Login details to direct access the database:

RDS End Point: pafsohdatabase.coq9yrrfvnw8.us-east-1.rds.amazonaws.com

RDS Database Name: pafsohdatabase

User: admin

Password: 123456789

Access Instructions for Evaluators:

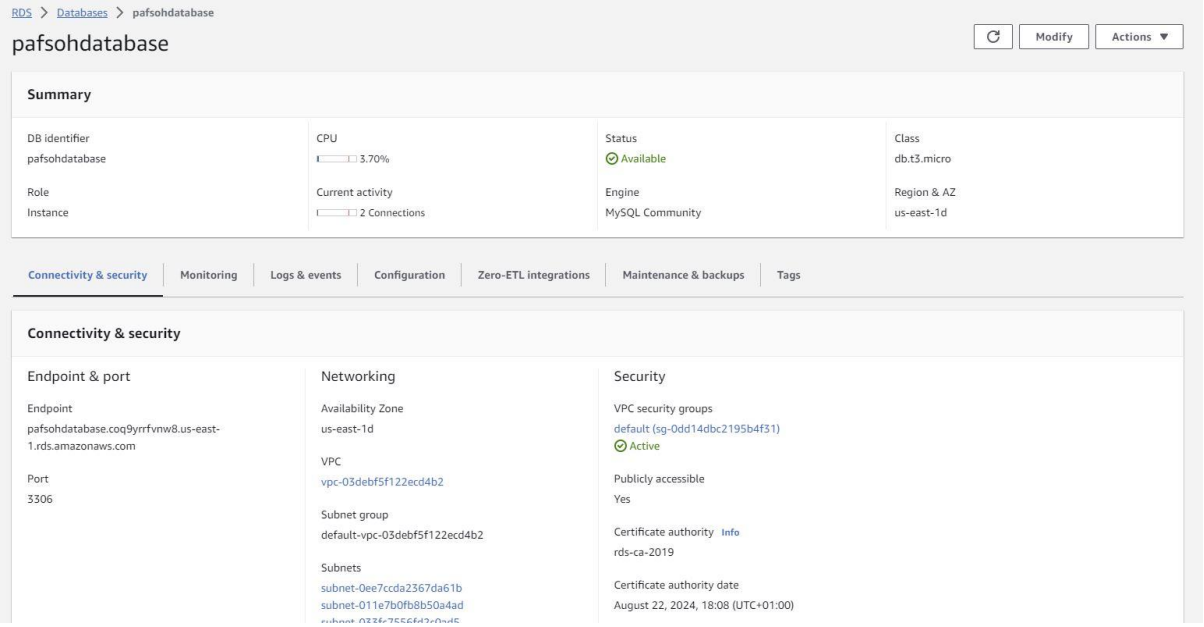
1. **MySQL Workbench Access:**
    - Launch MySQL Workbench on your system.
2. **Database Connection:**
    - Open MySQL Workbench.
    - Click on the "+" icon next to "MySQL Connections" to create a new connection.
    - Enter the following details:
        - Connection Name: PAFSOH_Database
        - Hostname: [Your AWS Server Hostname]
        - Port: 3306
        - Username: [Your Username]
        - Password: [Your Password]
3. **Connecting to the Database:**
    - Click "Test Connection" to ensure successful connectivity.
    - Once successful, click "OK" to save the connection.
4. **Accessing the Database:**
    - Double-click on the newly created connection ("PAFSOH_Database") to open it.
5. **Exploring Database Schema:**
    - Navigate to the "SCHEMAS" tab on the left panel.
    - Find and select the "PAFSOH_Database" schema.
6. **Reviewing Tables:**
    - All tables are listed under the selected schema. You can explore table structures and relationships.
7. **Running SQL Queries:**

- Go to the "SQL" tab to execute queries.
- Copy and paste provided SQL queries into the editor.
- Execute queries to interact with the database.

Important Notes:

- Ensure MySQL Workbench is installed on your system.
- Replace placeholders like [Your AWS Server Hostname], [Your Username], and [Your Password] with actual connection details

AWS Snapshot:



Introduction

PAFSOH Fashion, in its pursuit of sustainable growth and efficiency, has embarked on the development of a modern database system. This report outlines the approach taken in the system's development, highlighting key features and the connection between user interfaces (UIs) and SQL queries.

In developing the system, we prioritized addressing the challenges faced by PAFSOH Fashion, including operational inefficiencies, limited accessibility, production waste, data security concerns, and reporting complexities. The chosen approach focuses on creating a centralized, scalable, and secure database system capable of supporting the company's expansion plans.

System features, UIs, and SQL quarries

The seamless functionality of the PAFSOH Fashion modern database system relies on a robust and intricate connection between User Interfaces (UIs) and the underlying SQL queries. This section delves into the details of this connection, highlighting the essential aspects that contribute to the system's efficiency and user-centric design.

1. **User Interface Design:**

- **Manager UIs:**
    - Custom-designed UIs for managers provide a comprehensive view of critical aspects such as customer details, inventory, transactions, and employee information.
    - Each UI incorporates intuitive elements, including buttons, forms, and dropdowns, to facilitate user interactions.
- **Shop Floor Staff UIs:**

    - Tailored UIs for shop floor staff focus on essential functionalities, including customer details, inventory management, transaction logs, and personal profile editing.
    - The UIs are designed with simplicity and efficiency in mind, optimizing the user experience.
- **Customer UIs:**
    - Customer-specific dashboards and information pages offer a personalized view, allowing customers to access their details and interact with the system seamlessly.
    - The UIs prioritize simplicity and clarity to enhance user engagement.
- **Supplier UIs:**
    - Supplier-specific UIs focus on providing suppliers with easy access to their profiles and relevant details.
    - The design emphasizes clarity and ease of use for effective collaboration.

2. **User Actions and UI-Triggered Operations:**

- **Manager Operations:**
    - Managers can perform CRUD operations on customer details, inventory, transactions, and employee information through designated UI elements.
    - UI-triggered actions, such as adding a new product to the inventory or editing an employee record, prompt corresponding SQL queries for seamless backend execution.
- **Shop Floor Staff Operations:**
    - Shop floor staff interact with UIs to view customer details, manage inventory, access transaction logs, and edit personal profiles.
    - CRUD operations, initiated through UI elements, lead to the execution of specific SQL queries.
- **Customer Interactions:**
    - Customers engage with UIs to view personalized dashboards, access customer information, and register through the "Create Customer" UI.
    - UI-triggered actions, like placing an online order or updating personal details, invoke the execution of SQL queries for data manipulation.
- **Supplier Engagement:**
    - Suppliers utilize UIs to access their profiles and relevant information, fostering effective communication.
    - UI-driven actions, such as updating supplier details, prompt the execution of SQL queries to modify the backend database.

3. **SQL Queries Corresponding to UI Functions:**

- Each UI function corresponds to a set of SQL queries that define the backend operations required to fulfil user actions.
- For instance, a "Delete" button on a manager's UI for employee details triggers a SQL DELETE query, ensuring the removal of the specified record from the database.

4. **Cross-Referencing UI Elements with SQL Query Numbers:**

- UI elements, such as buttons or forms, are assigned unique identifiers or numbers that cross-reference with a list of SQL queries.
- This cross-referencing ensures a clear connection between user-triggered actions and the SQL queries executed in response.

5. **Testing and Optimization:**

- Rigorous testing of UI-embedded functionalities and corresponding SQL queries is conducted using tools like MySQL Workstation.
- Optimization efforts focus on refining the efficiency and performance of both UI interactions and SQL query executions.

UI & Queries:

Manager dashboard - The access for the Manager is level 1.

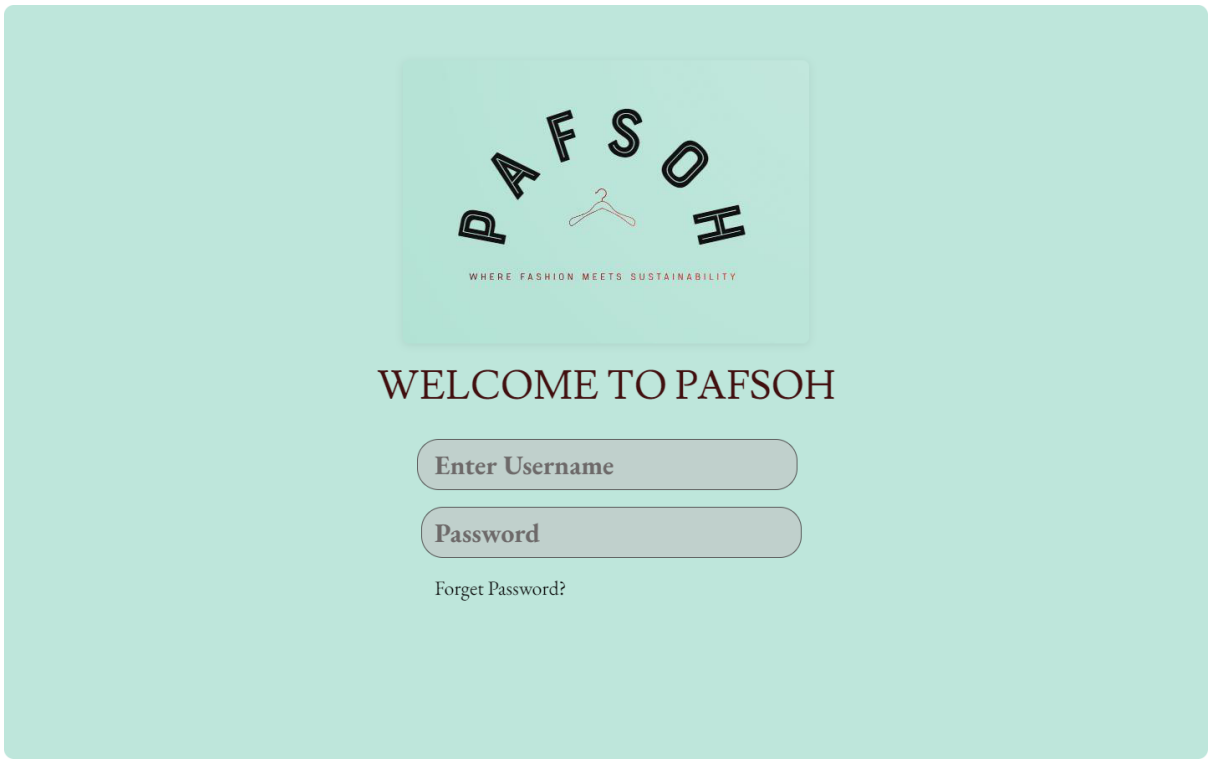


Figure 1 - Logging Page for all users

Manager - Customer Details:

1. **Create (C):**

-- Assume Manager can add new customers

INSERT INTO Customer (CustomerID, FirstName, LastName, PhoneNumber, Email, Gender) VALUES (1,'New', 'Customer', '555-123-4567', 'new.customer@example.com', 'Other');

2. **Read (R):**

-- Retrieve all customer details

SELECT * FROM Customer;

3. **Update (U):**

-- Assume Manager can update customer information

UPDATE Customer SET Email = 'updated.email@example.com' WHERE CustomerID = 1;

4. **Delete (D):**

-- Assume Manager can delete a customer

DELETE FROM Customer WHERE CustomerID = 1;

Manager - Employee Details:

1. **Create (C):**

-- Assume Manager can add new employees

INSERT INTO Employee (EmployeeID, FirstName, LastName, Role, PhoneNumber, Email, Gender) VALUES (1,'New', 'Employee', 'Staff', '123-456-7890', 'new.employee@example.com', 'Male');

2. **Read (R):**

-- Retrieve all employee details

SELECT * FROM Employee;

3. **Update (U):**

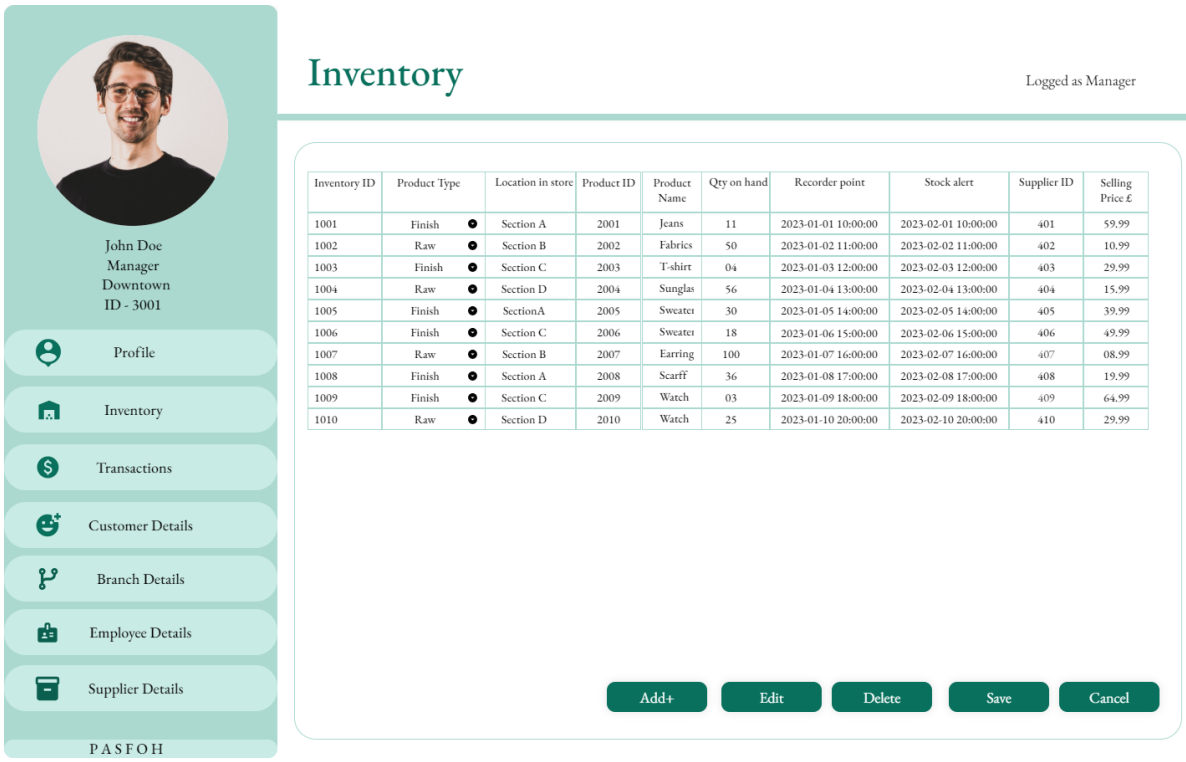
-- Assume Manager can update employee information

UPDATE Employee SET Email = 'updated.employee@example.com' WHERE EmployeeID = 1;

4. **Delete (D):**

-- Assume Manager can terminate an employee

DELETE FROM Employee WHERE EmployeeID = 1;



Manager - Inventory:

1. **Create (C):**

-- Assume Manager can add new products to inventory

INSERT INTO Inventory (InventoryID, QuantityOnHand, RecorderPoint, StockAlerts, LocationInStore) VALUES (1,50, NOW(), NOW(), 'Section A',);

2. **Read (R):**

-- Retrieve all inventory details

SELECT * FROM Inventory;

3. **Update (U):**

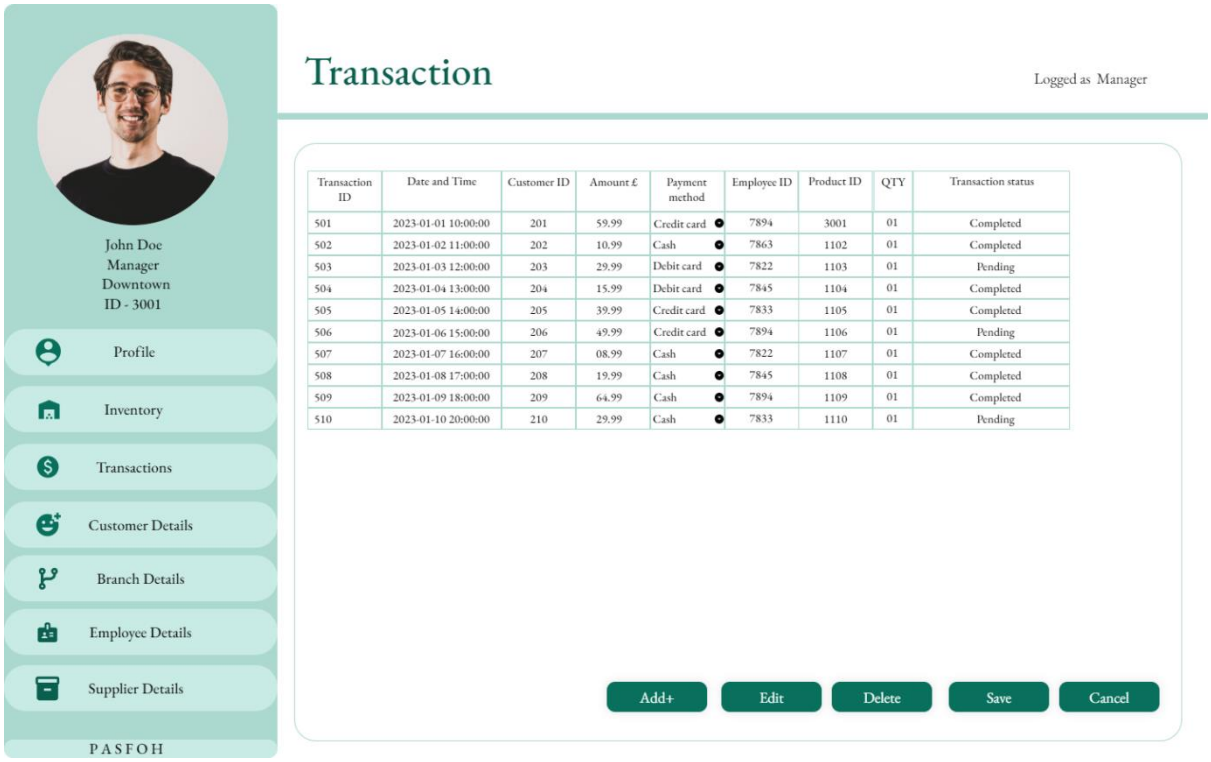
-- Assume Manager can update inventory information

UPDATE Inventory SET QuantityOnHand = 60 WHERE InventoryID = 1;

4. **Delete (D):**

-- Assume Manager can remove a product from inventory

DELETE FROM Inventory WHERE InventoryID = 1;



Manager - Transaction History:

1. **Create (C):**

-- Assume Manager can record new transactions

INSERT INTO TransactionHistory (TransactionID, DateAndTime) VALUES (1,NOW());

2. **Read (R):**

-- Retrieve all transaction history details

SELECT * FROM TransactionHistory;

3. **Update (U):**

-- Assume Manager can update transaction information

UPDATE TransactionHistory SET DateAndTime = NOW() WHERE TransactionID = 1;

4. **Delete (D):**

-- Assume Manager can delete a transaction record

DELETE FROM TransactionHistory WHERE TransactionID = 1;

Floor Staff dashboard - The access for the Employee is level 1.

Shop Floor Staff - Customer Details:

1. **Create (C):**

-- Assume Shop Floor Staff can add new customers

INSERT INTO Customer (CustomerID, FirstName, LastName, PhoneNumber, Email, Gender) VALUES (2,'New', 'Customer', '555-123-4567', 'new.customer@example.com', 'Other');

2. **Read (R):**

-- Retrieve all customer details

SELECT * FROM Customer;

3. **Update (U):**

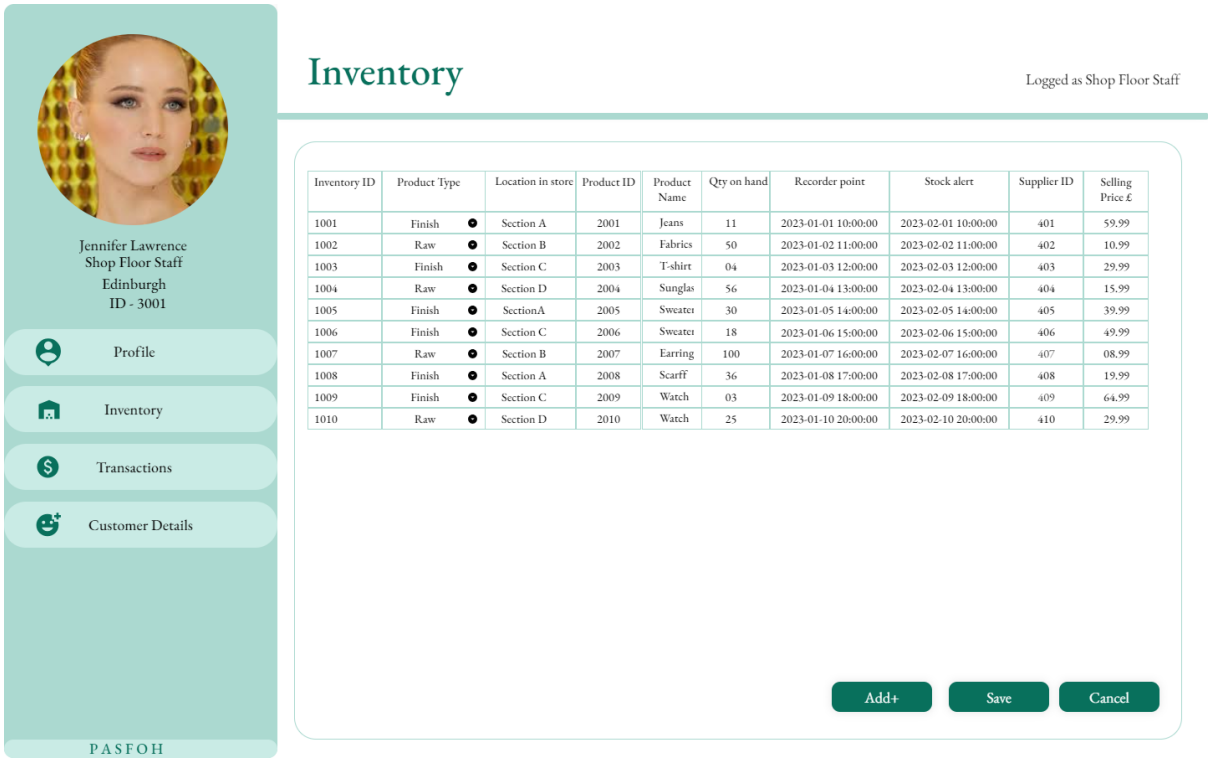
-- Assume Shop Floor Staff can update customer information

UPDATE Customer SET Email = 'updated.email@example.com' WHERE CustomerID = 2;

4. **Delete (D):**

-- Assume Shop Floor Staff can delete a customer

DELETE FROM Customer WHERE CustomerID = 2;



Shop Floor Staff - Inventory:

1. **Create (C):**

-- Assume Shop Floor Staff can add new products to inventory

INSERT INTO Inventory (InventoryID, QuantityOnHand, RecorderPoint, StockAlerts, LocationInStore) VALUES (1, 50, NOW(), NOW(), 'Section A');

2. **Read (R):**

-- Retrieve all inventory details

SELECT * FROM Inventory;

3. **Update (U):**

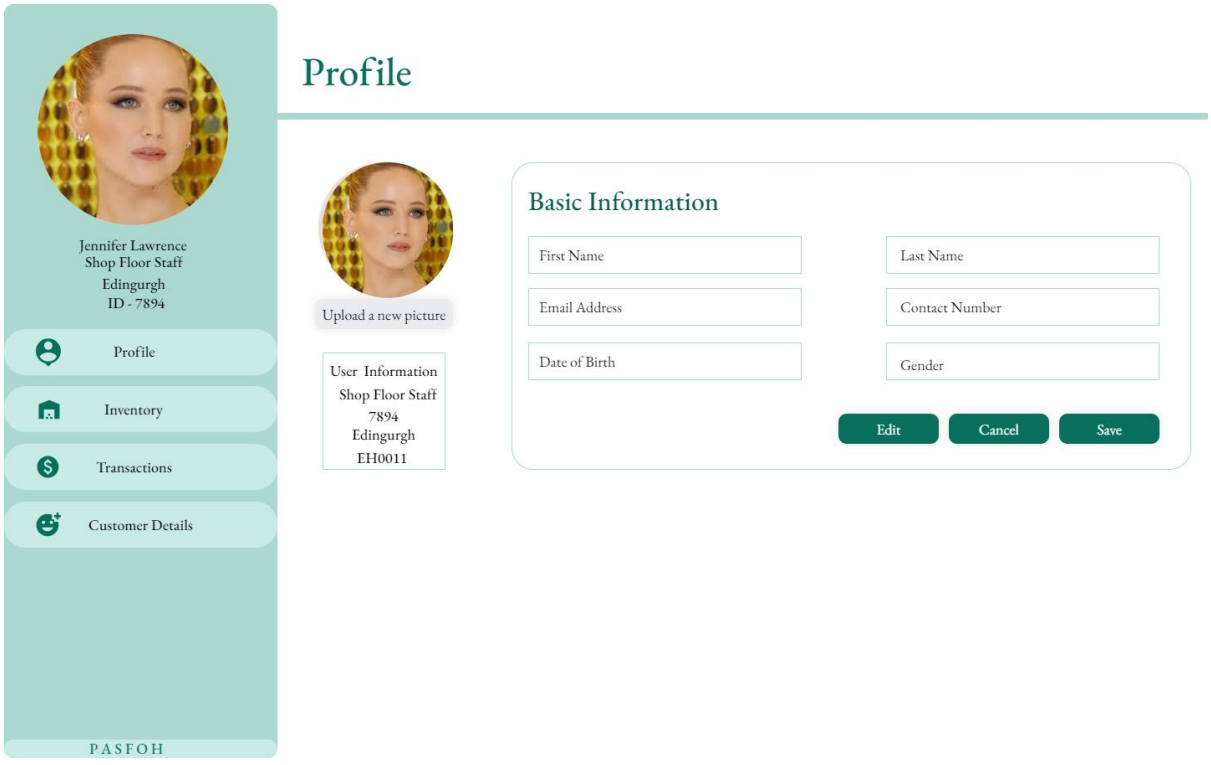
-- Assume Shop Floor Staff can update inventory information

UPDATE Inventory SET QuantityOnHand = 60 WHERE InventoryID = 1;

4. **Delete (D):**

-- Assume Shop Floor Staff can remove a product from inventory

DELETE FROM Inventory WHERE InventoryID = 1;



Shop Floor Staff - Profile Editing (Employee Table):

1. **Create (C):**

-- Assume Shop Floor Staff can add new employees

INSERT INTO Employee (EmployeeID, FirstName, LastName, Role, PhoneNumber, Email, Gender) VALUES (3, 'New', 'Employee', 'Staff', '123-456-7890', 'new.employee@example.com', 'Male');

2. **Read (R):**

-- Retrieve all employee details

SELECT * FROM Employee;

3. **Update (U):**

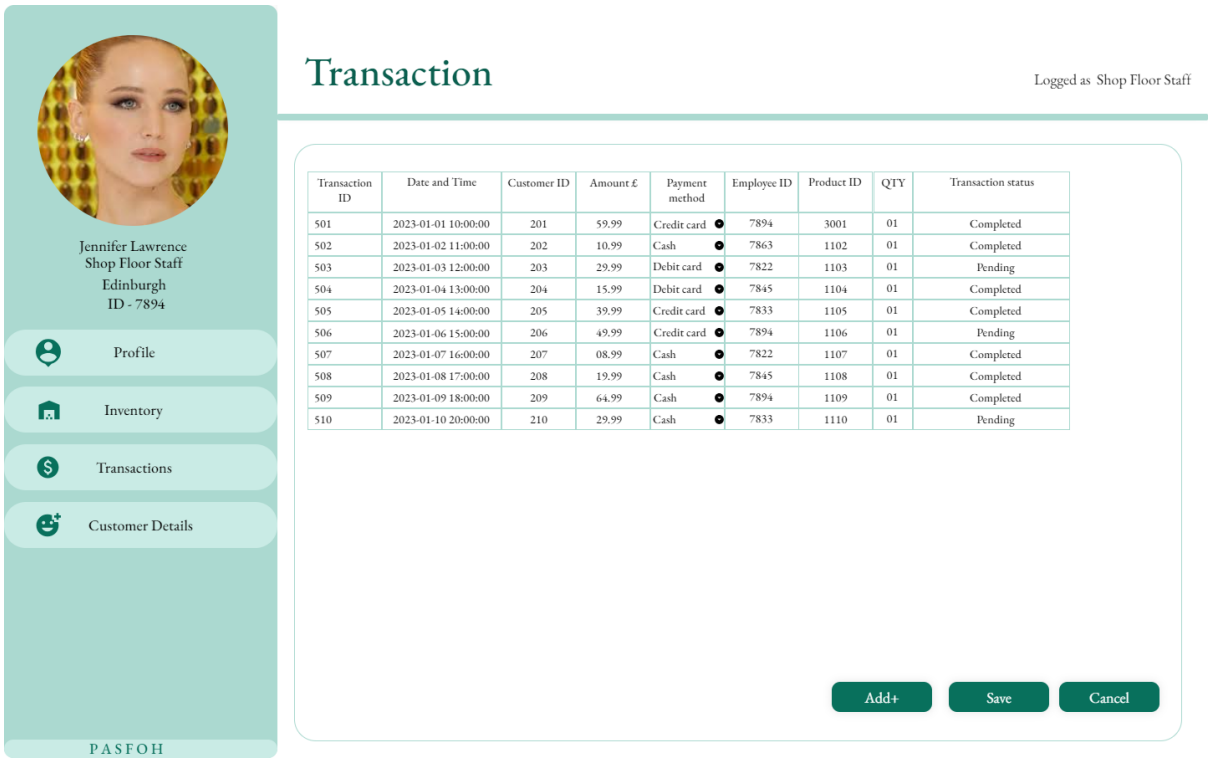
-- Assume Shop Floor Staff can update employee information

UPDATE Employee SET Email = 'updated.employee@example.com' WHERE EmployeeID = 3;

4. **Delete (D):**

-- Assume Shop Floor Staff can terminate an employee

DELETE FROM Employee WHERE EmployeeID = 3;



Shop Floor Staff - Transaction History:

1. **Create (C):**

-- Assume Shop Floor Staff can record new transactions

INSERT INTO TransactionHistory (TransactionID, DateAndTime) VALUES (4, NOW());

2. **Read (R):**

-- Retrieve all transaction history details

SELECT * FROM TransactionHistory;

3. **Update (U):**

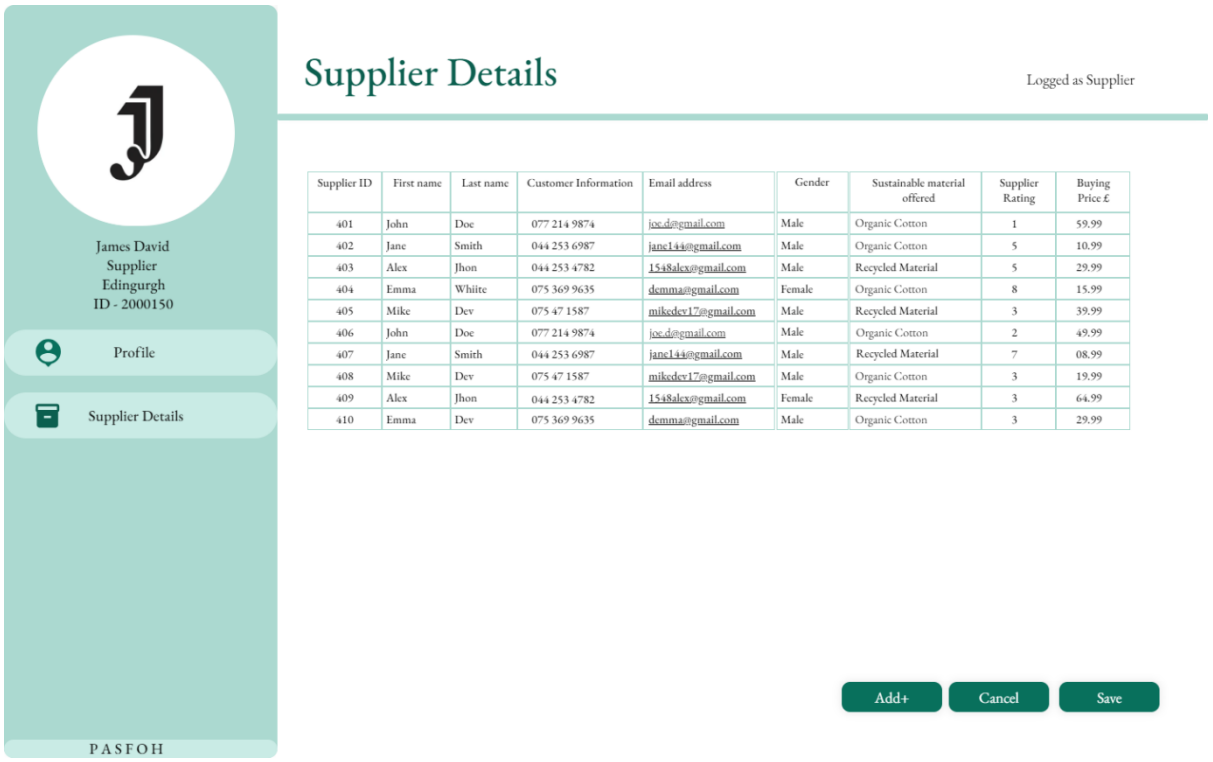
-- Assume Shop Floor Staff can update transaction information

UPDATE TransactionHistory SET DateAndTime = NOW() WHERE TransactionID = 4;

4. **Delete (D):**

-- Assume Shop Floor Staff can delete a transaction record

DELETE FROM TransactionHistory WHERE TransactionID = 4;



Employee - Supplier Details:

1. **Create (C):**

-- Assume employee can add a new supplier

INSERT INTO Supplier (SupplierID, FirstName, LastName, PhoneNumber, Email, Gender, SustainableMaterialOffered, SupplierRating) VALUES (5, 'New', 'Supplier', '555-987-6543', 'new.supplier@example.com', 'Other', 'Eco-Friendly Material', 5);

2. **Read (R):**

-- Retrieve all supplier details

SELECT * FROM Supplier;

3. **Update (U):**

-- Assume employee can update supplier information

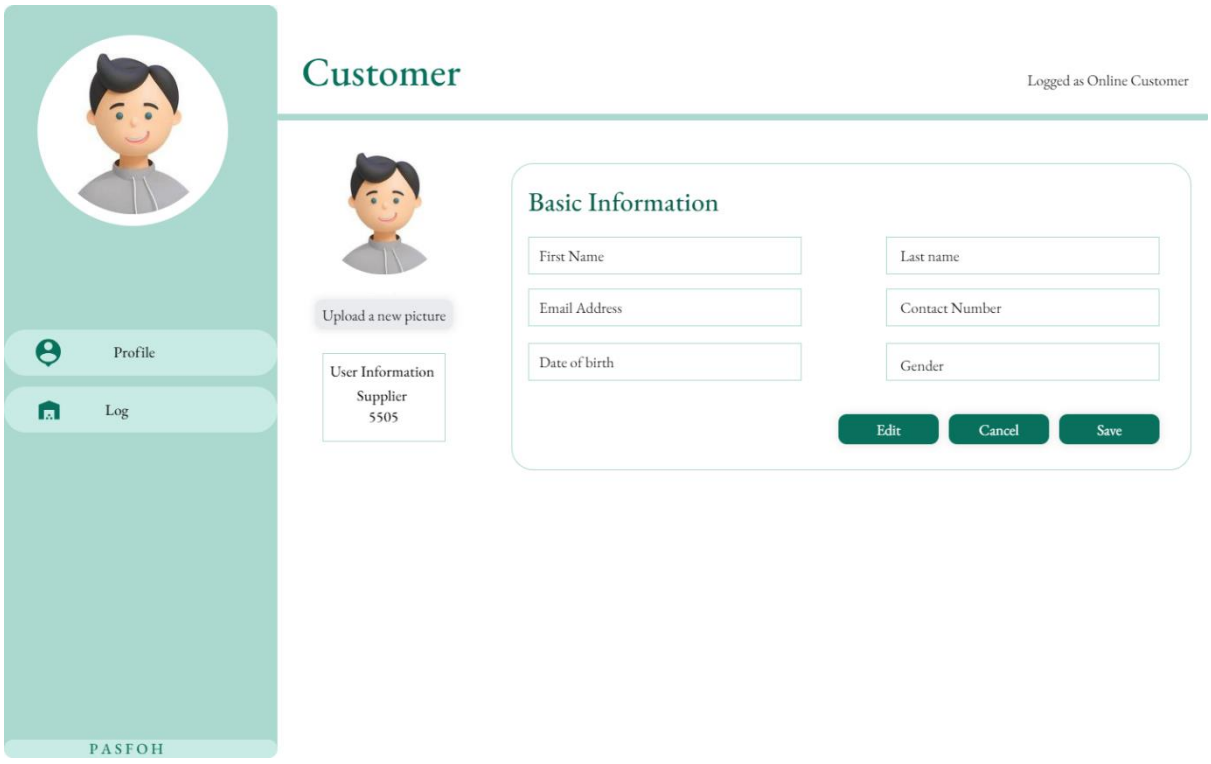
UPDATE Supplier SET Email = 'updated.email@example.com' WHERE SupplierID = 5;

4. **Delete (D):**

-- Assume employee can remove a supplier

DELETE FROM Supplier WHERE SupplierID = 5;

Customer dashboard - The access for the Customer is level 3.



Customer Dashboard - Customer Information Page:

1. **Create (C):**

-- Assume customer can update their own information

INSERT INTO Customer (CustomerID, FirstName, LastName, PhoneNumber, Email, Gender) VALUES (1, 'New', 'Customer', '555-123-4567', 'new.customer@example.com', 'Other');

2. **Read (R):**

-- Retrieve specific customer information based on

CustomerID SELECT * FROM Customer WHERE CustomerID = 1;

3. **Update (U):**

-- Assume customer can update their own information

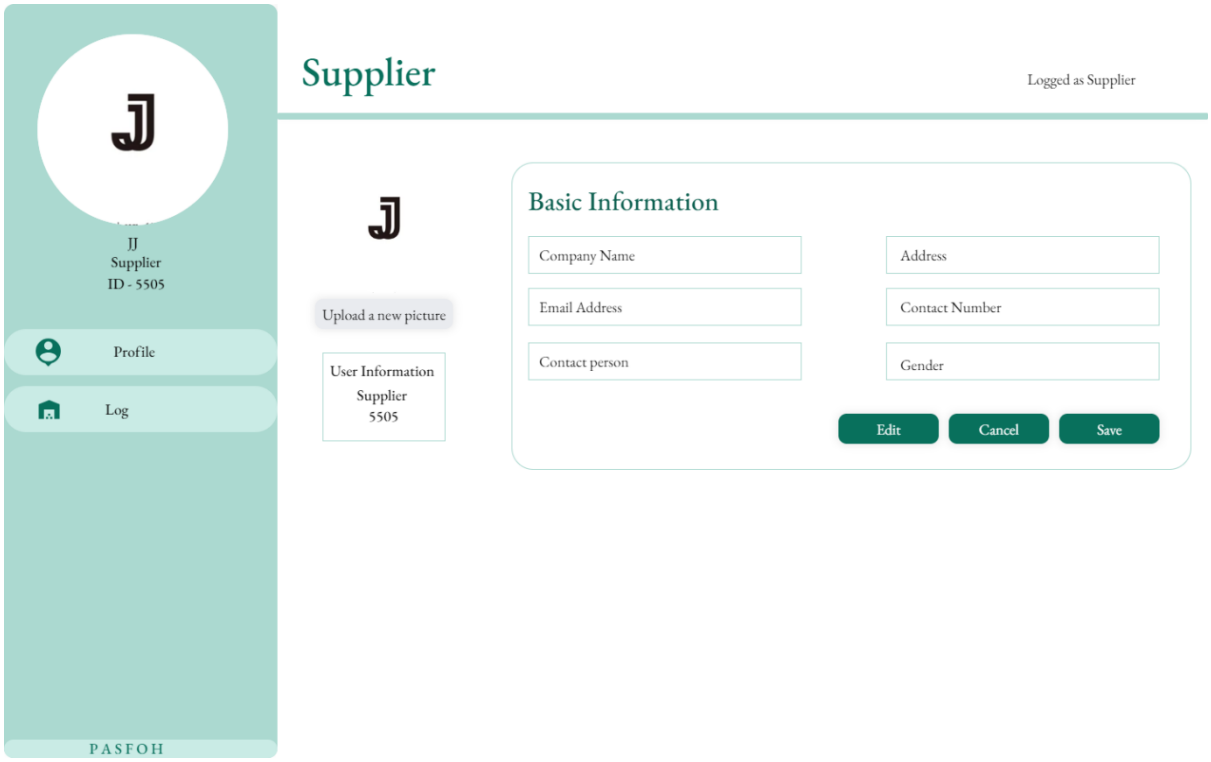
UPDATE Customer SET Email = 'updated.email@example.com' WHERE CustomerID = 1;

4. **Delete (D):**

-- Assume customer can deactivate their account

DELETE FROM Customer WHERE CustomerID = 1;

Supplier dashboard - The access for the Customer is level 3.



Supplier Profile Page:

1. **Create (C):**

-- Assume supplier can create their own profile

INSERT INTO Supplier (SupplierID, FirstName, LastName, PhoneNumber, Email, Gender, SustainableMaterialOffered, SupplierRating) VALUES (1,'New', 'Supplier', '555-987-6543', 'new.supplier@example.com', 'Other', 'Eco-Friendly Material', 5);

2. **Read (R):**

-- Retrieve specific supplier information based on SupplierID

SELECT * FROM Supplier WHERE SupplierID = 1;

3. **Update (U):**

-- Assume supplier can update their own information

UPDATE Supplier SET Email = 'updated.email@example.com' WHERE SupplierID = 1;

4. **Delete (D):**

-- Assume supplier can deactivate their profile

DELETE FROM Supplier WHERE SupplierID = 1;

Access Control:

The different access levels for the four main user roles:

1. **Manager (Access Level 1):**
    - Full access to Customer details (CRUD)
    - Full access to Inventory (CRUD)
    - Full access to Transaction History (CRUD)
    - Full access to Employee details (CRUD)
    - Limited access to Supplier details (Read-only)
2. **Employee (Access Level 2):**
    - Limited access to Customer details (Read-only)
    - Limited access to Inventory (CRUD)
    - Limited access to Transaction History (CRUD)
    - Limited access to Employee details (CRUD)
    - Limited access to Supplier details (CRUD)
3. **Customer (Access Level 3):**
    - Full access to their own Customer information (CRUD)
    - Limited access to Inventory (Read-only)
    - Limited access to Transaction History (Read-only)
    - No access to Employee details
    - No access to Supplier details
4. **Supplier (Access Level 3):**
    - Limited access to their own Supplier information (CRUD)
    - No access to Customer details
    - No access to Inventory
    - No access to Transaction History
    - No access to Employee details

These access levels provide a hierarchy where the Manager has the highest level of access, followed by the Employee, and then the Customer and Supplier having the least access.

Conclusion:

During this database implementation assessment, our team embarked on the task of translating the logical design developed in Assessment 1 into a functional database system. Employing MySQL as our database management system and hosting the database on AWS servers, we meticulously crafted tables representing various entities such as Customer, Supplier, Employee, Inventory, Transaction History, and more.

The application's user interface was strategically designed to cater to the distinct needs of four main user roles – Manager, Employee, Customer, and Supplier. Access levels were intricately defined, with the Manager possessing the highest privileges, followed by the Employee, Customer, and Supplier, each with progressively limited access. This hierarchical structure ensures data security and integrity while allowing users to perform essential tasks seamlessly.

For the Manager, the ability to manage Customer details, Inventory, Transaction History, and Employee details was implemented. Employees, with a slightly lower access level, were granted limited access to Customer details while maintaining comprehensive control over Inventory, Transaction History, Employee details, and Supplier details.

Customers and Suppliers, with the least access privileges, could manage their respective profiles and had restricted access to relevant information. Customers had full control over their information, a limited view of Inventory and Transaction History, and no access to Employee or Supplier details. Suppliers, on the other hand, could perform CRUD operations on their own details but had no access to Customer information, Inventory, Transaction History, or Employee details.

This comprehensive database implementation not only brought our logical design to life but also ensured that the system is robust, secure, and aligned with the specific needs of each user role. The structure and content of the MySQL database, coupled with well-defined user interfaces, serve as a foundation for the future development of a front-end application.

As a team, we have not only met the requirements outlined in the assessment but have strived for excellence in ensuring the fitness for purpose and the quality of functionality. The culmination of our efforts is a database system that lays the groundwork for a dynamic and responsive application, poised to meet the demands of our envisioned eco-system.

This report, along with the MySQL database hosted on AWS servers, stands as a testament to our commitment to delivering a robust and well-designed database implementation. We look forward to feedback from the assessors and are confident that our work meets the highest standards set by the University of Dundee.