Project Report

TPG team: Jeffery Zeng, Jiaxin Sun, Yutong Tian,

1 Languages and Frameworks Used

- Front-end: TypeScript, React
- Back-end: Python, Flask

2 Schema Changes

- Changed the password attribute of the Person table from varchar(100) to varchar(512) to accommodate the storage of hashed passwords.
- Set the default value of Item.hasPieces as FALSE for below checks of Piece numbers.
- Add ON DELETE CASCADE to ensure consistency of data in test:
  - FOREIGN KEY (roleID) REFERENCES Role(roleID)
  - FOREIGN KEY (orderID) REFERENCES Ordered(orderID)
- Add an index idx_item_category to Item table to speed up the search.

3 Additional Constraints

- Triggers
  - Add triggers to trace the change of hasPieces in Item table
  - Add triggers to restrict the selectable options for the status attribute in the Delivered table.
- Website Access
  - Limit Feature 7 (Prepare) and 10 (Update) to volunteers and staff only.
  - Limit Feature 6 (shopping) to staff only.

4 Main Queries

For each feature implemented, present the main SQL queries:

- Feature 1: Login & User Session Handling

```sql
-- SQL query to check if the username already exists
SELECT *
FROM Person
WHERE userName = %s;

-- SQL query to insert a new user into the Person table
INSERT INTO Person (userName, password, fname, lname,
   email)
VALUES (%s, %s, %s, %s, %s);

-- SQL query to insert phone numbers into the PersonPhone
    table
INSERT INTO PersonPhone (userName, phone)
VALUES (%s, %s);

-- SQL query to insert user role into the Act table
INSERT INTO Act (userName, roleID)
VALUES (%s, %s);

-- SQL query to fetch user information for login
SELECT *
FROM Person
WHERE userName = %s;

-- SQL query to fetch the role of the user
SELECT roleID
FROM Act
WHERE userName = %s;
```

- Feature 2: Find Single Item

```sql

-- SQL query to find item locations excluding pieces
   ready for delivery
SELECT p.pieceNum, p.pDescription, p.length, p.width, p.
   height, p.roomNum, p.shelfNum, p.pNotes
FROM Piece p
WHERE p.ItemID = %s AND NOT (p.roomNum = -1 AND p.
   shelfNum = -1);
```

- Feature 3: Find Order Items

```sql

-- SQL query to find items in the specified order
-- with additional piece information
SELECT
    i.ItemID,
    i.iDescription,
    p.pieceNum,
    p.pDescription,
    p.length,
```

```
        p.width,
        p.height,
        p.roomNum,
        p.shelfNum
    FROM Item i
    NATURAL JOIN ItemIn ii
    NATURAL JOIN Ordered o
    NATURAL JOIN Piece p
    WHERE o.orderID = %s;
```

- Feature 4: Accept Donation

```

    -- SQL query to check if the user is a registered donor
    SELECT * FROM Act WHERE userName = %s AND roleID = 'donor';

    -- SQL query to check if the user is a registered donor
    SELECT * FROM Act WHERE userName = %s AND roleID = 'donor';

    -- SQL query to insert a new item record
    INSERT INTO Item (iDescription, color, isNew, hasPieces,
    material, mainCategory, subCategory)
    VALUES (%s, %s, %s, %s, %s, %s, %s);

    -- SQL query to insert a piece associated with the item
    INSERT INTO Piece (ItemID, pieceNum, pDescription, length,
    width, height, roomNum, shelfNum, pNotes)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s);

    -- SQL query to insert the donation record into DonatedBy
    INSERT INTO DonatedBy (ItemID, userName, donateDate)
    VALUES (%s, %s, %s);
```

- Feature 5: Start an Order

```
     -- check if the client exists
     SELECT * FROM Person WHERE userName = %s

     -- Ensure clients' name exists in Act
    INSERT IGNORE INTO Act (userName, roleID) VALUES (%s, 'client')

     INSERT INTO Ordered (orderDate, orderNotes, supervisor, client) VALUES (%s, %s, %s, %s)
```

- Feature 6: Add to Current Order (Shopping)

```
     -- Check items availability
     SELECT EXISTS(SELECT * FROM ItemIn WHERE itemID = %s)
```

```
3
4       -- SQL query to add an item to the current order
5       INSERT INTO ItemIn (ItemID, orderID, found) VALUES (%s, %
            s, FALSE)
```

- Feature 7: Prepare Order

```
1
2       -- SQL query to search orders by order number
3       SELECT * FROM Ordered WHERE orderID = %s;
4
5       -- SQL query to search orders by client username
6       SELECT * FROM Ordered WHERE client = %s;
7
8       -- Check if the order has submitted items
9       SELECT EXISTS(SELECT * FROM ItemIn WHERE orderID = %s)
10
11      -- Check if the order has been prepared
12      SELECT EXISTS(SELECT * FROM Delivered WHERE orderID = %s)
13
14      -- SQL query to update the location of all pieces
15      -- associated with the order to (-1, -1)
16      UPDATE Piece
17      SET roomNum = -1, shelfNum = -1
18      WHERE ItemID IN (
19          SELECT ItemID FROM ItemIn WHERE orderID = %s
20      );
21
22      -- add delivery person
23      -- random select one staff or volunteer
24      SELECT userName
25      FROM Act
26      WHERE roleID IN ('staff', 'volunteer')
27      ORDER BY RAND()
28      LIMIT 1
29
30      INSERT INTO Delivered (userName, orderID, status, date)
31              VALUES (%s, %s, %s, %s)
```

- Feature 8: User's Tasks

```
1
2       -- SQL query to retrieve orders
3       -- where the current user is a client
4       SELECT o.orderID, o.orderDate, o.orderNotes, o.supervisor
            ,
5       d.username AS deliverer, d.status, d.date
6       FROM Ordered o
7       LEFT JOIN Delivered d ON d.orderID = o.orderID
8       WHERE o.client = %s;
9
10      -- SQL query to retrieve orders where the
```

```
    -- current user is a supervisor
    SELECT o.orderID, o.orderDate, o.orderNotes, o.supervisor
       ,
    d.username AS deliverer, d.status, d.date
    FROM Ordered o
    LEFT JOIN Delivered d ON d.orderID = o.orderID
    WHERE o.supervisor = %s;

    -- SQL query to retrieve delivered orders with
    -- order date and notes
    SELECT o.orderID, o.orderDate, o.orderNotes,
    o.supervisor, d.username AS deliverer, d.status, d.date
    FROM Delivered d
    NATURAL JOIN Ordered o
    WHERE d.userName = %s;
```

- Feature 9: Rank System

```

    -- Defining the task of volunteer is delivery
    -- SQL query for ranking volunteers' deliveries in a
       specific time period
    SELECT userName, COUNT(*) AS con
    FROM delivered NATURAL JOIN act
    Where roleID = 'volunteer' AND date >= %s AND date <= %s
    GROUP BY userName
    Order BY con DESC
```

- Feature 10: Update Enabled

```

    -- SQL query to update order status
    UPDATE delivered
    SET status = %s, date = %s
    WHERE orderID = %s AND userName = %s
```

5 Difficulties Encountered

1. Frontend and backend development

   - One of the first difficulties we encountered was a cookie loss issue related to Cross-Origin Resource Sharing (CORS). This problem arose when our frontend and backend servers were hosted on different domains.

   - To resolve this issue, we added the necessary request headers in our frontend server during requests. Additionally, we enabled CORS in our Flask backend by installing the 'flask_cors' package and configuring it appropriately to allow requests from our frontend origin.

- This experience taught us the importance of understanding CORS and its implications on cookie handling and session management when dealing with separate frontend and backend services.
- We also learned to thoroughly test interactions between the frontend and backend in a cross-origin context to preemptively identify and address similar issues.

2. When checking for existence, we found that using "SELECT * FROM table" directly along with cursor.fetchone() is not a good approach. If the query could return multiple rows, any unconsumed results from a previous query might lead to an "unread result found" error when attempting to close the cursor. Using the query "SELECT EXISTS(...)" can resolve this issue and make the query more efficient, as it stops after finding the first matching row instead of returning all rows.

6 Team Member Contributions

- Jeffery Zeng: Feature 7 & 8
- Jiaxin Sun: Feature 5 & 6
- Yutong Tian: Feature 9 & 10