Fresh Value Groceries

Anita Ruangrotsakun and Lindsey Olmstead

URL to website

https://fresh-value-groceries.herokuapp.com/

Project Outline

Fresh Value Groceries is a neighborhood supermarket with 50 employees that serves hundreds of customers and averages approximately $1,500,000 in sales each week. Our database-driven management website assists the general manager with the following tasks:

- Managing items in inventory
- Keeping a record of current employees
- Creating and scheduling work shifts
- Maintaining a record of customers and customer rewards
- Managing customer orders placed online for pick up

Database Outline

In this section, you will find a description of each of the entities and their attributes in our database. An Entity-Relationship Diagram and Schema that illustrate the relationships between these entities can be found below.

- **Employees:** maintains list of current employees
  - *One to many relationship with EmployeeShifts*
    - An Employee can be assigned to multiple Shifts
    - Multiple Employees can work the same Shift
    - *This relationship is used as part of the implementation of the M:M relationship between Employee and Shift*
  - *One to many relationship with Orders*
    - Each Order has only one Employee to service it
    - An Employee can serve multiple Orders
  - **EmployeeID:** int, NOT NULL, unique, Primary Key
    - Unique identifier for an Employee
  - **Name:** varchar
    - Employee's first and last name
  - **HourlyWage:** decimal
    - Hourly wage (in USD)
  - **Responsibilities:** varchar
    - Description of Employee's work duties
  - **SickDays:** int

        - Number of sick days remaining

- **Shifts:** list of shifts an Employee may work
    - *One to many relationship with EmployeeShifts*
        - An Employee can be assigned to multiple Shifts
        - Multiple Employees can work the same Shift
        - *This relationship is used as part of the implementation of the M:M relationship between Employee and Shift*
    - **ShiftID:** int, NOT NULL, unique, Primary Key
        - Unique identifier for a shift
    - **Day:** varchar
        - Day of the week for the scheduled shift
    - **StartTime:** time
        - Time that shift starts (24 hour clock)
    - **EndTime:** time
        - Time that shift ends (24 hour clock)

- **EmployeeShifts:** composite entity to manage the M:M relationship between Employees and Shifts
    - *One to many relationship with Shifts*
        - An EmployeeShift must correspond to one Shift
        - A Shift can correspond to many EmployeeShifts
    - *One to many relationship with Employees*
        - An EmployeeShift must correspond to one Employee
        - An Employee can correspond to many EmployeeShifts
    - **EmployeeShiftID:** int, NOT NULL, Primary Key
        - Unique identifier for an employee-shift entry
    - **EmployeeID:** int, NOT NULL, Foreign Key
        - Refers to an Employee working a given shift
    - **ShiftID:** int, NOT NULL, Foreign Key
        - Refers to a specific shift

- **Customers:** records past customers and their reward points for shopping at Fresh Value Groceries
    - *One to many relationship with Orders*
        - A Customer can place multiple Orders
        - An Order is linked to only one Customer
    - **CustomerID:** int, NOT NULL, unique, Primary Key
        - Unique identifier for a Customer
    - **Name:** varchar
        - Customer's first and last name
    - **PhoneNumber:** varchar
        - Customer's phone number

    - **RewardsPts:** int
        - Current point total that a Customer earns for shopping

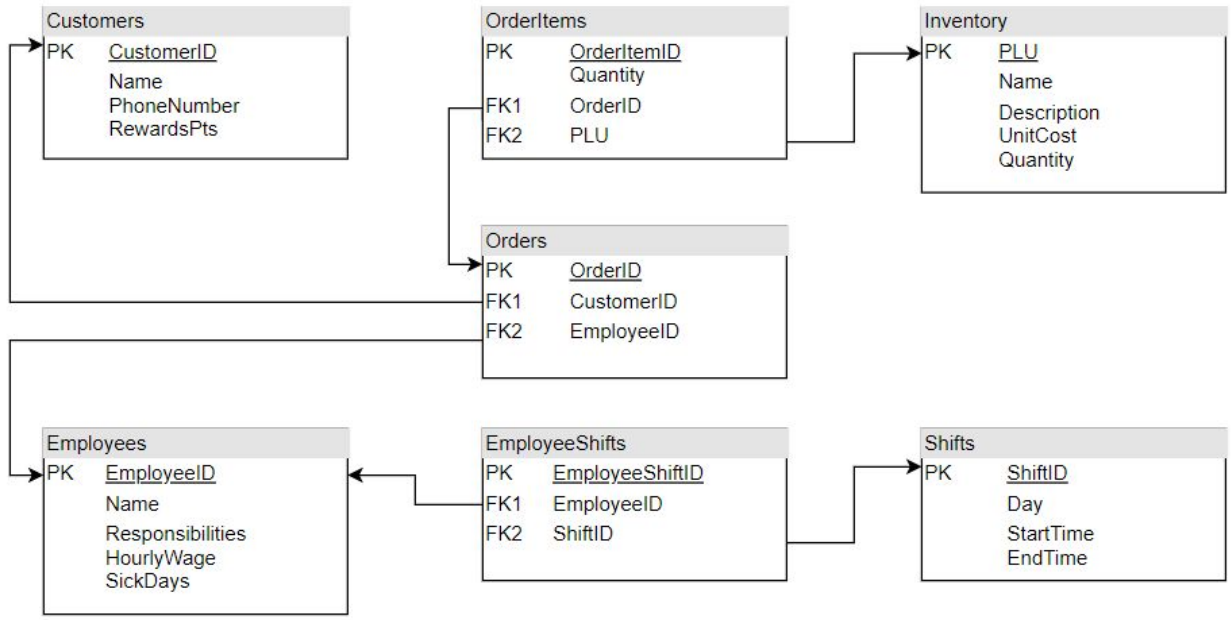
- **Inventory:** records the available grocery items in stock
    - *One to many relationship with OrderItems*
        - An Inventory item can be present in multiple OrderItems
        - Each OrderItem corresponds to one item from Inventory
        - *This relationship is used as part of the implementation of the M:M relationship between Order and Inventory*
    - **PLU:** int, NOT NULL, unique, Primary Key
        - Unique identifier for an item
    - **Name:** varchar
        - Name of an item in inventory
    - **Description:** varchar
        - Description of what the item is
    - **UnitCost:** decimal
        - Cost (in USD) of a single unit of the item
    - **Quantity:** int
        - Number of these items available in stock

- **Orders:** keeps track of customer orders
    - *One to many relationship with Customers*
        - Each Customer can have multiple Orders
        - Each Order can only have one customer
    - *One to many relationship with OrderItems*
        - An Order contains one or more OrderItems
        - An OrderItem is linked to only one Order
        - *This relationship is used as part of the implementation of the M:M relationship between Order and Inventory*
    - *One to many relationship with Employees*
        - Each Order has only one Employee to service it
        - An Employee can serve multiple Orders
    - **OrderID:** int, NOT NULL, Primary Key
        - Unique identifier for an Order
        - Querying an OrderID should return a list of items that are in that Order
    - **CustomerID:** int, NOT NULL, Foreign Key
        - Refers to the unique identifier of the Customer
    - **EmployeeID:** int, NOT NULL, Foreign Key
        - Refers to the unique identifier of the Employee working on a given order

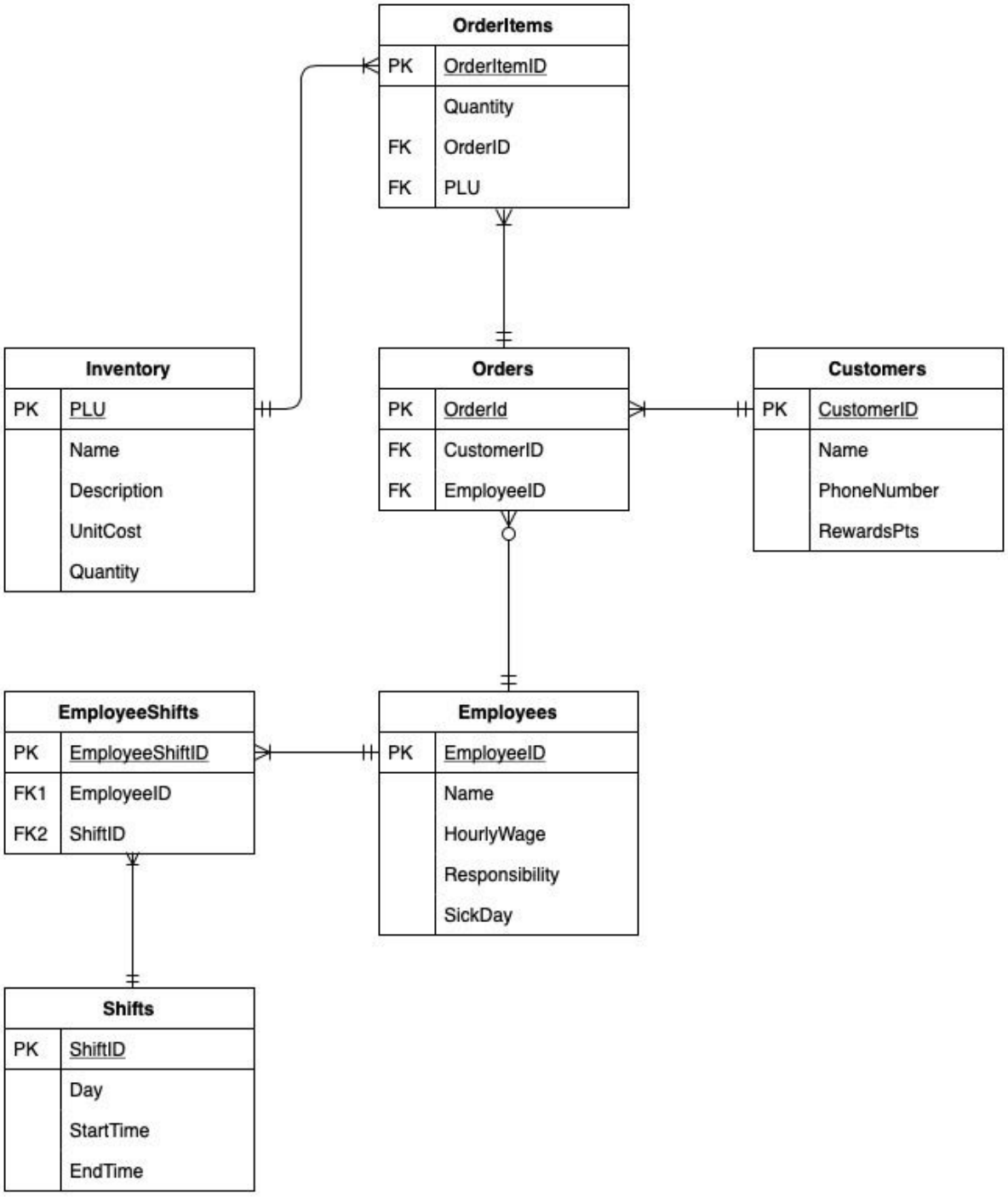
- **OrderItems**: a composite entity that lists the items included in a customer's order
    - *One to many relationship with Orders*

        - An Order contains one or more OrderItems
        - An OrderItem is linked to only one Order
        - *This relationship is used as part of the implementation of the M:M relationship between Order and Inventory*
    - *One to many relationship with Inventory*
        - An Inventory item can be present in multiple OrderItems
        - Each OrderItem corresponds to one item from Inventory
        - *This relationship is used as part of the implementation of the M:M relationship between Order and Inventory*
    - **OrderItemID**: int, NOT NULL, Primary Key
        - Unique identifier for an item included in an Order
    - **Quantity**: int
        - Number of this item requested by the customer
    - **OrderID**: int, Foreign Key
        - Refers to a specific Order placed by a Customer
    - **PLU**: int, Foreign Key
        - Refers to a specific Inventory item

Schema

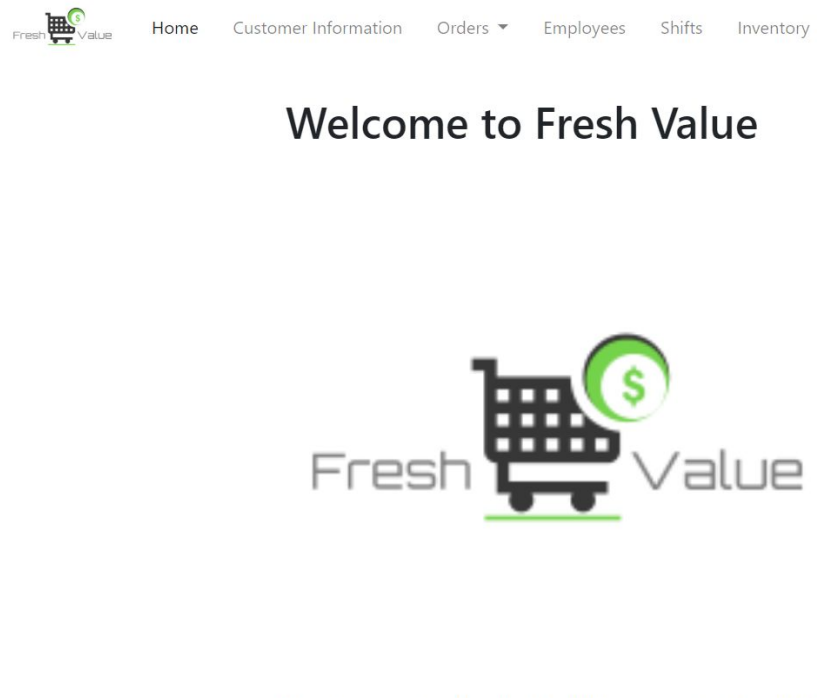


Entity-Relationship Diagram

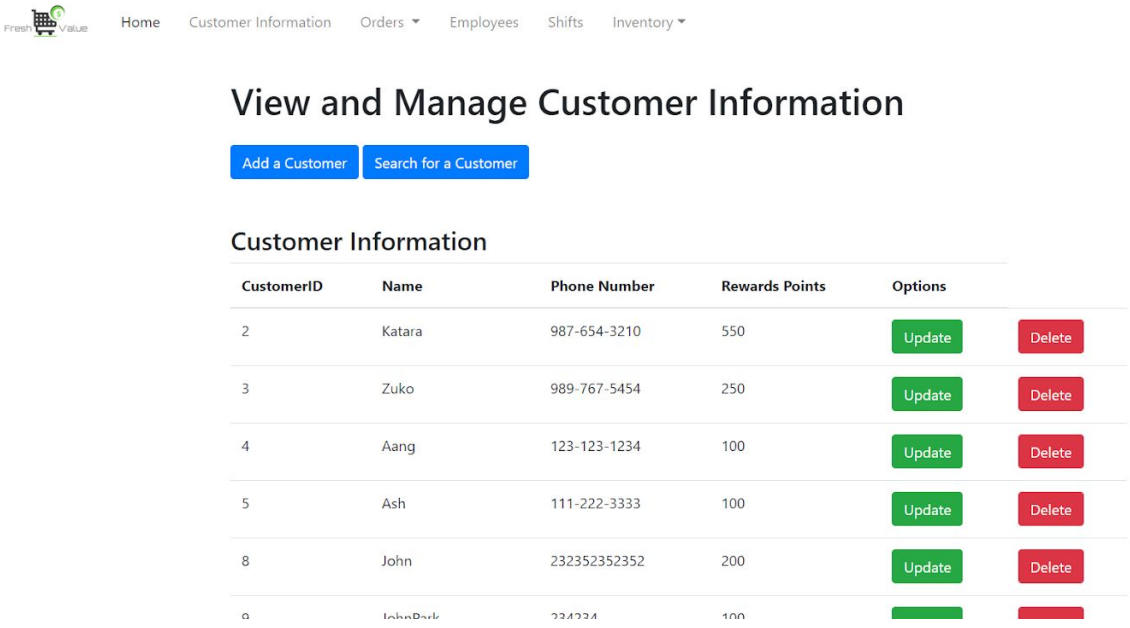


UI Screenshots

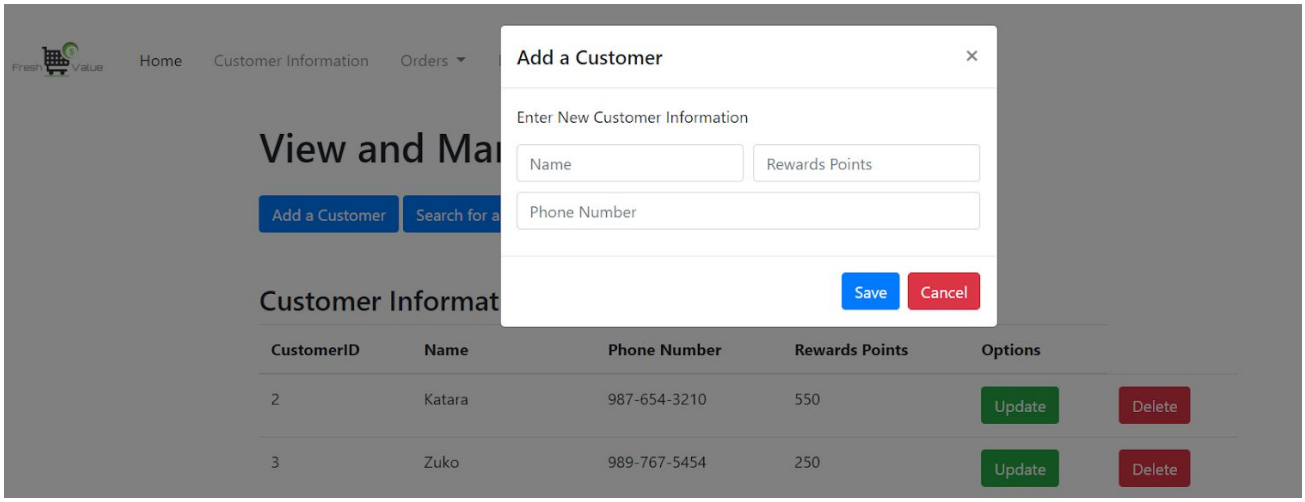
Home Page



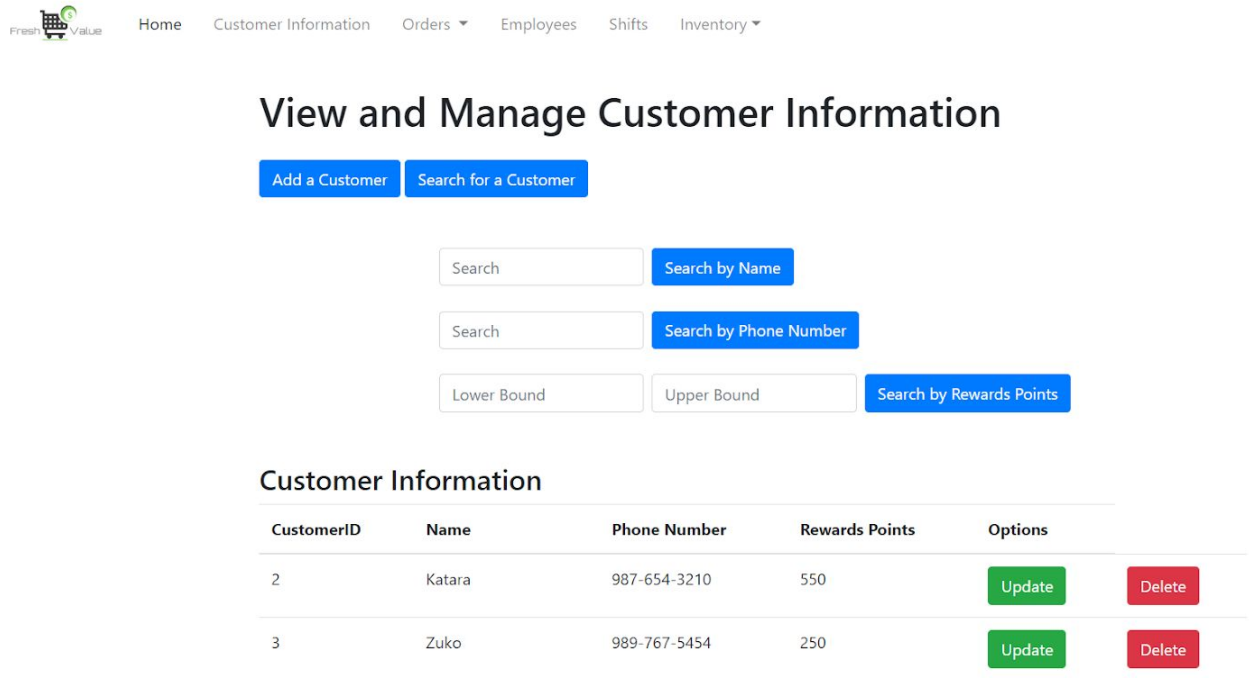
Display Customers Table with Ability to Update and Delete Rows



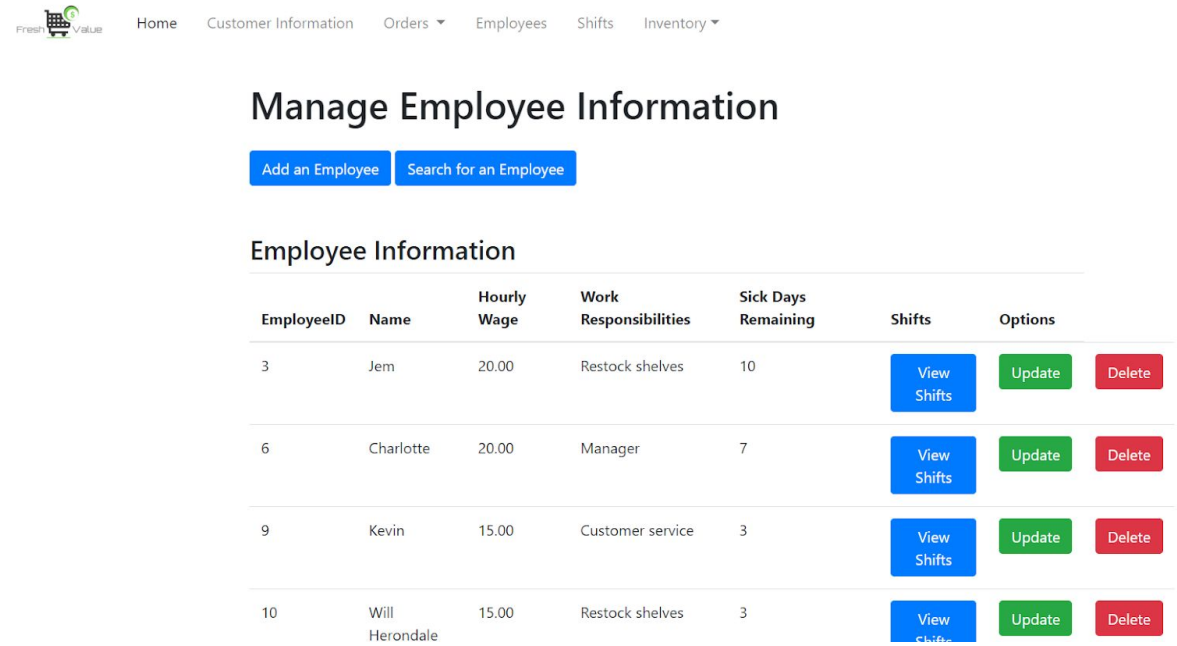
Insert a New Customer Into the Customers Table



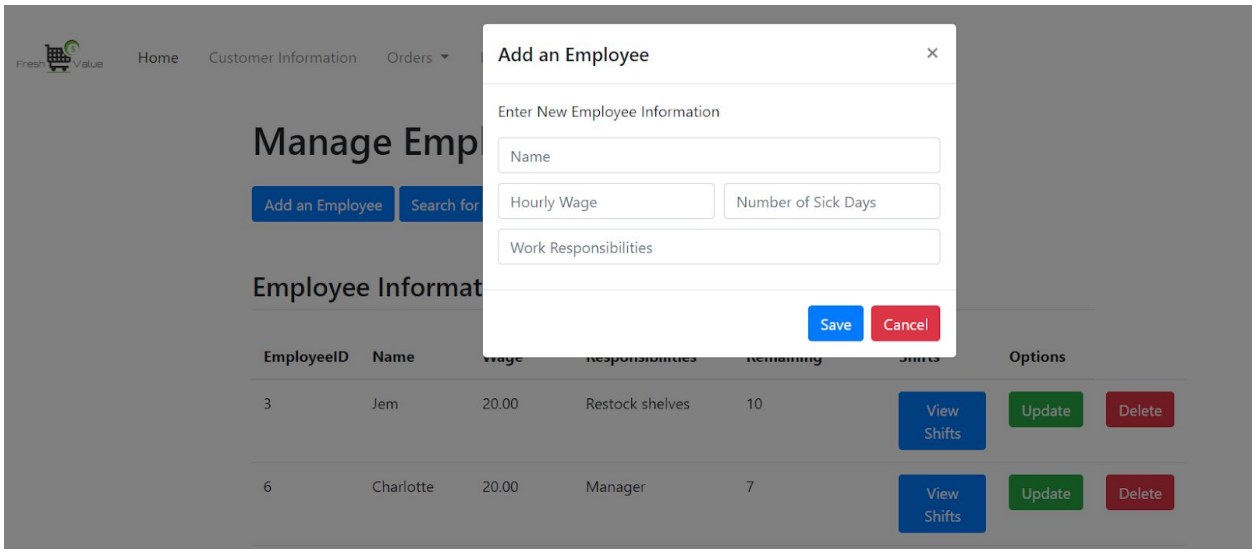
Search for Customers by Name, Phone Number, or Rewards Points



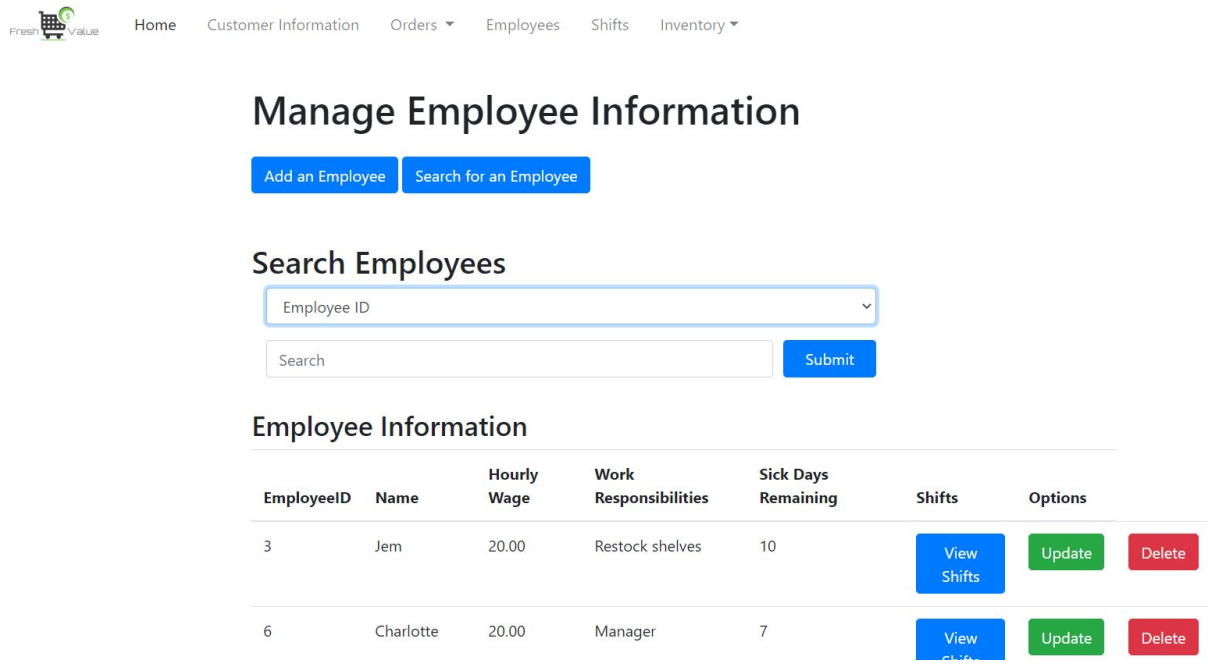
Display Employees Table with Ability to Update and Delete Rows and View Shifts an Employee is Assigned to Work



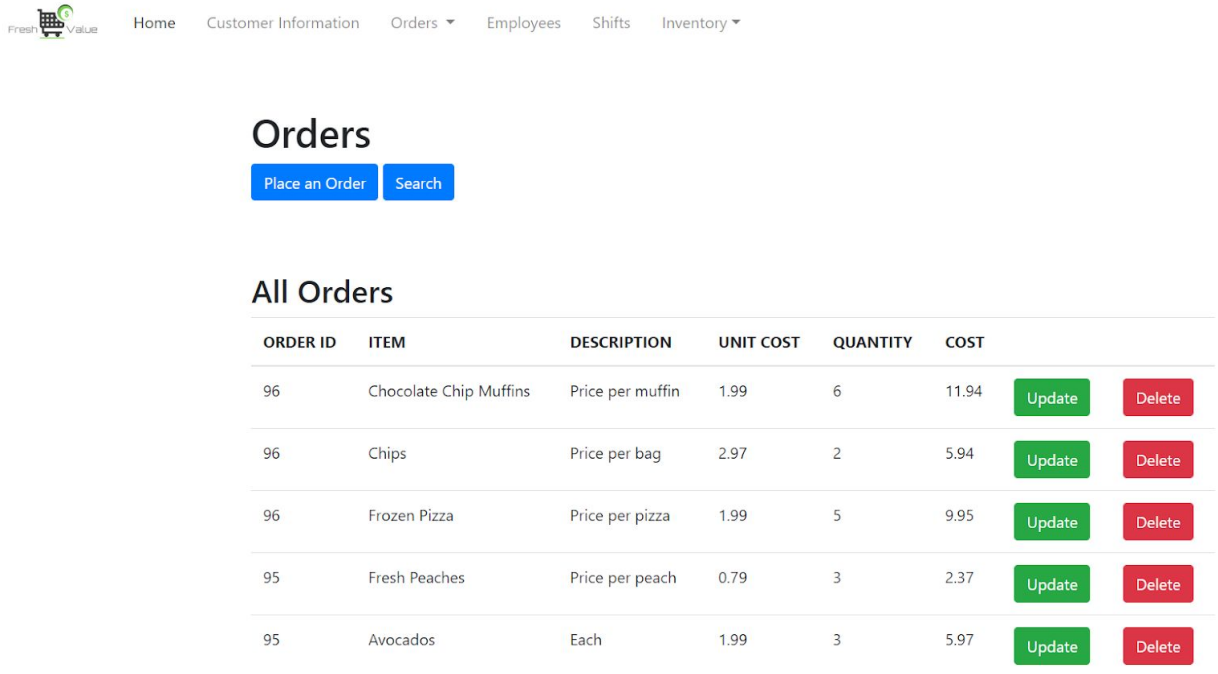
Insert a New Employee into the Employees Table



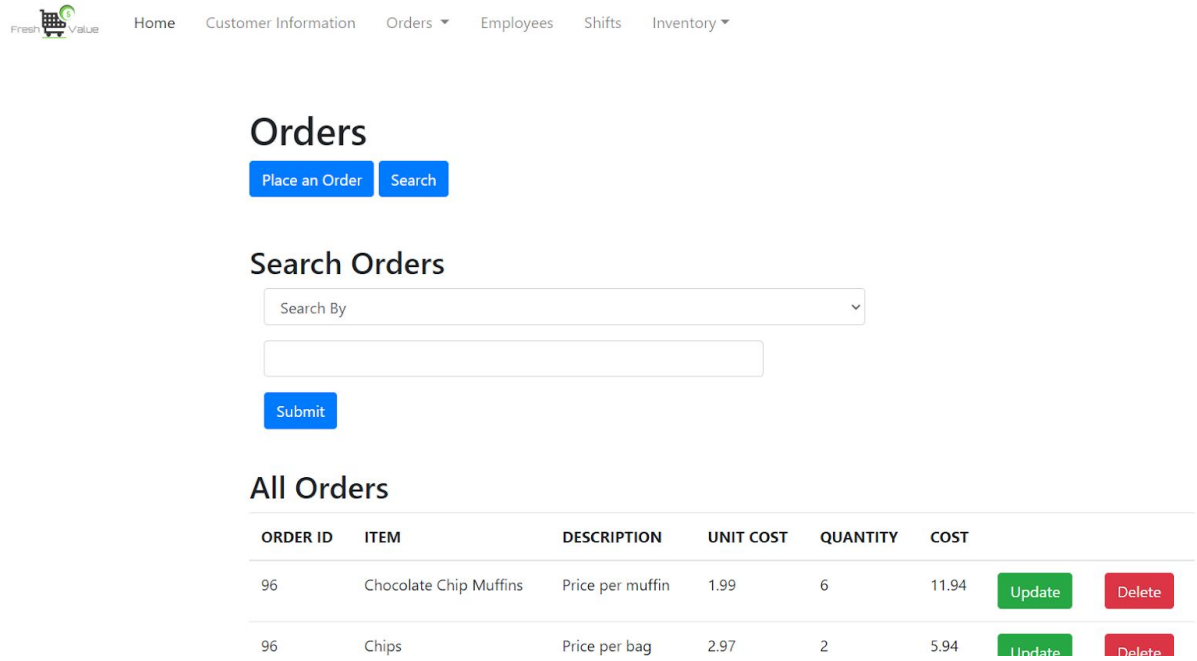
Search for Employees by Employee ID, Name, Work Responsibilities, Hourly Wage, or Number of Sick Days



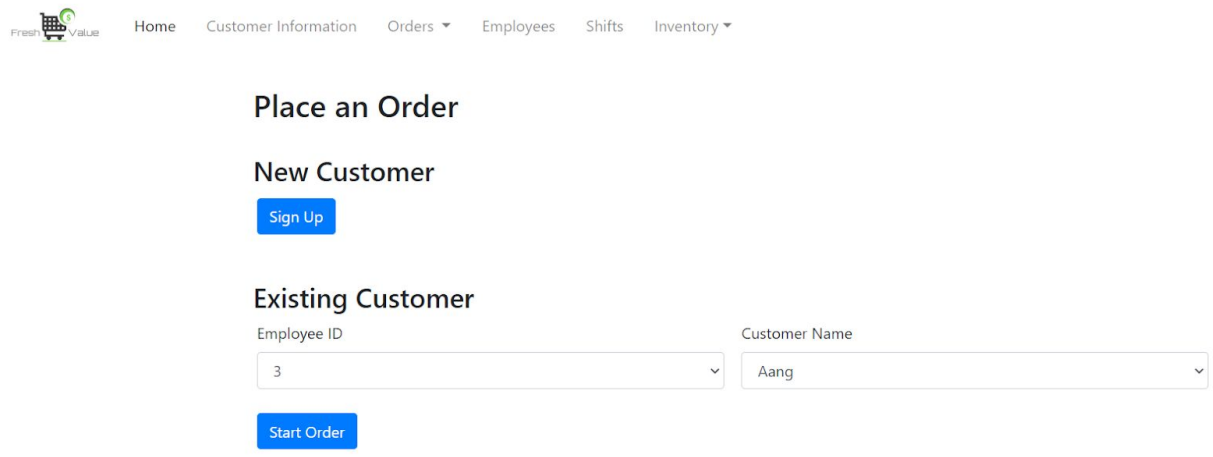
Display Orders Table with Ability to Update and Delete Rows



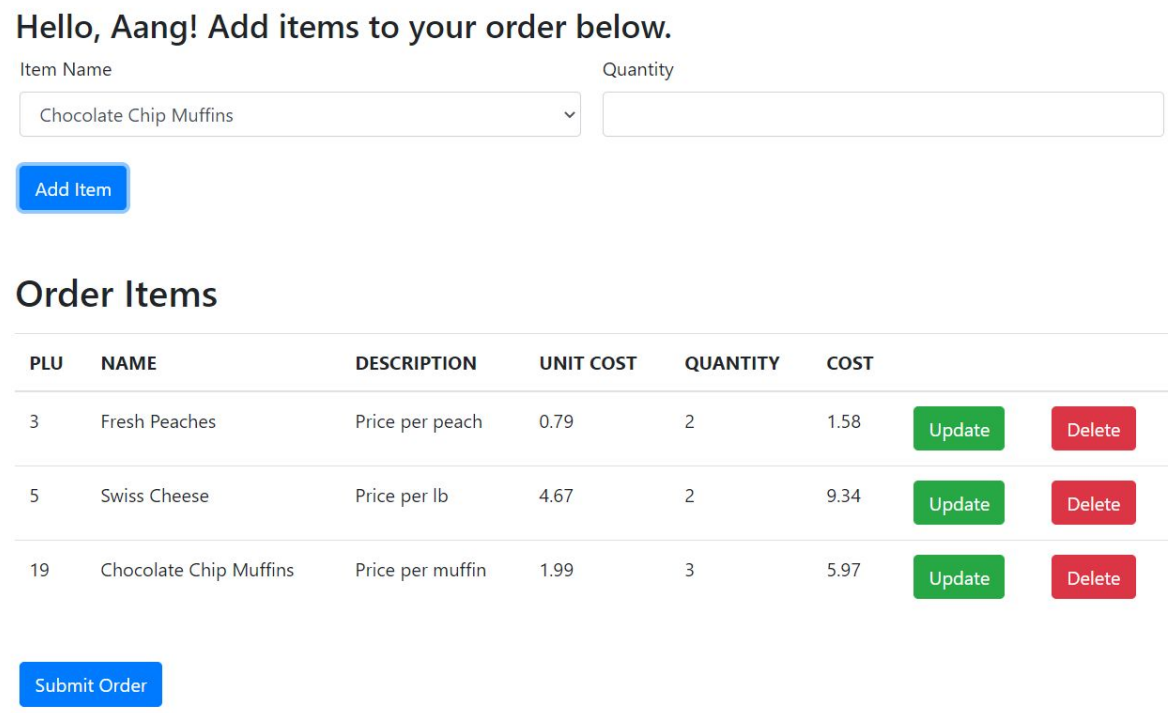
Search For Orders by Rewards ID (Customer ID), Customer Name, Customer Phone Number, or Employee Name



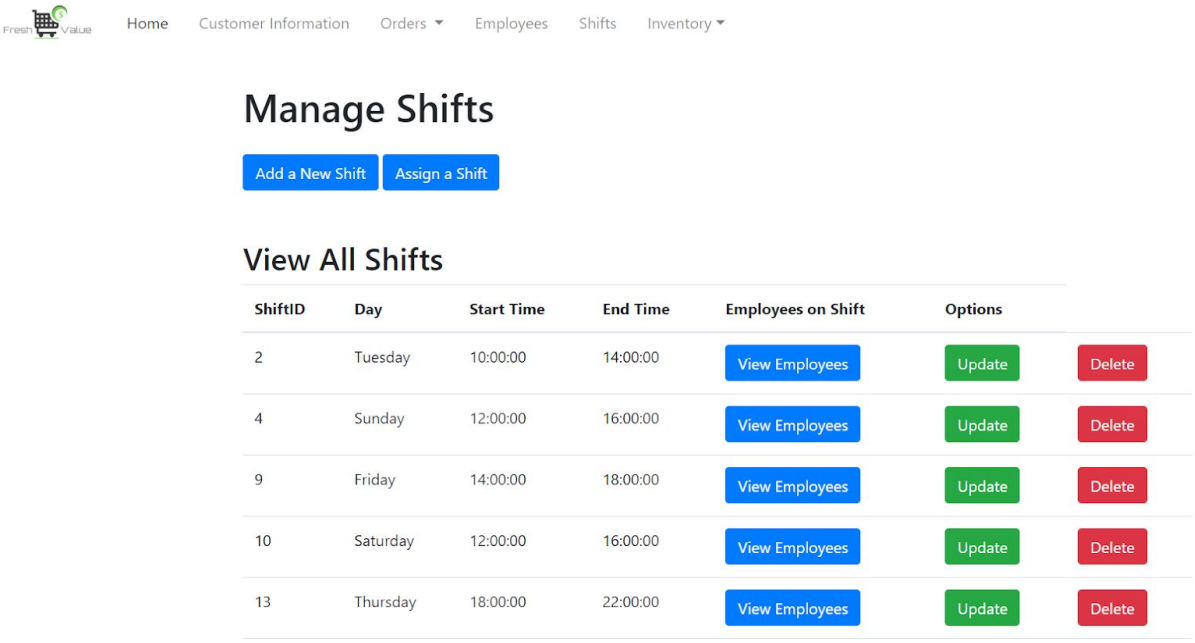
Start a New Order by Selecting a Customer and Assigning an Employee



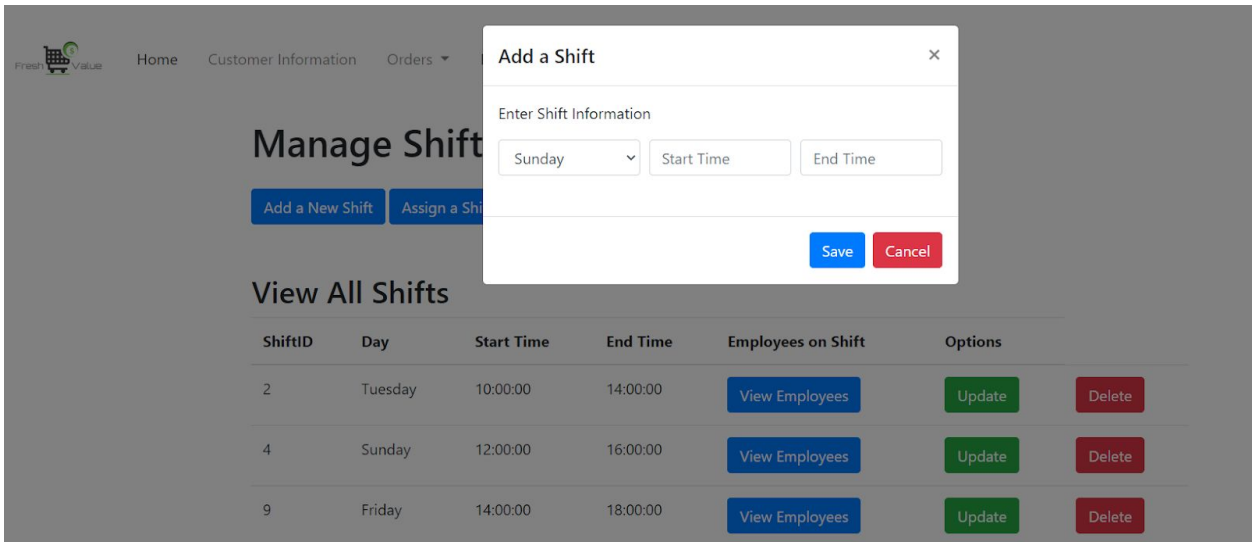
Add Items to Order on the “Place an Order” Page with the Ability to Update or Delete Items in The Order



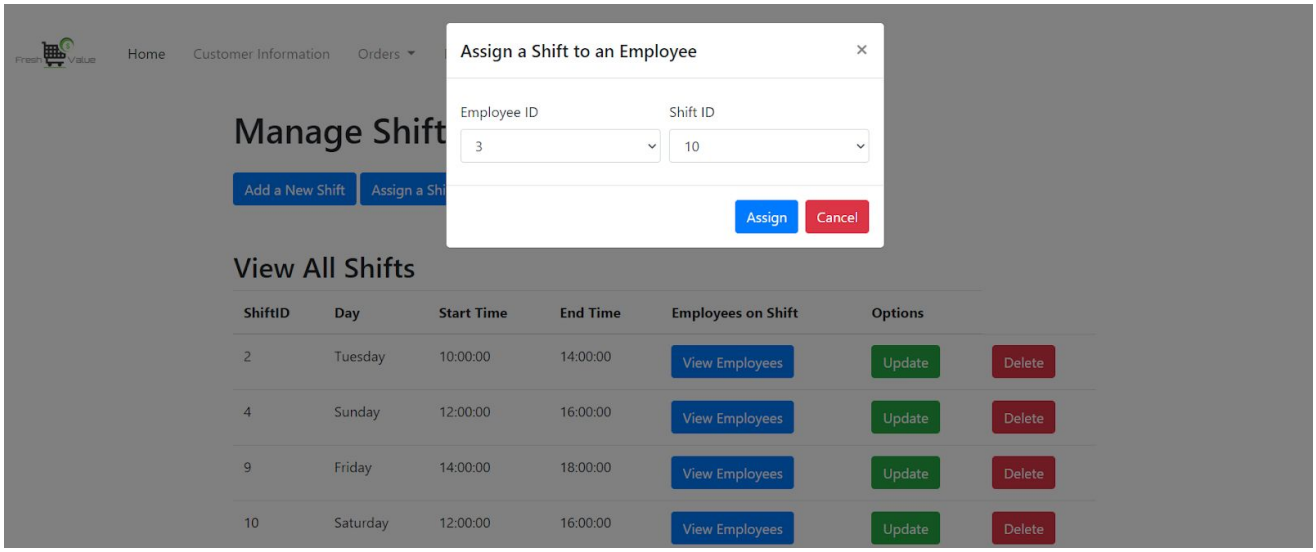
Display Shifts Table with Ability to Update and Delete Rows and to View Employees Assigned to a Shift



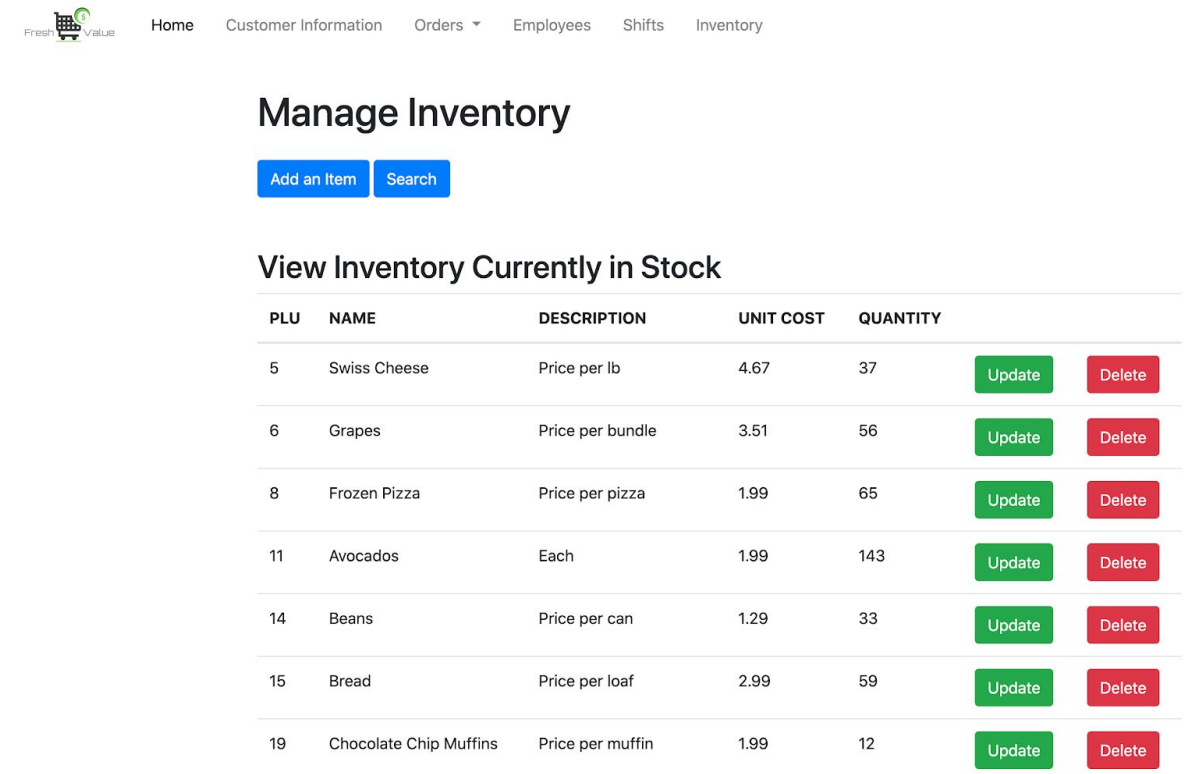
Insert a New Shift Into the Shifts Table



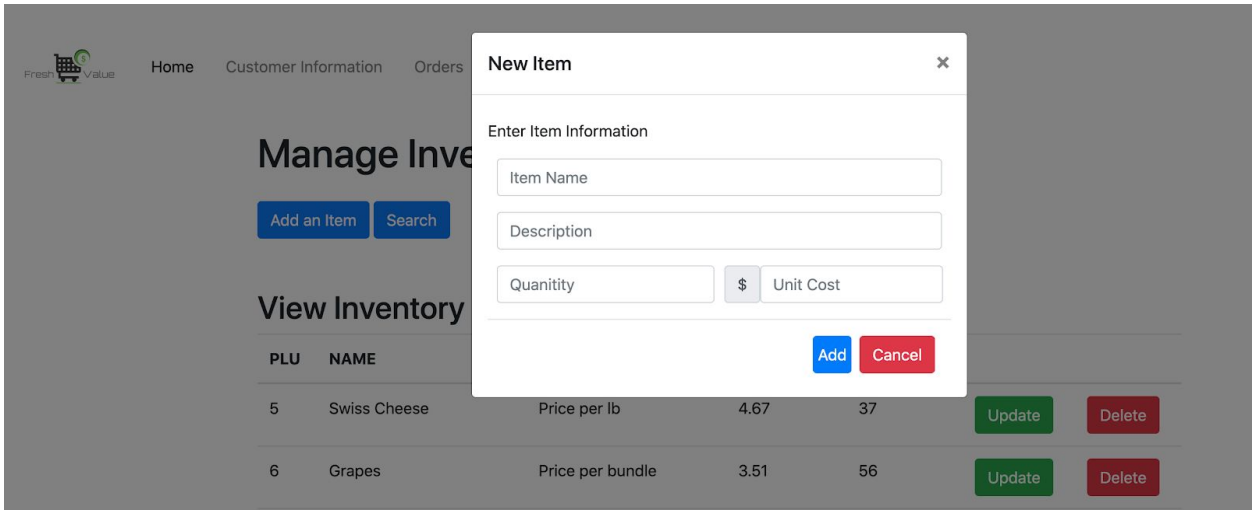
Insert a New Row Into the EmployeeShifts Table by Selecting Employee ID and Shift ID



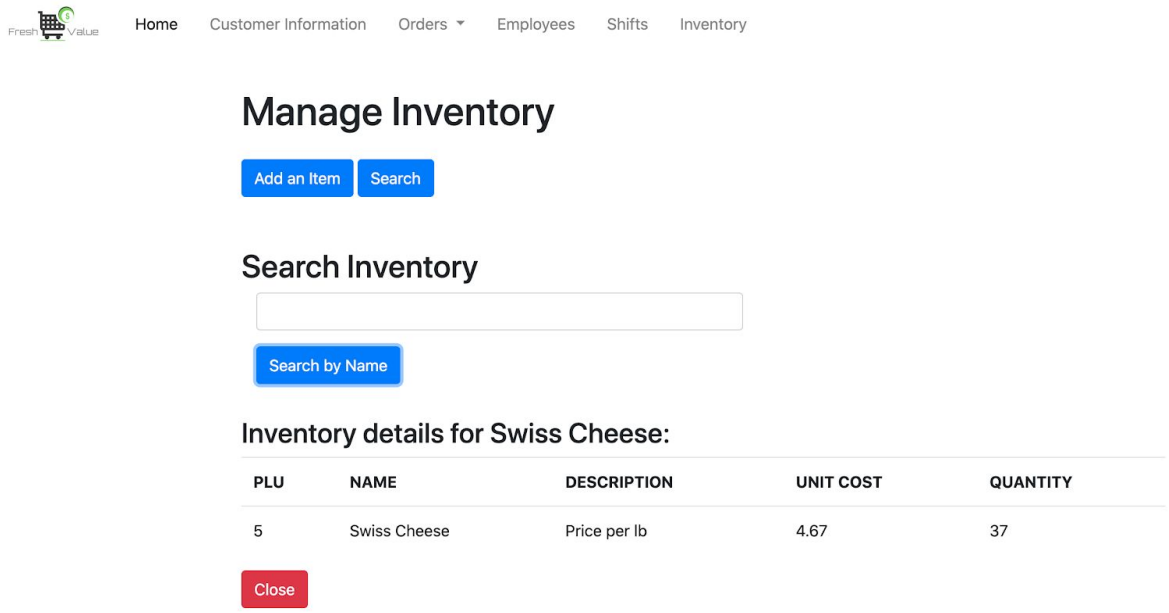
Display Inventory Table with Ability to Update and Delete Rows



Insert a New Item into the Inventory Table



Search for an Item in Inventory by Name



Summary of Learning and Development Process

Throughout the quarter, we have received feedback from our peers and TA regarding our project and have made several changes based on that feedback.

When we first started our project design, we were unclear on how to implement our many to many relationships. Based on guidance from our TA, we added the EmployeeShifts and OrderItems composite tables to implement the many to many relationships for Employees and Shifts and Orders and Inventory tables. Additionally, we eliminated the Manager attribute from the Employees entity as it was not necessary for our application.

As we continued to draft our Project and Database Outlines, our peers pointed out naming inconsistencies throughout our outlines and diagrams. We updated our entity and attribute names for consistency across our ERD, schema, and outlines. Based on additional feedback, we added a Primary Key to the EmployeeShifts table.

After solidifying our designs, we began creating the user interface. After receiving feedback, we created the Manage Orders page to allow users to view and search for specific orders. We also reorganized the layout of the pages to include Update and Delete buttons for each row in the table rather than only at the top of the page to make it clear to the user which line items they would be changing. We also pared down the number of header buttons to only Add and Search options to provide a cleaner user interface.

We then wrote the SQL queries required for our project. After reviewing peer feedback, we corrected the syntax of our queries in order to utilize the auto_increment feature when inserting new items into each table.

Finally, we utilized Flask, a micro-framework for Python, to implement CRUD functionality throughout our website. We ran into a few issues with the Place an Order page throughout this process but eventually got it working. We also added error handling to all of our forms based on peer feedback, which greatly improved the usability of our pages. Our peers also found an error in two of the search functionalities on the Orders page which are now resolved. Additionally, we removed the Order Inventory page from the website because its purpose was not apparent and it seemed to cause confusion for many of our peers.

We believe that our final project meets all of the project specifications and is now fully functioning. The peer reviews were very helpful throughout the design process and helped us catch many errors that we may have otherwise missed.