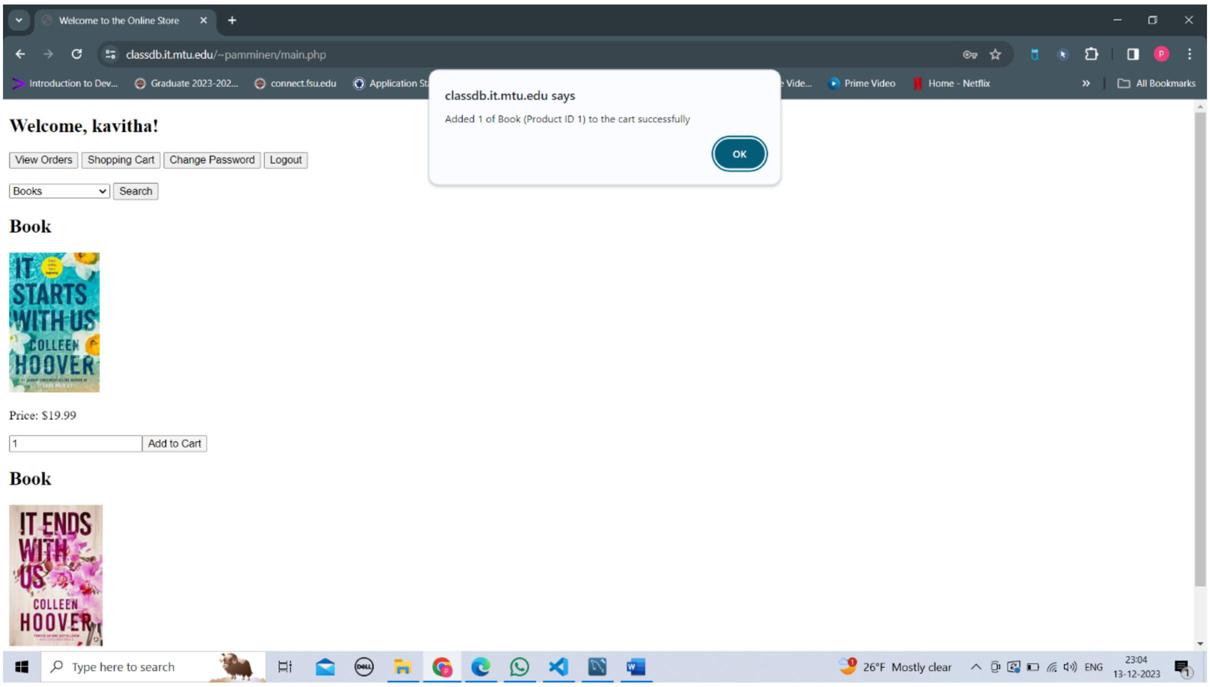
Book

Price: $19.99

Book

Price: $29.99

9. After entering quantity and pressed Add to Cart. User receive confirmation.



classdb.it.mtu.edu says

Added 1 of Book (Product ID 1) to the cart successfully

Welcome, kavitha!

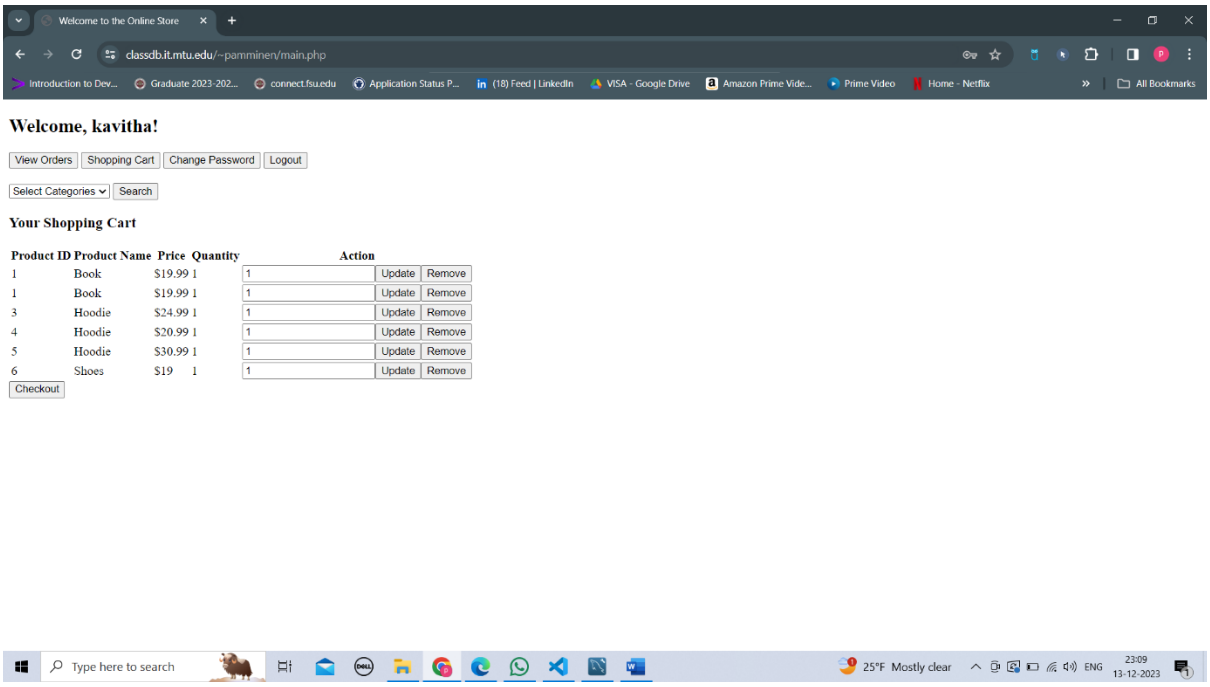
Book

Price: $19.99

1

Book

10. Click Shopping Cart in the main page



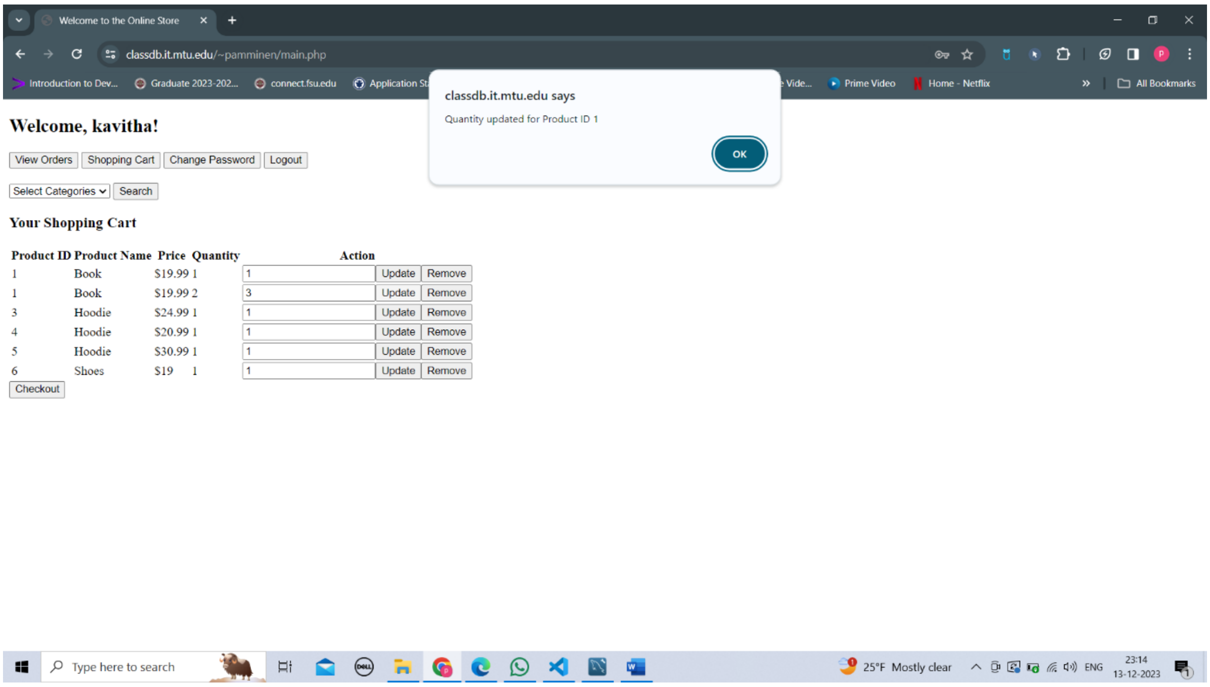
Welcome, kavitha!

Your Shopping Cart

Product ID	Product Name	Price	Quantity	Action
1	Book	$19.99	1	1 Update Remove
1	Book	$19.99	1	1 Update Remove
3	Hoodie	$24.99	1	1 Update Remove
4	Hoodie	$20.99	1	1 Update Remove
5	Hoodie	$30.99	1	1 Update Remove
6	Shoes	$19	1	1 Update Remove

11. Click update to update the quantity. User receives confirmation about the change.

I’m updating shoes it shows a pop-up



classdb.it.mtu.edu says

Quantity updated for Product ID 1

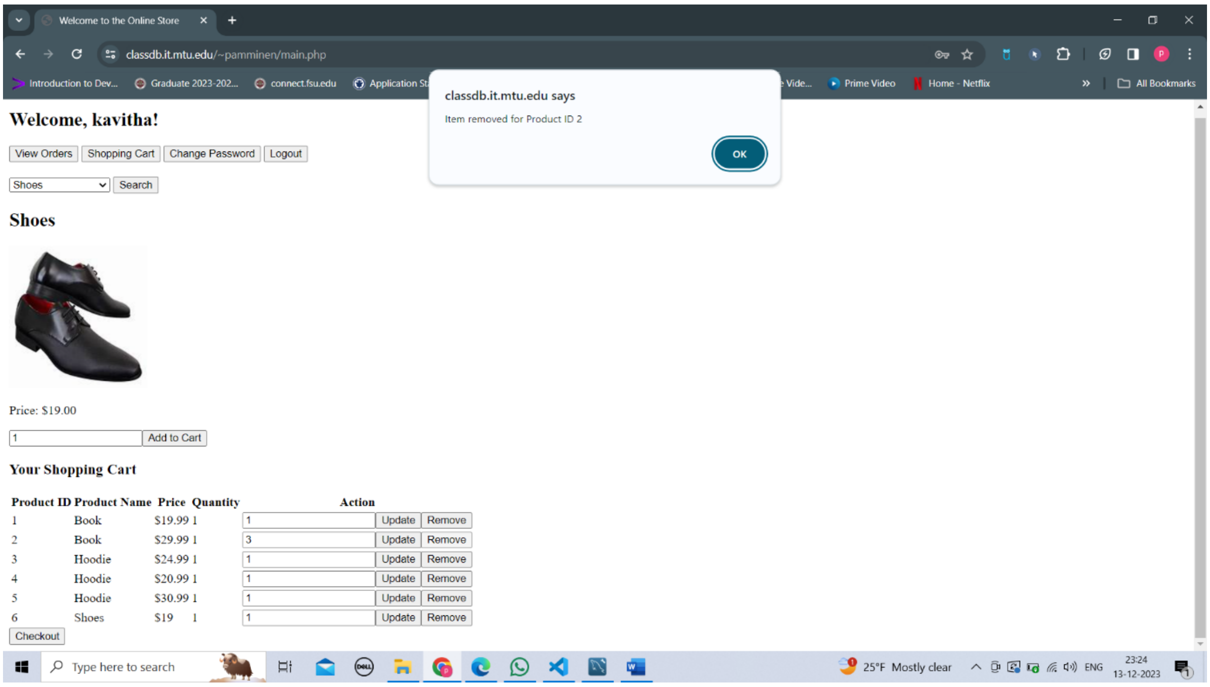
Welcome, kavitha!

Your Shopping Cart

Product ID	Product Name	Price	Quantity	Action
1	Book	$19.99	1	1 Update Remove
1	Book	$19.99	2	3 Update Remove
3	Hoodie	$24.99	1	1 Update Remove
4	Hoodie	$20.99	1	1 Update Remove
5	Hoodie	$30.99	1	1 Update Remove
6	Shoes	$19	1	1 Update Remove

Checkout

12. Click remove to remove an item. User receive confirmation about the change



classdb.it.mtu.edu says

Item removed for Product ID 2

Welcome, kavitha!

Shoes

Price: $19.00

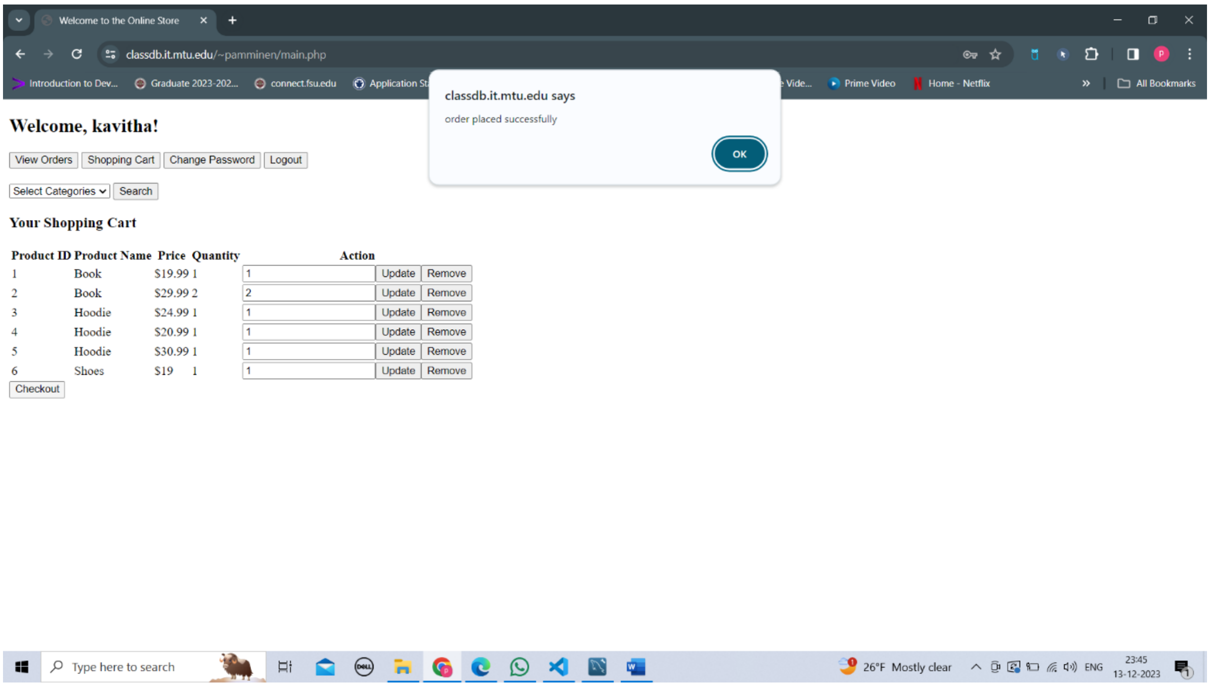
Your Shopping Cart

Product ID	Product Name	Price	Quantity	Action
1	Book	$19.99	1	1 Update Remove
2	Book	$29.99	1	3 Update Remove
3	Hoodie	$24.99	1	1 Update Remove
4	Hoodie	$20.99	1	1 Update Remove
5	Hoodie	$30.99	1	1 Update Remove
6	Shoes	$19	1	1 Update Remove

Checkout

Removing pop-up

13. Click Checkout



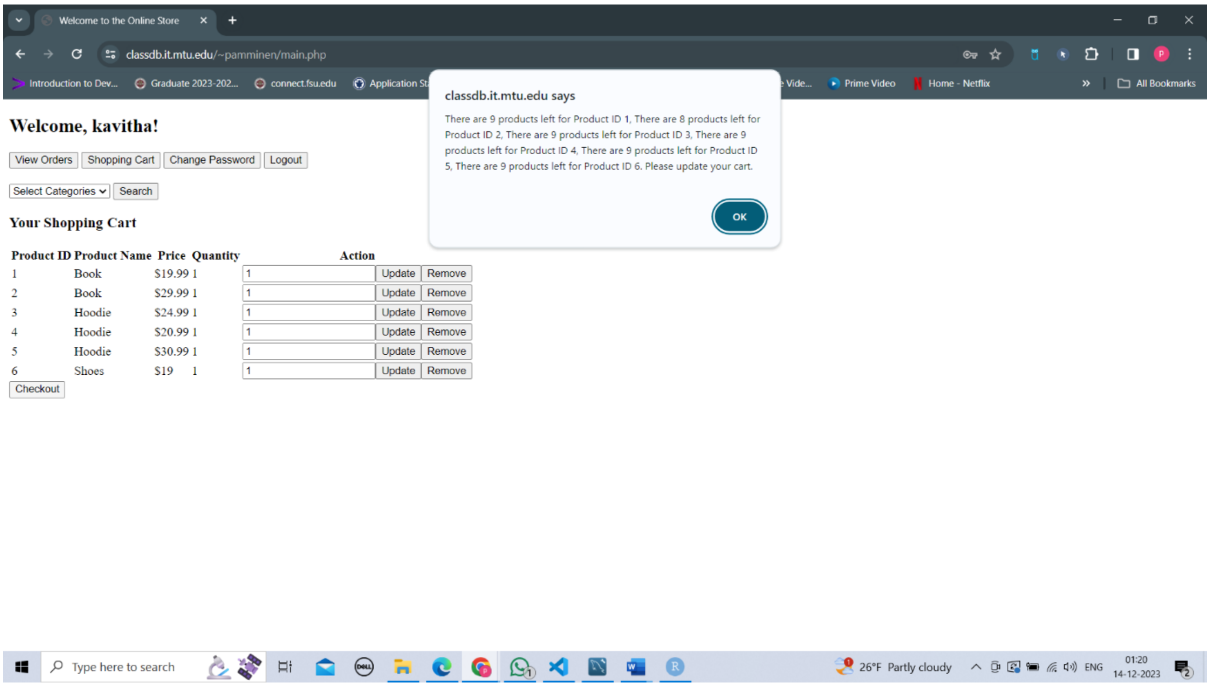
classdb.it.mtu.edu says

order placed successfully

Welcome, kavitha!

Your Shopping Cart

Product ID	Product Name	Price	Quantity	Action
1	Book	$19.99	1	1
2	Book	$29.99	2	2
3	Hoodie	$24.99	1	1
4	Hoodie	$20.99	1	1
5	Hoodie	$30.99	1	1
6	Shoes	$19	1	1



classdb.it.mtu.edu says

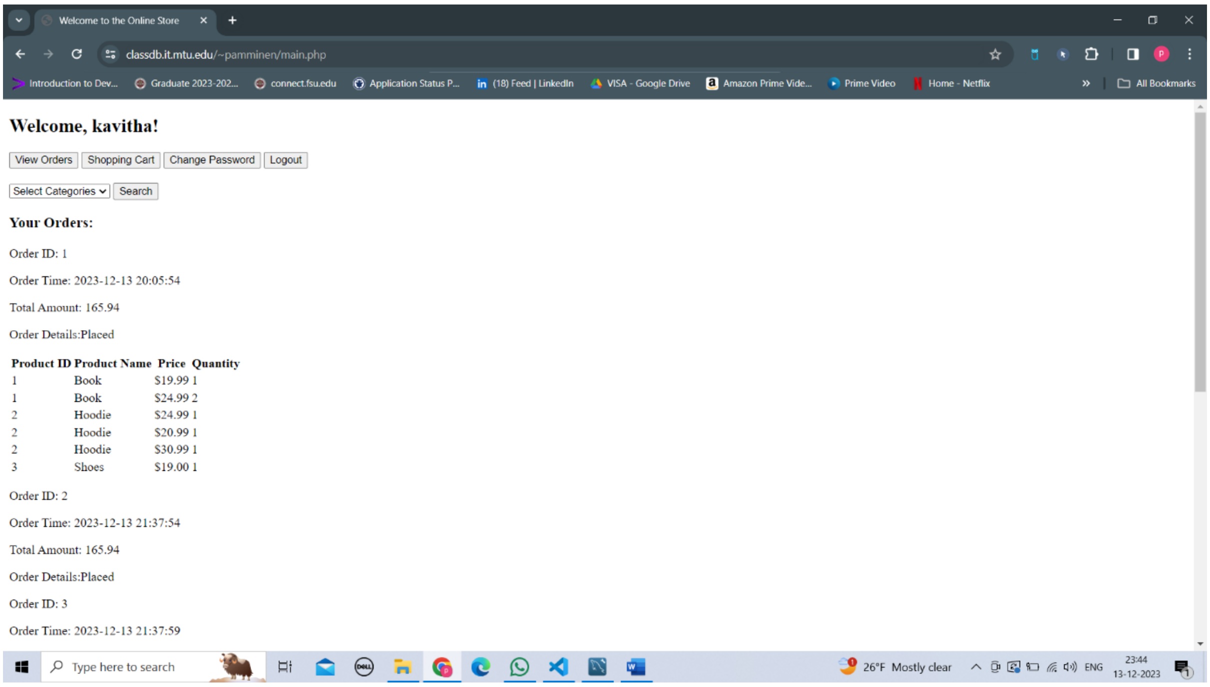
There are 9 products left for Product ID 1, There are 8 products left for Product ID 2, There are 9 products left for Product ID 3, There are 9 products left for Product ID 4, There are 9 products left for Product ID 5, There are 9 products left for Product ID 6. Please update your cart.

Welcome, kavitha!

Your Shopping Cart

Product ID	Product Name	Price	Quantity	Action
1	Book	$19.99	1	1
2	Book	$29.99	1	1
3	Hoodie	$24.99	1	1
4	Hoodie	$20.99	1	1
5	Hoodie	$30.99	1	1
6	Shoes	$19	1	1

14. Click View Order



Welcome, kavitha!

Your Orders:

Order ID: 1

Order Time: 2023-12-13 20:05:54

Total Amount: 165.94

Order Details:Placed

Product ID	Product Name	Price	Quantity
1	Book	$19.99	1
1	Book	$24.99	2
2	Hoodie	$24.99	1
2	Hoodie	$20.99	1
2	Hoodie	$30.99	1
3	Shoes	$19.00	1

Order ID: 2

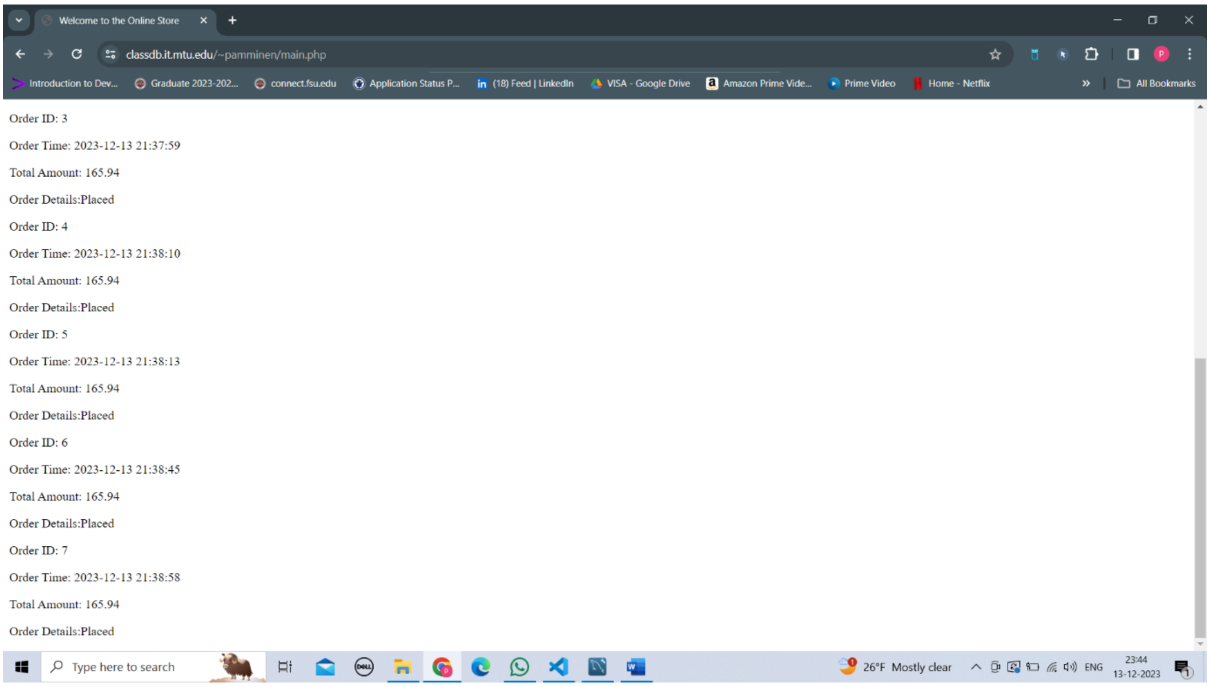
Order Time: 2023-12-13 21:37:54

Total Amount: 165.94

Order Details:Placed

Order ID: 3

Order Time: 2023-12-13 21:37:59



Order ID: 3

Order Time: 2023-12-13 21:37:59

Total Amount: 165.94

Order Details:Placed

Order ID: 4

Order Time: 2023-12-13 21:38:10

Total Amount: 165.94

Order Details:Placed

Order ID: 5

Order Time: 2023-12-13 21:38:13

Total Amount: 165.94

Order Details:Placed

Order ID: 6

Order Time: 2023-12-13 21:38:45

Total Amount: 165.94

Order Details:Placed

Order ID: 7

Order Time: 2023-12-13 21:38:58

Total Amount: 165.94

Order Details:Placed

3. a. (5p) Describe how transactions are used in your project. Copy the related code.

In the provided code, transactions are not explicitly used. Transactions are typically employed when you want to ensure that a series of database operations are treated as a single, atomic unit. This is important when multiple queries must succeed or fail together. If one query fails within the transaction, all changes are rolled back. If all queries succeed, the changes are committed.

```
function updatePassword($username, $newPassword)
{
  try {
    $dbh = connectDB();
    $dbh->beginTransaction(); // Start the transaction

    $hashedPassword = hash('sha256', $newPassword);

    $statement = $dbh->prepare("UPDATE users SET password = :password WHERE username = 
:username");
    $statement->bindParam(":username", $username);
    $statement->bindParam(":password", $hashedPassword);

    if ($statement->execute()) {
      $dbh->commit(); // Commit the transaction if successful
      $dbh = null;
      return true;
    } else {
      $errorInfo = $statement->errorInfo();
      error_log("Update Password Error: " . implode(" - ", $errorInfo));
      $dbh->rollBack(); // Rollback the transaction on failure
      $dbh = null;
      return false;
    }
  } catch (PDOException $e) {
    error_log("Update Password Exception: " . $e->getMessage());
    $dbh->rollBack(); // Rollback the transaction on exception
    $dbh = null;
    return false;
  }
}
```

The provided PHP code exemplifies the use of transactions in the context of updating a user's password securely and atomically. Transactions are initiated with `$dbh->beginTransaction();`, marking the beginning of a sequence of database operations that are treated as a single, indivisible unit. In this specific scenario, the code prepares an `UPDATE` query to modify the user's password, hashed using the SHA-256 algorithm, in the `users` table. The `execute()` method is employed to execute the SQL query, and if successful, the changes are committed to the database using `$dbh->commit();`. In the event of an error during execution, the code captures and logs the error information, triggers a rollback with `$dbh->rollBack();`, ensuring that any changes made during the transaction are reverted to maintain data consistency. Additionally, the code accounts for exceptions, logging them and initiating a rollback if any occur during the process. This robust approach ensures that the password update operation is executed atomically, guarding against partial or inconsistent changes in the database.

b. (5p) Describe how your code prevent SQL injection. Copy the related code.

My code uses parameterized queries, which is an effective method to prevent SQL injection. Parameterized queries ensure that user input is treated as data, not as executable code. Here's an example from My login function

```
function loginUser($username, $password) {
    try {
        $dbh = connectDB();
        $statement = $dbh->prepare("SELECT * FROM customer WHERE username = :username");
        $statement->bindParam(":username", $username);
        $statement->execute();
        $user = $statement->fetch(PDO::FETCH_ASSOC);
        $dbh = null;

        // Rest of the code...
    } catch (PDOException $e) {
        // Handle database connection errors
        echo "Error: " . $e->getMessage();
        return false;
    }
}
```

The code snippet showcases a robust defense against SQL injection through the implementation of parameterized queries in the login functionality. By utilizing the PDO (PHP Data Objects) prepare statement, the SQL query is constructed with a designated placeholder, `:username`, serving as a secure receptacle for user input. This deliberate separation of user input from the raw SQL query prevents the input from being interpreted as executable code, mitigating the risk of SQL injection attacks. The subsequent use of `bindParam` ensures that the actual user-supplied username is bound to the designated placeholder, establishing a clear boundary between user input and the SQL query. Upon execution with the `execute` method, the prepared statement treats the bound parameter as data rather than executable instructions, reinforcing the code's resilience against SQL injection vulnerabilities. This adherence to parameterized queries exemplifies a fundamental security measure, promoting the safe handling of user input in database interactions within the PHP application.

c. (5p) Describe how you encrypt the password. Copy the related code.

Describe how you encrypt the password. Copy the related code.

ChatGPT

Your code uses the password_hash function, which is a secure way to hash passwords. Here's the relevant part of your code:

php

Copy code

```
function registerUser($newUsername, $newPassword, $newFirstname, $newLastname, $newEmail, $newMobile, $newAddress)
{
    try {
        $dbh = connectDB();
        $hashedPassword = password_hash($newPassword, PASSWORD_DEFAULT);
        $statement = $dbh->prepare("INSERT INTO customer (username, password, first_name, last_name, email, mobile, address1) VALUES (:username, :password, :firstname, :lastname, :email, :mobile, :address1)");

        $statement->bindParam(":username", $newUsername);
        $statement->bindParam(":password", $hashedPassword);
        $statement->bindParam(":firstname", $newFirstname);
        $statement->bindParam(":lastname", $newLastname);
        $statement->bindParam(":email", $newEmail);
        $statement->bindParam(":mobile", $newMobile);
        $statement->bindParam(":address1", $newAddress);

        // Rest of the code...
    } catch (PDOException $e) {
        // Log the error or redirect to an error page
        error_log("Error in registerUser: " . $e->getMessage());
        echo "Error: Unable to register the user.";
    }
}
```

In this code snippet:

The password_hash function is used to securely hash the user's password before storing it in the database. The hashed password is stored in the password column of the user_cust table.

php

Copy code

```
$hashedPassword = password_hash($newPassword, PASSWORD_DEFAULT);
```

The hashed password is then bound to the :password placeholder in the prepared SQL statement.

php

Copy code

```
$statement->bindParam(":password", $hashedPassword);
```

This is a secure way to store passwords because it uses a strong hashing algorithm and includes automatic salt generation. The PASSWORD_DEFAULT algorithm will be updated as needed to maintain security.

When authenticating users during login, you'll use the password_verify function to compare the entered password with the stored hashed password. This is also a secure method and helps protect against password-related security risks.

d.

Sessions play a crucial role in persisting user data across multiple requests in my code. The $_SESSION superglobal serves as a storage mechanism for retaining user-related information. The following snippet from your login process exemplifies the utilization of sessions:

```
if (loginUser($username, $password)) {
    // Set the session variable to retain the logged-in username
    $_SESSION["username"] = $username;
    // Redirect to main.php or cust_main.php as necessary
    header("LOCATION: main.php");
    return;
}
```

In the provided code, sessions are efficiently utilized for user authentication. Upon successful login, the user's username is stored in the `$_SESSION["username"]` variable, ensuring persistence across pages during the session. The code employs a conditional check to confirm successful logins before initiating the session and redirecting the user to the main page. This implementation enhances security and user experience by securely managing user-related data throughout their interaction with the web application.