DATABASE MANAGEMENT SYSTEMS

BAKERY DATABASE

31.05.2021

2019103016

Diya Dhandapani

2019103037

Mullapudi Renuka Devi

OVERVIEW

INTRODUCTION

The project ***"Bakery Database"*** is developed to maintain the records of products available , customers and their orders and the employees.

FRONT END :	html , CSS , Bootstrap Frameworks
BACK END :	Node js
DATABASE SERVER:	MySQL Server 8.0
DATABASE TOOLS:	MySQL Shell 8.0.25

RELATIONS UNDER THE BAKERY DATABASE:

The system contains the following relations:

- Customers
- Employee
- Products
- Order_Desp
- Order_list

CREATING BAKERY DATABASE:

```
MySQL  localhost:33060+ ssl  temp  SQL > CREATE DATABASE BAKERY;
Query OK, 1 row affected (0.0046 sec)
MySQL  localhost:33060+ ssl  temp  SQL > USE BAKERY;
Default schema set to `BAKERY`.
Fetching table and column names from `bakery` for auto-completion... Press ^C to stop.
```

CUSTOMERS:

ATTRIBUTES:

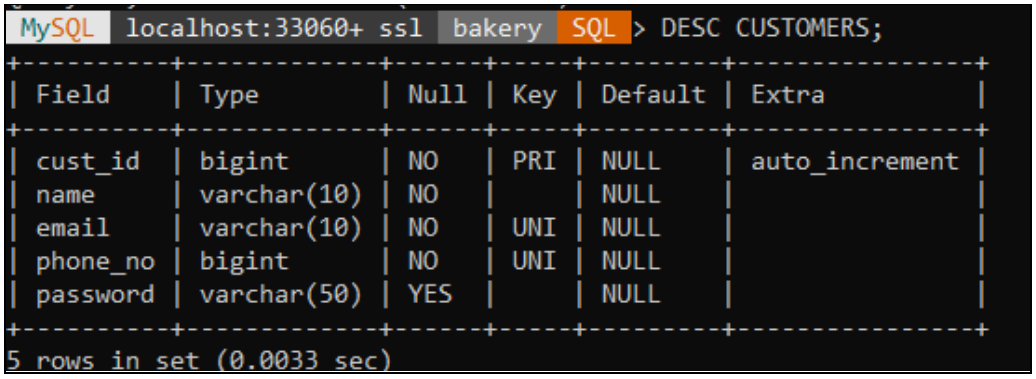
- Cust_id - : Customer ID (Primary Key)
- name : - Customer Name
- Email:- Customer Email
- Phone_no:- Customer Phone Number
- Password:- Customer Password

This table holds details of the customers.

FUNCTIONALITIES:

- The customer can login to the existing account using their email id and password or sign up to create a new account
- After logging in , the customer can view all their previous orders. They can also view products and place and edit new orders.

All attributes are non-transitively determined by the Primary Key(Super Key) CUST_ID. Hence, it is in Boyce-Codd Normal Form.



```
MySQL  localhost:33060+ ssl  bakery  SQL > DESC CUSTOMERS;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| cust_id  | bigint      | NO   | PRI | NULL    | auto_increment |
| name     | varchar(10) | NO   |     | NULL    |                |
| email    | varchar(10) | NO   | UNI | NULL    |                |
| phone_no | bigint      | NO   | UNI | NULL    |                |
| password | varchar(50) | YES  |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
5 rows in set (0.0033 sec)
```

EMPLOYEE/ADMIN:

ATTRIBUTES:

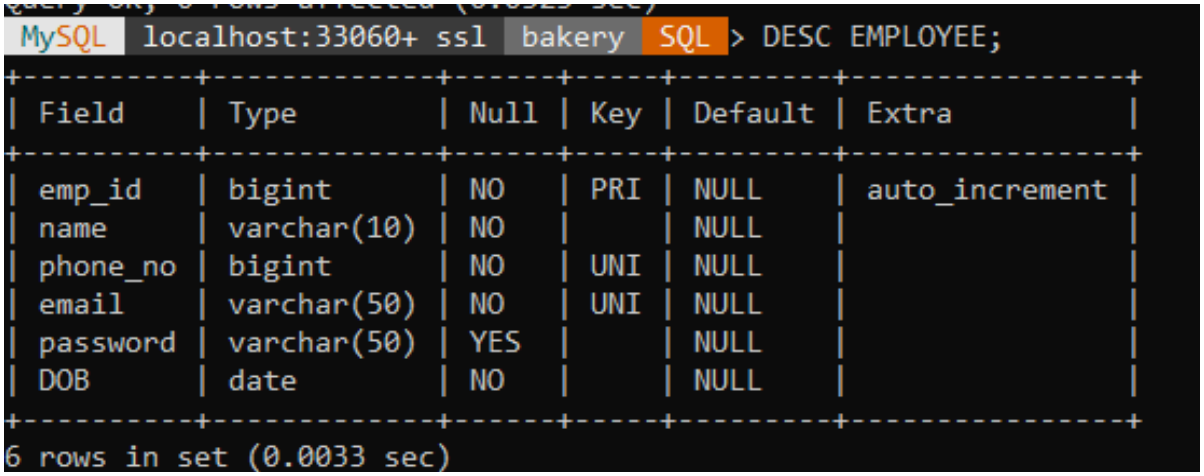
- Emp_id:- Employee ID (Primary Key)
- Name :- Employee Name
- Email :- Employee Email
- Phone_no :- Employee phone Number
- Password : - Employee Password
- DOB :- Date of Birth of employee

This table holds the details of the Employees.

FUNCTIONALITIES:

- The respective employee can login to the existing account using their employee ID and password
- After logging in , the employee can view and edit all the products and only the orders assigned to them

All attributes are non-transitively determined by the Primary Key(Super Key) EMP_ID. Hence, it is in Boyce-Codd Normal Form.



```
MySQL  localhost:33060+ ssl  bakery  SQL > DESC EMPLOYEE;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| emp_id   | bigint      | NO   | PRI | NULL    | auto_increment |
| name     | varchar(10) | NO   |     | NULL    |                |
| phone_no | bigint      | NO   | UNI | NULL    |                |
| email    | varchar(50) | NO   | UNI | NULL    |                |
| password | varchar(50) | YES  |     | NULL    |                |
| DOB      | date        | NO   |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
6 rows in set (0.0033 sec)
```

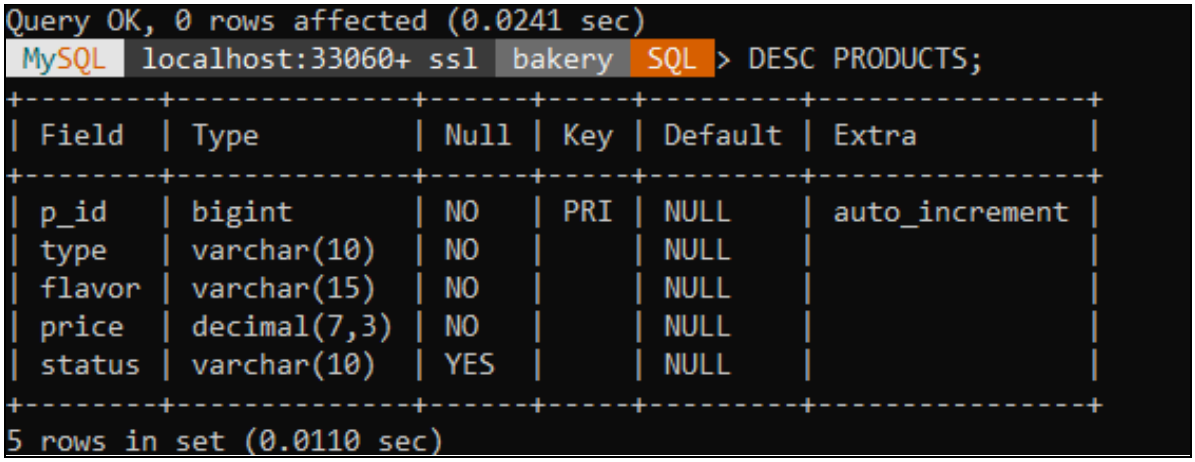
PRODUCTS:

ATTRIBUTES:

- P_id :- Product ID (Primary Key)
- Type :- Product Type
- Flavour :- Product Flavour
- Price :- cost of product
- Status : - Status (available or not available)

This table holds the details of each product. The product table can be edited only by the employee through the user interface.

All attributes are non-transitively determined by the Primary Key(Super Key) P_ID. Hence, it is in Boyce-Codd Normal Form.



```
Query OK, 0 rows affected (0.0241 sec)
MySQL localhost:33060+ ssl bakery SQL > DESC PRODUCTS;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| p_id   | bigint       | NO   | PRI | NULL    | auto_increment |
| type   | varchar(10)  | NO   |     | NULL    |                |
| flavor | varchar(15)  | NO   |     | NULL    |                |
| price  | decimal(7,3) | NO   |     | NULL    |                |
| status | varchar(10)  | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
5 rows in set (0.0110 sec)
```

ORDER LIST:

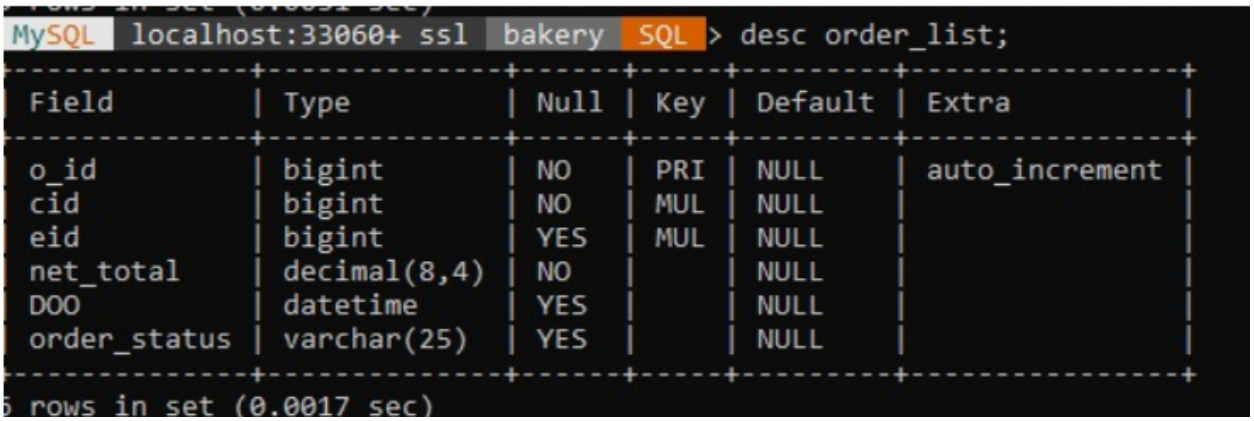
ATTRIBUTES:

- O_id :- Order ID (Primary Key)
- Cid :- Customer ID (Foreign key references to cust_id in Customers)
- Eid :- Employee/Admin ID (Foreign key references to Employees)
- DOO :- Date and Time of order placed
- Order_status :- Order status (confirmed, waiting etc)
- Net_total :- Total billing Amount

This table contains each and every order made by the customer. Although it doesn't describe the order thoroughly, it provides the necessary information.

The order status can be altered only by the respective employee, through the user interface, before it is delivered.

All attributes are non-transitively determined by the Primary Key(Super Key) CUST_ID. Hence, it is in Boyce-Codd Normal Form.



```
MySQL  localhost:33060+ ssl  bakery  SQL > desc order_list;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| o_id         | bigint       | NO   | PRI | NULL    | auto_increment |
| cid          | bigint       | NO   | MUL | NULL    |                |
| eid          | bigint       | YES  | MUL | NULL    |                |
| net_total    | decimal(8,4) | NO   |     | NULL    |                |
| DOO          | datetime     | YES  |     | NULL    |                |
| order_status | varchar(25)  | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
6 rows in set (0.0017 sec)
```

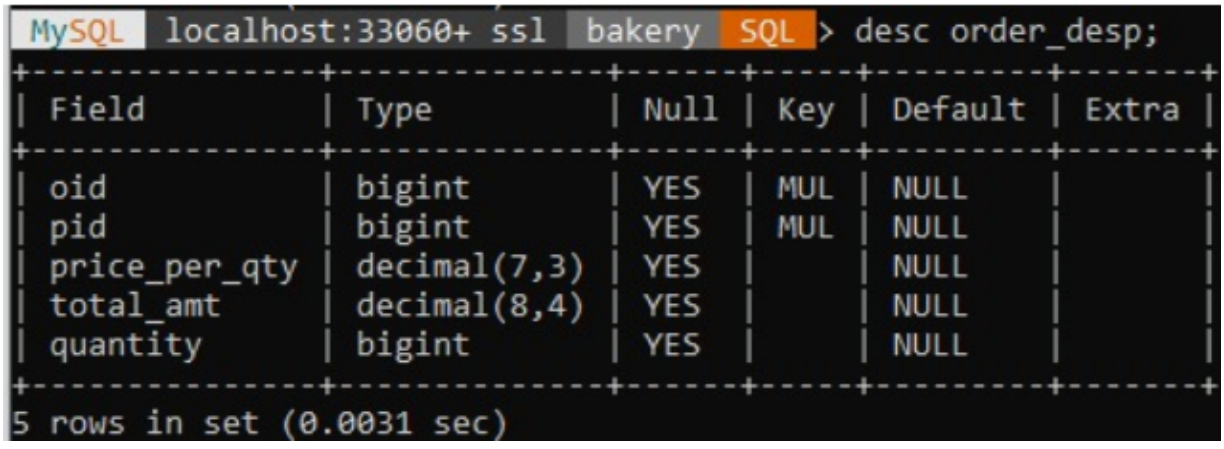
ORDER DESCRIPTION:

ATTRIBUTES:

- O_id :- Order ID (Foreign key references to o_id in Order List)
- Pid :- Product ID (Foreign key references to pid in Products)
- Prie_per_qty :- Price per one item
- Total_amt :- Total price (quantity * price of item)
- Quantity :- quantity added by the customer

The order description describes each order of the order list. It can be modified by the customer through the user interface.

It provides a thorough description of each order and is also used to generate the VIEW for the bill.



```
MySQL localhost:33060+ ssl bakery SQL > desc order_desp;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| oid           | bigint       | YES  | MUL | NULL    |       |
| pid           | bigint       | YES  | MUL | NULL    |       |
| price_per_qty | decimal(7,3) | YES  |     | NULL    |       |
| total_amt     | decimal(8,4) | YES  |     | NULL    |       |
| quantity      | bigint       | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
5 rows in set (0.0031 sec)
```

SETTING UP PROCEDURES

A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database. For our database we have included a procedure called 'CALCULATE BILL', to compute the bill amount for each order. The procedure is called from the server using the Node JS script.

The procedure takes the 'Order ID' as a parameter and calculates the total from the 'Order Desp' table for the respective order id.

```
MySQL  localhost:33060+ ssl  bakery  SQL > DELIMITER //
MySQL  localhost:33060+ ssl  bakery  SQL > CREATE PROCEDURE CALCULATE_BILL (IN order_id BIGINT)
                                         -> BEGIN
                                         -> SELECT sum(total_amt) AS "TOTAL" FROM ORDER_DESP WHERE oid = order_id;
                                         -> END//
Query OK, 0 rows affected (0.0157 sec)
```

SETTING UP VIEWS

VIEWS are virtual tables that do not store any data of their own but display data stored in other tables. A view can contain all or a few rows from a table. For our database we have included a view called 'BILL', that stores data from the 'Products ' and 'Order Desp' tables. The view is created on the server using the Node JS script.

The view is created as necessary to display order details and reflects any changes that may come up from editing orders.

```
MySQL  localhost:33060+ ssl  bakery  SQL > desc bill;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| p_id         | bigint       | NO   |     | 0       |       |
| type         | varchar(10)  | NO   |     | NULL    |       |
| flavor       | varchar(15)  | NO   |     | NULL    |       |
| price_per_qty| decimal(7,3) | YES  |     | NULL    |       |
| total_amt    | decimal(8,4) | YES  |     | NULL    |       |
| quantity     | bigint       | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
6 rows in set (0.0047 sec)
```

REQUIREMENTS

FUNCTIONAL REQUIREMENTS:

The system maintains the data of :

- Customer and Employee Details
- Details of various products
- Orders booked by the customers (current and previous)
- Billing details of every order placed

The system also provides the following actions:

- Editing (removing, adding ,modifying)
  - the products in the cart - **customer's privilege**
  - the orders placed by customers - **employee's privilege**
  - The product list in the database - **employee's privilege**

NON FUNCTIONAL REQUIREMENTS:

The system provides the following :

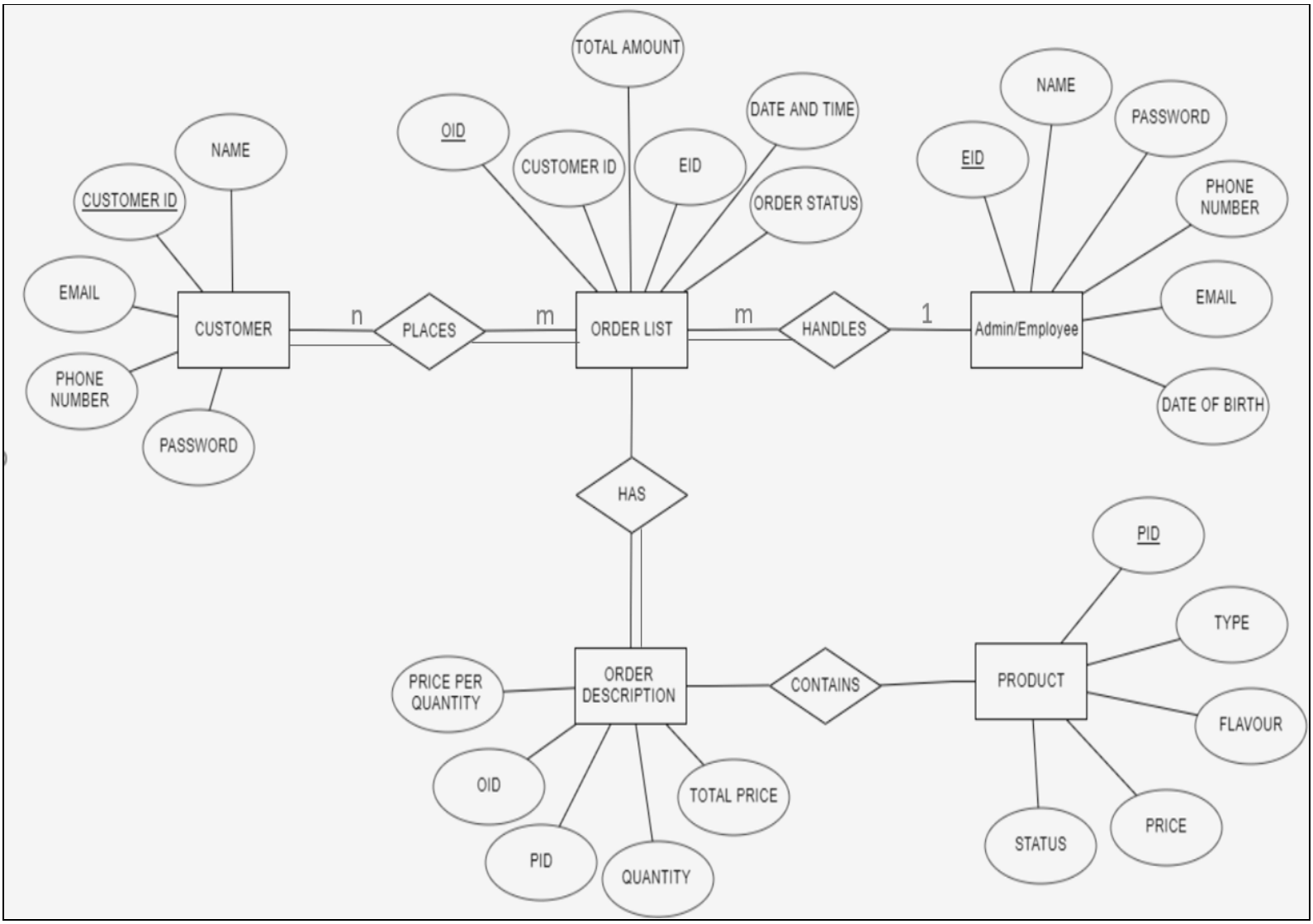
- USER INTERFACE:

  A friendly User Interface has been developed for better customer experience. In this case bootstrap frameworks have been used for a responsive web page.

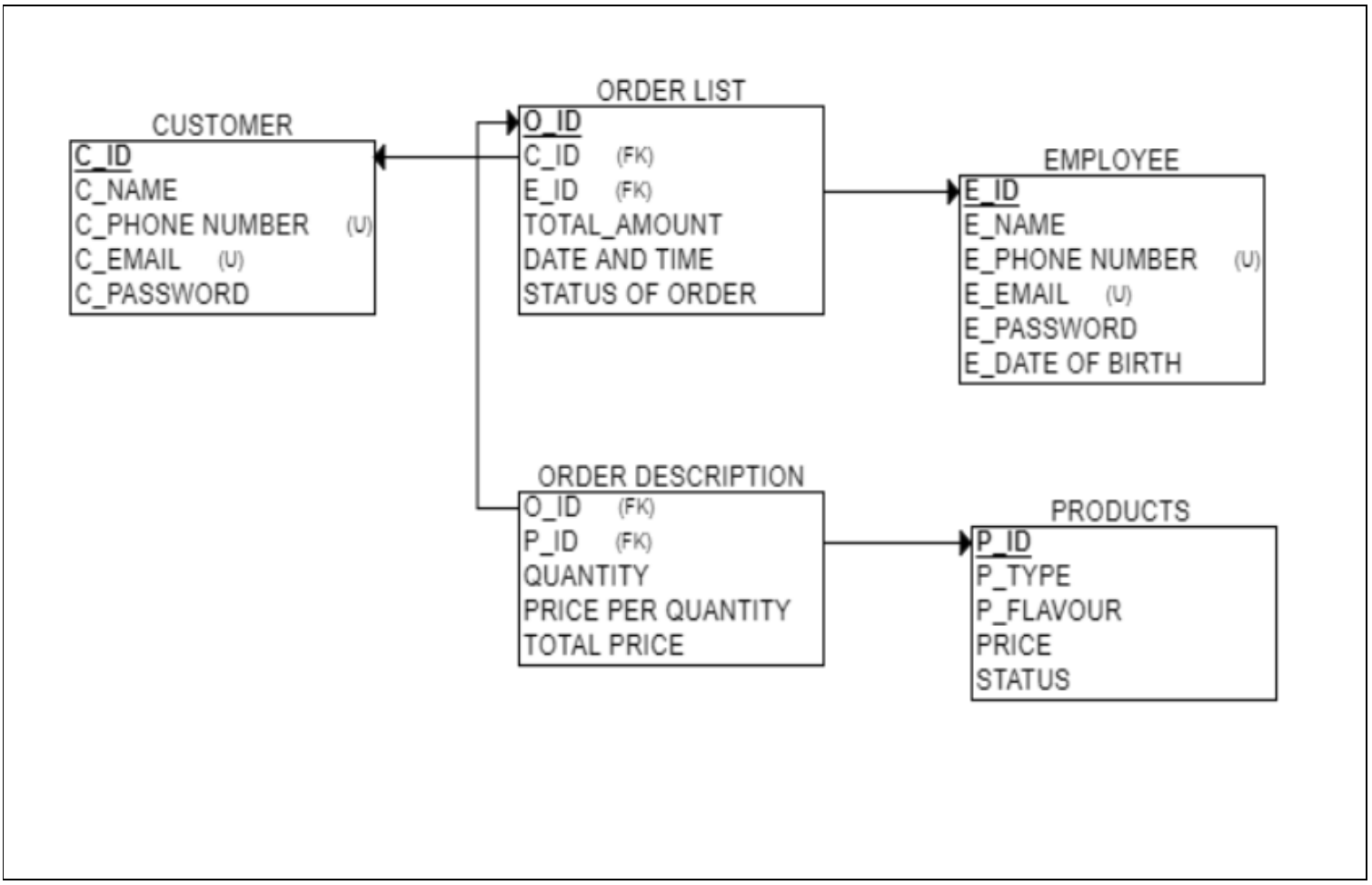
- A VIEW is used to display order details
- A PROCEDURE is used to calculate the amount of the bill

ER DIAGRAM AND RELATIONAL SCHEMA

ER DIAGRAM:



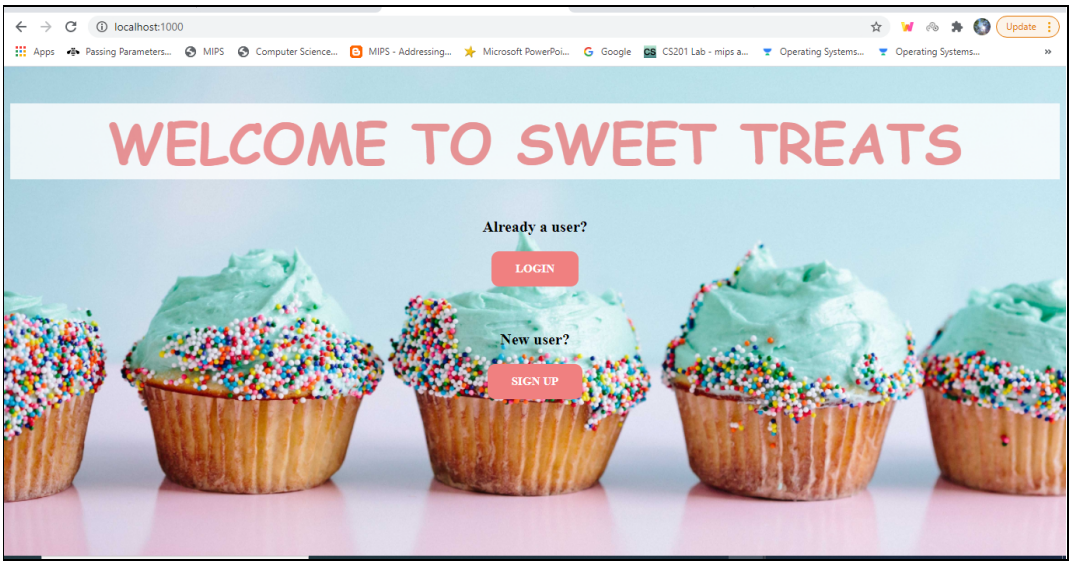
RELATIONAL SCHEMA:



User Interface Design

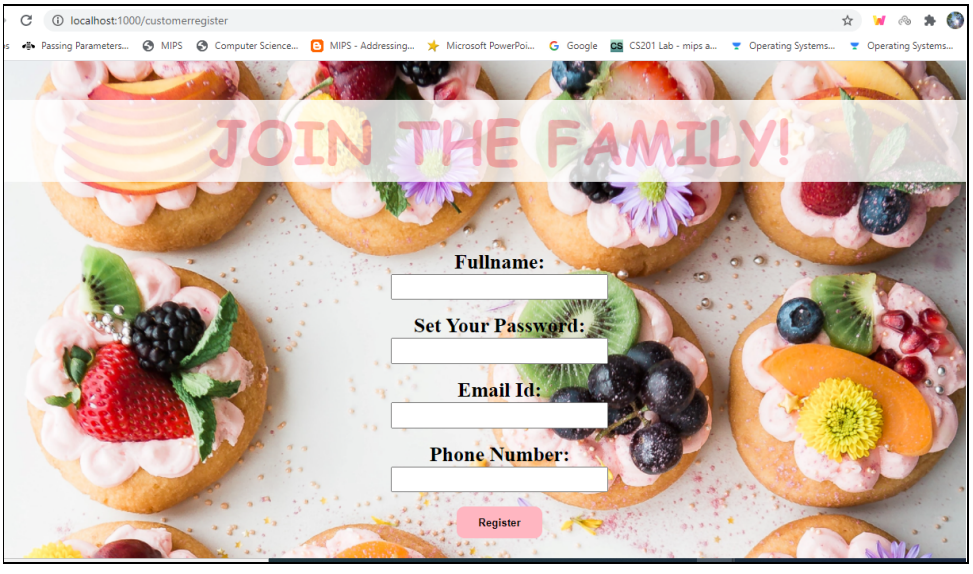
CUSTOMER INTERFACE:

Landing Page:



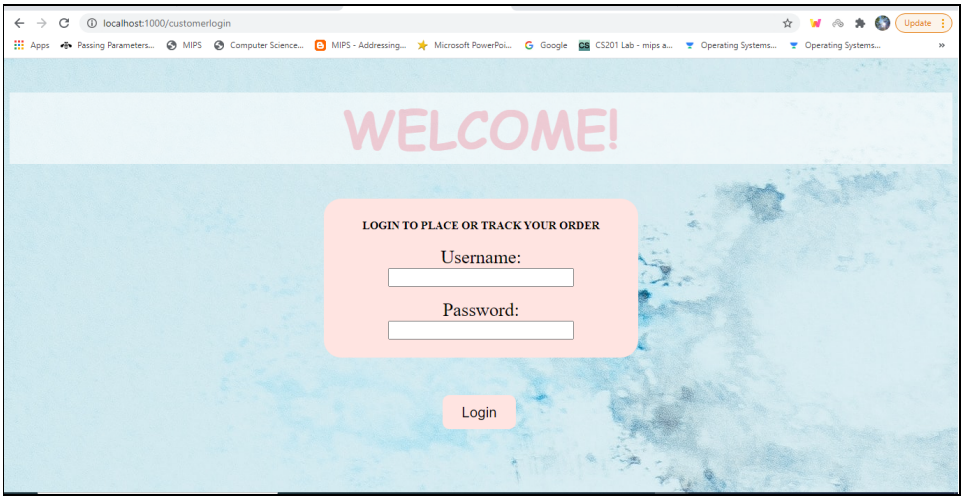
Sign up Page :

The User/Customer can create a new account by providing basic details like name, phone number, email and set a password. After successfully signing up, they can log in to their account. The email ID and phone number must be unique.



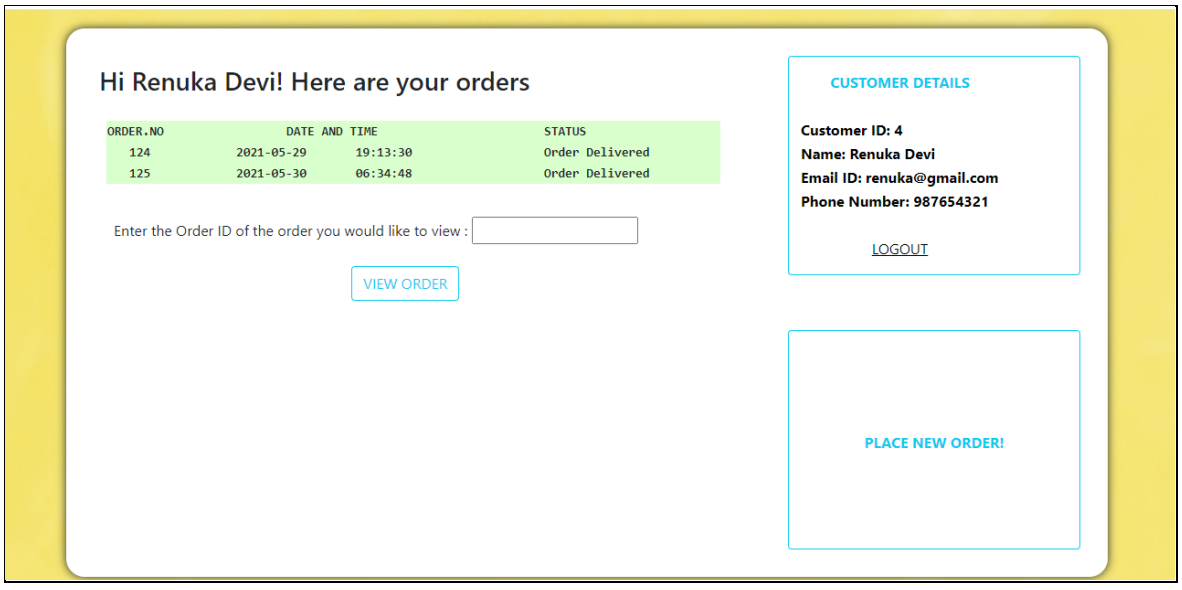
Login Page:

Here , The Customer/User must enter the correct credentials of one's own account which will redirect them to another page where the Users/Customers can place, edit new orders or view previous orders.



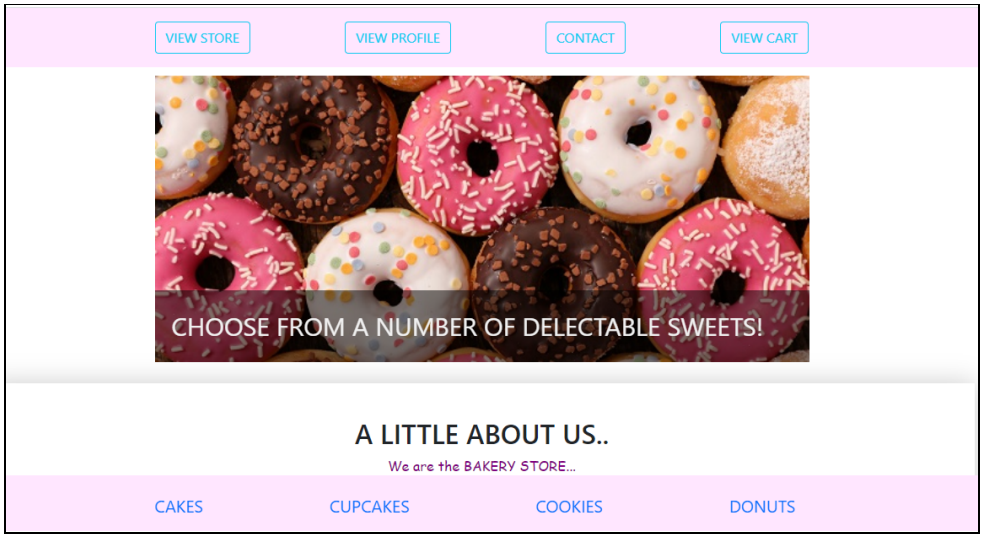
View All Orders:

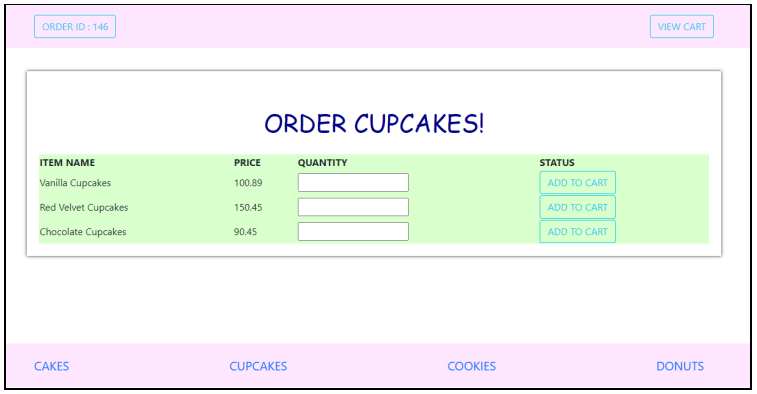
Here the Users/Customers can view all the details of their orders and can visit the product page to place a new order. The customer can also choose to logout here if they wish.



Products Page:

The Users/Customers can view all the products available by choosing the category they desire (lower navigation bar) and choose the items along with the required quantities.



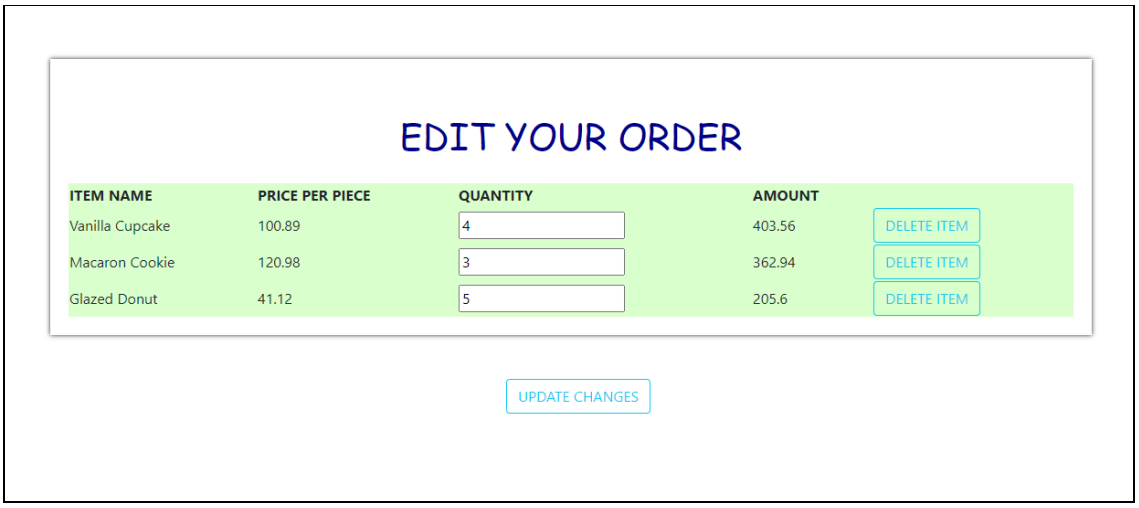


Cart, Edit , Place Order:

The Users/Customers can add their required products to the cart , edit the quantities or remove them , and confirm their order.

View cart:

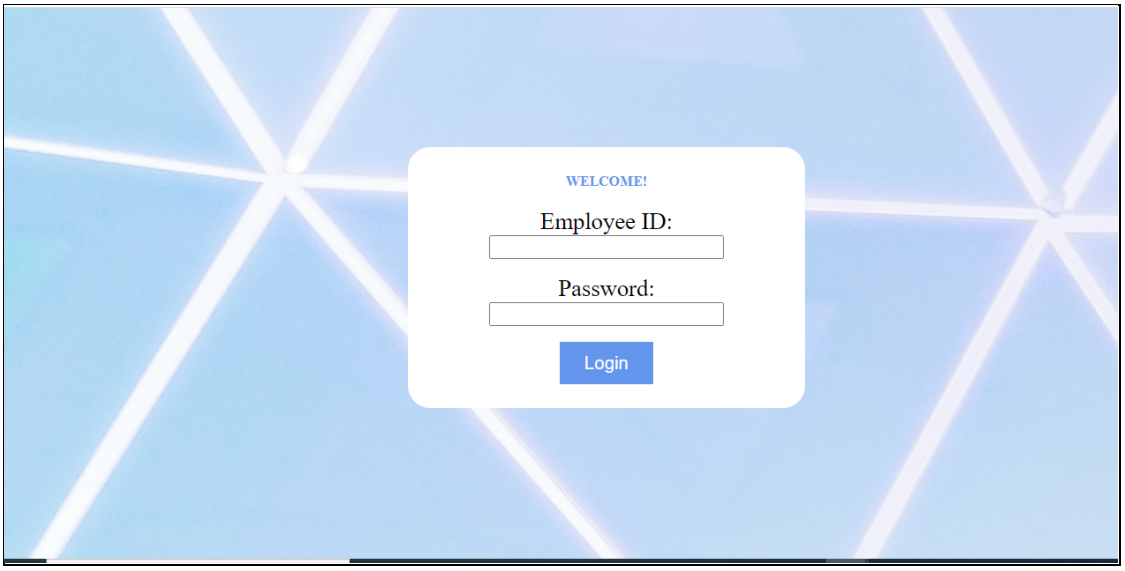
Editing cart:



EMPLOYEE/ADMIN INTERFACE

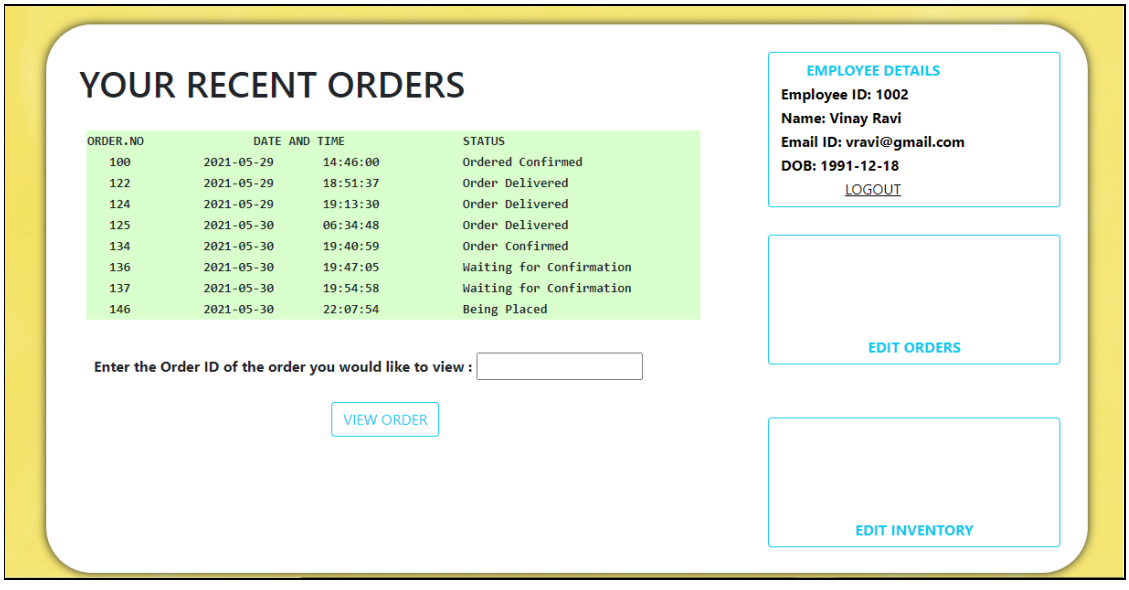
Login Page:

Here , The Employee/Admin must enter the correct credentials of one's own account which will redirect them to another page where they can edit or view the inventory or their orders.



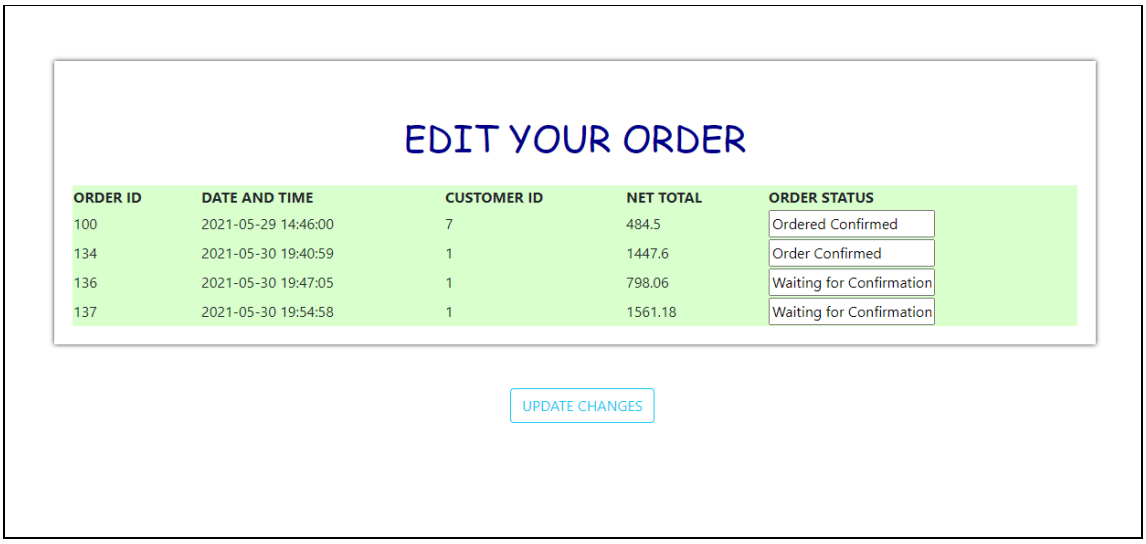
View All Orders:

Here the Employee/Admin can view all the details of their orders and can also choose to edit the orders or the inventory as well. The user can also choose to logout here if they wish.



Editing Orders:

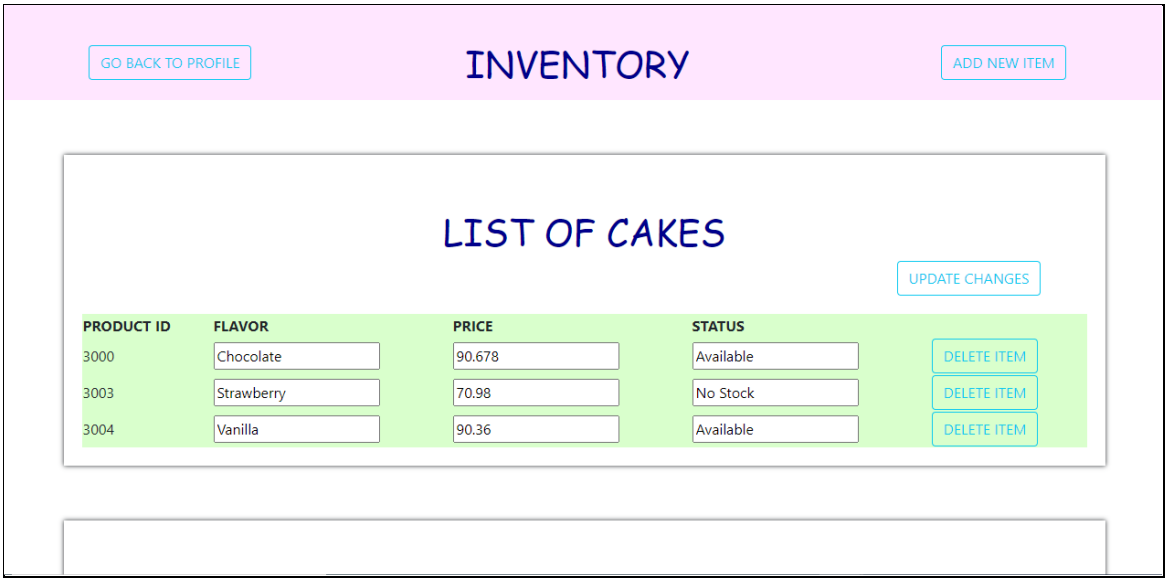
Here the Employee/Admin has the privilege to alter the order status of any order, assigned to them, that hasn't been delivered yet.



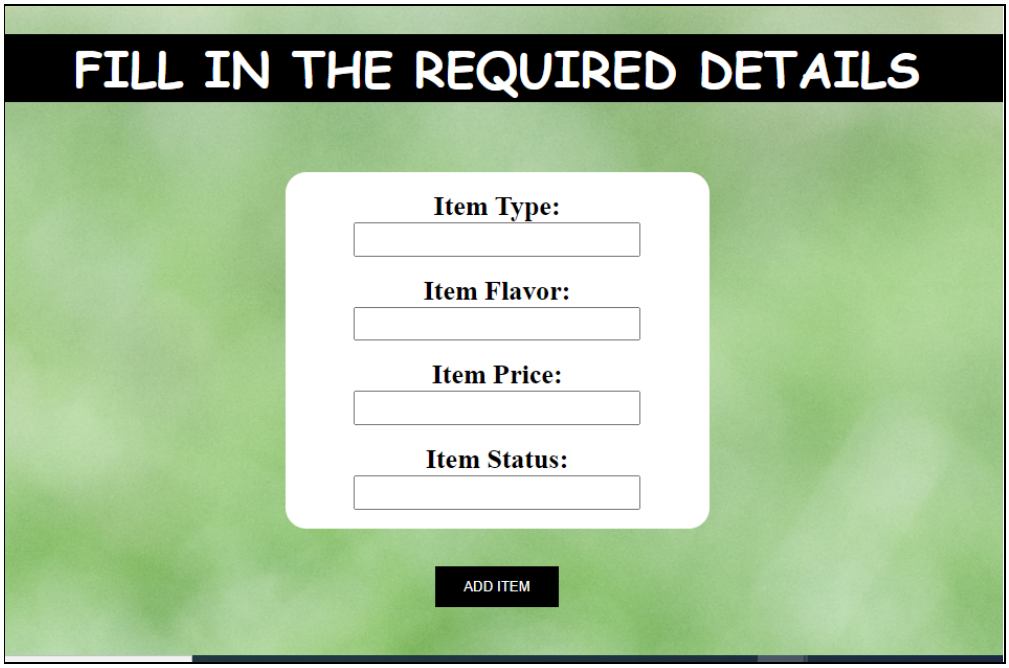
Editing the Inventory:

Here the Employee/Admin has the privilege to alter the details of existing products and even add new products.

Editing existing products:



Adding new products:



Implementation:

1. Connecting to the database and testing the connection

```
/CREATE DB CONNECTION
onst db = mysql.createConnection({
    host: "localhost",
    port: "3306",
    user: "root",
    password: "ddd1810",
    database: "bakery"
});

/TEST DB CONNECTION
b.connect((err) => {
    if (err) throw err;
    else {
        console.log("Bakery DB connected...");
    }
});
```

2. Setting up the server

```
/SETTING UP SERVER AND TESTING CONNECTION
app.listen(1000, () => {
    console.log('Listening on Port 1000...');
});
```

3. User registration

```
//GETTING DATA FROM REGISTRATION PAGE
app.post('/register', (req, res) => {

    name = req.body.Fullname;           //getting all the data
    password = req.body.Setpassword;
    email = req.body.emailid;
    phone_no = req.body.PhoneNumber;

    let sql = 'SELECT * FROM CUSTOMERS WHERE email = ? OR phone_no = ?';   //comparing the the em
    db.query(sql, [email, phone_no], (err, data, fields) => {
        if (err) console.log("ERROR WHEN CHECKING");

        if (data.length >= 1) {
            var msg = email + " or " + phone_no + " already exists";
            res.render('customerregister', { alertMsg: msg });
        }

        else {
            let sql = 'INSERT INTO CUSTOMERS (name,email,phone_no,password) VALUES (?,?,?,?)';
            db.query(sql, [name,email,phone_no,password], (err, data, fields) => {
                if (err) throw err;

                else {
                    console.log("USER REGISTERED");
                }
            });
            res.redirect('/customerlogin');
        }
    });
});
```

4. Verifying data for the login page :

Based on the data entered by the user in the form mentioned in the user interface, it is verified with the data in our database and authenticated. Else the necessary message is displayed.

For customer :

```
/VERIFYING DATA FROM LOGIN PAGE
pp.post('/login', (req, res) => {

    email = req.body.username;
    password = req.body.password;

    let sql = 'SELECT * FROM CUSTOMERS WHERE email = ? AND password = ?';
    db.query(sql, [email, password], (err, data, field) => {
        if (err) throw err;

        else if (data.length > 0) {

                req.session.loggedin = true;
                req.session.username = email;
            //var user = email;

            res.redirect('/customerprofile');
            console.log("USER LOGGED IN");
        }

        else {
            res.render('customerlogin', { alertMsg: "Your Email Address or password is wrong" });
        }

    });

);
```

For employee:

```
//AUTHENTICATING EMPLOYEE LOGIN
app.post('/emp-login', (req, res) => {
    e_id = req.body.emp_id;
    password = req.body.password;

    let sql = 'SELECT * FROM EMPLOYEE WHERE emp_id = ? AND password = ?';
    db.query(sql, [e_id, password], (err, data, field) => {
        if (err) throw err;

        else if (data.length > 0) {

            req.session.loggedin = true;
            req.session.user = e_id;

            res.redirect('/employeeprofile');
            console.log("EMPLOYEE LOGGED IN");
        }

        else {
            res.render('employeelogin', { alertMsg: "Your Employee ID or password is wrong" });
        }

    });


});
```

5. Displaying the user profile - their details and orders

This displays the order details from the 'Order_List' table for the respective user.

It only shows the essential details to the user. The user can choose to view any specific order thoroughly if they wish (explained in the following part). The details are displayed using an 'ejs' template.

For customer:

```
137     sql = 'SELECT o_id, DOO, order_status FROM ORDER_LIST WHERE cid = (SELECT cust_id FROM CUSTOMERS WHERE email = ?)';
138     db.query(sql, [req.session.username], (err, data, fields) => {
139
140         if (err) throw err;
141
142         else {
143
144             console.log(data.order_status);
145             for (i = 0; i < data.length; i++)
146                 data[i].DOO = dateFormat(data[i].DOO, 'yyyy-mm-dd       HH:MM:ss');
147
148
149             res.render('customerprofile.ejs', { displayData: data, detail });
150         }
151     });
152 });
```

For employee:

```
574     sql = 'SELECT o_id, DOO, order_status FROM ORDER_LIST WHERE eid = ?';
575     db.query(sql, [req.session.user], (err, data, fields) => {
576
577         if (err) throw err;
578
579         else {
580
581             for (i = 0; i < data.length; i++)
582                 data[i].DOO = dateFormat(data[i].DOO, 'yyyy-mm-dd       HH:MM:ss');
583
584             console.log(detail);
585             res.render('employeeprofile.ejs', { displayData: data, detail});
586         }
587     });
588
589 });
```

6. Viewing a specific order

The user (customer/employee), can have a detailed view of any specific order if they wish. For this purpose, the VIEW is created as shown.

```
154  //DISPLAYING ALL PREVIOUS ORDERS
155  app.post('/view-order', (req, res) => {
156
157      if(o == null)
158          var o = req.body.order_id;
159
160      let sql = 'CREATE OR REPLACE VIEW BILL AS SELECT p.type, p.flavor,' +
161      'o.price_per_qty, o.total_amt, o.quantity FROM PRODUCTS p, ORDER_DESP o WHERE p.p_id = o.pid AND o.oid = ? ';
162      db.query(sql, [o], (err, data, fields) => {
163
164          if (err) throw err;
165          else {
166              console.log('viewing order' + o);
167
168              sql = 'SELECT * FROM BILL';
169              db.query(sql, (err, data, fields) => {
170
171                  if (err) throw err;
172
173                  else
174                      // res.render('cart.ejs', {displayData : data});
175                      displayData = data;
```

The procedure is also called to calculate the bill amount as necessary.

```
177                      total_q = 0;
178                      for (i = 0; i < data.length; i++) {
179                          total_q += data[i].quantity;
180                      }
181                  });
182
183                  sql = 'CALL CALCULATE_BILL(?)';
184                  db.query(sql, [o], (err, data, fields) => {
185
186                      if (err) throw err;
187
188                      else {
189
190                          t = data[0];
191                          console.log(displayData, t);
192                          res.render('order.ejs', { displayData, t, total_q, o });
193                      }
194                  });
195              }
196          });
197
198  });
199
```

CUSTOMER SPECIFIC FUNCTIONALITIES

7. Placing a new order

When the customer wishes to place a new order, a new entry is added to the 'Order List' table and populated as necessary.

```
221
222
223             sql = 'INSERT INTO ORDER_LIST (cid,eid,net_total,DOO,order_status) VALUES (?,?,?,?,?)';
224             db.query(sql, [custid, e, t, date_o, o_status], (err, data, fields) => {
225
226                 if (err) throw err;
227                 else
228                     console.log('INSERTED INTO ORDER LIST');
229             });
230
231             //console.log(custid,date_o);
232
233             sql = 'SELECT o_id FROM ORDER_LIST WHERE cid = ? AND DOO = ?';
234             db.query(sql, [custid, date_o], (err, data, fields) => {
235
236                 order = data[0].o_id;
237
238                 if (err) throw err;
239             });
240         }
241
242     });
243 });
```

8. Adding items to the cart

When the customer adds products into the cart according to their requirements, a new entry is added to the 'Order Desp' table with the respective 'order id'.

```
319 //ADDING A PRODUCT TO THE CART
320 app.post('/add', (req, res) => {
321
322     id = req.body.prod;
323     pid = id[0];
324     q = req.body.qty;
325
326     for (i = 0; i < q.length; i++) {
327         console.log(q[i]);
328     }
329     console.log(pid);
330
331     sql = 'SELECT price, type FROM PRODUCTS WHERE p_id = ?';
332     db.query(sql, [pid], (err, data, field) => {
333
334     if (err) throw err;
335
336     else {
337         p = data[0].price;
338         page = data[0].type;
339         total = 0;
340         for (i = 0; i < q.length; i++) {
341             total = (p * q[i]);
```

```
342
343                 if (total != 0) {
344                     quant = q[i];
345                     break;
346                 }
347             }
348             console.log(total, quant, pid);
349
350             sql = 'INSERT INTO ORDER_DESP ( oid, pid, price_per_qty, total_amt, quantity) VALUES (?,?,?,?,?)';
351             db.query(sql, [order, pid, p, total, quant], (err, data, result) => {
352
353                 if (err) throw err;
354
355                 else {
356                     console.log('ORDER ADDED');
357                     var msg = "ITEM ADDED";
358                 }
359             });
360         }
361
362         });
363     });
364
```

9. Displaying cart

The cart is displayed in the same manner as **6. Viewing a specific order.**

The concept of VIEWS and PROCEDURES is implemented here as well.

10. Deleting an item from the cart

The customer can delete items as they wish. This gets reflected in the 'Order Desp' table for the specific order.

```
435 //DELETING ITEM
436 app.get('/:id/delete', (req, res) => {
437
438     console.log('IN DELETE-ITEM');
439    // type = req.body.type;
440    // flavor = req.body.flavor;
441     p_id = req.params.id;
442     console.log(p_id,'DELETING ITEM');
443
444     sql = 'DELETE FROM ORDER_DESP WHERE oid = ? AND pid = ?';
445     db.query(sql, [order, p_id], (err, data, fields) => {
446
447         if (err) throw err;
448
449         else {
450             console.log('ITEM DELETED');
451
452         }
453     });
454
455 });
```

11. Modifying quantities of specific products placed in the cart

Any changes made to the quantity of the product is reflected in the 'Order Desp' table as well.

```
464  //UPDATING ORDER CHANGES
465  app.post('/update-change', (req, res) => {
466
467      console.log('Making changes to the order');
468
469      food_type = req.body.type;
470      food_flavor = req.body.flavor;
471
472      new_quant = req.body.quant;
473
474      console.log(food_type,food_flavor,new_quant,order);
475
476      sql = 'SELECT * FROM BILL';
477      db.query(sql, (err, data, fields) => {
478
479          if (err) throw err;
480
481          else {
482
483              console.log(data);
484              for (i = 0; i < data.length; i++)
485                  if (new_quant[i] != data[i].quantity) {
486
```

```
483              console.log(data);
484              for (i = 0; i < data.length; i++)
485                  if (new_quant[i] != data[i].quantity) {
486
487                      console.log(new_quant[i], order, food_type[i], food_flavor[i]);
488
489                      sql = 'UPDATE ORDER_DESP SET quantity = ? WHERE oid = ? AND pid =' +
490                      '(SELECT p_id FROM PRODUCTS WHERE type = ? AND flavor = ?)';
491                      db.query(sql, [new_quant[i], order, food_type[i], food_flavor[i]], (err, data, flavor) => {
492
493                          if (err) throw err;
494
495                          else
496                              console.log('CHANGES UPDATED');
497                      });
498                  }
499
500              res.redirect('/cart');
501          }
502
503      });
504
505  });
```

EMPLOYEE SPECIFIC FUNCTIONALITIES

12. Editing 'Order Status' of specific orders

The employee can alter the status of any order assigned to them, before it is delivered. The changes are reflected in the 'Order List' table and can be viewed by the customer as well.

```
614
615  //UPDATING CHANGES AFTER CHANGING ORDER STATUS
616  app.post('/update-order-change', (req, res) => {
617
618      console.log('Employee making changes to the order');
619
620      var s = 'Order Delivered';
621      o_id = req.body.oid;
622      o_status = req.body.status;
623
624      console.log(o_id,o_status);
625
626     let sql = 'SELECT o_id, order_status FROM ORDER_LIST WHERE eid = ? AND order_status != ?';
627      db.query(sql, [req.session.user, s], (err, data, fields) => {
628
629          if (err) throw err;
630
631          else {
632
633              for (i = 0; i < data.length; i++)
634                  if (o_status[i] != data[i].order_status) {
```

```
635
636                      console.log(o_status[i], o_id[i]);
637
638                      sql = 'UPDATE ORDER_LIST SET order_status = ? WHERE o_id = ?';
639                      db.query(sql, [o_status[i], o_id[i]], (err, data, flavor) => {
640
641                          if (err) throw err;
642
643                          else
644                              console.log('ORDER CHANGES UPDATED');
645                      });
646                  }
647
648              res.redirect('/employeeprofile');
649          }
650
651      });
652
653  });
```

13. Editing the details of existing products

The employee can choose to modify the 'Flavor, price or status' of a product already existing. These changes are reflected in the 'Products' table and can be viewed by the customer as well.

```
692
693     //UPDATING INVENTORY CHANGES
694     app.post('/:id/update/inv', (req, res) => {
695
696         type = req.params.id;
697         console.log('IN UPDATE INVENTORY');
698         pid = req.body.pid;
699         flavor = req.body.fl;
700         price = req.body.p;
701         status = req.body.s;
702         console.log(type, flavor, status, price);
703
704         let sql = 'SELECT * FROM PRODUCTS WHERE type = ?';
705         db.query(sql, [type], (err, data, fields) => {
706
707             if (err) throw err;
708
709             else {
710
```

```
711             for (i = 0; i < data.length; i++) {
712
713                 if (data[i].flavor != flavor[i] || data[i].price != price[i] || data[i].status != status[i]) {
714                     sql = 'UPDATE PRODUCTS SET flavor = ?, price = ?, status =? WHERE p_id = ?';
715                     db.query(sql, [flavor[i], price[i], status[i], pid[i]], (err, data, fields) => {
716
717                         if (err) throw err;
718
719                         else
720                             console.log(pid[i] + 'updated');
721                     });
722                 }
723
724             }
725
726             res.redirect('/edit_inv');
727         }
728     });
729 });
730
731 //DELETING ITEM FROM INVENTORY
```

14. Adding a new product to the list

The employee can enter the required details and thus add to the 'Products' table. The addition can be viewed by the customer after the required changes are made in the database.

```
669
670   //ADDING NEW ITEM TO INVENTORY
671   app.get('/add-new-item', (req, res, next) => {
672       res.render('add_new_item.ejs');
673   });
674
675   app.post('/add-new-item', (req, res, next) => {
676
677       type = req.body.type;
678       flavor = req.body.flavor;
679       price = req.body.price;
680       status = req.body.status;
681
682       let sql = 'INSERT INTO PRODUCTS (type,flavor,price,status) VALUES (?,?,?,?)';
683       db.query(sql, [type,flavor,price,status], (err, data, fields) => {
684
685           if (err) throw err;
686
687           else
688               res.redirect('/edit_inv');
689       });
690
691   });
692
```

15. Deleting from the products table

The employee can also delete existing products and the necessary changes are reflected in the 'Products' table.

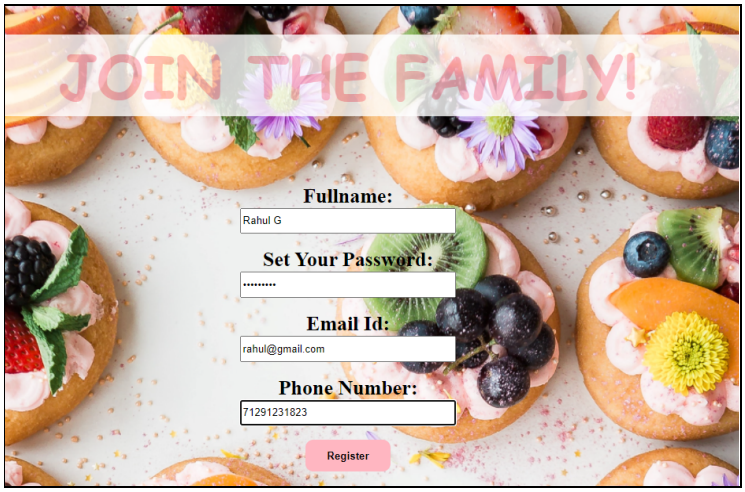
```
730
731   //DELETING ITEM FROM INVENTORY
732   app.get('/:id/delete/inv', (req, res) => {
733
734       console.log('IN DELETE-ITEM FROM INVENTORY');
735       p_id = req.params.id;
736       console.log(p_id, 'DELETING ITEM');
737
738       sql = 'DELETE FROM PRODUCTS WHERE p_id = ?';
739       db.query(sql, [p_id], (err, data, fields) => {
740
741           if (err) throw err;
742
743           else {
744               console.log('ITEM DELETED FROM INVENTORY');
745
746           }
747       });
748
749   });
```

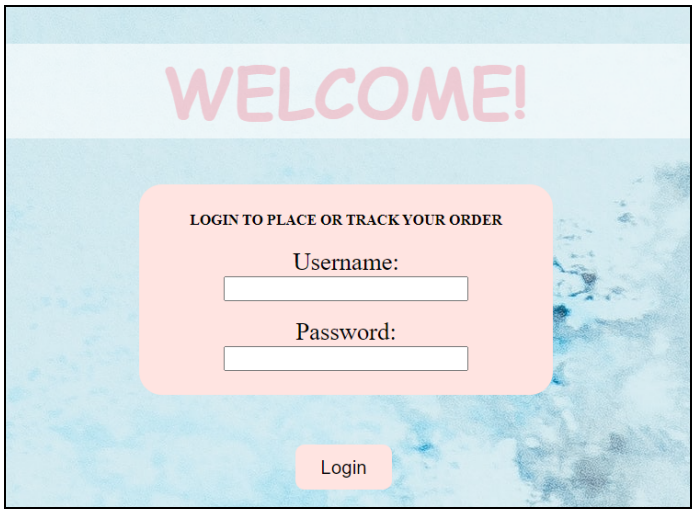
Results:

Customer Interface:

- **User Registration -**

After a successful registration - User gets directed to the Login Page

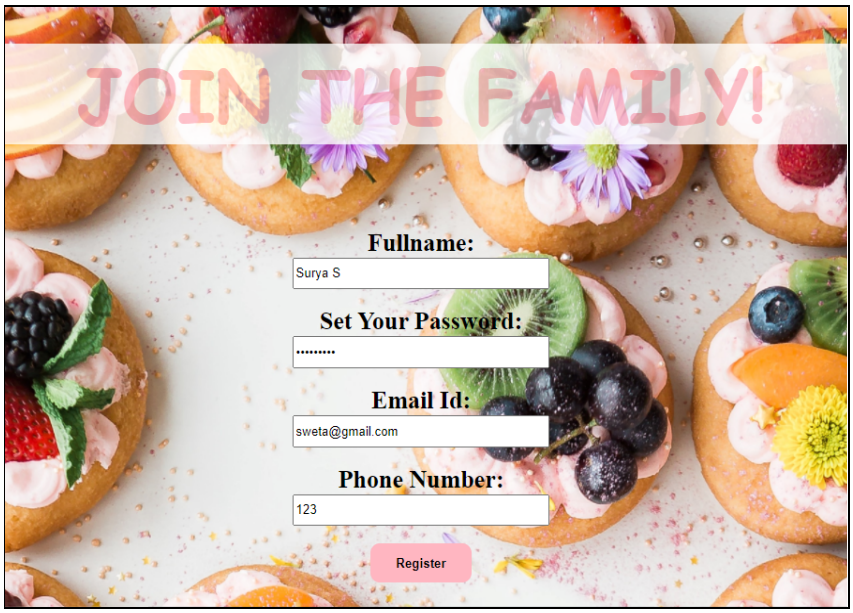


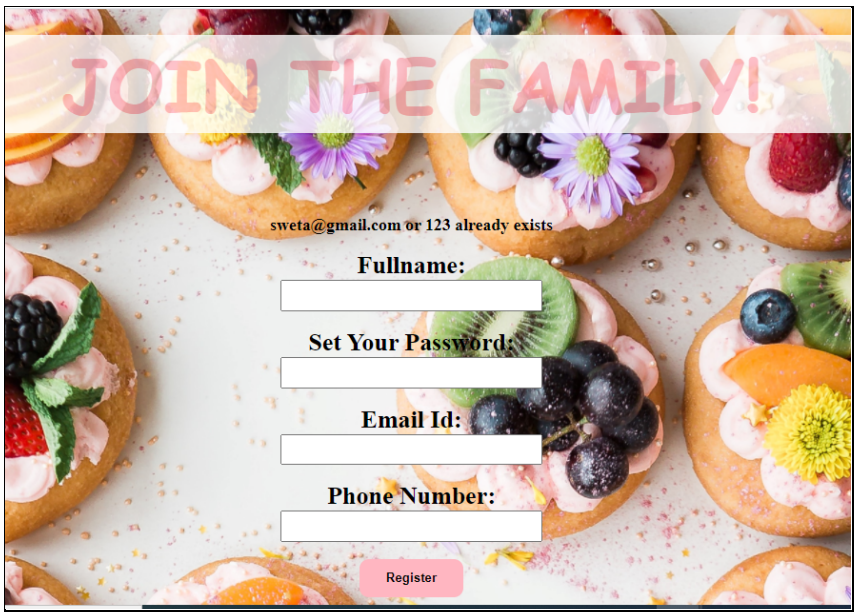


Customers table:

```
MySQL  localhost:33060+ ssl  bakery  SQL > select * from customers;
+---------+-----------------+----------------------+--------------+------------+
| cust_id | name            | email                | phone_no     | password   |
+---------+-----------------+----------------------+--------------+------------+
|       1 | Diya Dhandapani | diyadhan01@gmail.com |    123456789 | 123456789  |
|       4 | Renuka Devi     | renuka@gmail.com     |    987654321 | 123456789  |
|       5 | Sweta           | sweta@gmail.com      | 377298131336 | 1234567890 |
|       6 | Deepa Dan       | deepa@gmail.com      |  62900820717 | 123456789  |
|       7 | Vaishnavi       | vaish@gmail.com      |   9289290129 | 123456789  |
|       8 | Rahul G         | rahul@gmail.com      |  71291231823 | 123456789  |
+---------+-----------------+----------------------+--------------+------------+
```

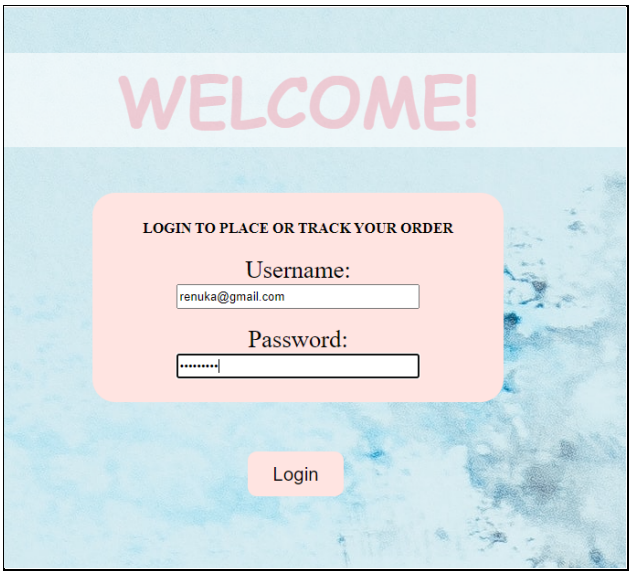
After an unsuccessful registration - User gets an alert message

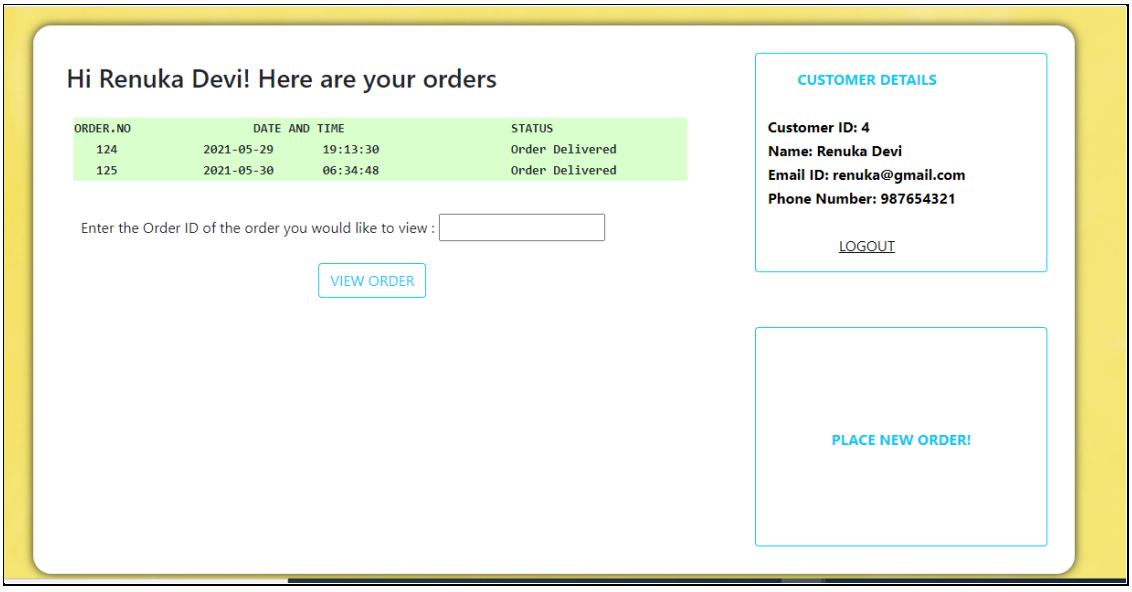




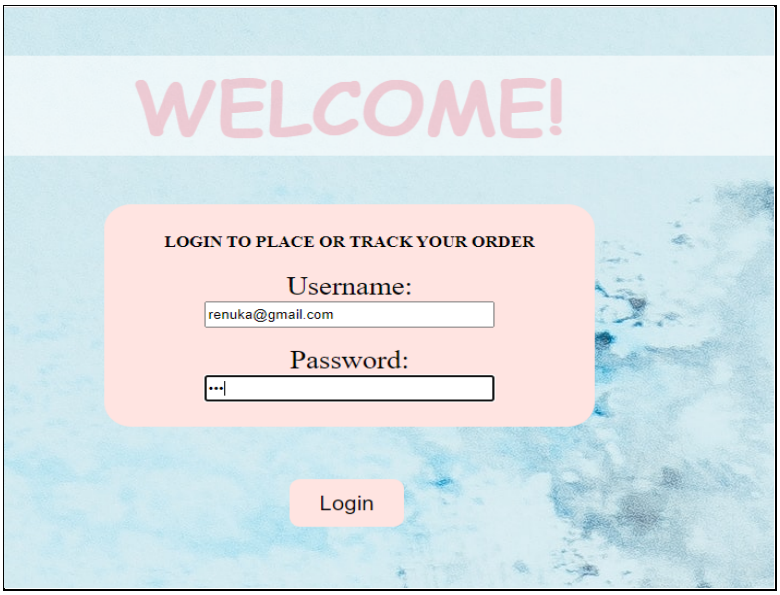
- **User Login -**

After a successful login - User gets directed to the Orders Page

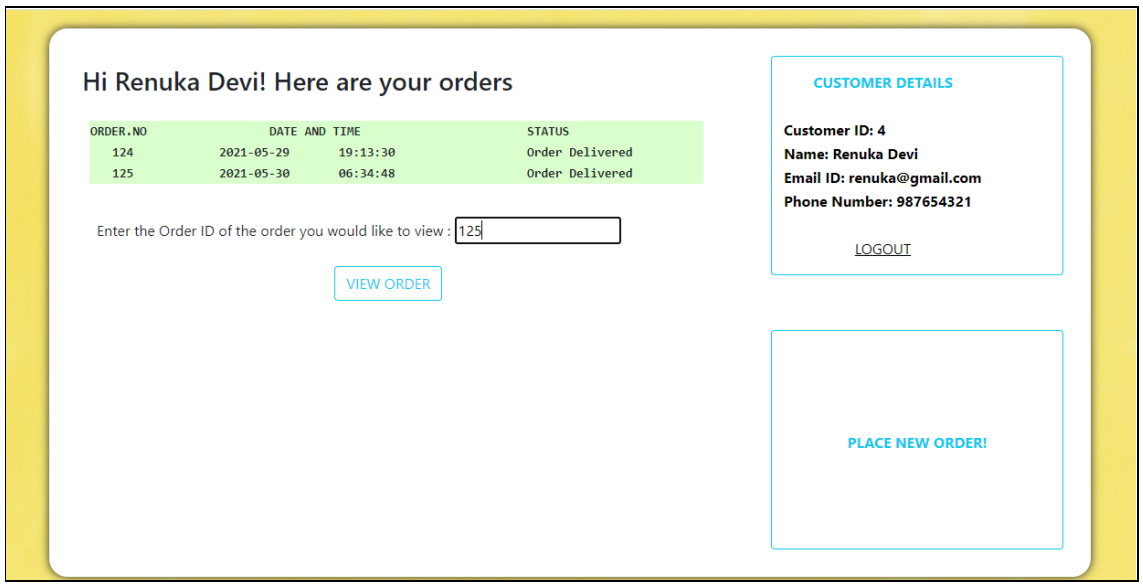




After an successful login - User gets an alert message



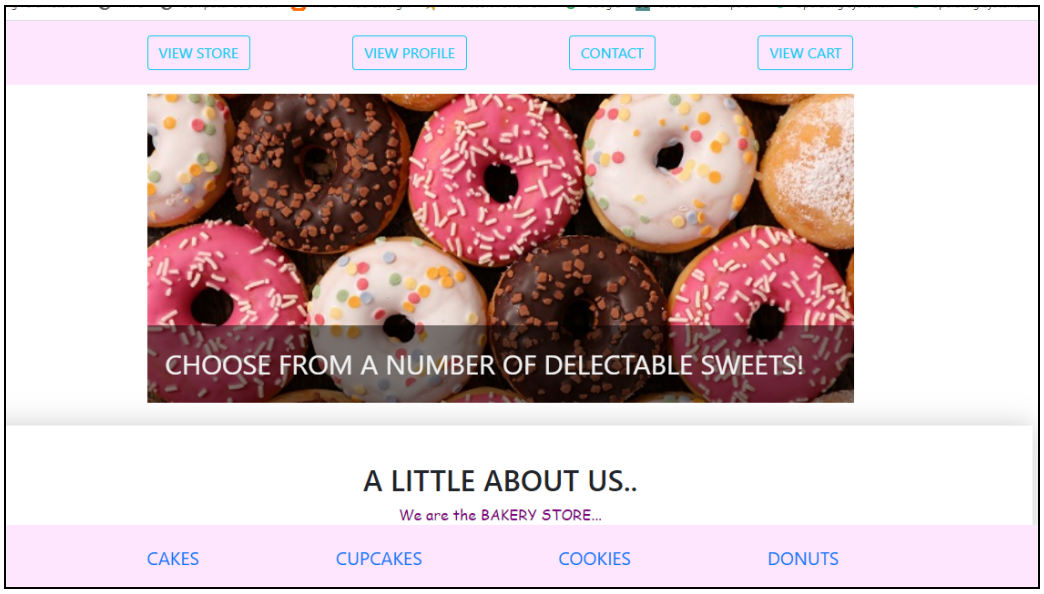
View Order Details:



View [BILL] created to show the order details:

```
MySQL  localhost:33060+ ssl  bakery  SQL > select * from bill;
+--------+--------------+---------------+-----------+----------+
| type   | flavor       | price_per_qty | total_amt | quantity |
+--------+--------------+---------------+-----------+----------+
| Cake   | Chocolate    |        89.456 |  357.8240 |        4 |
| Cookie | Peanut Butter |       190.780 |  381.5600 |        5 |
+--------+--------------+---------------+-----------+----------+
2 rows in set (0.0025 sec)
```

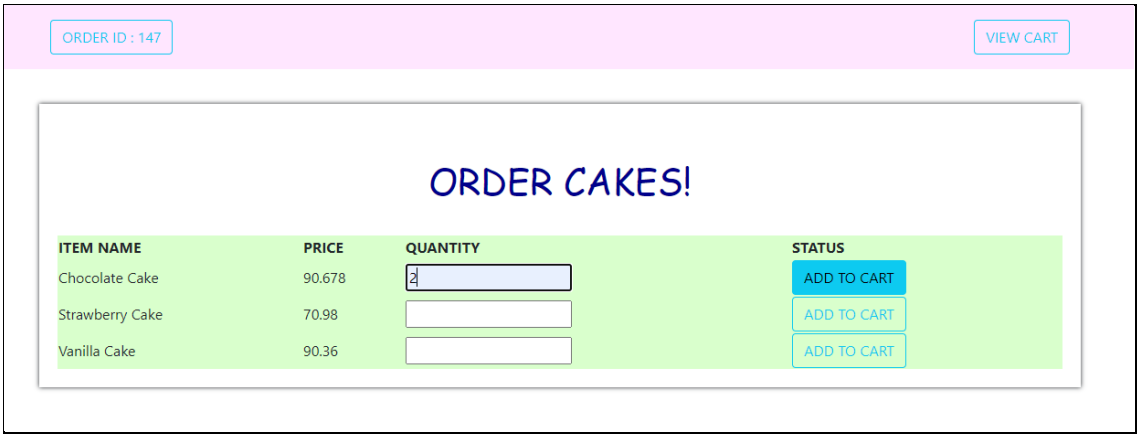
Placing a new order:



New order is added to order list:

```
MySQL  localhost:33060+ ssl  bakery  SQL > SELECT * FROM ORDER_LIST;
+------+-----+------+-----------+---------------------+--------------------------+
| o_id | cid | eid  | net_total | DOO                 | order_status             |
+------+-----+------+-----------+---------------------+--------------------------+
|   70 |   1 | 1000 |  178.9120 | 2021-05-28 20:36:22 | Order Delivered          |
|   73 |   7 | 1000 |  504.4500 | 2021-05-28 20:41:15 | Order Delivered          |
|   74 |   7 | 1000 |  504.4500 | 2021-05-28 20:52:49 | Order Delivered          |
|   76 |   1 | 1000 |  601.8000 | 2021-05-28 22:07:39 | Order Delivered          |
|   79 |   1 | 1000 |  163.5600 | 2021-05-29 09:06:21 | Order Delivered          |
|   81 |   6 | 1000 |  451.3500 | 2021-05-29 09:52:28 | Waiting for Confirmation |
|  100 |   7 | 1002 |  484.5000 | 2021-05-29 14:46:00 | Ordered Confirmed        |
|  122 |   1 | 1002 |  584.4600 | 2021-05-29 18:51:37 | Order Delivered          |
|  124 |   4 | 1002 |  645.7200 | 2021-05-29 19:13:30 | Order Delivered          |
|  125 |   4 | 1002 |  739.3840 | 2021-05-30 06:34:48 | Order Delivered          |
|  134 |   1 | 1002 | 1447.6000 | 2021-05-30 19:40:59 | Order Confirmed          |
|  136 |   1 | 1002 |  798.0600 | 2021-05-30 19:47:05 | Waiting for Confirmation |
|  137 |   1 | 1002 | 1561.1800 | 2021-05-30 19:54:58 | Waiting for Confirmation |
|  147 |   4 | 1002 |    0.0000 | 2021-05-30 23:05:23 | Being Placed             |
+------+-----+------+-----------+---------------------+--------------------------+
14 rows in set (0.0016 sec)
```

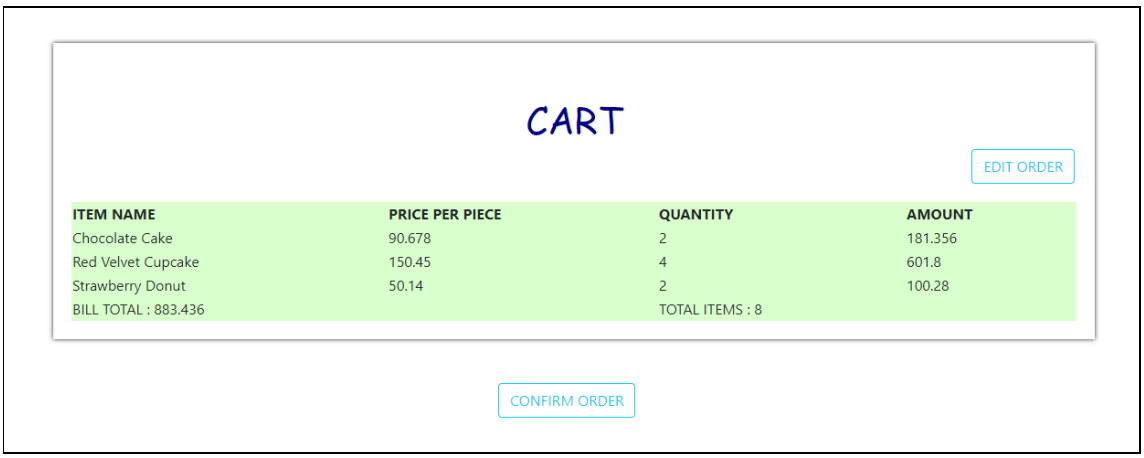
Adding products:



Order Desp table after adding to cart:

```
MySQL  localhost:33060+ ssl  bakery  SQL > SELECT * FROM ORDER_DESP;
+-----+------+---------------+-----------+----------+
| oid | pid  | price_per_qty | total_amt | quantity |
+-----+------+---------------+-----------+----------+
|  70 | 3000 |        89.456 |  178.9120 |        2 |
|  73 | 3005 |       100.890 |  504.4500 |        5 |
|  76 | 3006 |       150.450 |  601.8000 |        4 |
|  79 | 3012 |        40.890 |  163.5600 |        4 |
|  81 | 3006 |       150.450 |  451.3500 |        3 |
| 100 | 3006 |       150.450 |  300.9000 |        1 |
| 100 | 3011 |        45.900 |  183.6000 |        3 |
| 122 | 3005 |       100.890 |  403.5600 |        4 |
| 122 | 3007 |        90.450 |  180.9000 |        3 |
| 124 | 3003 |        70.980 |  283.9200 |        5 |
| 124 | 3008 |        90.450 |  361.8000 |        4 |
| 125 | 3000 |        89.456 |  357.8240 |        4 |
| 125 | 3009 |       190.780 |  381.5600 |        5 |
| 134 | 3009 |       190.780 |  763.1200 |        4 |
| 134 | 3011 |        50.140 |  200.5600 |        4 |
| 134 | 3010 |       120.980 |  483.9200 |        4 |
| 136 | 3005 |       100.890 |  403.5600 |        4 |
| 136 | 3016 |        78.900 |  394.5000 |        5 |
| 137 | 3009 |       190.780 |  763.1200 |        4 |
| 137 | 3005 |       100.890 |  403.5600 |        4 |
| 137 | 3016 |        78.900 |  394.5000 |        5 |
| 147 | 3000 |        90.678 |  181.3560 |        2 |
+-----+------+---------------+-----------+----------+
```

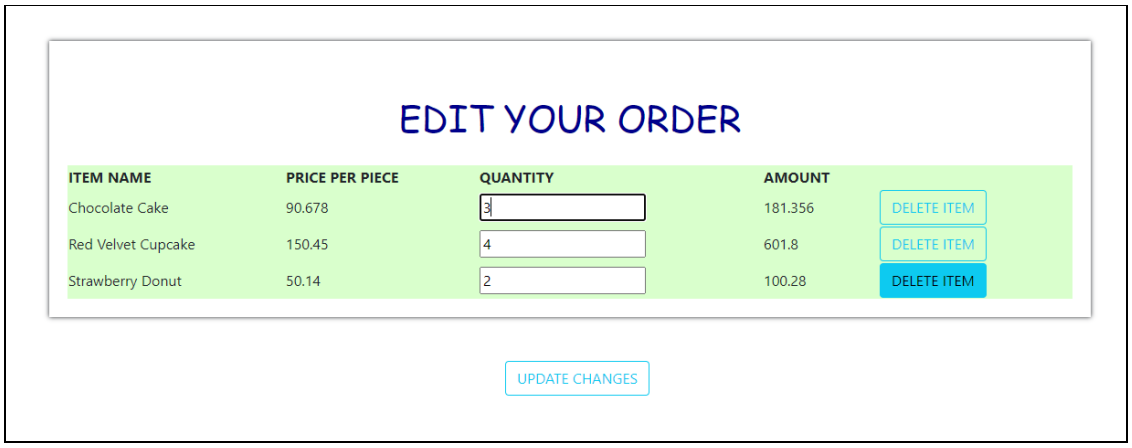
View the cart :



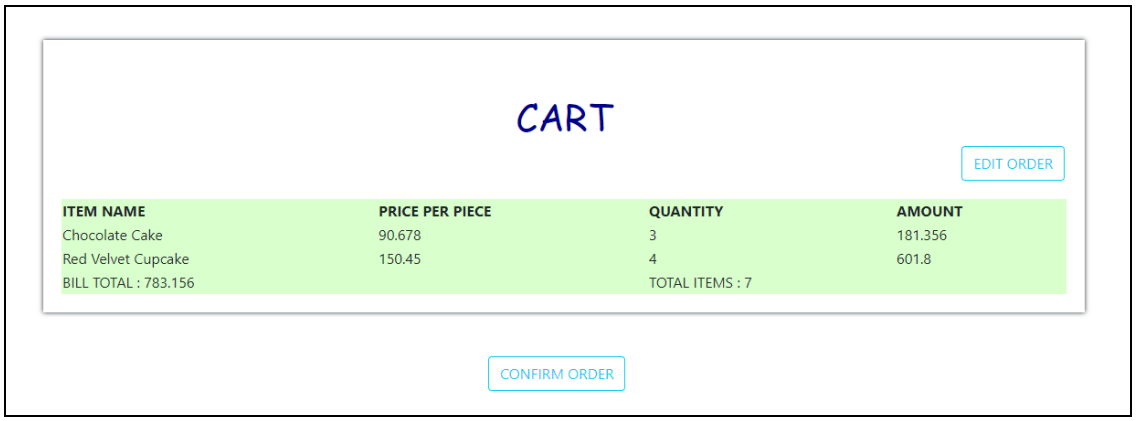
VIEW [BILL] CREATED TO DISPLAY THE CART:

```
MySQL  localhost:33060+ ssl  bakery  SQL > select * from bill;
+------+---------+------------+---------------+-----------+----------+
| p_id | type    | flavor     | price_per_qty | total_amt | quantity |
+------+---------+------------+---------------+-----------+----------+
| 3000 | Cake    | Chocolate  |        90.678 |  181.3560 |        2 |
| 3006 | Cupcake | Red Velvet |       150.450 |  601.8000 |        4 |
| 3011 | Donut   | Strawberry |        50.140 |  100.2800 |        2 |
+------+---------+------------+---------------+-----------+----------+
3 rows in set (0.0018 sec)
```

Editing the cart:



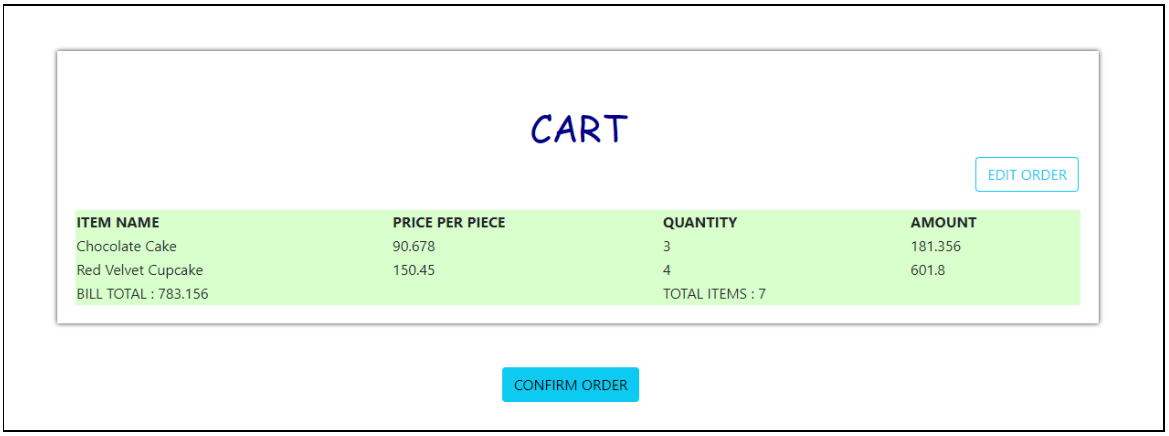
Cart after editing:



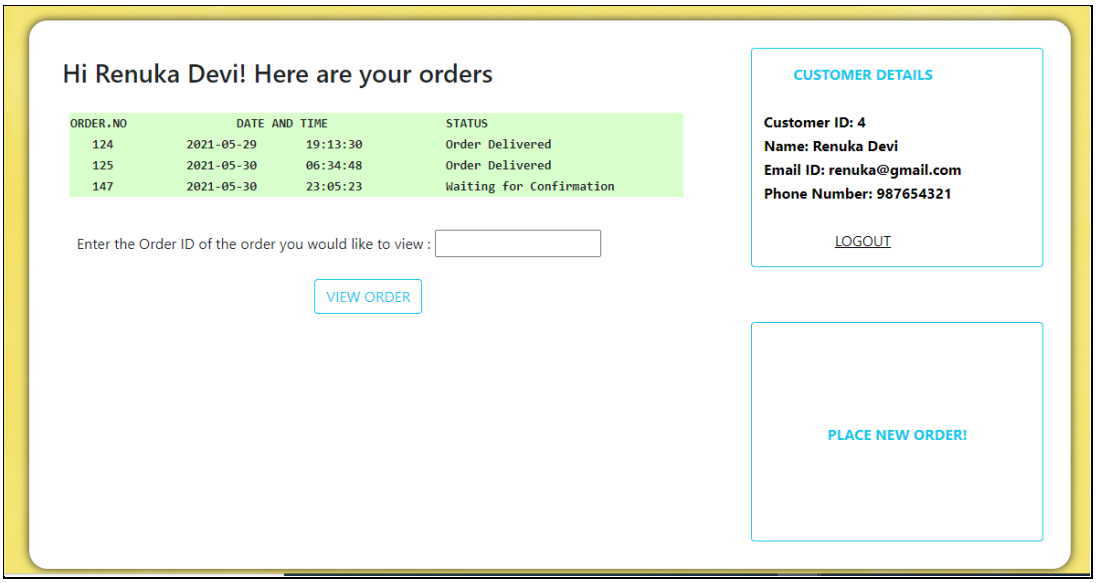
Order Desp table after editing:

```
| 137 | 3005 |       100.890 |  403.5600 |        4
| 137 | 3016 |        78.900 |  394.5000 |        5
| 147 | 3000 |        90.678 |  181.3560 |        3
| 147 | 3006 |       150.450 |  601.8000 |        4
+-----+------+---------------+-----------+----------
23 rows in set (0.0015 sec)
```

After Confirming the order:



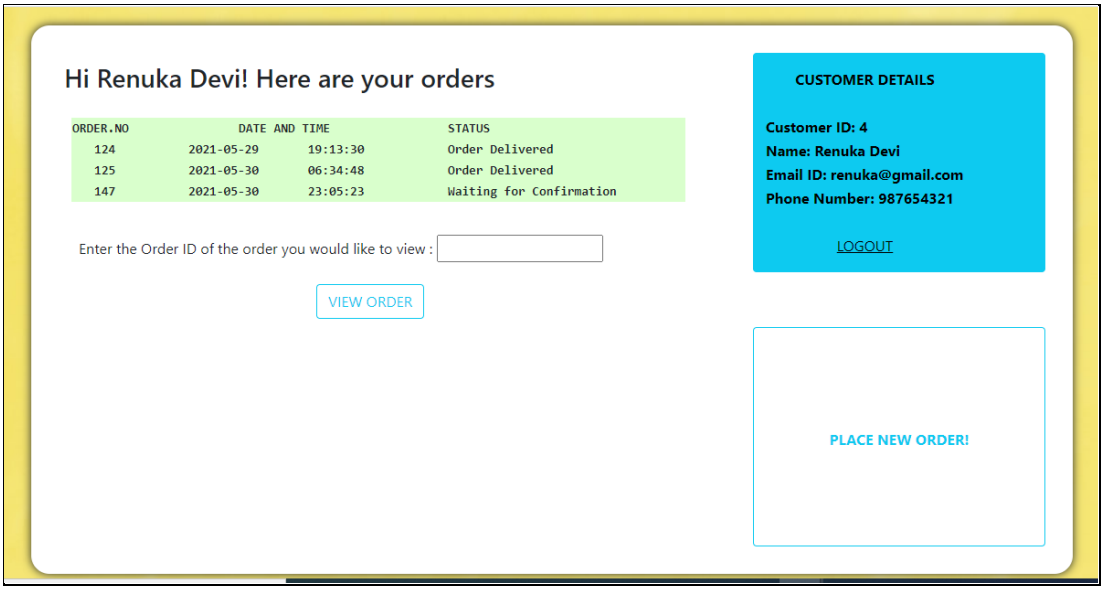
Order added to customer's order list:

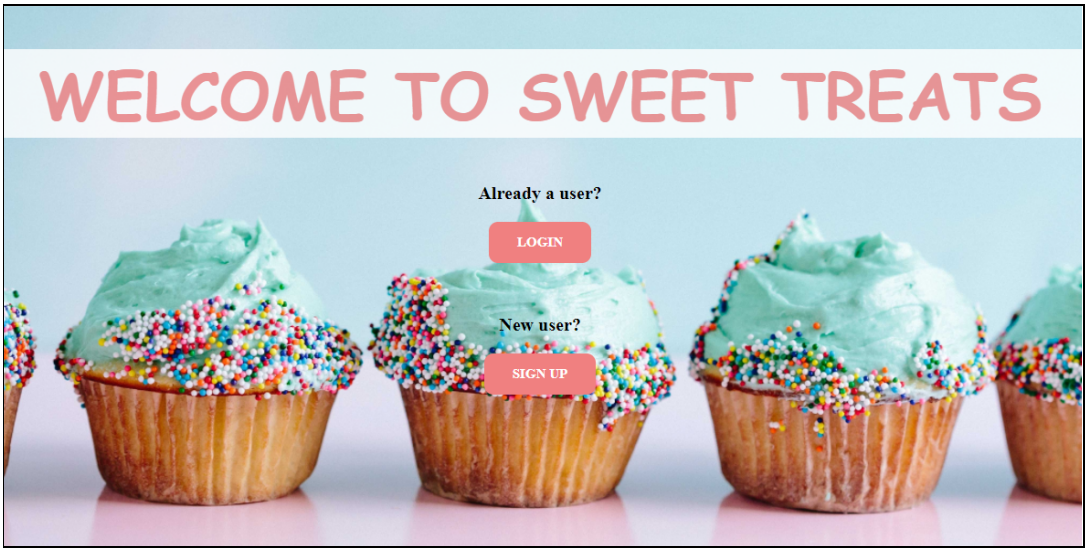


Order List table:

```
|  136 |   1 | 1002 |  798.0600 | 2021-05-30 19:47:05 | Waiting for Confirmation |
|  137 |   1 | 1002 | 1561.1800 | 2021-05-30 19:54:58 | Waiting for Confirmation |
|  147 |   4 | 1002 |  783.1560 | 2021-05-30 23:05:23 | Waiting for Confirmation |
+------+-----+------+-----------+---------------------+--------------------------+
14 rows in set (0.0013 sec)
```

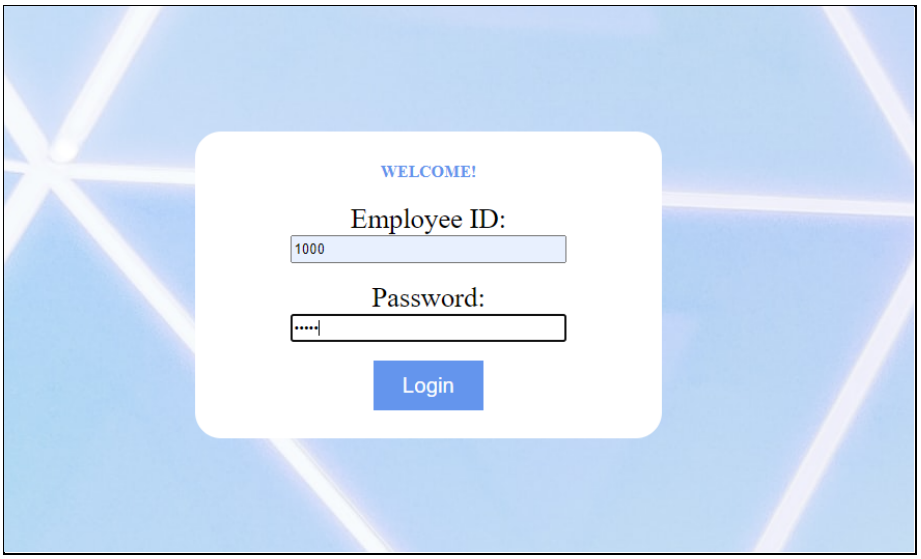
Customer logging out - User is directed to the landing page:

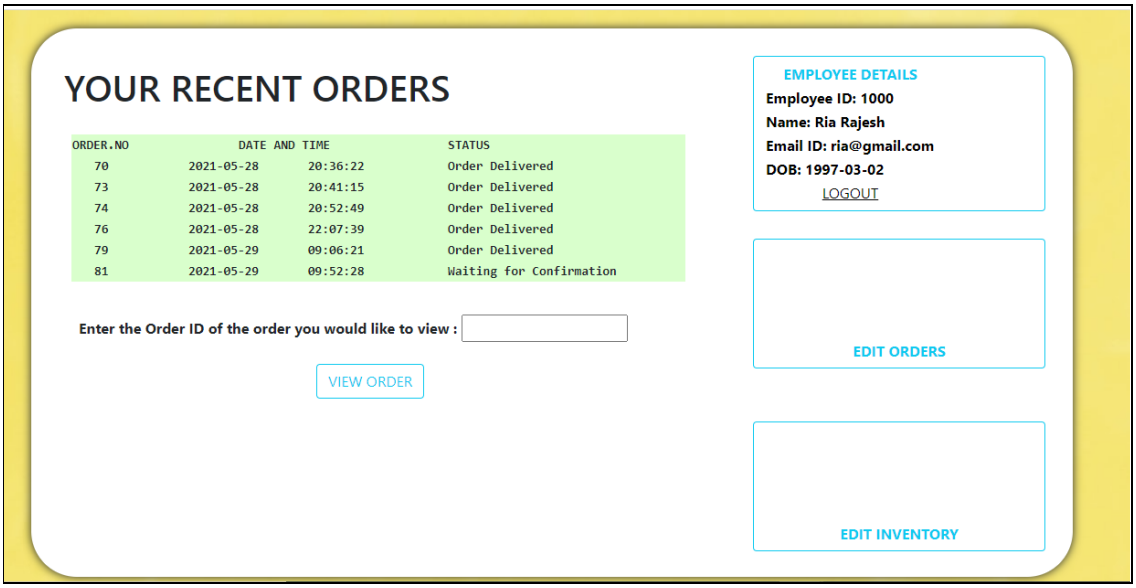




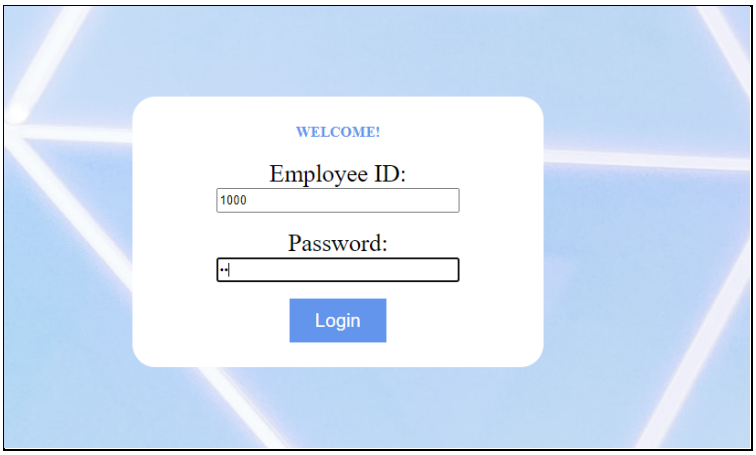
Employee Interface:

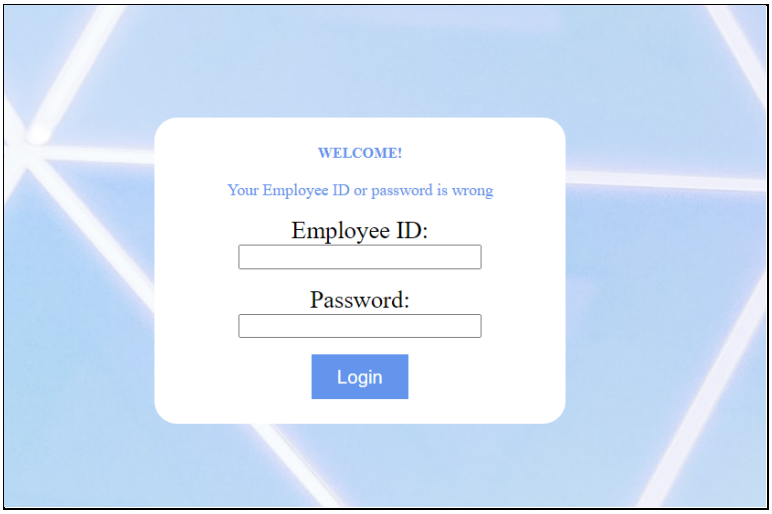
After a successful login - Employee gets directed to their profile



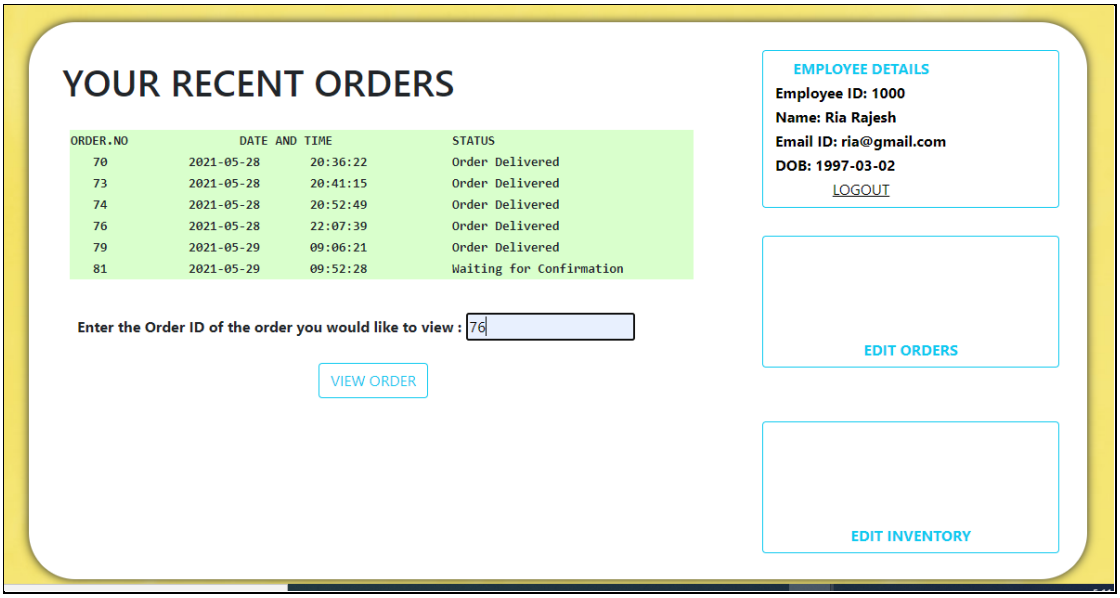


After an unsuccessful login - Employee gets an alert message





View Order Details:



View created as a result:

```
MySQL  localhost:33060+ ssl  bakery  SQL > select * from bill;
+---------+------------+---------------+-----------+----------+
| type    | flavor     | price_per_qty | total_amt | quantity |
+---------+------------+---------------+-----------+----------+
| Cupcake | Red Velvet |       150.450 |  601.8000 |        4 |
+---------+------------+---------------+-----------+----------+
1 row in set (0.0018 sec)
```

Editing Inventory:

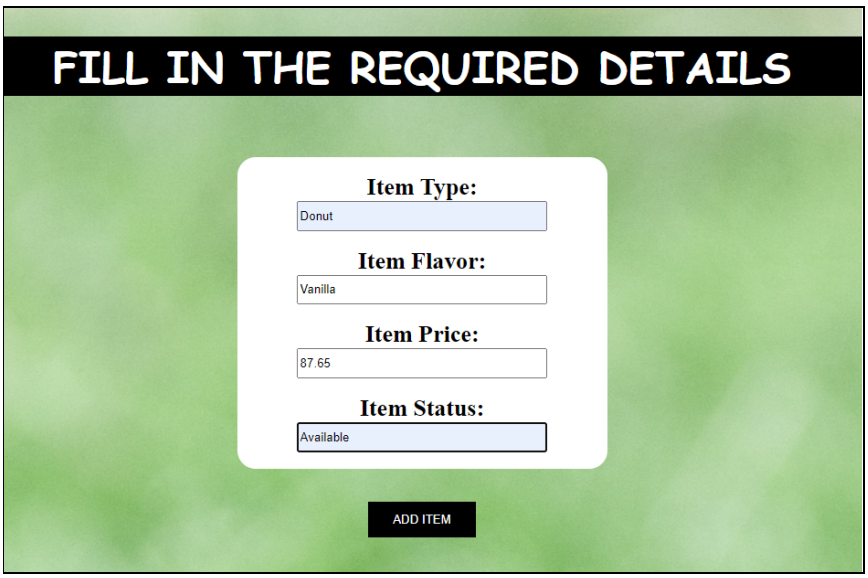
Before editing:

```
MySQL  localhost:33060+ ssl  bakery  SQL > SELECT *
+------+-------+------------+--------+-----------+
| p_id | type  | flavor     | price  | status    |
+------+-------+------------+--------+-----------+
| 3011 | Donut | Strawberry | 50.140 | Available |
| 3012 | Donut | Glazed     | 41.120 | Available |
| 3013 | Donut | Powder     | 30.900 | Available |
| 3016 | Donut | Chocolate  | 78.900 | Available |
+------+-------+------------+--------+-----------+
4 rows in set (0.0011 sec)
```

After editing:

```
+------+-------+------------+--------+-----------+
| p_id | type  | flavor     | price  | status    |
+------+-------+------------+--------+-----------+
| 3011 | Donut | Strawberry | 50.140 | Available |
| 3012 | Donut | Glazed     | 42.670 | Available |
| 3016 | Donut | Chocolate  | 78.900 | Available |
+------+-------+------------+--------+-----------+
3 rows in set (0.0014 sec)
```

Adding new products :



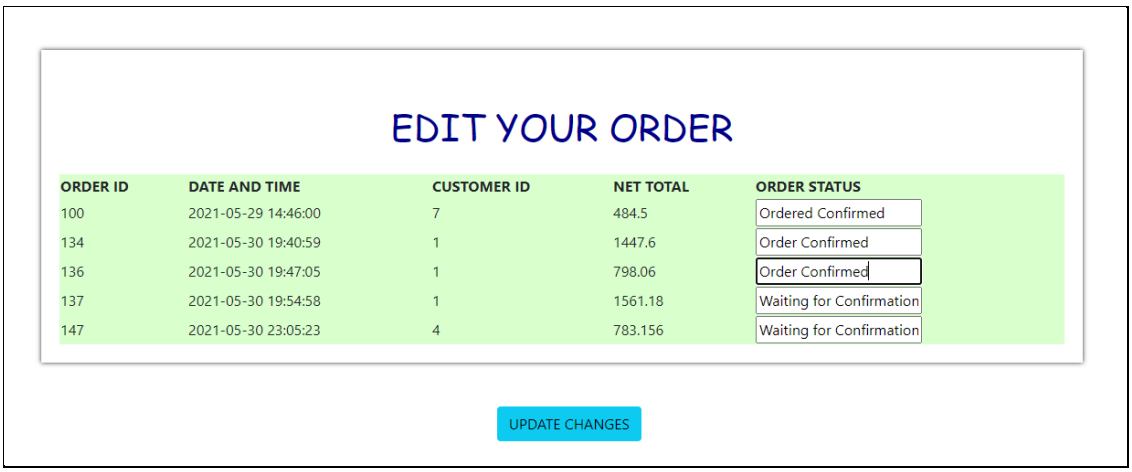
In the products table:

```
MySQL  localhost:33060+ ssl  bakery  SQL > select * fro
+------+-------+-----------------+--------+-----------+
| p_id | type  | flavor          | price  | status    |
+------+-------+-----------------+--------+-----------+
| 3011 | Donut | Strawberry      | 50.140 | Available |
| 3012 | Donut | Glazed          | 42.670 | Available |
| 3016 | Donut | Chocolate       | 78.900 | Available |
| 3017 | Donut | Vanilla Custard | 87.650 | Available |
+------+-------+-----------------+--------+-----------+
4 rows in set (0.0013 sec)
MySQL  localhost:33060+ ssl  bakery  SQL >
```

Editing Orders:

Before editing:

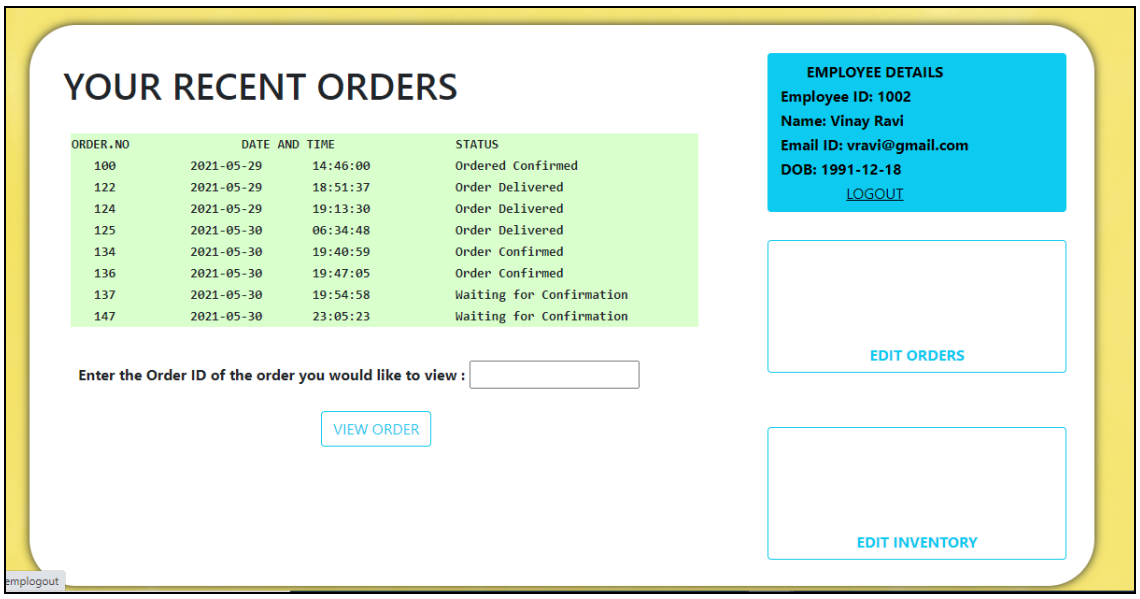
```
MySQL  localhost:33060+ ssl  bakery  SQL > select * from order_list where o_id = 136;
+------+-----+------+-----------+---------------------+--------------------------+
| o_id | cid | eid  | net_total | DOO                 | order_status             |
+------+-----+------+-----------+---------------------+--------------------------+
|  136 |   1 | 1002 |  798.0600 | 2021-05-30 19:47:05 | Waiting for Confirmation |
+------+-----+------+-----------+---------------------+--------------------------+
1 row in set (0.0011 sec)
```

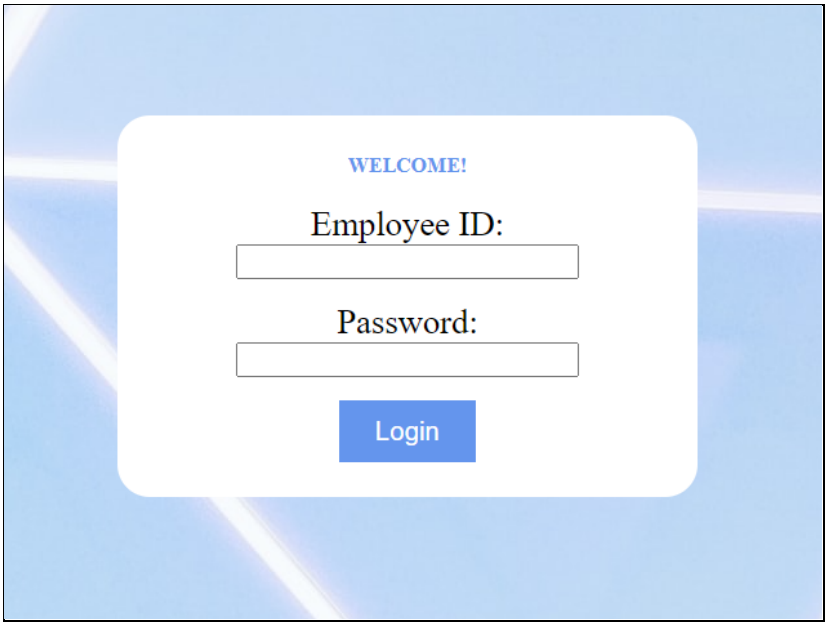


After editing, the order list table:

```
MySQL  localhost:33060+ ssl  bakery  SQL > select * from order_list where o_id = 136;
+------+-----+------+-----------+---------------------+-----------------+
| o_id | cid | eid  | net_total | DOO                 | order_status    |
+------+-----+------+-----------+---------------------+-----------------+
|  136 |   1 | 1002 |  798.0600 | 2021-05-30 19:47:05 | Order Confirmed |
+------+-----+------+-----------+---------------------+-----------------+
1 row in set (0.0015 sec)
MySQL  localhost:33060+ ssl  bakery  SQL >
```

Employee logging out - Employee is directed to the login page:





Conclusion

With the basics we have learnt in the theory classes and labs and given the opportunity , we were able to develop a bakery database which has a friendly user interface, where

- the users can log in/sign up, view all the products and add the required quantities of respective products to the cart, edit and remove the items in the cart as necessary
- The employees view the orders of users/customers and can update the status of their order. The employee can also update (add, remove, modify) the product list as necessary.

We also learnt to navigate the MySQL shell, VsCode and Sublime Text Editor, Node JS : implementing modules (express, ejs etc), routes , templates and also gained ample knowledge about the BootStrap Framework.

Working on this project has given us better exposure to the working of databases, communication between the client and server and building interactive and efficient user interfaces.

We hope to further enhance our knowledge in the area of web development and work on more projects involving the above mentioned technologies and frameworks.

References

- https://www.w3schools.com/html/default.asp
- https://www.w3schools.com/css/default.asp
- https://www.w3schools.com/nodejs/default.asp
- https://getbootstrap.com/
- https://www.youtube.com/watch?v=TlB_eWDSMt4&t=2s
- https://www.youtube.com/watch?v=EN6Dx22cPRI&t=77s
- https://codingstatus.com/create-registration-and-login-form-in-node-js-mysql/
- https://www.geeksforgeeks.org/express-js-express-static-function/
- https://www.geeksforgeeks.org/how-to-serve-static-content-using-node-js/
- https://www.geeksforgeeks.org/use-ejs-as-template-engine-in-node-js/
- https://stackoverflow.com/questions/11187961/date-format-in-node-js
- https://www.w3schools.com/nodejs/nodejs_mysql.asp
- http://expressjs.com/en/api.html#express
- https://dev.to/achowba/build-a-simple-app-using-node-js-and-mysql-19me