**PROJECT REPORT**

**ONLINE AUCTION SYSTEM**

**CONTRIBUTORS:**

MANAS C BAVASKAR: 221080004

AARYA A BODAS: 221080008

ASSIGNMENT 1

ASSIGNMENT 2

ASSIGNMENT 3

a)

Write TRIGGERS on tables for entered data validations, updating derived attributes, updating foreign key data updates, etc. See that how TRIIGERS will help to make correct, non-repeated data entry.

Triggers:

1. Unique Buyer and Seller username

TRIGGER CODE:

DELIMITER //

CREATE TRIGGER trg\_unique\_username

BEFORE INSERT ON Buyers

FOR EACH ROW

BEGIN

    DECLARE username\_count INT;

    SELECT COUNT(\*) INTO username\_count FROM Buyers WHERE Username = NEW.Username;

    IF username\_count > 0 THEN

        SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Username already exists in Buyers table';

    END IF;

END //

CREATE TRIGGER trg\_unique\_username\_seller

BEFORE INSERT ON Sellers

FOR EACH ROW

BEGIN

    DECLARE username\_count INT;

    SELECT COUNT(\*) INTO username\_count FROM Sellers WHERE Username = NEW.Username;

    IF username\_count > 0 THEN

        SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Username already exists in Sellers table';

    END IF;

END //

DELIMITER ;

TRIGGER EXPLANATION:

These triggers ensure that a username is unique within the respective tables **Buyers** and **Sellers**. Here's what each trigger does:

1. **Trigger for Buyers Table (trg\_unique\_username):**
   * This trigger is fired before inserting a new row into the **Buyers** table.
   * It checks if the **Username** being inserted already exists in the **Buyers** table.
   * If a record with the same **Username** already exists, it raises an error using **SIGNAL SQLSTATE** with a custom error message.
   * This prevents the insertion of a duplicate username into the **Buyers** table.
2. **Trigger for Sellers Table (trg\_unique\_username\_seller):**
   * This trigger is similar to the previous one but applies to the **Sellers** table.
   * It is fired before inserting a new row into the **Sellers** table.
   * It checks if the **Username** being inserted already exists in the **Sellers** table.
   * If a record with the same **Username** already exists, it raises an error using **SIGNAL SQLSTATE** with a custom error message.
   * This ensures that usernames are unique within the **Sellers** table.

Here's the test data insertion:

INSERT INTO Buyers (Buyer\_ID, Username, Password, Email, Address, Account\_Balance)  VALUES (99, 'raj\_sharma', 'password123', 'raj@example.com', '123 Street', 1000.00);

INSERT INTO Sellers (Seller\_ID, Username, Password, Email, Address, Account\_Balance)  VALUES (99, 'fashion\_hub', 'password456', 'fashion@example.com', '456 Avenue', 2000.00);

If we try to insert a duplicate username for either buyers or sellers, the triggers will prevent it and raise an error.

1. Set Last\_Bid Default Value

TRIGGER CODE:

DELIMITER //

CREATE TRIGGER set\_last\_bid\_default BEFORE INSERT ON items

FOR EACH ROW

BEGIN

    IF NEW.Last\_bid IS NULL THEN

        SET NEW.Last\_bid = NEW.Starting\_Price;

    END IF;

END//

DELIMITER ;

TRIGGER EXPLANATION:

This trigger ensures that when a new row is inserted into the **Items** table, the **Last\_bid** field is set to the **Starting\_Price** if it's initially NULL. Here's an explanation of the trigger:

**Trigger Description:**

* **Trigger Name:** **set\_last\_bid\_default**
* **Event:** **BEFORE INSERT** on the **Items** table.
* **Trigger Type:** **FOR EACH ROW**, meaning it will execute for each row being inserted.
* **Action:** Sets the **Last\_bid** field to the **Starting\_Price** if **Last\_bid** is initially NULL for the row being inserted.

**Explanation:**

1. **IF Statement:**
   * Checks if the **Last\_bid** field for the new row being inserted is NULL (**IF NEW.Last\_bid IS NULL**).
2. **Setting Default Value:**
   * If the **Last\_bid** is indeed NULL, it sets the **Last\_bid** field to the **Starting\_Price** for that item (**SET NEW.Last\_bid = NEW.Starting\_Price;**).

**Purpose:**

* This trigger ensures that when a new item is added to the **Items** table and no previous bids have been made (thus **Last\_bid** is NULL), the **Last\_bid** field is initialized with the **Starting\_Price** of the item. This helps to maintain consistency in the data and ensures that there is always a starting point for bidding on an item.

1. Update Last Bid trigger

TRIGGER CODE:

DELIMITER //

CREATE TRIGGER trg\_update\_last\_bid

AFTER INSERT ON Bids

FOR EACH ROW

BEGIN

    UPDATE Items

    SET Last\_Bidder = NEW.Bidder\_ID,

        Last\_Bid = NEW.Bid\_Amount

    WHERE Item\_ID = NEW.Item\_ID;

END //

DELIMITER ;

TRIGGER EXPLANATION:

This trigger, named **trg\_update\_last\_bid**, is designed to update the **Last\_Bidder** and **Last\_Bid** fields in the **Items** table whenever a new bid is inserted into the **Bids** table. Here's a breakdown of how it works:

**Trigger Description:**

* **Trigger Name:** **trg\_update\_last\_bid**
* **Event:** **AFTER INSERT** on the **Bids** table.
* **Trigger Type:** **FOR EACH ROW**, indicating that the trigger will execute once for each row that is inserted into the **Bids** table.
* **Action:** Updates the **Last\_Bidder** and **Last\_Bid** fields in the **Items** table based on the information of the newly inserted bid.

**Explanation:**

1. **UPDATE Statement:**
   * The trigger executes an **UPDATE** statement on the **Items** table.
   * It sets the **Last\_Bidder** field in the **Items** table to the **Bidder\_ID** of the newly inserted bid (**SET Last\_Bidder = NEW.Bidder\_ID**).
   * It also updates the **Last\_Bid** field in the **Items** table to the **Bid\_Amount** of the newly inserted bid (**SET Last\_Bid = NEW.Bid\_Amount**).
2. **WHERE Clause:**
   * The **UPDATE** statement applies these changes only to the row in the **Items** table where the **Item\_ID** matches the **Item\_ID** of the newly inserted bid (**WHERE Item\_ID = NEW.Item\_ID**).

**Purpose:**

* This trigger ensures that whenever a new bid is made on an item (**Bids** table), the corresponding **Last\_Bidder** and **Last\_Bid** fields in the **Items** table are updated accordingly. This helps to keep track of the latest bid and the bidder for each item without requiring manual updates.

EXAMPLE:

INSERT INTO Bids (Bid\_ID, Bidder\_ID, Item\_ID, Bid\_Amount, Bid\_Time, Bid\_Status, Bid\_Increment) VALUES (123, 1, 19, 200.00, NOW(), 'Active', 0.00);

SELECT Last\_Bidder, Last\_Bid FROM Items WHERE Item\_ID = 19;

After inserting a new bid into the **Bids** table with **Item\_ID** 19, the trigger will automatically update the **Last\_Bidder** and **Last\_Bid** fields in the **Items** table. We can then retrieve this updated information using the **SELECT** statement provided.

1. Update Buyer and Seller balance

TRIGGER CODE:

DELIMITER //

CREATE TRIGGER trg\_update\_buyer\_balance

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

    DECLARE current\_balance DECIMAL(10, 2);

    SELECT Account\_Balance INTO current\_balance FROM Buyers WHERE Buyer\_ID = NEW.Buyer\_ID;

    IF current\_balance >= NEW.Transaction\_Amount THEN

        UPDATE Buyers

        SET Account\_Balance = current\_balance - NEW.Transaction\_Amount

        WHERE Buyer\_ID = NEW.Buyer\_ID;

    ELSE

        SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient balance for the transaction';

    END IF;

END //

DELIMITER ;

Top of Form

DELIMITER //

CREATE TRIGGER trg\_update\_seller\_balance

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

    UPDATE Sellers

    SET Account\_Balance = Account\_Balance + NEW.Transaction\_Amount

    WHERE Seller\_ID = NEW.Seller\_ID;

END //

DELIMITER ;

TRIGGER EXPLANATION:

These two triggers, **trg\_update\_buyer\_balance** and **trg\_update\_seller\_balance**, are designed to update the account balances of buyers and sellers respectively after a new transaction is inserted into the **Transactions** table.

**Trigger trg\_update\_buyer\_balance:**

* **Trigger Name:** trg\_update\_buyer\_balance
* **Event:** AFTER INSERT on the Transactions table.
* **Trigger Type:** FOR EACH ROW, meaning it executes once for each row that is inserted into the Transactions table.
* **Purpose:** Update the account balance of the buyer who made the transaction.

**Explanation:**

1. **DECLARE Statement:**
   * Declares a local variable **current\_balance** of type DECIMAL(10, 2) to store the current account balance of the buyer.
2. **SELECT Statement:**
   * Retrieves the current account balance of the buyer (**Account\_Balance**) from the Buyers table based on the **Buyer\_ID** of the newly inserted transaction (**NEW.Buyer\_ID**).
3. **IF Statement:**
   * Checks if the current balance (**current\_balance**) is sufficient for the transaction amount (**NEW.Transaction\_Amount**).
4. **UPDATE Statement (IF condition is met):**
   * If the current balance is sufficient, it updates the buyer's account balance by subtracting the transaction amount.
5. **SIGNAL Statement (IF condition is not met):**
   * If the current balance is insufficient, it raises an error using SIGNAL SQLSTATE with a custom error message indicating "Insufficient balance for the transaction".

**Trigger trg\_update\_seller\_balance:**

* **Trigger Name:** trg\_update\_seller\_balance
* **Event:** AFTER INSERT on the Transactions table.
* **Trigger Type:** FOR EACH ROW.
* **Purpose:** Update the account balance of the seller who made the transaction.

**Explanation:**

1. **UPDATE Statement:**
   * Updates the account balance of the seller by adding the transaction amount (**NEW.Transaction\_Amount**) to their existing account balance.

**Tests:**

* After inserting a new transaction into the Transactions table, you can verify the updated account balances of the buyer and seller by querying the Buyers and Sellers tables respectively.

EXAMPLE:

INSERT INTO Transactions (Transaction\_ID, Buyer\_ID, Seller\_ID, Item\_ID, Transaction\_Amount, Transaction\_Time, Payment\_Method, Transaction\_Status) VALUES (456, 1, 2, 19, 500.00, NOW(), 'Credit Card', 'Success');

SELECT Account\_Balance FROM Buyers WHERE Buyer\_ID = 1;

SELECT Account\_Balance FROM Sellers WHERE Seller\_ID = 2;

1. Set Winning Bid After Auction Update

TRIGGER CODE:

DELIMITER //

CREATE TRIGGER after\_auction\_update\_set\_winning\_bid\_trigger

AFTER UPDATE ON Auctions

FOR EACH ROW

BEGIN

    DECLARE max\_bid\_id INT;

    DECLARE bidder\_id INT;

    DECLARE winning\_bid DECIMAL(10, 2);

    IF NEW.Auction\_Status = 'Closed' AND OLD.Auction\_Status != 'Closed' THEN

        -- Selecting the highest bid amount for the item

        SELECT MAX(Bid\_Amount) INTO winning\_bid FROM Bids WHERE Item\_ID = NEW.Item\_ID;

        IF winning\_bid IS NOT NULL THEN

            -- Selecting the maximum bid ID and incrementing it by 1 for the new bid

            SELECT COALESCE(MAX(Bid\_ID), 0) + 1 INTO max\_bid\_id FROM Bids;

            -- Selecting the bidder ID associated with the highest bid

            SELECT Bidder\_ID INTO bidder\_id FROM Bids WHERE Item\_ID = NEW.Item\_ID AND Bid\_Amount = winning\_bid ORDER BY Bid\_Time DESC LIMIT 1;

            -- Inserting the winning bid into the Bids table

            IF bidder\_id IS NOT NULL THEN

                INSERT INTO Bids (Bid\_ID, Bidder\_ID, Item\_ID, Bid\_Amount, Bid\_Time, Bid\_Status, Bid\_Increment)

                VALUES (max\_bid\_id, bidder\_id, NEW.Item\_ID, winning\_bid, NOW(), 'Winning', 0);

            END IF;

        END IF;

    END IF;

END //

DELIMITER ;

TRIGGER EXPLANATION:

This trigger, named **after\_auction\_update\_set\_winning\_bid\_trigger**, is designed to automatically set the winning bid for an item in the **Bids** table after an auction is closed.

**Trigger Description:**

* **Trigger Name:** after\_auction\_update\_set\_winning\_bid\_trigger
* **Event:** AFTER UPDATE on the **Auctions** table.
* **Trigger Type:** FOR EACH ROW, meaning it executes once for each row that is updated in the **Auctions** table.
* **Purpose:** Automatically sets the winning bid for an item in the **Bids** table when the corresponding auction is closed.

**Explanation:**

1. **IF Statement:**
   * Checks if the **Auction\_Status** has changed to 'Closed' (**IF NEW.Auction\_Status = 'Closed' AND OLD.Auction\_Status != 'Closed'**), indicating that the auction has been closed.
2. **Selecting the Winning Bid:**
   * If the auction is closed, it selects the highest bid amount (**MAX(Bid\_Amount)**) for the item associated with the closed auction and stores it in the **winning\_bid** variable.
3. **Handling Winning Bid Existence:**
   * If a winning bid exists (**winning\_bid IS NOT NULL**), it proceeds to select the maximum bid ID and increments it by 1 to generate a new bid ID for the winning bid.
   * It then selects the bidder ID associated with the highest bid amount for the item.
4. **Inserting Winning Bid:**
   * If a bidder ID is found (**bidder\_id IS NOT NULL**), it inserts a new row into the **Bids** table with the generated bid ID, bidder ID, item ID, winning bid amount, current timestamp, 'Winning' status, and zero bid increment.

**Test:**

* The provided test updates the **Auction\_Status** of a specific auction (**Auction\_ID = 24**) to 'Closed' and then selects the winning bid from the **Bids** table for the item associated with that auction.

**Example:**

UPDATE Auctions SET Auction\_Status = 'Closed' WHERE Auction\_ID = 24;

SELECT \* FROM Bids WHERE Item\_ID = (SELECT Item\_ID FROM Auctions WHERE Auction\_ID = 24) AND Bid\_Status = 'Winning';

1. Insert Transaction on Auction End

TRIGGER CODE:

DELIMITER //

CREATE TRIGGER end\_auction\_insert\_transactions\_trigger

BEFORE UPDATE ON Auctions

FOR EACH ROW

BEGIN

    DECLARE new\_transaction\_id INT;

    -- Check if the auction end time is in the past or now

    IF OLD.Auction\_Status != 'Closed' AND NEW.Auction\_Status = 'Closed' AND NEW.Auction\_End\_Time <= NOW() THEN

        -- Generate a new transaction ID by incrementing the maximum transaction ID

        SET new\_transaction\_id = (SELECT COALESCE(MAX(Transaction\_ID), 0) + 1 FROM Transactions);

        -- Fetch the Last\_Bidder and Last\_Bid from the Items table based on the Item\_ID associated with the auction

        SET @bidder\_username := (SELECT Last\_Bidder FROM Items WHERE Item\_ID = NEW.Item\_ID);

        SET @last\_bid := (SELECT Last\_Bid FROM Items WHERE Item\_ID = NEW.Item\_ID);

        -- Fetch the Buyer\_ID using the bidder's username

        SET @buyer\_id := (SELECT Buyer\_ID FROM Buyers WHERE Username = @bidder\_username);

        -- Insert a new transaction into the Transactions table

        INSERT INTO Transactions (Transaction\_ID, Buyer\_ID, Seller\_ID, Item\_ID, Transaction\_Amount, Transaction\_Time, Payment\_Method, Transaction\_Status)

        VALUES (

            new\_transaction\_id, -- New transaction ID

            @buyer\_id, -- Buyer ID based on Last\_Bidder

            (SELECT Seller\_ID FROM Items WHERE Item\_ID = NEW.Item\_ID), -- Seller ID from the auction

            NEW.Item\_ID, -- Item ID from the auction

            @last\_bid, -- Last bid amount from the auction

            NOW(), -- Current timestamp

            'Online', -- Payment method (assuming online)

            'Completed' -- Transaction status (assuming completed)

        );

        -- Update the buyer's account balance by subtracting the last bid amount

        UPDATE Buyers

        SET Account\_Balance = Account\_Balance - @last\_bid

        WHERE Buyer\_ID = @buyer\_id;

    END IF;

END //

DELIMITER ;

TRIGGER EXPLANATION:

This trigger, named **end\_auction\_insert\_transactions\_trigger**, is designed to automatically insert a transaction into the **Transactions** table and update the buyer's account balance when an auction is closed.

**Trigger Description:**

* **Trigger Name:** end\_auction\_insert\_transactions\_trigger
* **Event:** BEFORE UPDATE on the **Auctions** table.
* **Trigger Type:** FOR EACH ROW, indicating that the trigger will execute once for each row that is updated in the **Auctions** table.
* **Purpose:** Automatically inserts a transaction into the **Transactions** table and updates the buyer's account balance when the auction is closed and its end time has passed.

**Explanation:**

1. **IF Statement:**
   * Checks if the **Auction\_Status** has changed to 'Closed' (**IF OLD.Auction\_Status != 'Closed' AND NEW.Auction\_Status = 'Closed'**) and if the **Auction\_End\_Time** is less than or equal to the current time (**AND NEW.Auction\_End\_Time <= NOW()**), indicating that the auction has ended.
2. **Generating Transaction ID:**
   * Generates a new transaction ID by incrementing the maximum transaction ID found in the **Transactions** table.
3. **Fetching Bidder Information:**
   * Retrieves the **Last\_Bidder** and **Last\_Bid** from the **Items** table based on the **Item\_ID** associated with the auction.
4. **Fetching Buyer ID:**
   * Retrieves the **Buyer\_ID** using the bidder's username obtained in the previous step.
5. **Inserting Transaction:**
   * Inserts a new row into the **Transactions** table with the generated transaction ID, buyer ID, seller ID (obtained from the **Items** table), item ID, last bid amount, current timestamp, payment method ('Online'), and transaction status ('Completed').
6. **Updating Buyer's Account Balance:**
   * Updates the buyer's account balance by subtracting the last bid amount (**@last\_bid**).

**Test:**

* The provided test doesn't explicitly show how the trigger works but assumes that an auction with a specific **Auction\_ID** has ended, triggering the update on the **Auctions** table. The trigger then automatically inserts a transaction into the **Transactions** table and updates the buyer's account balance.

**Example:**

UPDATE Auctions

SET Auction\_Status = 'Closed'

WHERE Auction\_ID = 16

-- Check the latest transaction

SELECT \* FROM Transactions ORDER BY Transaction\_ID DESC LIMIT 1;

-- Check the buyer's account balance

SELECT \* FROM Buyers WHERE Buyer\_ID = buyer\_id;

After executing this update statement, the trigger will automatically insert a transaction into the **Transactions** table and update the buyer's account balance if the conditions specified in the trigger are met.

1. Update Auction Status

TRIGGER CODE:

DELIMITER $$

CREATE TRIGGER update\_auction\_status

BEFORE INSERT ON auctions

FOR EACH ROW

BEGIN

    IF NEW.Auction\_End\_Time <= NOW() THEN

        SET NEW.Auction\_Status = 'Closed';

    END IF;

END$$

DELIMITER ;

test

-- Insert a new auction with Auction\_End\_Time set to a past time

INSERT INTO auctions (Auction\_ID, Item\_ID, Auction\_Start\_Time, Auction\_End\_Time, Auction\_Status, Reserve\_Price)

VALUES (99, 1, '2024-05-01 12:00:00', '2024-05-15 12:00:00', 'pen', 100.00);

-- Check the inserted auction

SELECT \* FROM auctions WHERE Auction\_ID = 99;

TRIGGER EXPLANATION:

This trigger, named **update\_auction\_status**, is designed to automatically update the **Auction\_Status** field when a new auction is inserted into the **auctions** table.

**Trigger Description:**

* **Trigger Name:** update\_auction\_status
* **Event:** BEFORE INSERT on the **auctions** table.
* **Trigger Type:** FOR EACH ROW, indicating that the trigger will execute once for each row that is being inserted into the **auctions** table.
* **Purpose:** Automatically updates the **Auction\_Status** field to 'Closed' if the **Auction\_End\_Time** for the new auction is in the past or equals the current time.

**Explanation:**

1. **IF Statement:**
   * Checks if the **Auction\_End\_Time** for the new auction (**NEW.Auction\_End\_Time**) is less than or equal to the current time (**NOW()**).
2. **Updating Auction Status:**
   * If the **Auction\_End\_Time** is indeed in the past or equals the current time, it sets the **Auction\_Status** for the new auction (**NEW.Auction\_Status**) to 'Closed'.

**Test:**

* The provided test inserts a new auction into the **auctions** table with an **Auction\_End\_Time** that is set to a past time ('2024-05-15 12:00:00'). This should trigger the execution of the trigger, automatically updating the **Auction\_Status** to 'Closed'.

**Example:**

-- Insert a new auction with Auction\_End\_Time set to a past time

INSERT INTO auctions (Auction\_ID, Item\_ID, Auction\_Start\_Time, Auction\_End\_Time, Auction\_Status, Reserve\_Price)

VALUES (99, 1, '2024-05-01 12:00:00', '2024-05-15 12:00:00', 'pen', 100.00);

-- Check the inserted auction

SELECT \* FROM auctions WHERE Auction\_ID = 99;

After executing this insert statement, the trigger will automatically update the **Auction\_Status** to 'Closed' for the newly inserted auction because its **Auction\_End\_Time** is in the past. When you check the inserted auction, you should see the updated **Auction\_Status** reflecting this change.

1. Create Bid Entry on Update in items

TRIGGER CODE:

DELIMITER $$

DROP TRIGGER IF EXISTS create\_bid\_entry\_after\_update $$

CREATE TRIGGER create\_bid\_entry\_after\_update

AFTER UPDATE ON items

FOR EACH ROW

BEGIN

    DECLARE bidder\_id INT;

    DECLARE last\_bid DECIMAL(10, 2);

    DECLARE bid\_increment DECIMAL(10, 2);

    SET bidder\_id = (SELECT Buyer\_ID FROM buyers WHERE Username = NEW.Last\_Bidder);

    SELECT Last\_Bid INTO last\_bid FROM items WHERE Item\_ID = NEW.Item\_ID;

    SET bid\_increment = GREATEST(NEW.Last\_Bid - COALESCE(last\_bid, 0), 0);

    INSERT INTO bids (Bid\_ID, Bidder\_ID, Item\_ID, Bid\_Amount, Bid\_Time, Bid\_Status, Bid\_Increment)

    SELECT COALESCE(MAX(Bid\_ID), 0) + 1, bidder\_id, NEW.Item\_ID, NEW.Last\_Bid, NOW(), 'Outbid', bid\_increment FROM bids;

END$$

DELIMITER ;

TRIGGER EXPLANATION:

This trigger, named **create\_bid\_entry\_after\_update**, is designed to automatically create a new bid entry in the **bids** table after an update on the **items** table.

**Trigger Description:**

* **Trigger Name:** create\_bid\_entry\_after\_update
* **Event:** AFTER UPDATE on the **items** table.
* **Trigger Type:** FOR EACH ROW, indicating that the trigger will execute once for each row that is updated in the **items** table.
* **Purpose:** Automatically creates a new bid entry in the **bids** table when the **Last\_Bid** field of an item is updated.

**Explanation:**

1. **Declaration of Variables:**
   * Declares three variables: **bidder\_id** to store the ID of the bidder, **last\_bid** to store the last bid amount for the item, and **bid\_increment** to store the difference between the new bid amount and the previous bid amount.
2. **Fetching Bidder ID:**
   * Sets the **bidder\_id** variable by selecting the **Buyer\_ID** from the **buyers** table based on the username (**NEW.Last\_Bidder**) provided in the updated row.
3. **Fetching Last Bid:**
   * Selects the **Last\_Bid** from the **items** table based on the **Item\_ID** of the updated row and stores it in the **last\_bid** variable.
4. **Calculating Bid Increment:**
   * Calculates the bid increment by finding the difference between the new bid amount (**NEW.Last\_Bid**) and the previous bid amount (**last\_bid**). It ensures that the bid increment is always non-negative.
5. **Inserting Bid Entry:**
   * Inserts a new row into the **bids** table with the following values:
     + **Bid\_ID**: The maximum bid ID incremented by 1.
     + **Bidder\_ID**: The ID of the bidder fetched earlier.
     + **Item\_ID**: The **Item\_ID** of the updated row.
     + **Bid\_Amount**: The new bid amount (**NEW.Last\_Bid**).
     + **Bid\_Time**: The current timestamp.
     + **Bid\_Status**: 'Outbid' indicating that the previous bid has been outbid.
     + **Bid\_Increment**: The calculated bid increment.

**Test:**

* This trigger is automatically executed after an update is performed on the **items** table. You can simulate this by updating the **Last\_Bid** field of an item in the **items** table.

**Example:**

-- Update the Last\_Bid field of an item in the items table

UPDATE items SET Last\_Bid = 150.00 WHERE Item\_ID = 1;

-- This update will trigger the execution of the trigger create\_bid\_entry\_after\_update

After executing this update statement, the trigger will automatically create a new bid entry in the **bids** table based on the updated **Last\_Bid** value for the item with **Item\_ID** 1.

b) Write PROCEDURES to minimize the complexity of AGGREGATE functions used in SQL Query, to create NEW FUNCTIONS for which AGGREGATE function is not available, etc.

**PROCEDURES:**

1. **CalculateAverageTransaction**: This procedure calculates the average transaction amount for a given seller ID. It takes the seller ID as input and calculates the average transaction amount by querying the Transactions table for transactions made by that seller. The result is then returned.

PROCEDURE CODE:

DELIMITER //

CREATE PROCEDURE CalculateAverageTransaction(IN sellerID INT)

BEGIN

    DECLARE avgTransaction DECIMAL(10, 2);

    SELECT AVG(Transaction\_Amount) INTO avgTransaction

    FROM Transactions

    WHERE Seller\_ID = sellerID;

    SELECT avgTransaction;

END //

DELIMITER ;

PROCEDURE EXAMPLE:

CALL CalculateAverageTransaction(1);

1. **CountBidsForItem**: This procedure counts the number of bids for a given item ID. It takes the item ID as input and calculates the count by querying the Bids table for bids made on that item. The result is then returned.

PROCEDURE CODE:

DELIMITER //

CREATE PROCEDURE CountBidsForItem(IN itemID INT)

BEGIN

    DECLARE bidCount INT;

    SELECT COUNT(\*) INTO bidCount

    FROM Bids

    WHERE Item\_ID = itemID;

    SELECT bidCount;

END //

DELIMITER ;

PROCEDURE EXAMPLE:

CALL CountBidsForItem(1);

1. **FindMaxBidAmountForItem**: This function finds the maximum bid amount for a given item ID. It takes the item ID as input and calculates the maximum bid amount by querying the Bids table for bids made on that item. The maximum bid amount is then returned.

PROCEDURE CODE:

DELIMITER //

CREATE FUNCTION FindMaxBidAmountForItem(itemID INT) RETURNS DECIMAL(10, 2)

BEGIN

    DECLARE maxBid DECIMAL(10, 2);

    SELECT MAX(Bid\_Amount) INTO maxBid

    FROM Bids

    WHERE Item\_ID = itemID;

    RETURN maxBid;

END //

DELIMITER ;

PROCEDURE EXAMPLE:

SELECT FindMaxBidAmountForItem(1);

1. **CalculateTotalSalesAmount**: This procedure calculates the total sales amount for a given seller ID. It takes the seller ID as input and calculates the total sales amount by querying the Transactions table for transactions made by that seller. The result is then returned.

PROCEDURE CODE:

DELIMITER //

CREATE PROCEDURE CalculateTotalSalesAmount(IN sellerID INT)

BEGIN

    DECLARE totalSales DECIMAL(10, 2);

    SELECT SUM(Transaction\_Amount) INTO totalSales

    FROM Transactions

    WHERE Seller\_ID = sellerID;

    SELECT totalSales;

END //

DELIMITER ;

PROCEDURE EXAMPLE:

CALL CalculateTotalSalesAmount(1);

1. **CalculateAverageAuctionDuration**: This procedure calculates the average duration of auctions in a given category. It takes the category as input and calculates the average duration by querying the Auctions and Items tables, joining them on the Item\_ID, and filtering by the specified category. The duration of each auction is calculated using the difference between the auction start and end times. The average duration is then returned.

PROCEDURE CODE:

DELIMITER //

DROP PROCEDURE IF EXISTS CalculateAverageAuctionDuration;

DELIMITER //

CREATE PROCEDURE CalculateAverageAuctionDuration(IN category VARCHAR(50))

BEGIN

    DECLARE avgDuration DECIMAL(10, 2);

    SELECT AVG(TIMESTAMPDIFF(SECOND, Auctions.Auction\_Start\_Time, Auctions.Auction\_End\_Time)) INTO avgDuration

    FROM Auctions

    INNER JOIN Items ON Auctions.Item\_ID = Items.Item\_ID

    WHERE Items.Category = category;

    SELECT avgDuration;

END //

DELIMITER ;

PROCEDURE EXAMPLE:

CALL CalculateAverageAuctionDuration('Electronics');

1. **CountItemsSoldByBuyer**: This function counts the number of items sold by a given buyer ID. It takes the buyer ID as input and calculates the count by querying the Transactions table for transactions where the specified buyer is the buyer. The count is then returned.

PROCEDURE CODE:

DELIMITER //

CREATE FUNCTION CountItemsSoldByBuyer(buyerID INT) RETURNS INT

BEGIN

    DECLARE itemCount INT;

    SELECT COUNT(\*) INTO itemCount

    FROM Transactions

    WHERE Buyer\_ID = buyerID;

    RETURN itemCount;

END //

DELIMITER ;

PROCEDURE EXAMPLE:

SELECT CountItemsSoldByBuyer(1);

**Declare which part of the database has triggers and procedures defined:**

In most modern database systems, the Information Schema contains views related to triggers, providing insights into their definitions and associated objects. Commonly, these views include:

* **Triggers View**: This view typically contains metadata about triggers defined in the database, including their names, associated tables, events, and actions.
* **Trigger Columns View**: Some systems offer this view to provide details about the columns affected by each trigger.

**Procedures in the Information Schema:**

Similar to triggers, procedures also have their metadata exposed through the Information Schema, facilitating querying and analysis. Relevant views may include:

* **Routines View**: This view usually lists all routines (including procedures and functions) defined in the database, along with their names, types, and other attributes.
* **Routine Parameters View**: If procedures accept parameters, this view provides information about the parameters, such as names, data types, and positions.

TO VIEW ALL TRIGGERS:

SELECT

    TRIGGER\_NAME,

    EVENT\_OBJECT\_TABLE

FROM

    INFORMATION\_SCHEMA.TRIGGERS;



TO VIEW ALL PROCEDURES:

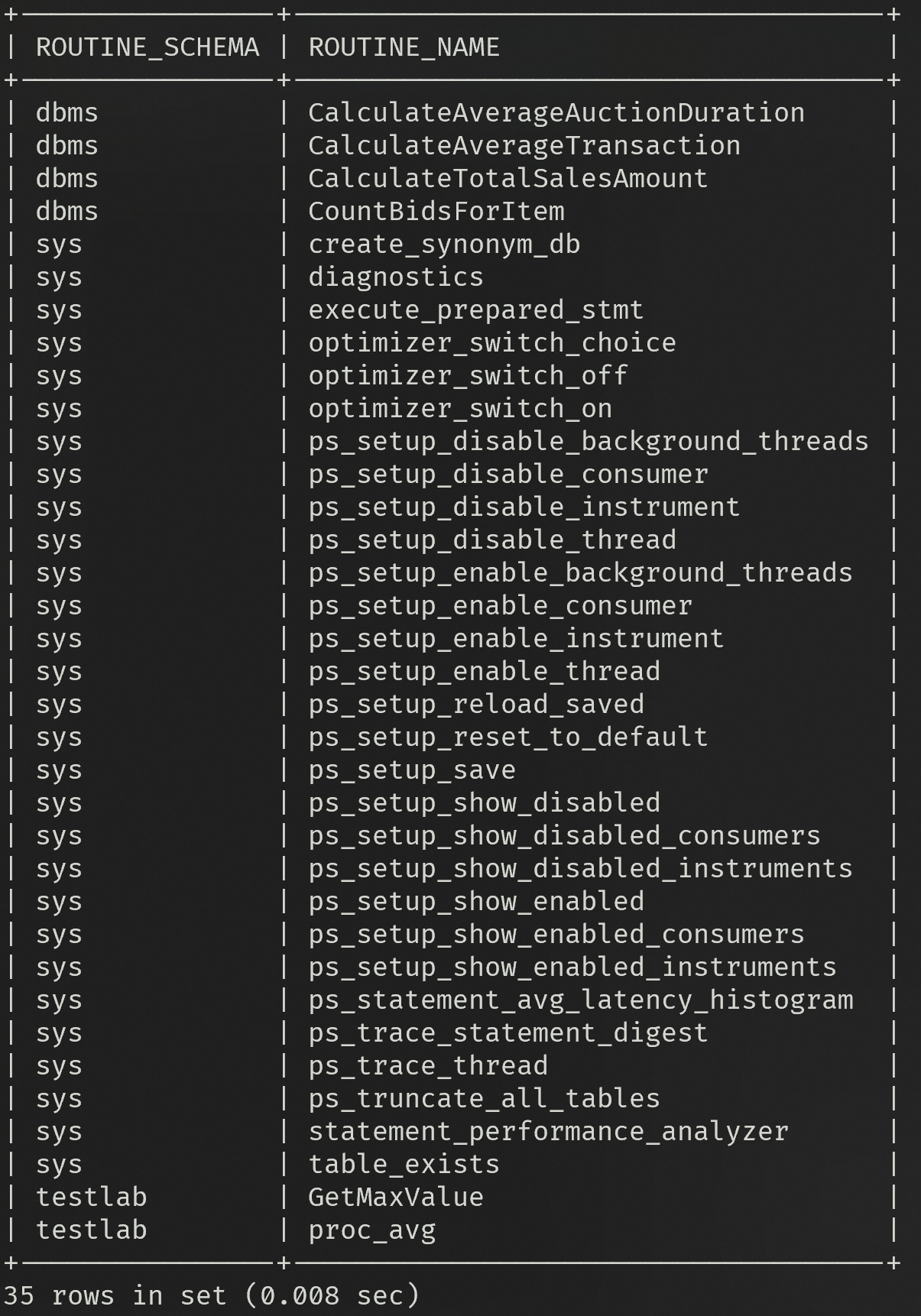
SELECT

  ROUTINE\_SCHEMA,

  ROUTINE\_NAME

FROM INFORMATION\_SCHEMA.ROUTINES

WHERE ROUTINE\_TYPE = 'PROCEDURE';



**Explain the Data Dictionary details for Triggers and Procedures.**

Triggers and procedures are defined within the schema of the online auction system database. Triggers are defined to enforce data integrity, handle automatic updates, and perform certain actions upon specific events in the database tables. Procedures are defined to encapsulate reusable logic for performing calculations or data analysis within the database.

Specifically, the triggers are defined for the following tables:

* Buyers
* Sellers
* Items
* Bids
* Auctions
* Transactions

The procedures are defined to perform various calculations and data analysis tasks within the database, such as calculating averages, counting bids, finding maximum bid amounts, and calculating total sales amounts.

**Triggers:**

1. **trg\_unique\_username:**
   * **Description:** Ensures uniqueness of usernames in the Buyers table.
   * **Event:** BEFORE INSERT ON Buyers
   * **Action:** Checks if the inserted username already exists in the Buyers table and signals an error if it does.
2. **trg\_unique\_username\_seller:**
   * **Description:** Ensures uniqueness of usernames in the Sellers table.
   * **Event:** BEFORE INSERT ON Sellers
   * **Action:** Checks if the inserted username already exists in the Sellers table and signals an error if it does.
3. **set\_last\_bid\_default:**
   * **Description:** Sets the Last\_Bid field to the Starting\_Price if it is null when inserting into the Items table.
   * **Event:** BEFORE INSERT ON Items
   * **Action:** If the Last\_Bid field is null, it sets it to the Starting\_Price.
4. **trg\_update\_last\_bid:**
   * **Description:** Updates the Last\_Bidder and Last\_Bid fields in the Items table after a bid is inserted.
   * **Event:** AFTER INSERT ON Bids
   * **Action:** Updates the Last\_Bidder and Last\_Bid fields in the Items table based on the inserted bid.
5. **trg\_update\_buyer\_balance:**
   * **Description:** Updates the Account\_Balance of the buyer after a transaction is inserted, checking for sufficient balance.
   * **Event:** AFTER INSERT ON Transactions
   * **Action:** Checks if the buyer has sufficient balance for the transaction and updates the Account\_Balance accordingly.
6. **trg\_update\_seller\_balance:**
   * **Description:** Updates the Account\_Balance of the seller after a transaction is inserted.
   * **Event:** AFTER INSERT ON Transactions
   * **Action:** Updates the Account\_Balance of the seller based on the inserted transaction.
7. **after\_auction\_update\_set\_winning\_bid\_trigger:**
   * **Description:** Sets the winning bid for an item after an auction is closed.
   * **Event:** AFTER UPDATE ON Auctions
   * **Action:** Sets the winning bid for an item if the auction status is changed to 'Closed'.
8. **end\_auction\_insert\_transactions\_trigger:**
   * **Description:** Inserts a transaction after an auction ends.
   * **Event:** BEFORE UPDATE ON Auctions
   * **Action:** Inserts a transaction and updates buyer's account balance after an auction ends.
9. **update\_auction\_status:**
   * **Description:** Updates the Auction\_Status to 'Closed' if the Auction\_End\_Time is in the past when inserting into the Auctions table.
   * **Event:** BEFORE INSERT ON Auctions
   * **Action:** Sets the Auction\_Status to 'Closed' if the Auction\_End\_Time is in the past.
10. **create\_bid\_entry\_after\_update:**

* **Description:** Inserts a bid entry after an item's Last\_Bid is updated.
* **Event:** AFTER UPDATE ON Items
* **Action:** Inserts a bid entry with the updated Last\_Bid value.

**Procedures:**

1. **CalculateAverageTransaction:**
   * **Description:** Calculates the average transaction amount for a given seller.
   * **Parameters:** sellerID (INT)
   * **Returns:** avgTransaction (DECIMAL)
2. **CountBidsForItem:**
   * **Description:** Counts the number of bids for a given item.
   * **Parameters:** itemID (INT)
   * **Returns:** bidCount (INT)
3. **FindMaxBidAmountForItem:**
   * **Description:** Finds the maximum bid amount for a given item.
   * **Parameters:** itemID (INT)
   * **Returns:** maxBid (DECIMAL)
4. **CalculateTotalSalesAmount:**
   * **Description:** Calculates the total sales amount for a given seller.
   * **Parameters:** sellerID (INT)
   * **Returns:** totalSales (DECIMAL)
5. **CalculateAverageAuctionDuration:**
   * **Description:** Calculates the average duration of auctions for a given category.
   * **Parameters:** category (VARCHAR)
   * **Returns:** avgDuration (DECIMAL)
6. **CountItemsSoldByBuyer:**
   * **Description:** Counts the number of items sold by a given buyer.
   * **Parameters:** buyerID (INT)
   * **Returns:** itemCount (INT)

**Write the SQL Queries for fetching/processing triggers and procedures details.**

**Fetch Triggers Details:**

-- Fetch all triggers in the database

SHOW TRIGGERS;

-- Fetch details of a specific trigger

SHOW CREATE TRIGGER trg\_name;

**Fetch Procedures Details:**

-- Fetch all procedures in the database

SHOW PROCEDURES;

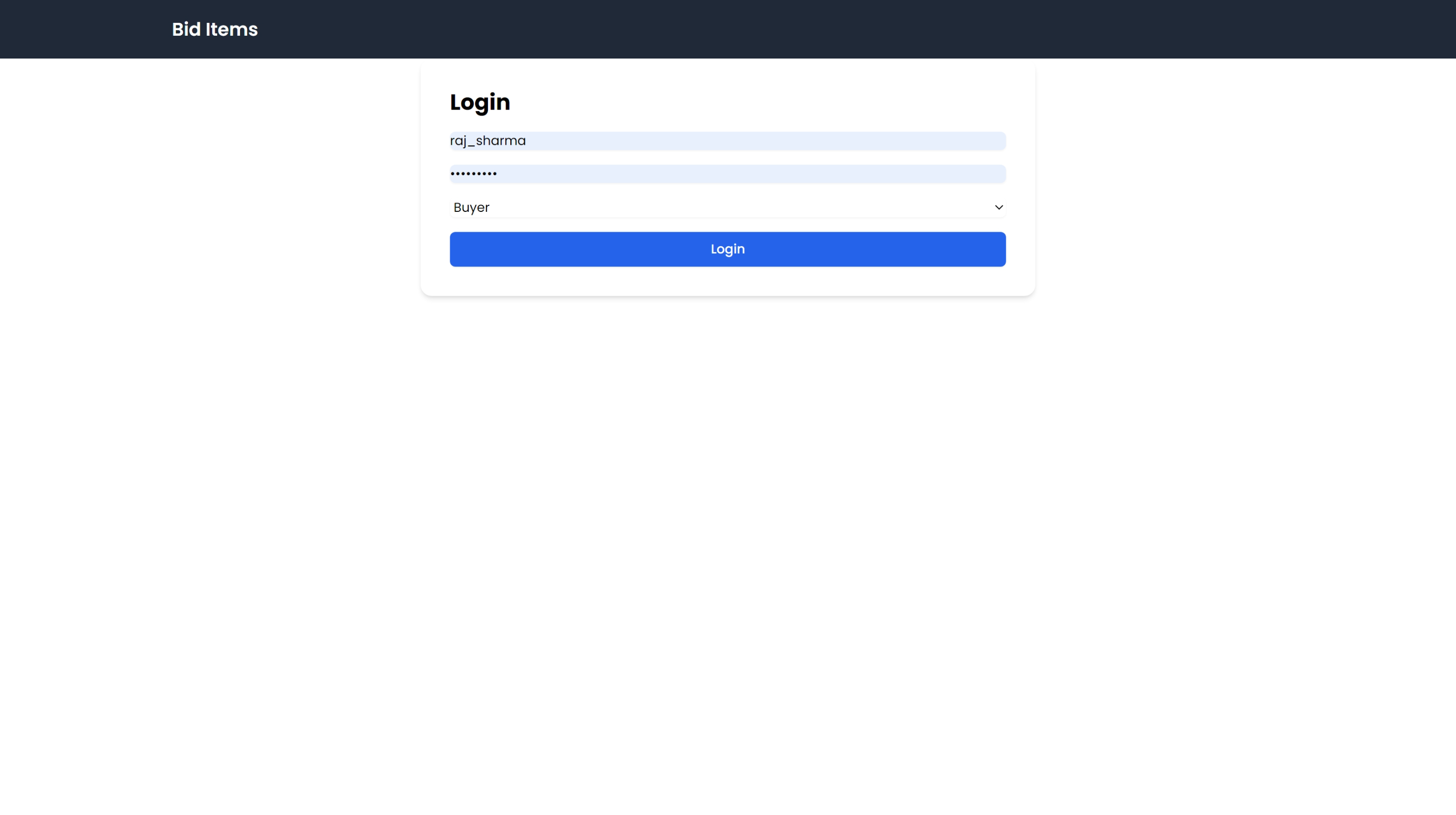
-- Fetch details of a specific procedure

SHOW CREATE PROCEDURE proc\_name;

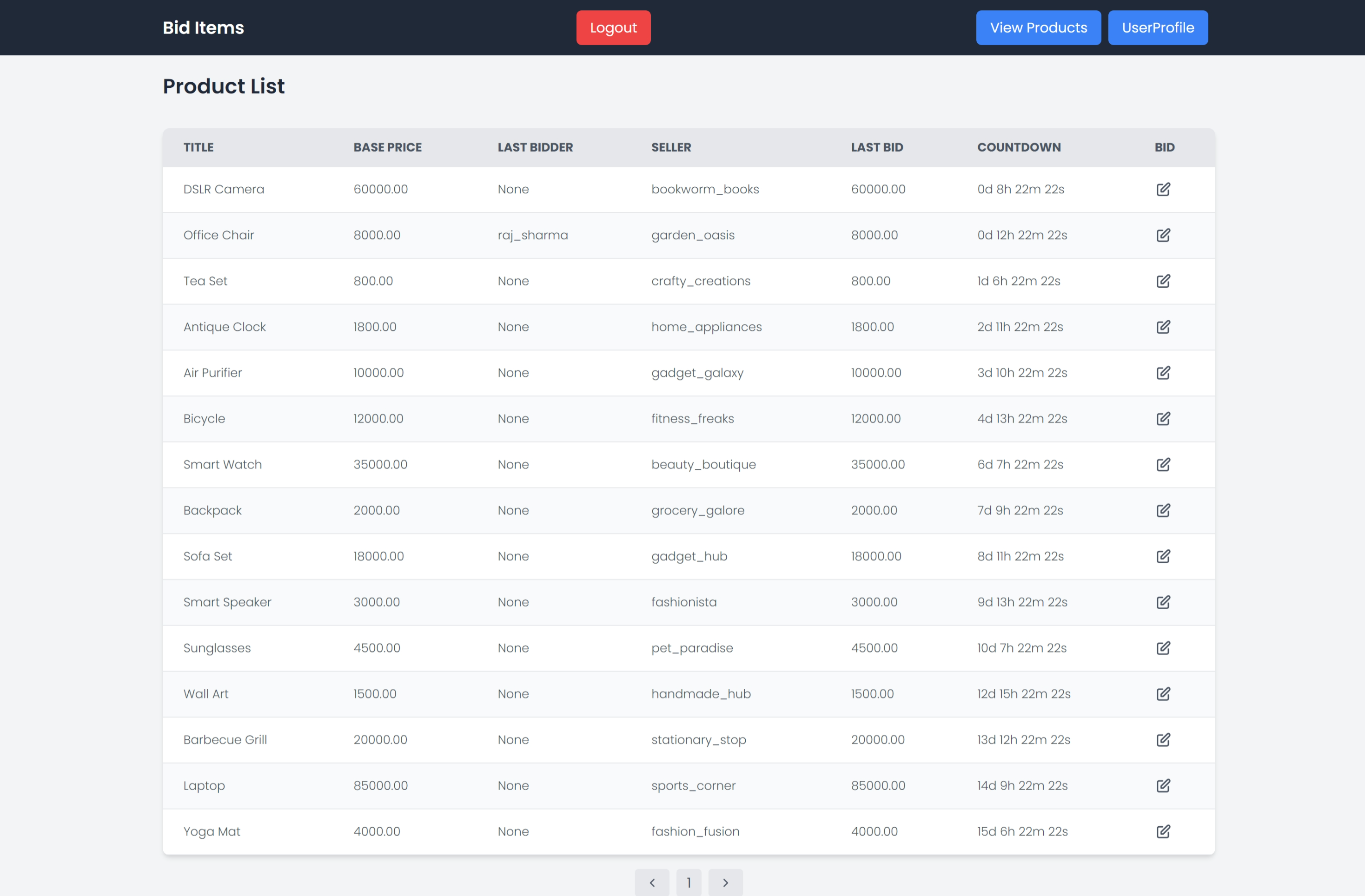
**ASSIGNMENT 4**

**BUYER PORTAL**

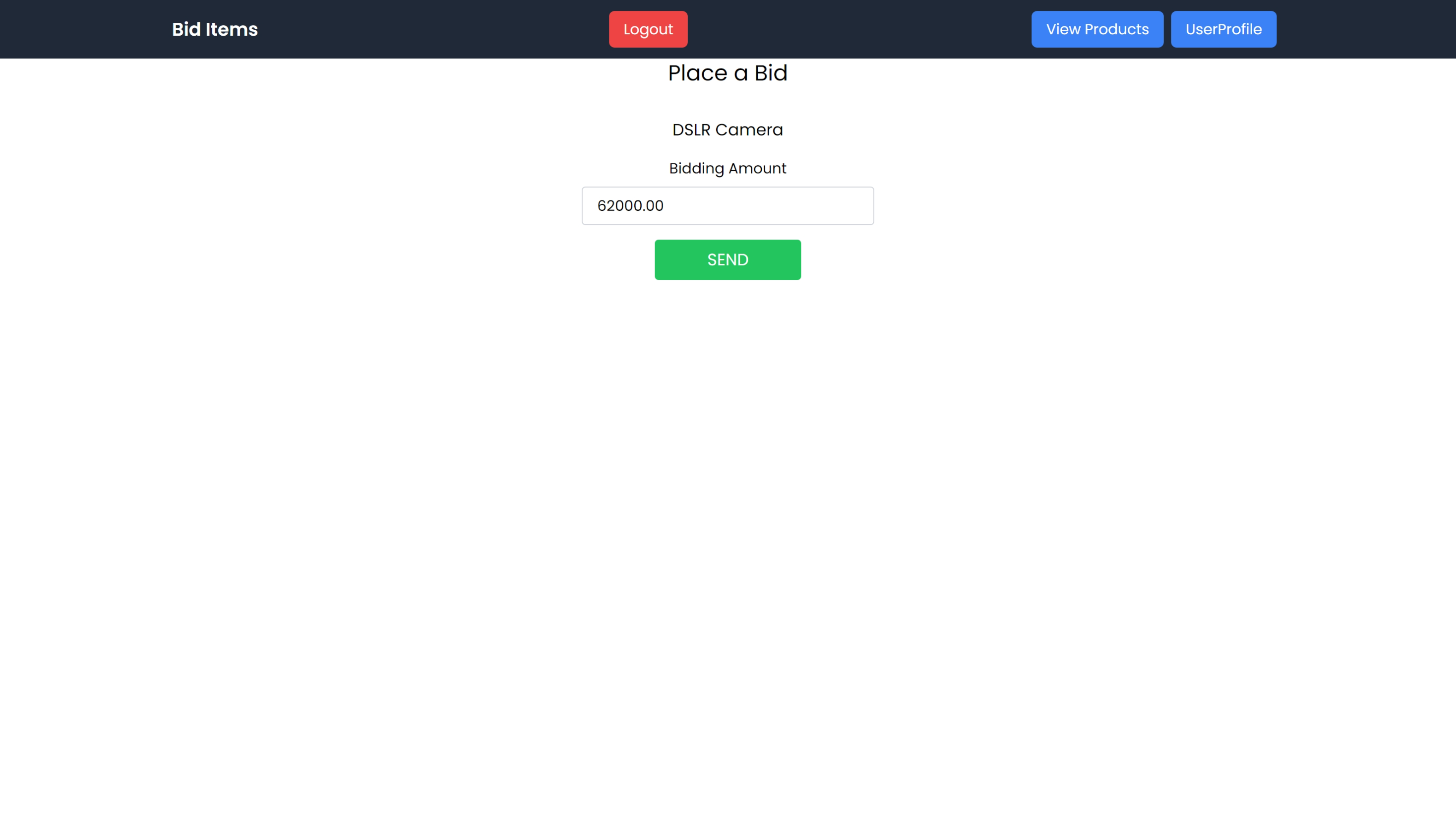
**Login Page:**

****

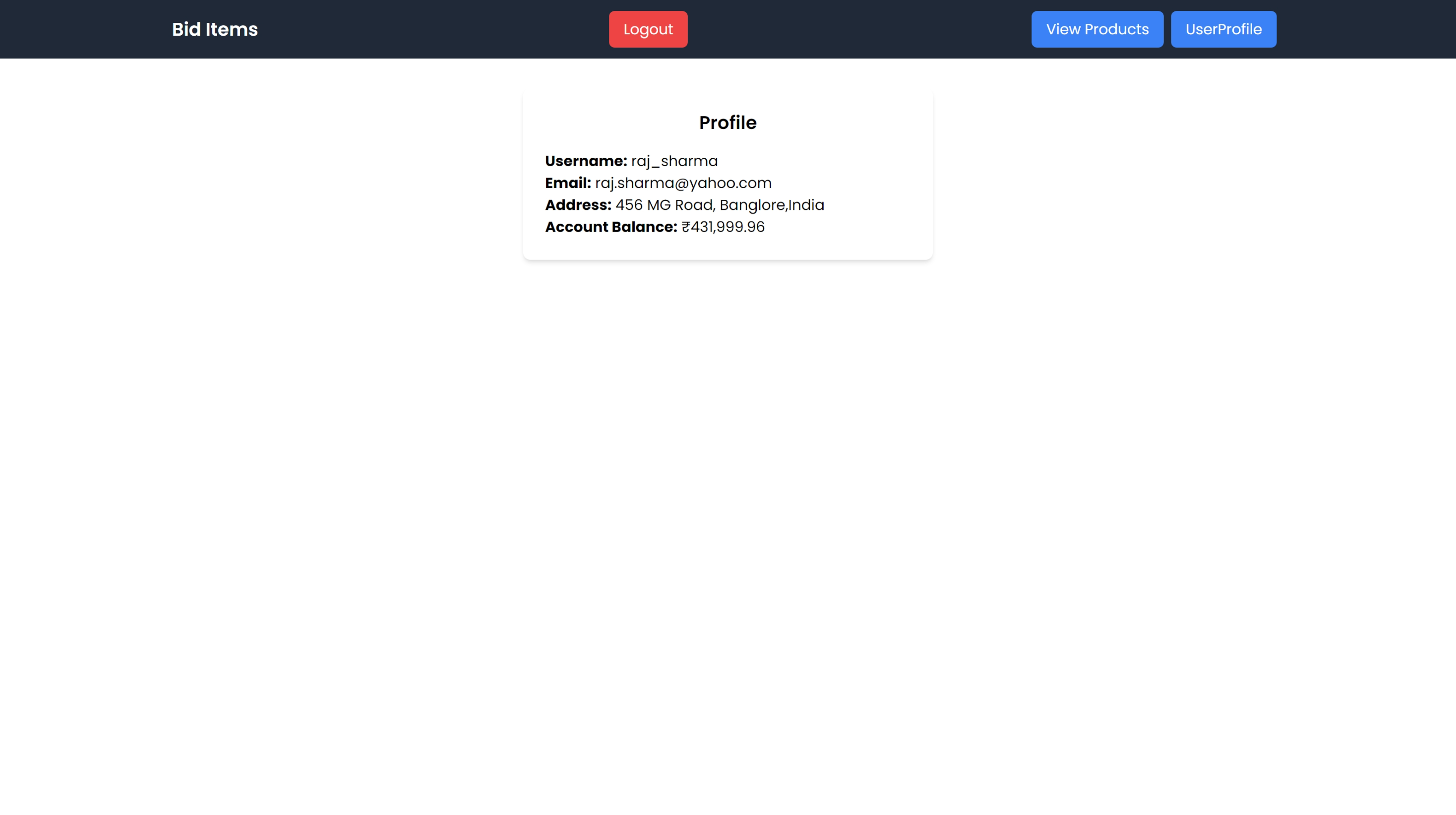
**View Products Page:**

****

**Bid on Product Page:**

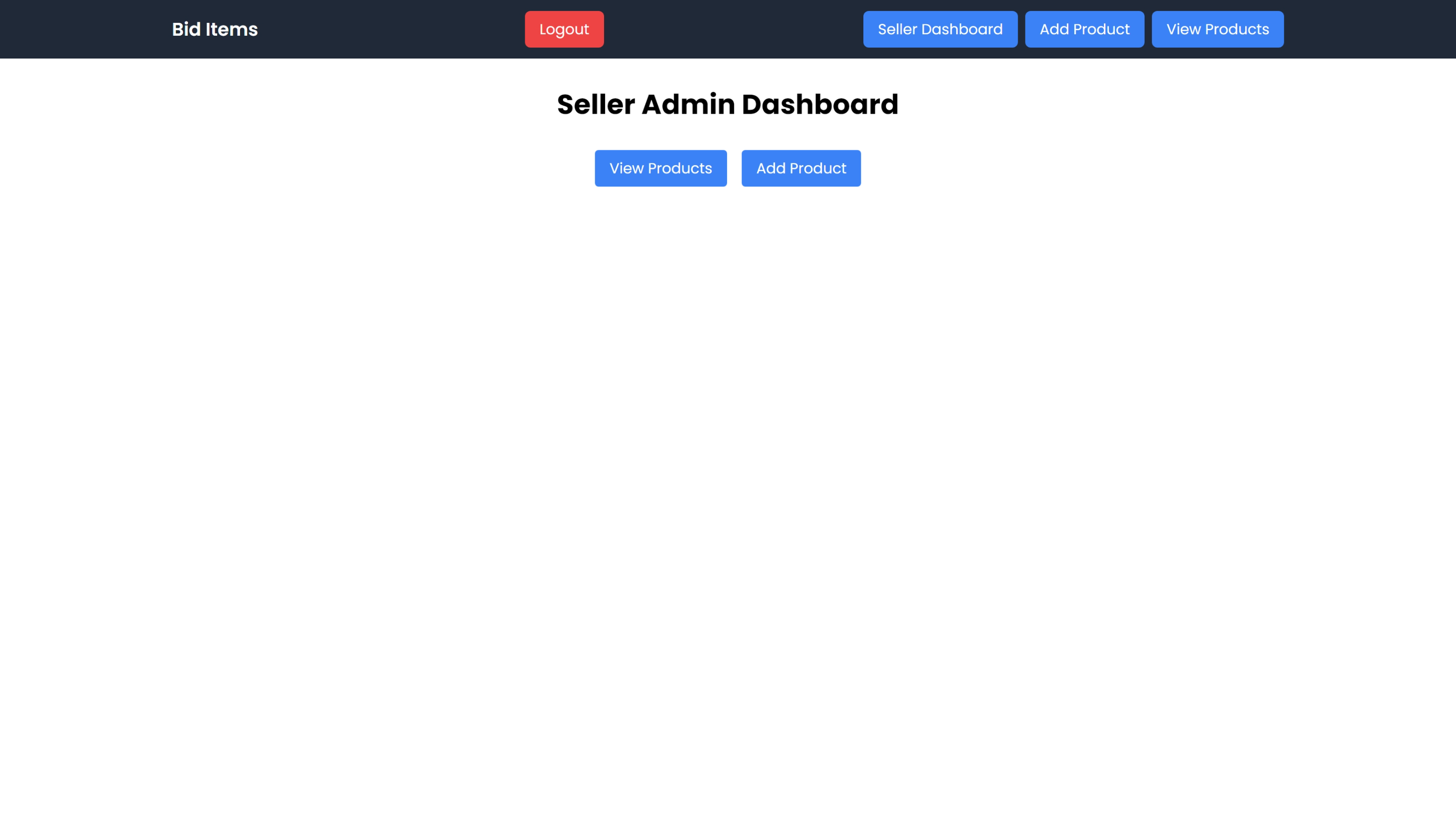
****

**User Profile Page:**

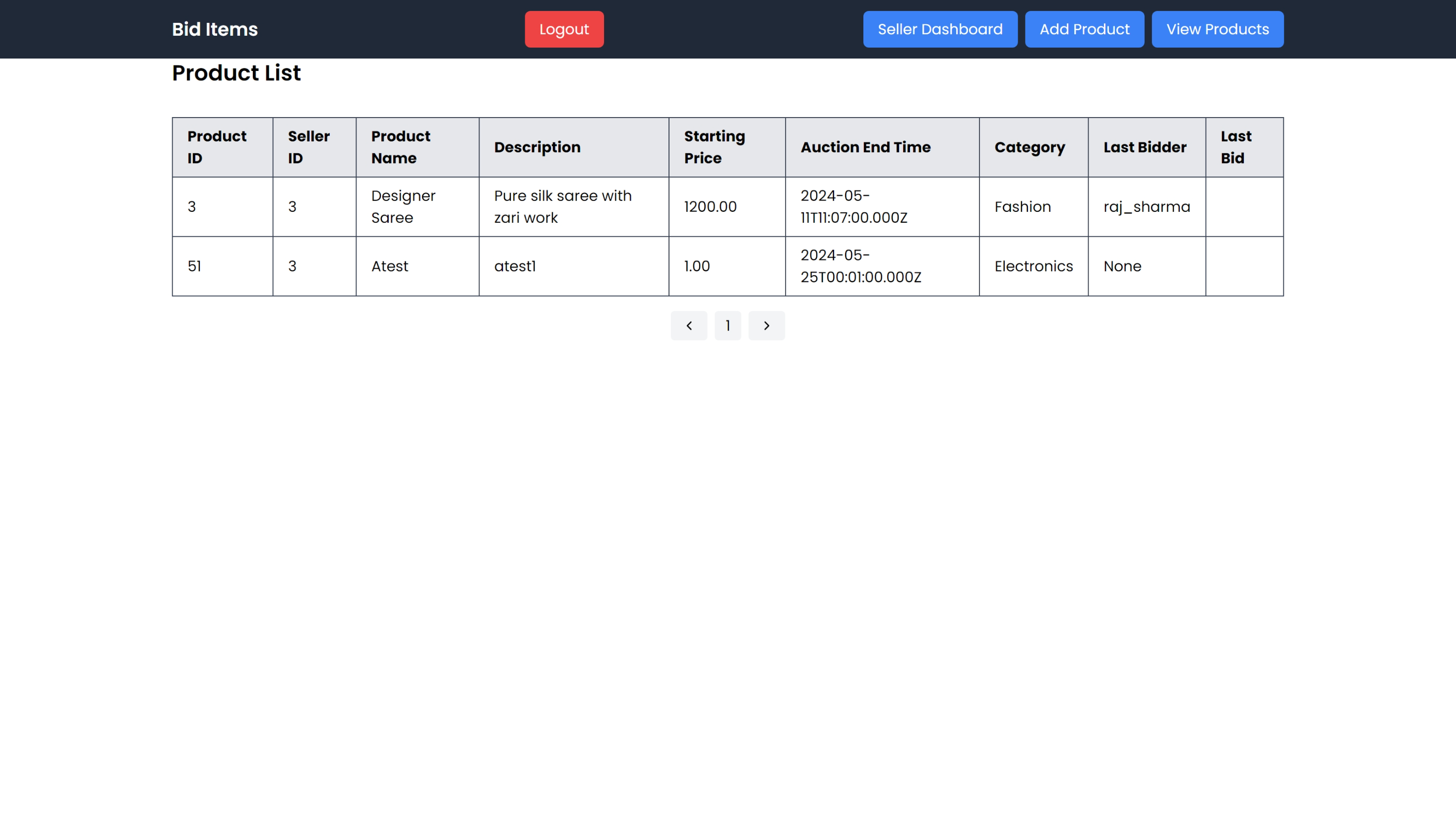
****

**SELLER PORTAL**

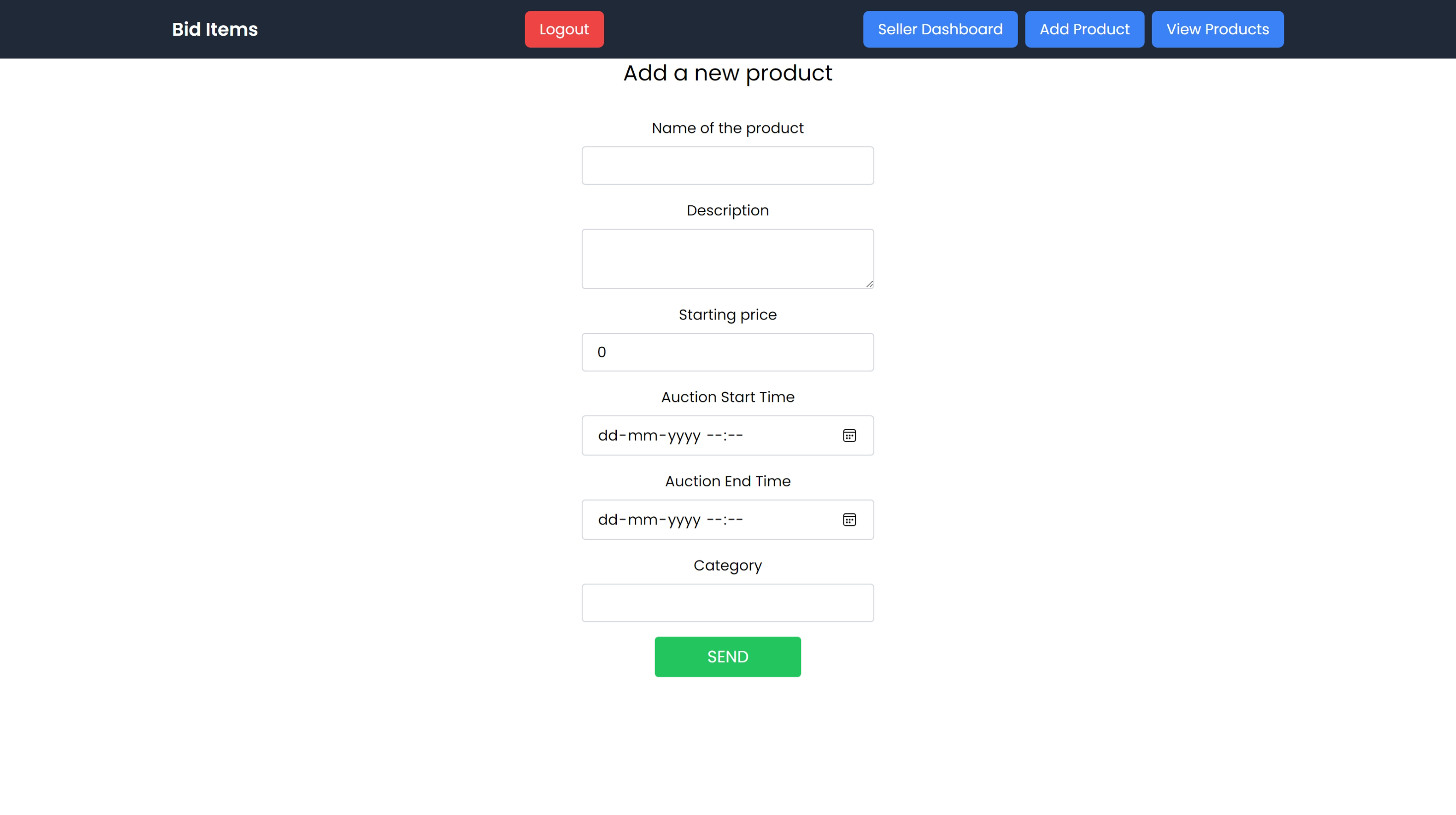
**SELLER DASHBOARD PAGE:**

****

**PRODUCTS LISTED BY SELLER PAGE:**

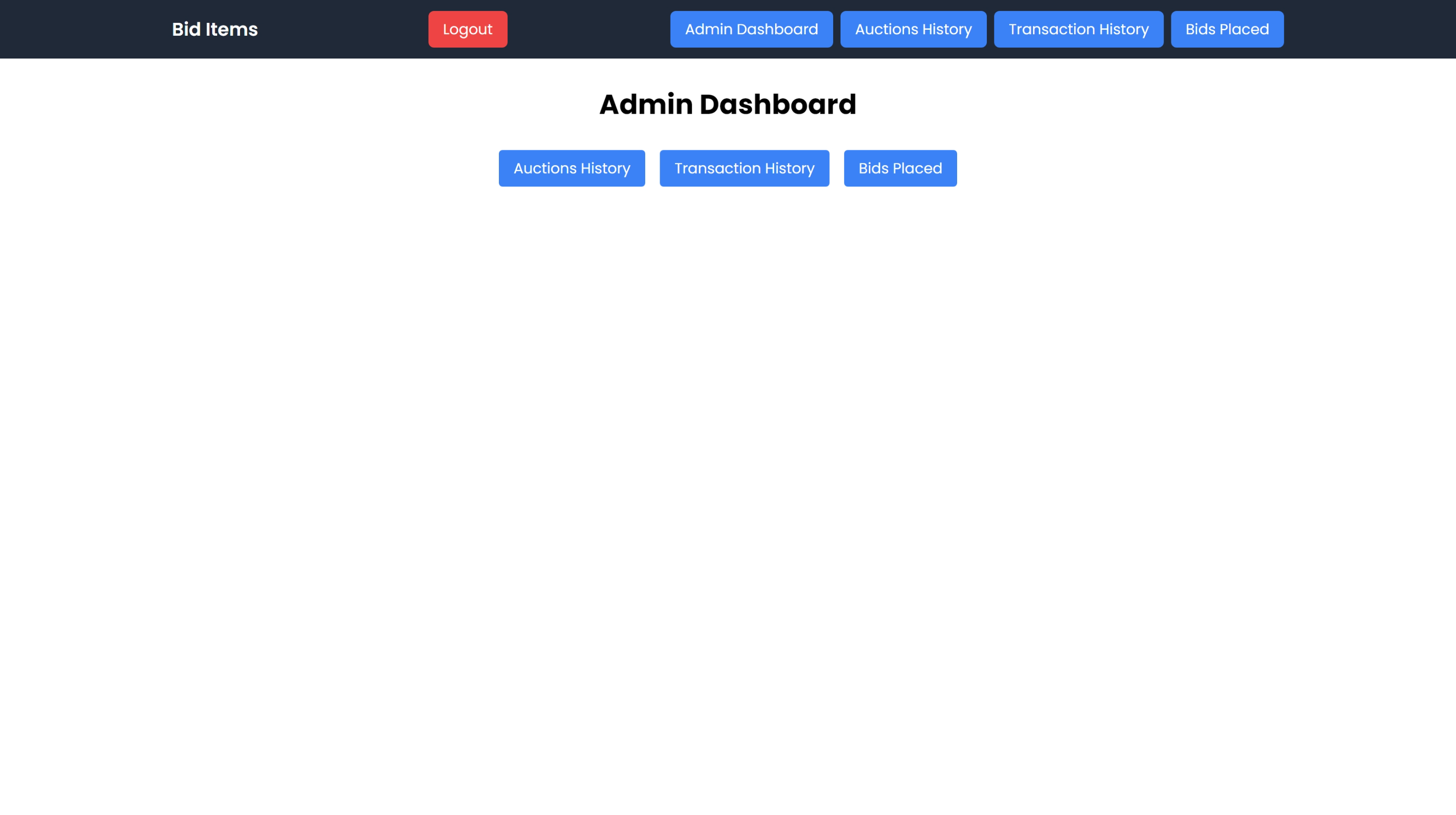
****

**LIST NEW PRODUCT PAGE:**

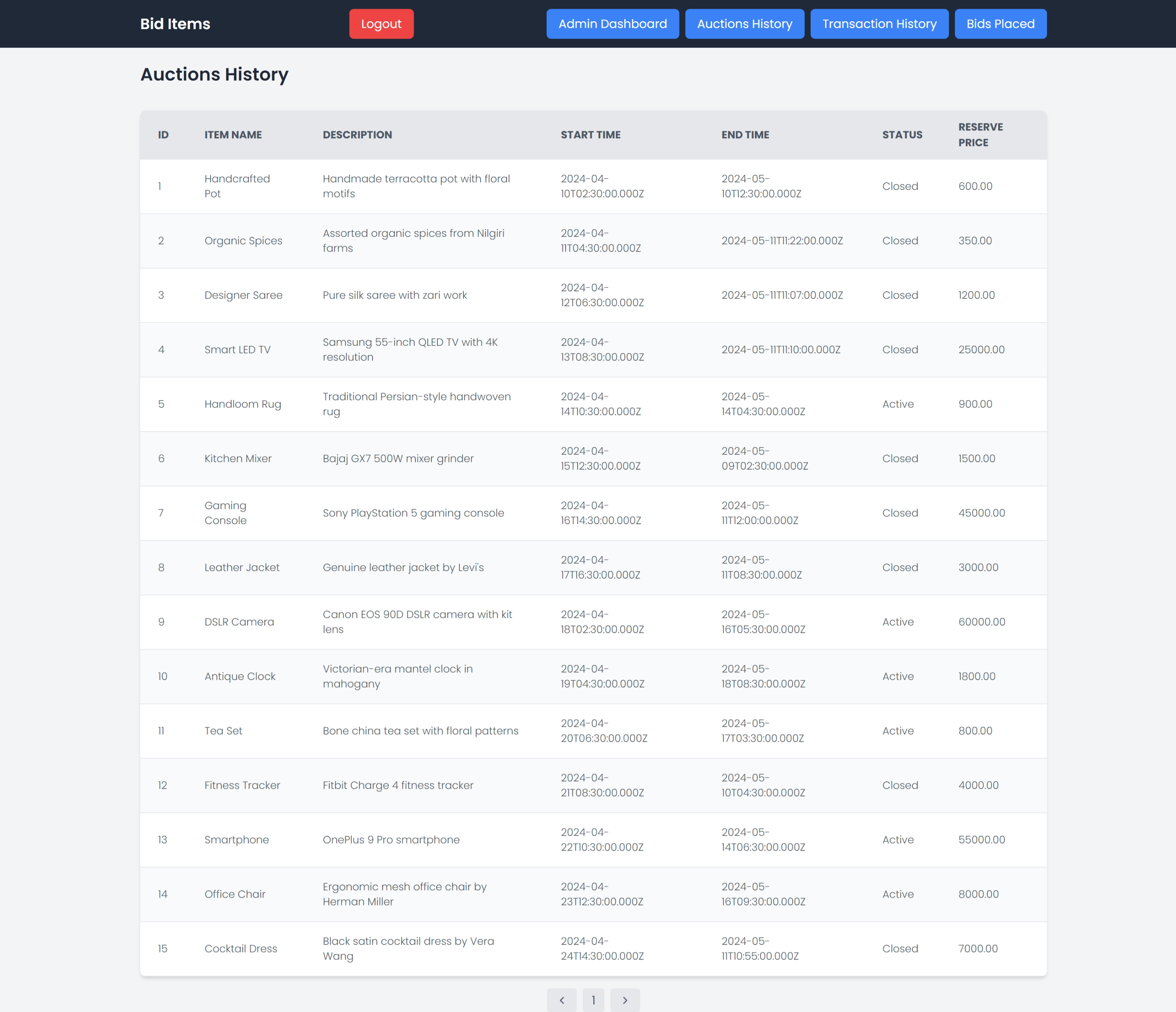
****

**ADMIN PORTAL**

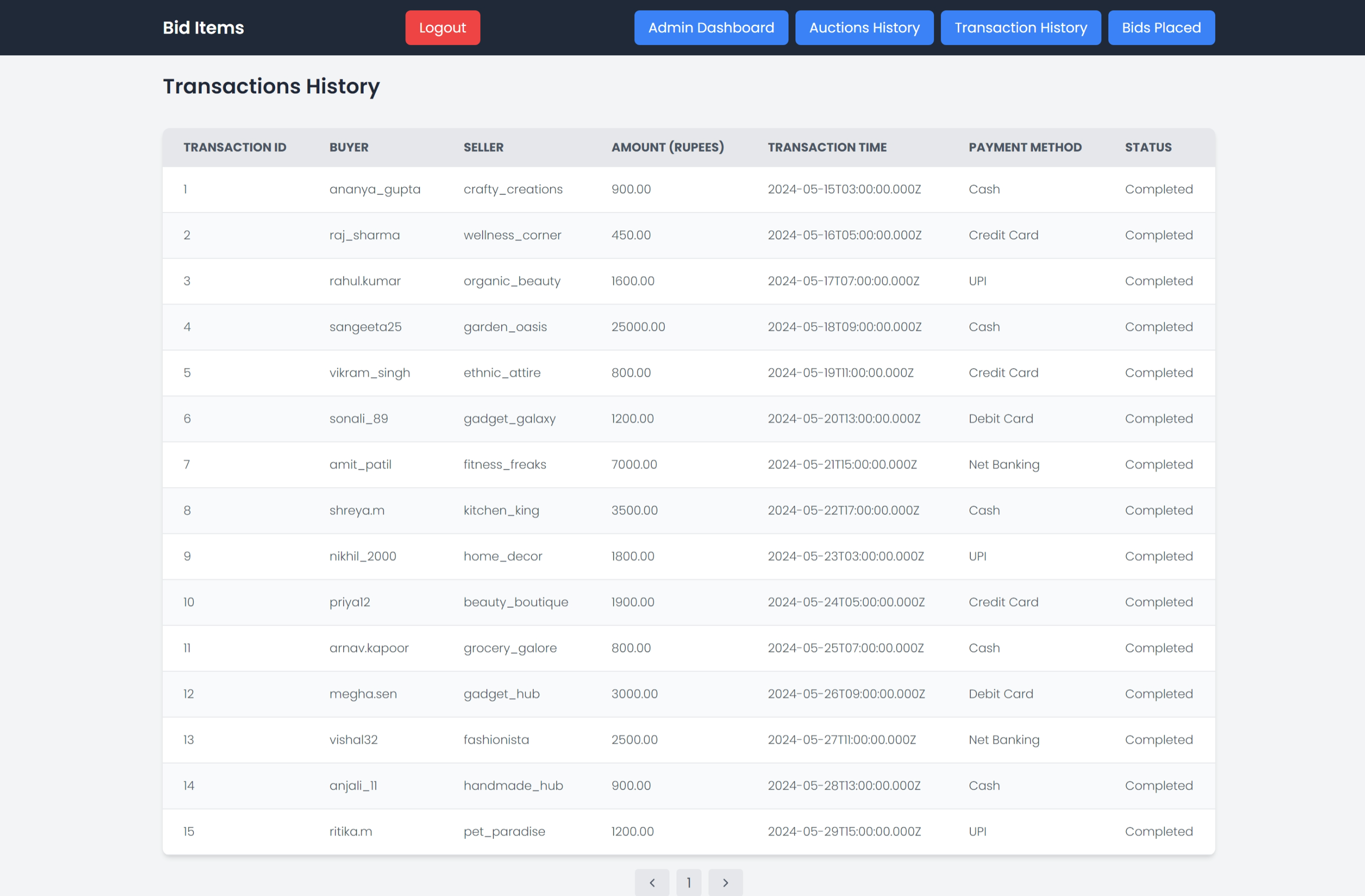
**ADMIN DASHBOARD PAGE:**

****

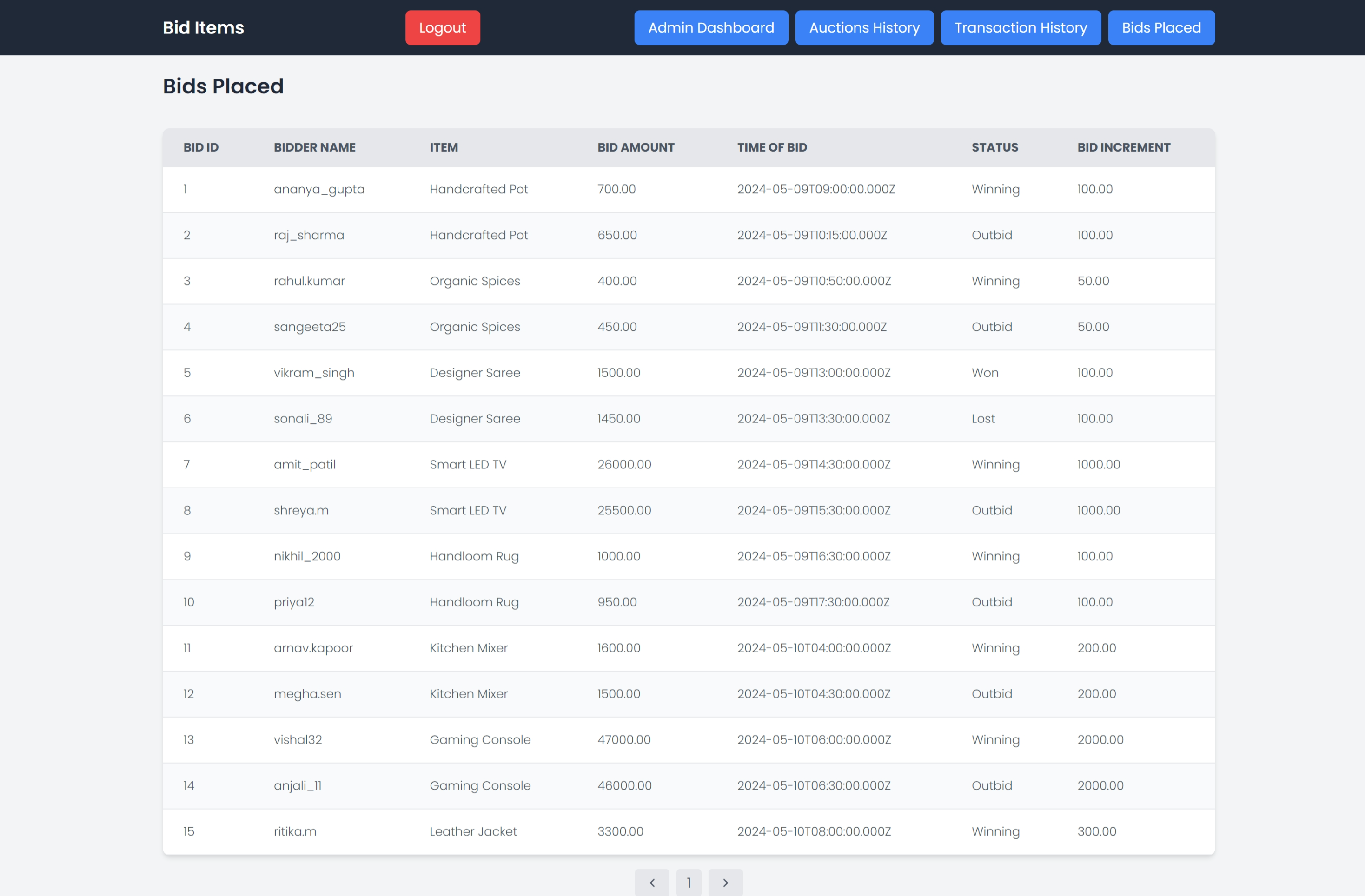
**AUCTIONS HISTORY PAGE:**

****

**TRANSACTIONS HISTORY PAGE:**

****

**BIDS PLACED PAGE:**

****