Project Final Report (Due 8/4/2020)

Summer 2020 | CS 157A

GNT-Market

Grocery Nutrient Tracker

Main theme of this web app is to help users to get a convenient and healthy selection for the grocery list. Users can generate their grocery list by selecting a dish, which internally converts a dish to ingredient food items and/or by adding food items directly. User customized dietary restrictions and preferences are taken into consideration when generating a grocery list.

Github:
https://github.com/ipark-CS/CS157A_Team11

Team 11
Tracy Ho & Inhee Park

Project Requirements

- Update Functional Requirements (based on the DB manipulating activities)
- Correct and copy from your Project Proposal & Requirements report.
- A list of detailed descriptions of users and how users interact with your application o Describe each individual function/feature, functional process and I/O.

About the GNT WebApp

Main theme of this web app is to help users to get a convenient and healthy selection for the grocery list. Users can generate their grocery list by selecting a dish, which internally converts a dish to ingredient food items and/or by adding food items directly. User customized dietary restrictions and preferences are taken into consideration when generating a grocery list. This app also provides quantitative nutritional distribution for their grocery list as well as qualitative beneficial nutrient information for body systems.

- **Stakeholders** : consumers
- **Application Domain** : dish, food, nutrient
- **Benefits to Users** : Quick and healthy tracker for users' grocery shopping list

System Environment

- **Structure of the system (3-tiered architecture)** :

Layer/Tier	Front-end	Middle-ware	Back-end
Role	Web Client	Web Server	DB Server
Software	Bootstrap	TomCat	MySQL
Application Language	HTML, CSS, JS	Java, JSP	SQL

- **HW/SW used** : macOS Sierra 10.12.6
- **RDBMS used** : MySQL Ver 8.0.19; MySQL Workbench 6.0.10
- **Application languages** : Java, SQL, JSP, HTML, CSS, JS

Function1. Dish ⇒ Grocery List ⇒ Nutrient Information

User inputs a desired dish from a menu in the app.

Function2. Grocery List ⇒ Nutrient Information

User inputs a list of grocery items in the app.
The app will provide a list of grocery items with nutritional distribution of each item.

Function3. Dietary Restriction

Users can set their dietary restrictions (e.g. food allergy) to the app.
Then if the user selects a grocery item that conflicts with their diet restriction, the app gives a warning.

Function4. User Creation

A user will be able to register their own account to use the web application with an email and password. Then they will be able to use the application for their own usage.

Function5. Nutrient Information

This will display a list of food items that contain a specific nutrient, giving the user an idea of what nutrients they are taking in their diet.

Function6. Create/Delete a Grocery Shopping List Card

Users can create a card that will contain all of the grocery items that they want to purchase. Items can be added to this card, deleted, or updated accordingly.

Project Design

- Corrected/UpdatedERD
- Convert your ERD into schemas. (Updated from your Design Report)
- Create and show at least 10 tables according to schemas and model the data stored in the database (Each table must contain at least 15 tuple instances.)
- Screenshots of each table from MySQL Workbench and its content (data stored) in MySQL.

Entity1. User:

The User entity represents the User of GNT-Market. A user would have a user_id, email, and password.

Entity2. Dish:

The Dish entity represents the dish that is made up of several different foods. It contains a food_id and name that says that it is, and a short description of it.

Entity3. Food:

The Food entity represents a single food item that may be used in a relation to other entities.

Entity4. GroceryList:

GroceryList is an entity to represent the food list that the user creates by selecting a dish (then converting to ingredient food list) or by adding a food list directly. This contains list_id and date (in datetime type).

Entity5. Nutrient:

Nutrient is an entity to represent individual nutrients. This contains nutrient_id and name to keep track of the different nutrients.

Relationship1. User_creates_GroceryList:

This relationship connects a grocery list to a user. Since a user creates a grocery list, a grocery list may only belong to one user.

Relationship2. User_marks_Food:

This relationship connects a user to a food item with conditions tied to the food item. The user may mark the food item as a Favorite and/or Restricted. A user may put conditions on multiple food items, and a food item may have conditions from multiple users.

Relationship3. Dish_has_Food:

This relationship represents what dish contains what food items. It connects dishes with food. A dish can have multiple food items, and a food item can be part of multiple dishes.

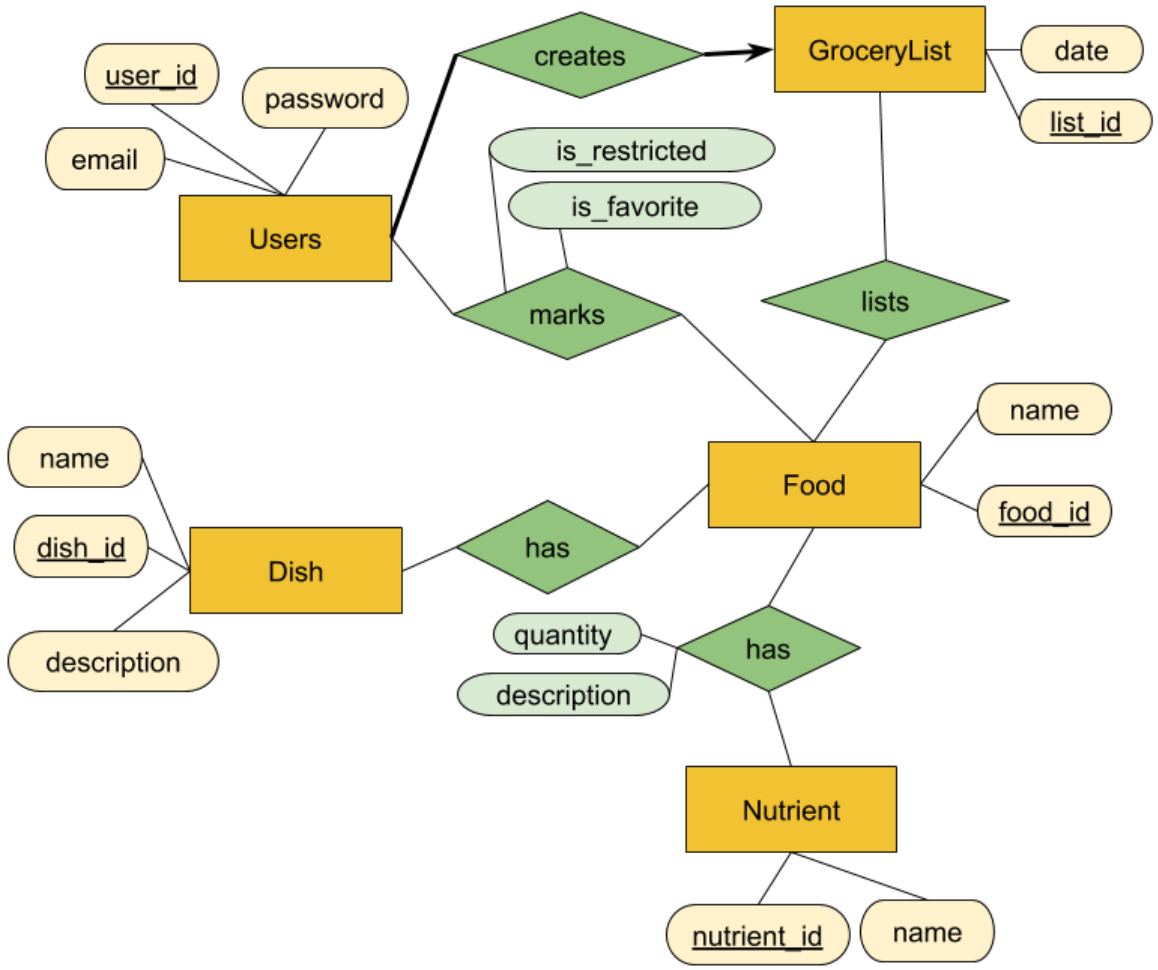
Relationship4. Food_has_Nutrient:

This relationship connects food with nutrients. A food item can have multiple nutrients and a nutrient can be found in many different food items. This will also be where the nutrient quantity for a food item will be stored.

Relationship5. Food_lists_GroceryList:

This relationship connects food with the user's grocery list. From the selected food items to the grocery list, we may suggest users to add their favorite food or give warning on the restricted food item if it’s selected.

Entity-Relationship Diagram (ERD)



Schemas for Entity Sets

E1. Users(user_id, password, email)
E2. Dish(dish_id, name, description)
E3. Food(food_id, name)
E4. GroceryList(list_id, date)
E5. Nutrient(nutrient_id, name)

Schemas for Relationships

R1. User_creates_GroceryList(user_id, list_id)
R2. User_marks_Food(user_id, food_id, is_restricted, is_favorite)
R3. Dish_has_Food(dish_id, food_id)
R4. Food_has_Nutrient(food_id, nutrient_id, quantity, description)
R5. Food_lists_GroceryList(food_id, list_id)

MySQL Workbench Tables

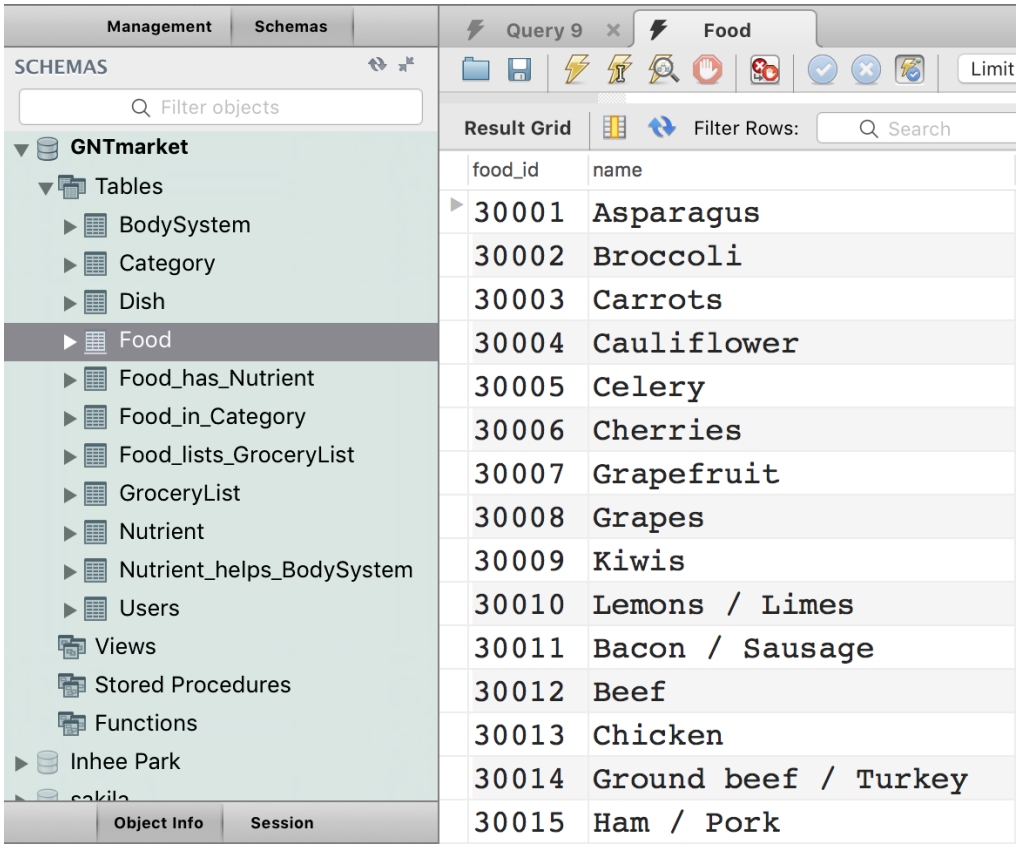
Entity1. Users(user_id, password, email)

user_id	email	password
1	testuser1@example.com	passtest123
2	testuser2@example.com	pass90
3	testuser3@example.com	sqlop90
4	testuser4@example.com	f4fopi5
5	testuser5@example.com	fiver
6	testuser6@example.com	qrghbdsed52
7	testuser7@example.com	septem
8	testuser8@example.com	passtest123
9	testuser9@example.com	sjsucs157a
10	testuser10@example.com	encrypted
11	testuser11@example.com	password
12	testuser12@example.com	defend
13	testuser13@example.com	the
14	testuser14@example.com	east
15	testuser15@example.com	wall!

Entity2. Dish(dish_id, name, description)

dish_id	name	description
1	Basic Salad	Assorted vegetables tossed together
2	Hamburger	Basic American Burger
3	Stir-fry Veggies	Assorted Grilled Vegetables
4	Mashed Califlower	Califlower mashed together
5	Vegetable Juice	Blended Vegetable Juice
6	Pasta	Basic Pasta Dish
7	Roasted Chicken	Baked Chicken
8	Ham & Eggs	Ham with Eggs
9	Carrot Juice	Blended Carrot Juice
10	Grape Juice	Blended Grape Juice
11	Vegetable Soup	Assorted veggie soup
12	Cherry Pie	Baked pie with cherries
13	Lemonade	Classic Lemon Drink
14	Steamed Asparagus	Steamed up asparagus
15	Beef & Broccoli	Cooked beef and broccoli

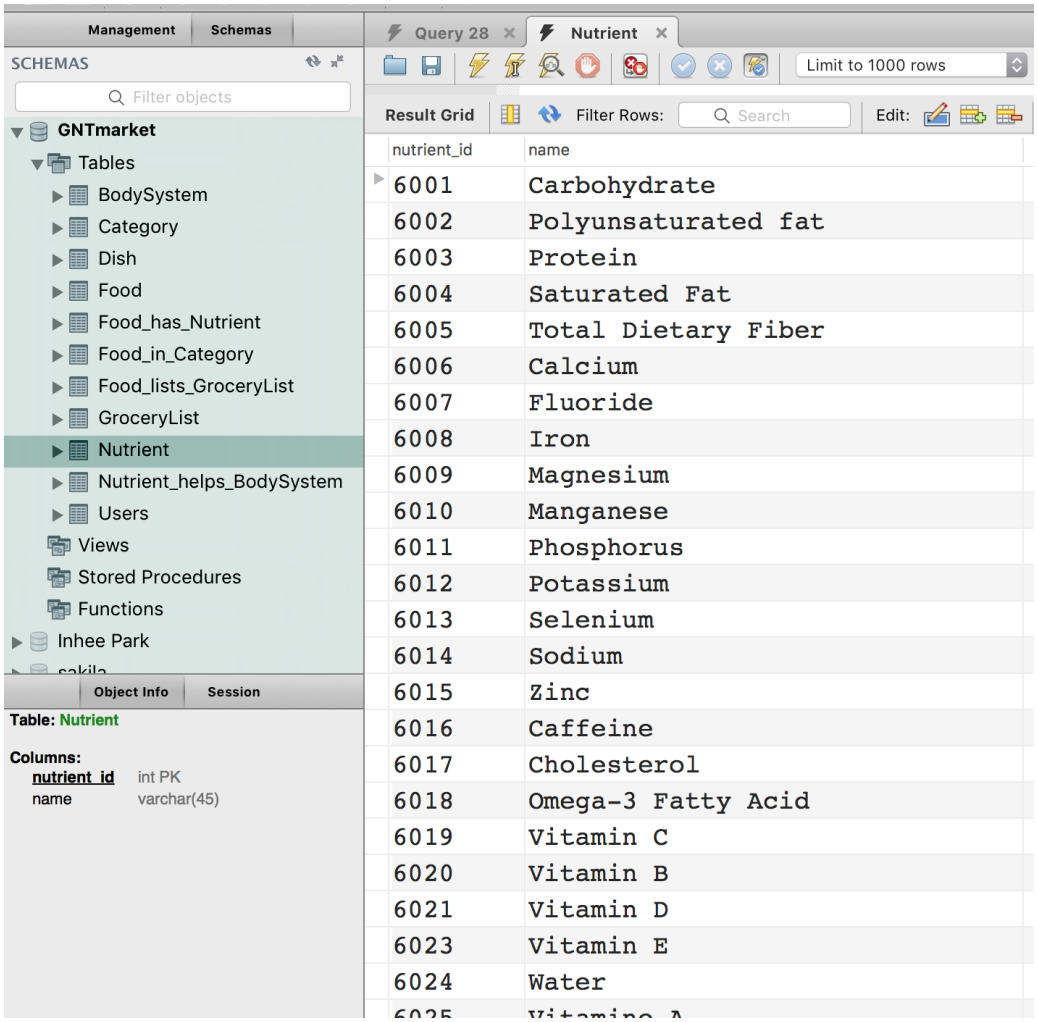
Entity3. Food(food_id, name)



Entity4. GroceryList(list_id, date)

list_id	date
1	2020-07-05 16:12:36
2	2020-07-05 16:12:41
3	2020-07-05 16:12:45
4	2020-07-05 16:12:48
5	2020-07-05 16:12:53
6	2020-07-05 16:12:56
7	2020-07-05 16:12:59
8	2020-07-05 16:13:02
9	2020-07-05 16:13:09
10	2020-07-05 16:13:11
11	2020-07-05 16:13:14
12	2020-07-05 16:13:17
13	2020-07-05 16:13:20
14	2020-07-05 16:13:25
15	2020-07-05 16:13:31

Entity5. Nutrient(nutrient_id, name)



Relationship1. User_creates_GroceryList(user_id, list_id)

user_id	list_id
1	1
2	2
3	3
4	4
5	5
6	6
7	15
8	14
9	13
10	12
11	11
12	10
13	9
14	8
15	7

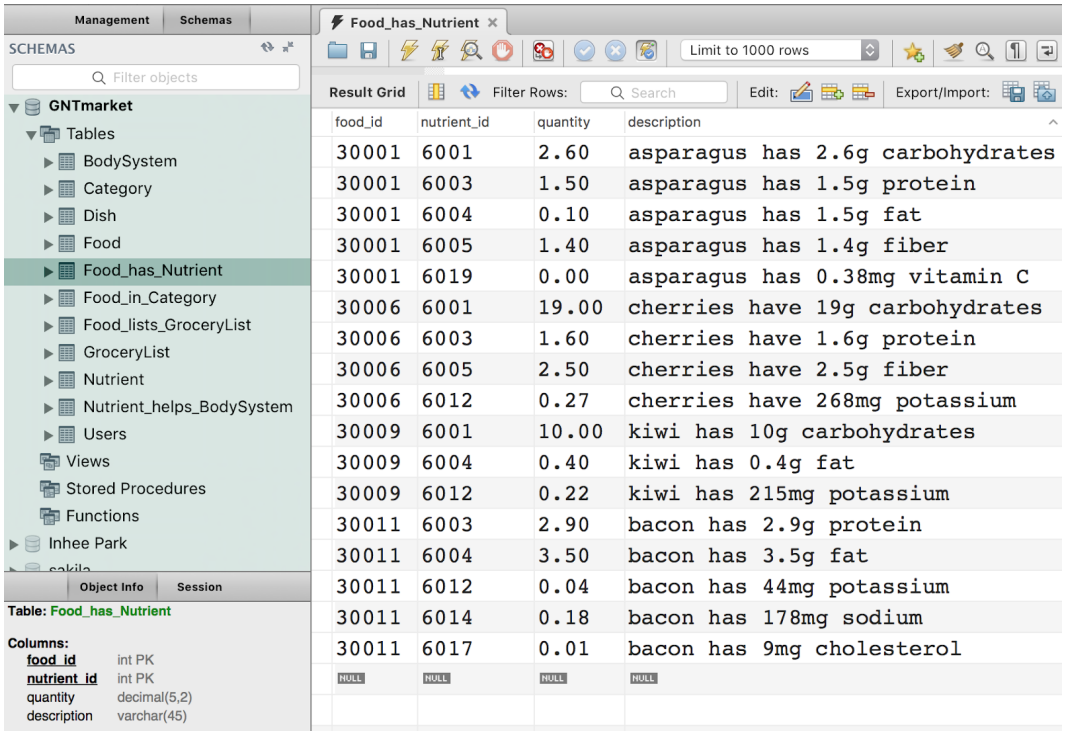
Relationship2. User_marks_Food(user_id, food_id, is_restricted, is_favorite)

user_id	food_id	is_restricted	is_favorite
1	1	0	1
2	1	0	0
3	3	1	0
1	11	1	0
4	5	0	1
5	5	0	1
7	6	0	1
9	4	0	1
9	5	0	1
10	9	0	1
11	12	1	0
4	8	1	0
12	12	0	1
13	11	1	0
15	15	0	1

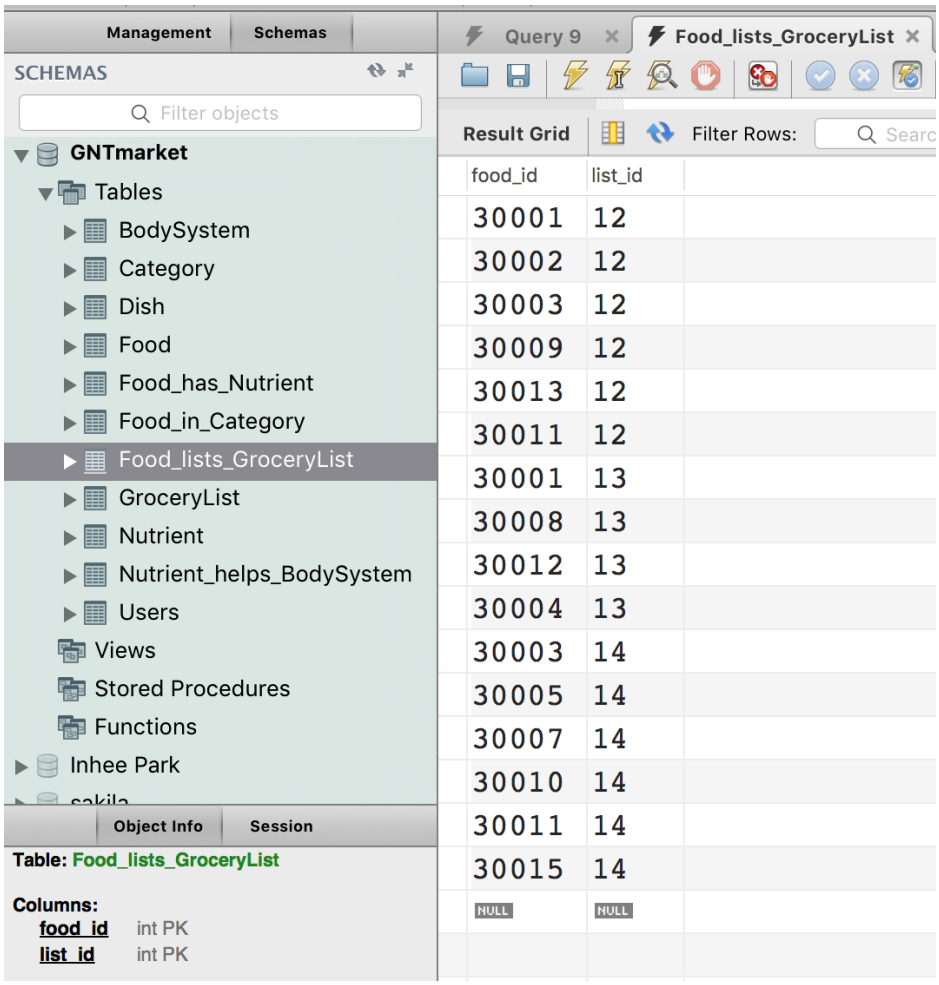
Relationship3. Dish_has_Food(dish_id, food_id)

dish_id	food_id
1	2
1	3
1	4
3	1
3	2
3	3
2	14
4	4
14	1
6	11
6	14
12	6
9	3
15	2
15	12

Relationship4. Food_has_Nutrient(food_id, nutrient_id, quantity, description)



Relationship5. Food_lists_GroceryList(food_id, list_id)



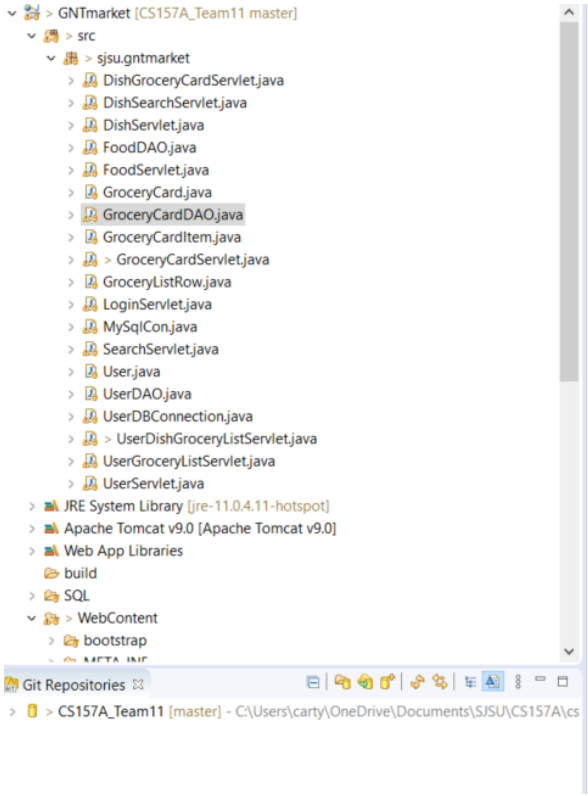
Project Implementation

- Explain and keep track of implementations based on your functional requirements and design.
- Take enough screenshots to demonstrate functions/features associated with query, insertion, updating, and deletion. (show snapshots of tables, GUI, execution results, and so on.)
- Explain and show procedures (step by step) of how to set up and run your system

Implementation Details

	JSP (Front-end ← → WebServer)	JavaServlet (WebServer ← → RDBMS)
Home/User	├── createAccount.jsp ├── index.jsp ├── error.jsp ├── login.jsp └── user.jsp	├── LoginServlet.java ├── User.java ├── UserDAO.java ├── UserDBConnection.java ├── UserDishGroceryListServlet.java ├── UserGroceryListServlet.java └── UserServlet.java
Food	├── food.jsp ├── grocerylist.jsp └── nutrient.jsp	├── FoodDAO.java ├── FoodServlet.java ├── GroceryCard.java ├── GroceryCardDAO.java ├── GroceryCardItem.java ├── GroceryCardServlet.java ├── GroceryListRow.java └── SearchServlet.java
Dish	├── dish.jsp	├── DishGroceryCardServlet.java ├── DishSearchServlet.java └── DishServlet.java

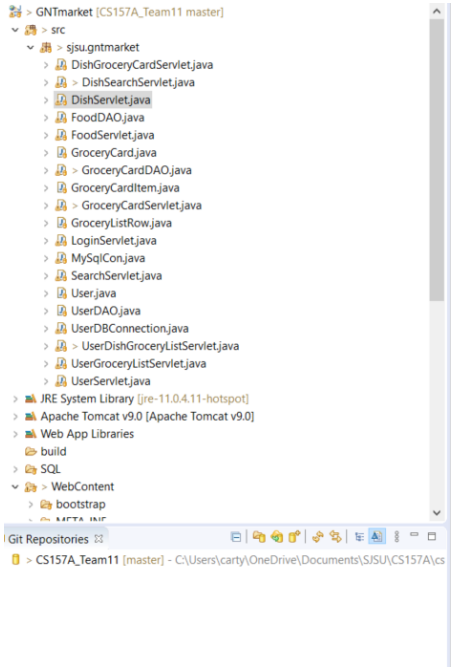
Database Access Object (using GroceryCardDAO.java)




```java
package sjsu.gntmarket;

import java.sql.Connection;

public class GroceryCardDAO {

    private String driver = "com.mysql.cj.jdbc.Driver";
    private Connection dbCon;

    private String dbURL;
    private String dbUser;
    private String dbPassword;

    public GroceryCardDAO(String dbURL, String dbUser, String dbPassword){
        this.dbURL = dbURL;
        this.dbUser = dbUser;
        this.dbPassword = dbPassword;
    }

    protected void mysqlConnect() throws SQLException{
        try{
            Class.forName(driver);
            dbCon = DriverManager.getConnection(dbURL, dbUser, dbPassword);
            System.out.println("Successfully connected to database from GroceryCardDAO");
            }
        catch(ClassNotFoundException e){
            System.err.println("Couldn't load driver.");
            System.out.println(e.getMessage( ));
            }
        catch(SQLException e){
            System.err.println("Couldn't connect to database.");
            System.out.println(e.getMessage( ));
            }
     }

    protected void mysqlDisconnect() throws SQLException{
        if (dbCon != null && !dbCon.isClosed()) {
            dbCon.close();
        }
    }
```

Throughout the program, we have created DAO (Data Access Object) java files to make calls to the MySQL database. This GroceryCardDAO is an example of one, showing that each of these DAO classes make connection and disconnection calls to the database in order to perform database operations.

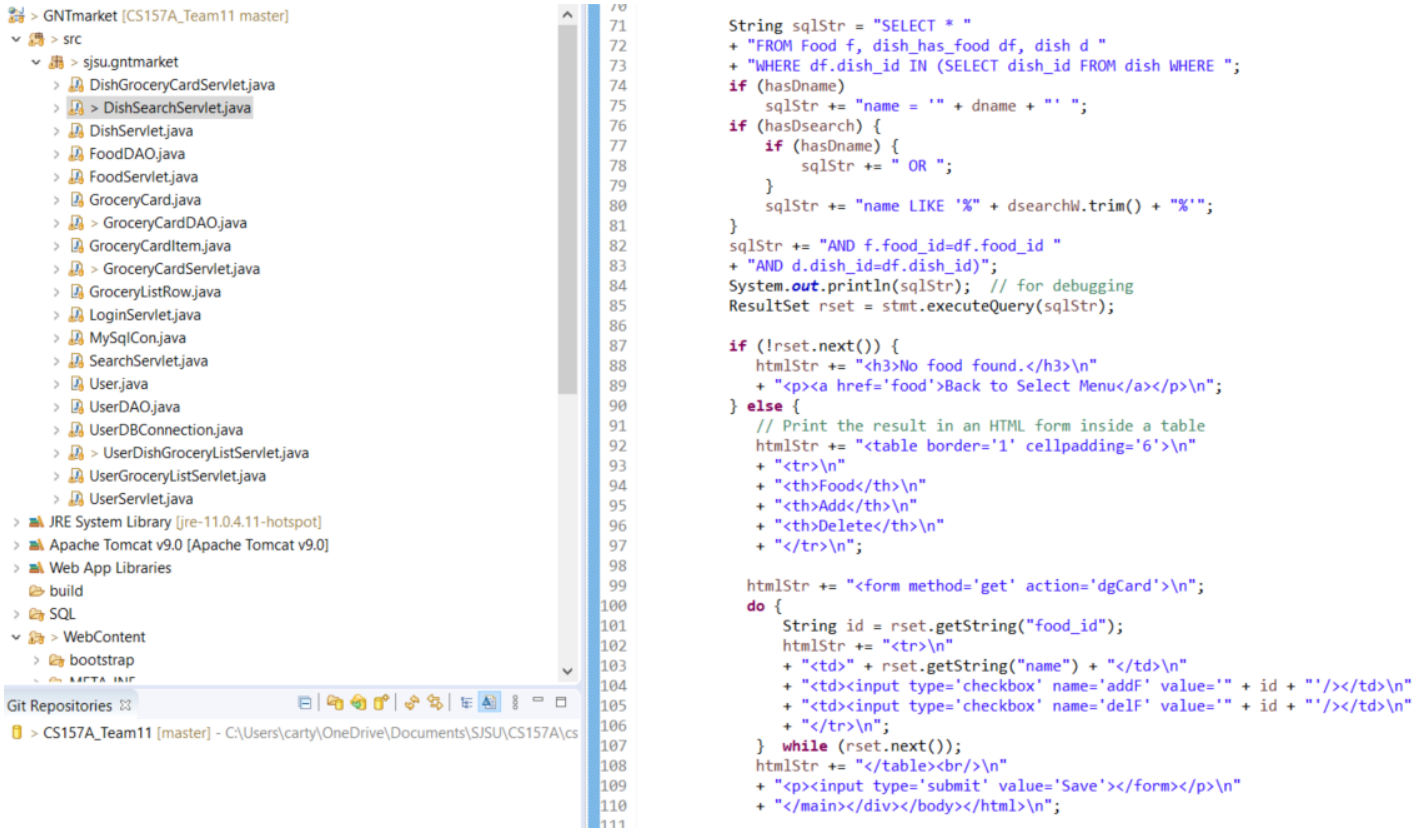
DishServlet.java




```java
// food ////////////////////////////////
String sqlStr = "SELECT name FROM dish";
ResultSet rset = stmt.executeQuery(sqlStr);

String htmlStr="<html lang='en'> <head> <meta charset='utf-8'> <title>CS157A-Team11| GNT-Market | Home</title> <!-- css --> <link

htmlStr += "<div class=\"container-fluid\">"
+ "<main role='main' class='container'>"
+ "<p><h3>Search Dish "
+ "<i class=\"fas fa-drumstick-bite\"></i> <i class=\"fas fa-hotdog\"></i> "
+ "<i class=\"fas fa-hamburger\"></i>"
+ " <i class=\"fas fa-pizza-slice\"></i> <i class=\"fas fa-stroopwafel\"></i>"
+ " <i class=\"fas fa-seedling\"></i>"
+ "</h3></p><br>"
+ "<form method='get' action='dishsearch'>\n"
+ "<select name='dname' id='dname' class='custom-select'>\n"
+ "<option value=''>Select Dish</option>\n";

while (rset.next()) {  // list all the food
   String name = rset.getString("name");
   htmlStr += "<option value='" + name + "'>" + name + "</option>";
}
htmlStr += "</select>\n"
+ "<input type='text' class='form-control'  onfocus=\"this.value=''\" name='dsearch' placeholder='Type Dish Name'/>\n"
+ "<br><button type='submit' class='btn btn-primary mb-2' value='Search'/>search"
+ "</button></form>\n";


// Show "View Shopping Cart" if the cart is not empty
HttpSession session = request.getSession(false); // check if session exists
if (session != null) {
   GroceryCard gCard;
   synchronized (session) {
      // Retrieve the shopping cart for this session, if any. Otherwise, create one.
      gCard = (GroceryCard) session.getAttribute("gCard");
      if (gCard != null && !gCard.isEmpty()) {
         htmlStr += "<P><a href='dgCard'>Grocery List</a></p>\n";
      }
   }
}
htmlStr += "</div></main></body></html>\n";
out.println(htmlStr);
```

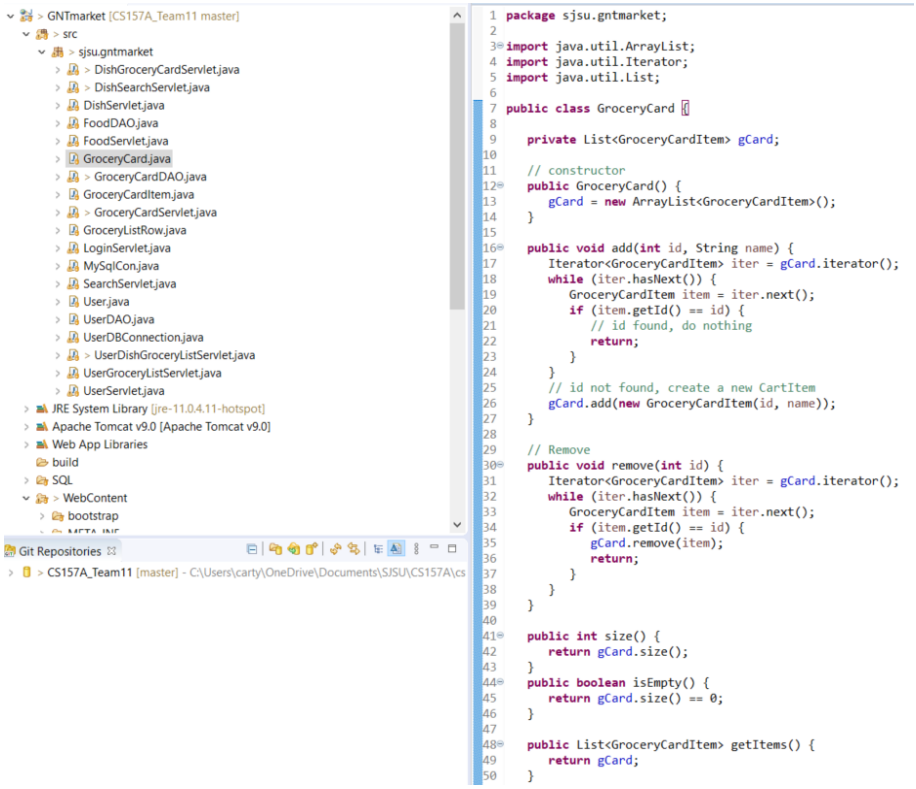
The DishServlet.java is used to present the user a list of Dishes that are currently in the database as a way to select what has already been added in the web application so they may get some ingredients used to make the dish in their grocery list. On the front end, it can be shown that dishes retrieved from the database are shown as options. With the value of what is selected being passed on.

DishSearchServlet.java



The DishSearchServlet.java is what is used to search for and retrieve a dish that has been stored in the database. When a dish is found it will display the option to add or delete the ingredients used for the dish in a user's grocery list. Which can be shown in the input with checkbox as text type. The SearchServlet.java for searching food items works very similarly.

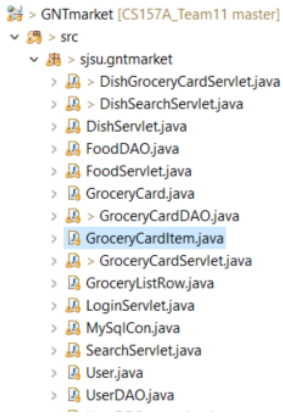
GroceryCard.java




```java
package sjsu.gntmarket;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class GroceryCard {

   private List<GroceryCardItem> gCard;

   // constructor
   public GroceryCard() {
      gCard = new ArrayList<GroceryCardItem>();
   }

   public void add(int id, String name) {
      Iterator<GroceryCardItem> iter = gCard.iterator();
      while (iter.hasNext()) {
         GroceryCardItem item = iter.next();
         if (item.getId() == id) {
            // id found, do nothing
            return;
         }
      }
      // id not found, create a new CartItem
      gCard.add(new GroceryCardItem(id, name));
   }

   // Remove
   public void remove(int id) {
      Iterator<GroceryCardItem> iter = gCard.iterator();
      while (iter.hasNext()) {
         GroceryCardItem item = iter.next();
         if (item.getId() == id) {
            gCard.remove(item);
            return;
         }
      }
   }

   public int size() {
      return gCard.size();
   }
   public boolean isEmpty() {
      return gCard.size() == 0;
   }

   public List<GroceryCardItem> getItems() {
      return gCard;
   }
}
```

This GroceryCard.java is our own developed data structure to hold information on what is currently contained in the grocery list. Here it can be seen that GroceryCardItem objects are stored in a List, which they can be added to or deleted from.

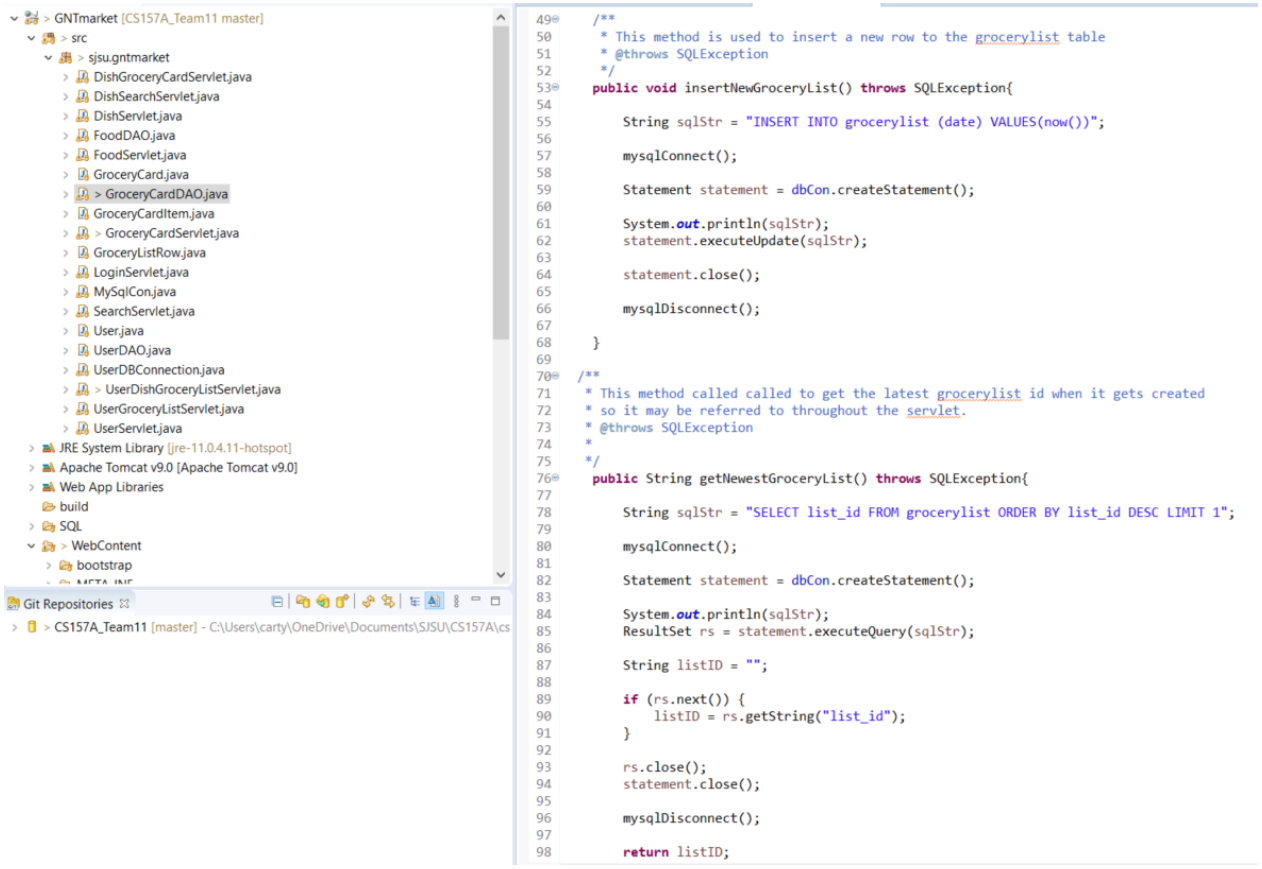
GroceryCardItem.java




```java
package sjsu.gntmarket;

public class GroceryCardItem {

   private int id;
   private String name;

   // Constructor
   public GroceryCardItem(int id, String name) {
      this.id = id;
      this.name = name;
   }

   public int getId() {
      return id;
   }

   public String getName() {
      return name;
   }
}
```

This GroceryCardItem.java is a simple class to hold the name and id number of a food item.

GroceryCardDAO.java



Looking more into the GroceryCardDAO.java, here when a new user creates their own grocery list it can be shown that a new grocery list row is inserted into the grocerylist table. After this is done, the most recently created grocery list_id is returned so it may be referred to when adding items to the user's grocery list.

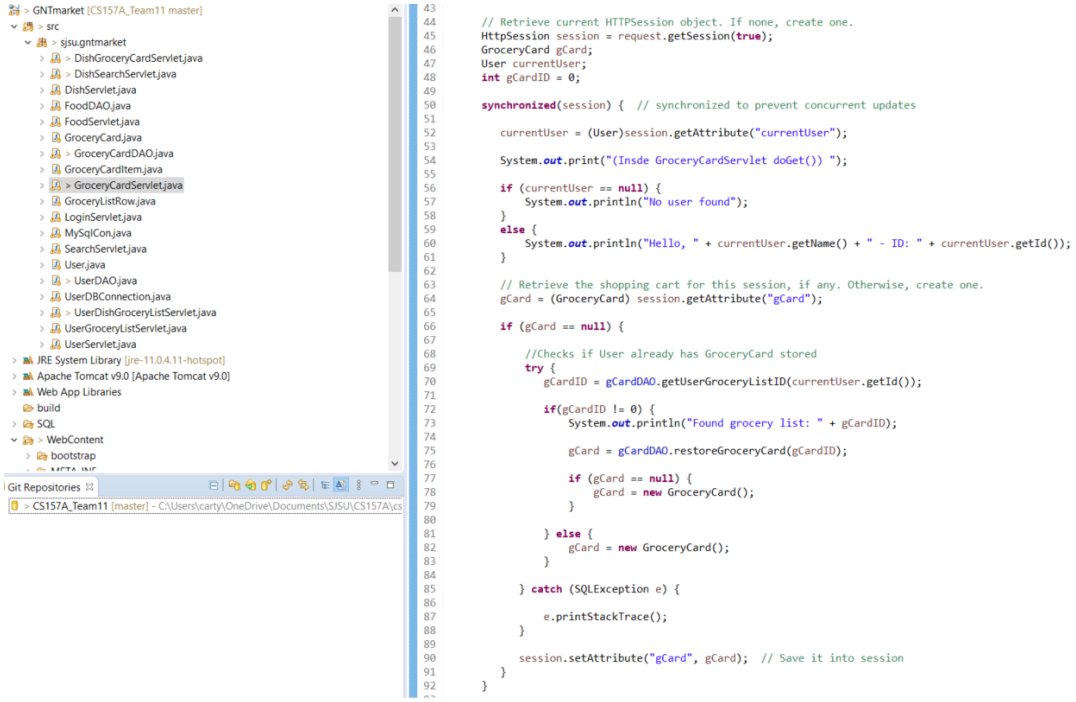
```java
/**
 *
 * This method removes a food item from a grocery list from taking the food item id to be removed
 * from the grocery list id.
 *
 * @param listID
 * @param foodID
 * @throws SQLException
 */
public void removeFromGroceryList(int listID, int foodID) throws SQLException {

    String sql = "DELETE FROM food_lists_grocerylist WHERE list_id = ? AND food_id = ?";

    mysqlConnect();

    PreparedStatement statement = dbCon.prepareStatement(sql);
    statement.setInt(1, listID);
    statement.setInt(2, foodID);

    System.out.println(sql);
    statement.executeUpdate();

    statement.close();

    mysqlDisconnect();

}
/**
 * This method brings back the grocery card that a user current has stored.
 *
 * @param listID
 * @return
 * @throws SQLException
 */
public GroceryCard restoreGroceryCard(int listID) throws SQLException {

    GroceryCard gCard = new GroceryCard();

    String sql = "SELECT food_id FROM food_lists_grocerylist WHERE list_id = ?";

    mysqlConnect();

    PreparedStatement statement = dbCon.prepareStatement(sql);
    statement.setInt(1, listID);

    System.out.println(sql);
    ResultSet rs = statement.executeQuery();
```

Looking more at it, there is a DELETE call made to remove an item from its associated grocery list in the database. Also partially shown is a call to the database to restore the grocery list that a user has already made when they log back into the application.

```java
/**
 * This method gets the latest created grocery list that the user has made.
 *
 * @param userID
 * @throws SQLException
 */
public int getUserGroceryListID(int userID) throws SQLException{

    String sql = "SELECT list_id FROM user_creates_grocerylist WHERE user_id = ? " +
                 "ORDER BY list_id DESC LIMIT 1";

    mysqlConnect();

    PreparedStatement statement = dbCon.prepareStatement(sql);
    statement.setInt(1, userID);

    ResultSet rs = statement.executeQuery();

    int listID = 0;

    if(rs.next()) {
        listID = rs.getInt("list_id");

        System.out.println("List ID: " + listID + " found for user: " + userID);
    }

    rs.close();
    statement.close();

    mysqlDisconnect();

    return listID;
}
```

Finally, a call is made to get the listID of the latest grocery list that a user has made so it may be used throughout the GroceryCardServlet.

GroceryCardServlet.java



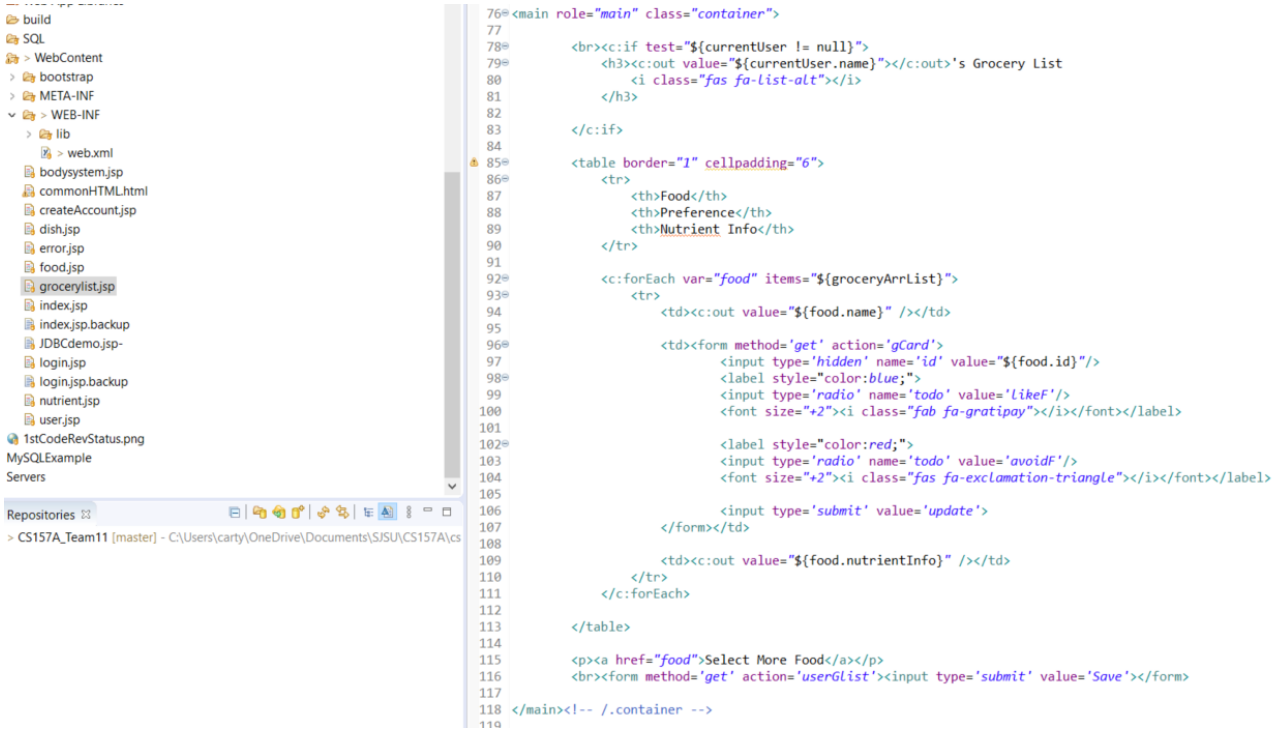

```
43
44      // Retrieve current HTTPSession object. If none, create one.
45      HttpSession session = request.getSession(true);
46      GroceryCard gCard;
47      User currentUser;
48      int gCardID = 0;
49
50      synchronized(session) {  // synchronized to prevent concurrent updates
51
52         currentUser = (User)session.getAttribute("currentUser");
53
54         System.out.print("(Insde GroceryCardServlet doGet()) ");
55
56         if (currentUser == null) {
57             System.out.println("No user found");
58         }
59         else {
60             System.out.println("Hello, " + currentUser.getName() + " - ID: " + currentUser.getId());
61         }
62
63         // Retrieve the shopping cart for this session, if any. Otherwise, create one.
64         gCard = (GroceryCard) session.getAttribute("gCard");
65
66         if (gCard == null) {
67
68             //Checks if User already has GroceryCard stored
69             try {
70                gCardID = gCardDAO.getUserGroceryListID(currentUser.getId());
71
72                if(gCardID != 0) {
73                    System.out.println("Found grocery list: " + gCardID);
74
75                    gCard = gCardDAO.restoreGroceryCard(gCardID);
76
77                    if (gCard == null) {
78                        gCard = new GroceryCard();
79                    }
80
81                } else {
82                    gCard = new GroceryCard();
83                }
84
85             } catch (SQLException e) {
86
87                e.printStackTrace();
88             }
89
90             session.setAttribute("gCard", gCard);  // Save it into session
91         }
92      }
```

Starting off in the GroceryCard Servlet are checks being made to see if the User already has a Grocery List stored in the database and if they do, retrieve it and restore it so it may be used throughout the application. Otherwise create a new one for the user.

```
/**
 *
 * This method is used to pass on the grocery list that a user has stored and display it on the grocery list page.
 * It sets the attribute to the current grocery list of the user that it made be used on the grocerylist.jsp page.
 *
 * @param request
 * @param response
 * @param gCard
 * @throws ServletException
 * @throws IOException
 * @throws SQLException
 */
public void displayGroceryList(HttpServletRequest request, HttpServletResponse response, GroceryCard gCard)
        throws ServletException, IOException, SQLException {


    List<GroceryListRow> groceryArrList = new ArrayList<GroceryListRow>();

    groceryArrList = foodDAO.getGroceryRows(gCard);

    request.setAttribute("gCard", gCard);
    request.setAttribute("groceryArrList", groceryArrList);

    HttpSession session = request.getSession();
    session.setAttribute("gCard", gCard);
    session.setAttribute("groceryArrList", groceryArrList);

    RequestDispatcher dispatcher = request.getRequestDispatcher("grocerylist.jsp");
    dispatcher.forward(request, response);
}
```

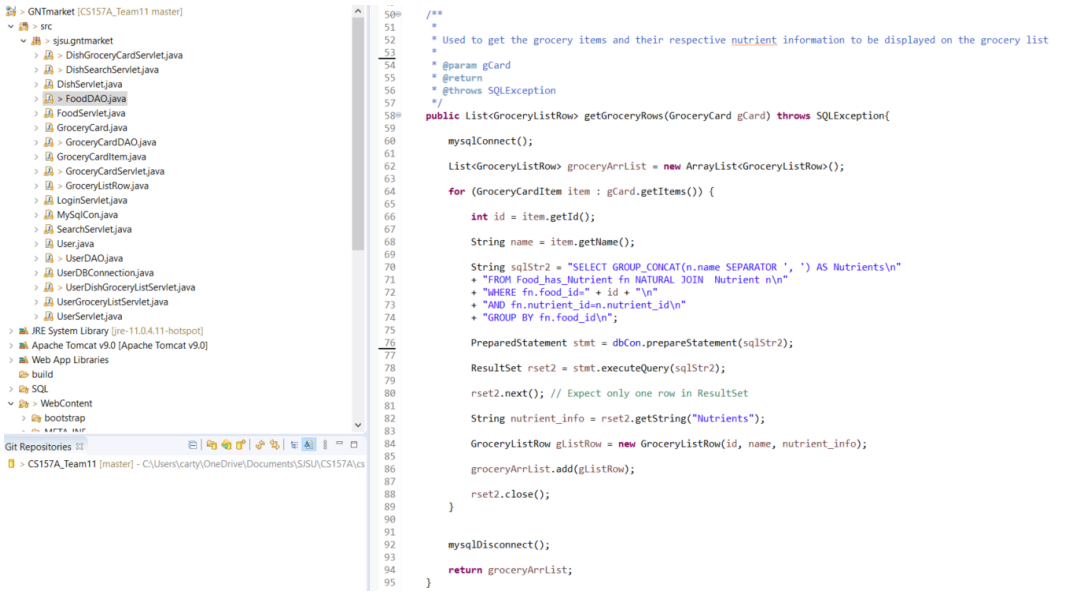
This method is what is called to get all the food information from a GroceryCard object to be displayed on the grocerylist.jsp The attribute list of grocery items is set to be used on other pages.

grocerylist.jsp



This is the front end of the grocerylist.jsp to that takes the groceryArrList attribute from the displayGroceryList method in the above image and displays the contents of the grocery list that a user has.

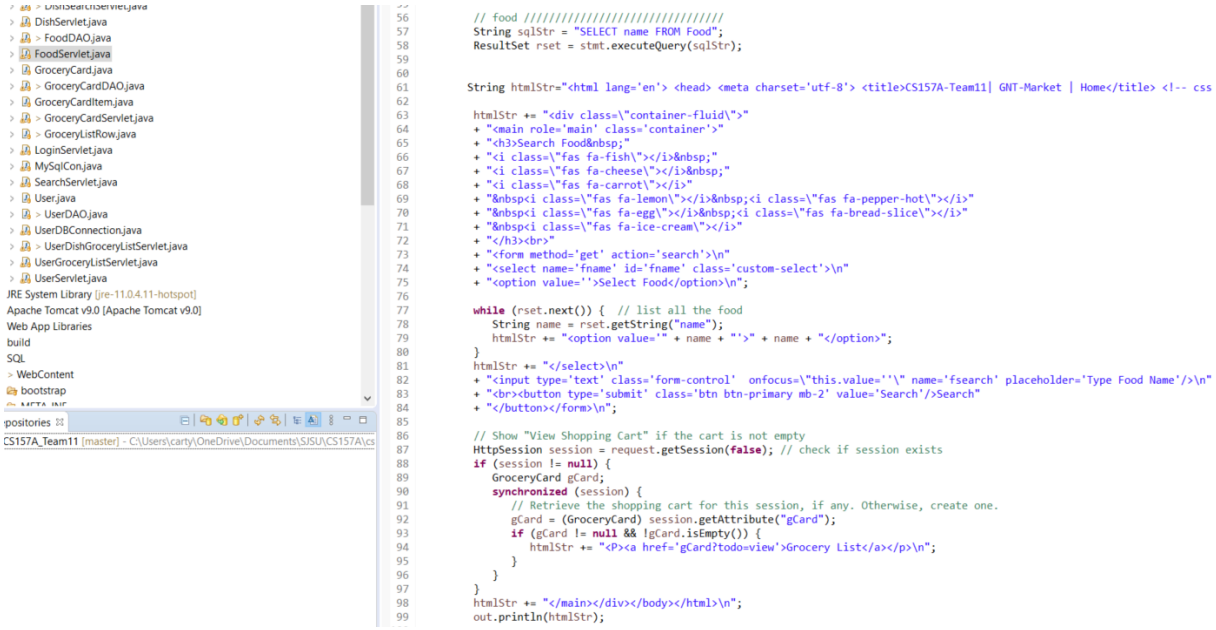
FoodDAO.java



This FoodDAO.java is a DAO class for the Food table. For this program it is only used to get the grocery

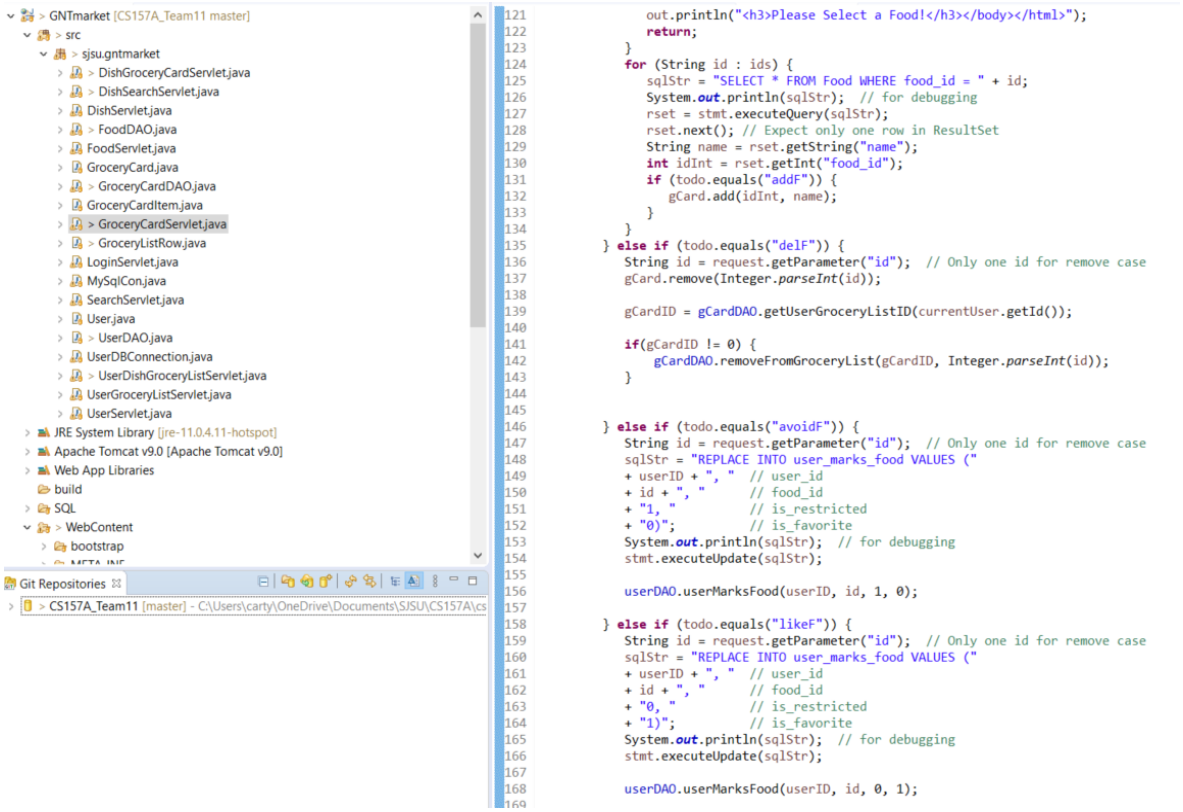
list items to be displayed in the grocery list page. The GroceryListRow class that can be seen is merely a class that extends the GroceryCardItem class to also hold the nutrient information for a particular food item.

FoodServlet.java



The FoodServlet.java is to make calls to the food table in the MySQL database to display all the food options that are stored in the database. The gets selected is then used as an option to be added to a grocery list.

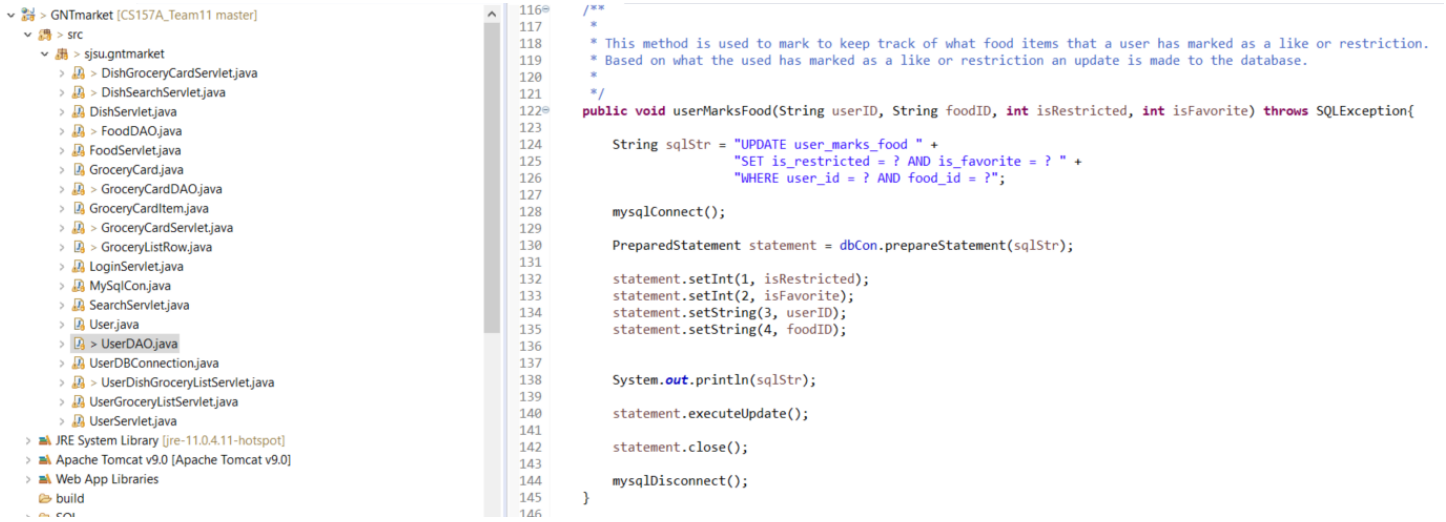
Marking food as Like or Restriction



```
            out.println("<h3>Please Select a Food!</h3></body></html>");
            return;
         }
         for (String id : ids) {
            sqlStr = "SELECT * FROM Food WHERE food_id = " + id;
            System.out.println(sqlStr);  // for debugging
            rset = stmt.executeQuery(sqlStr);
            rset.next(); // Expect only one row in ResultSet
            String name = rset.getString("name");
            int idInt = rset.getInt("food_id");
            if (todo.equals("addF")) {
               gCard.add(idInt, name);
            }
         }
      } else if (todo.equals("delF")) {
         String id = request.getParameter("id");  // Only one id for remove case
         gCard.remove(Integer.parseInt(id));

         gCardID = gCardDAO.getUserGroceryListID(currentUser.getId());

         if(gCardID != 0) {
             gCardDAO.removeFromGroceryList(gCardID, Integer.parseInt(id));
         }


      } else if (todo.equals("avoidF")) {
         String id = request.getParameter("id");  // Only one id for remove case
         sqlStr = "REPLACE INTO user_marks_food VALUES ("
         + userID + ", "  // user_id
         + id + ", "      // food_id
         + "1, "          // is_restricted
         + "0)";          // is_favorite
         System.out.println(sqlStr);  // for debugging
         stmt.executeUpdate(sqlStr);

         userDAO.userMarksFood(userID, id, 1, 0);

      } else if (todo.equals("likeF")) {
         String id = request.getParameter("id");  // Only one id for remove case
         sqlStr = "REPLACE INTO user_marks_food VALUES ("
         + userID + ", "  // user_id
         + id + ", "      // food_id
         + "0, "          // is_restricted
         + "1)";          // is_favorite
         System.out.println(sqlStr);  // for debugging
         stmt.executeUpdate(sqlStr);

         userDAO.userMarksFood(userID, id, 0, 1);
```

Here a user is able to mark as a like or restriction, then what gets selected gets updated in the user_marks_food table using the UserDAO.java object.

UserDAO.java



```
/**
 *
 * This method is used to mark to keep track of what food items that a user has marked as a like or restriction.
 * Based on what the used has marked as a like or restriction an update is made to the database.
 *
 */
public void userMarksFood(String userID, String foodID, int isRestricted, int isFavorite) throws SQLException{

    String sqlStr = "UPDATE user_marks_food " +
                    "SET is_restricted = ? AND is_favorite = ? " +
                    "WHERE user_id = ? AND food_id = ?";

    mysqlConnect();

    PreparedStatement statement = dbCon.prepareStatement(sqlStr);

    statement.setInt(1, isRestricted);
    statement.setInt(2, isFavorite);
    statement.setString(3, userID);
    statement.setString(4, foodID);


    System.out.println(sqlStr);

    statement.executeUpdate();

    statement.close();

    mysqlDisconnect();
}
```

As mentioned above, the UserDAO is an object that makes calls to user related tables, such as the user_marks_food in this case. Which is updating a food item as either like or restricted.

```java
/**
 * This method is called to insert a new user to the users table when a new account is created.
 *
 * @param user
 * @return
 * @throws SQLException
 */
public boolean createUser(User user) throws SQLException {
    String sql = "INSERT INTO users (email, password, name) VALUES (?, ?, ?)";
    mysqlConnect();

    PreparedStatement statement = dbCon.prepareStatement(sql);
    statement.setString(1, user.getEmail());
    statement.setString(2, user.getPassword());
    statement.setString(3, user.getName());

    boolean userCreated = (statement.executeUpdate() > 0);
    statement.close();
    mysqlDisconnect();

    return userCreated;
}
```

This createUser method is called when a user creates a new account, where it will make a call to the users table and add a new row for the new user.

```java
/**
 *
 * A method used to a return a user object once the user enters their credentials to login.
 *
 * @param email
 * @param password
 * @return
 * @throws SQLException
 */
public User getUser(String email, String password) throws SQLException {
    User user = null;

    String sql = "SELECT * FROM users WHERE email = ? AND password = ?";

    mysqlConnect();

    PreparedStatement statement = dbCon.prepareStatement(sql);
    statement.setString(1, email);
    statement.setString(2, password);

    ResultSet rs = statement.executeQuery();

    if(rs.next()) {
        int userID = rs.getInt("user_id");
        email = rs.getString("email");
        password = rs.getString("password");
        String name = rs.getString("name");

        user = new User(userID, email, password, name);

        System.out.println("Welcome, " + user.getName());
    }

    rs.close();
    statement.close();

    return user;
}
```

When a user logs in, a call is made to the users table to retrieve the user's information from their credentials.

createAccount.jsp

```
    <form action="insert" method="post" class="form-signin">
      <h1 class="h3 mb-3 font-weight-normal">Create New User</h1>
      <label for="name" class="sr-only">Name</label>
      <input type="text" id="name" name="name" class="form-control" placeholder="Name">
      <label for="email" class="sr-only">Email address</label>
      <input type="text" id="email" name="email" class="form-control" placeholder="Email address">
      <label for="pass" class="sr-only">Password</label>
      <input type="password" id="pass" name="password" class="form-control" placeholder="Password">
      <button class="btn btn-lg btn-primary btn-block" type="submit" value="Create New Account" />
      Create New Account</button>
      <p class="mt-5 mb-3 text-muted">© GNT-market</p>
</form>
```

On the front-end a form is used to create a new user, which the submit is then used to pass the information to the UserServlet. The login form is very similar to the create user form, only that it only has email address and password as input elements.

UserServlet.java

This is the UserServlet.java used to make calls regarding the user account depending on the method parameter that gets used in the URL, which the Switch-Case decides what method it should go to.

```
public void insertUser(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException, SQLException {

     String email = request.getParameter("email");
     String password = request.getParameter("password");
     String name = request.getParameter("name");

     User newUser = new User(email, password, name);
     userDAO.createUser(newUser);

     int newUserID = userDAO.restoreUserID(email, password);

     newUser.setId(newUserID);

     HttpSession session = request.getSession();
     session.setAttribute("currentUser", newUser);

     RequestDispatcher dispatcher = request.getRequestDispatcher("index.jsp");
     dispatcher.forward(request, response);

}

public void showNewUserForm(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    RequestDispatcher dispatcher = request.getRequestDispatcher("createAccount.jsp");
    dispatcher.forward(request, response);

}

public void loginPage(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        RequestDispatcher dispatcher = request.getRequestDispatcher("login.jsp");
        dispatcher.forward(request, response);

}

public void loginUser(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException, SQLException {

    String email = request.getParameter("email");
    String password = request.getParameter("password");

    User returningUser = userDAO.getUser(email, password);

    request.setAttribute("currentUser", returningUser);

    HttpSession session = request.getSession();
    session.setAttribute("currentUser", returningUser);
```

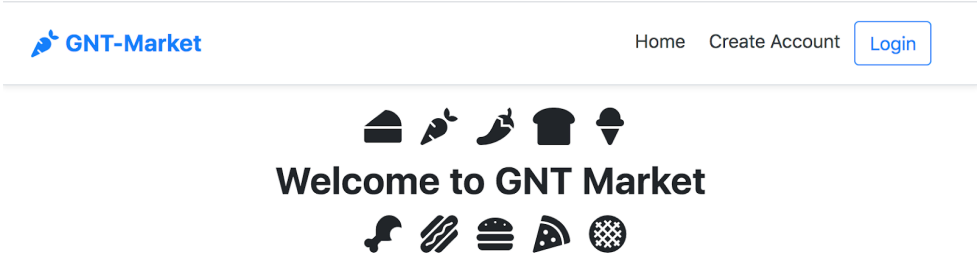
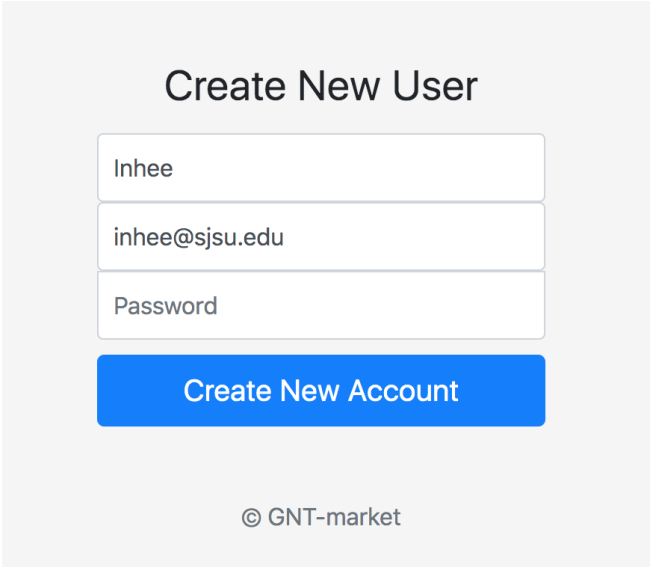
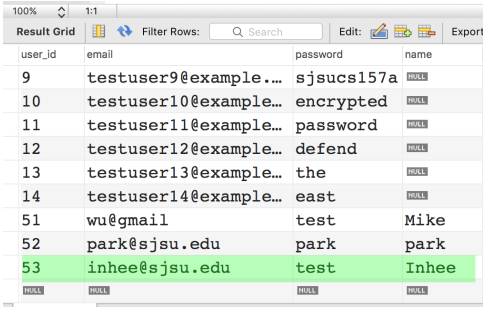
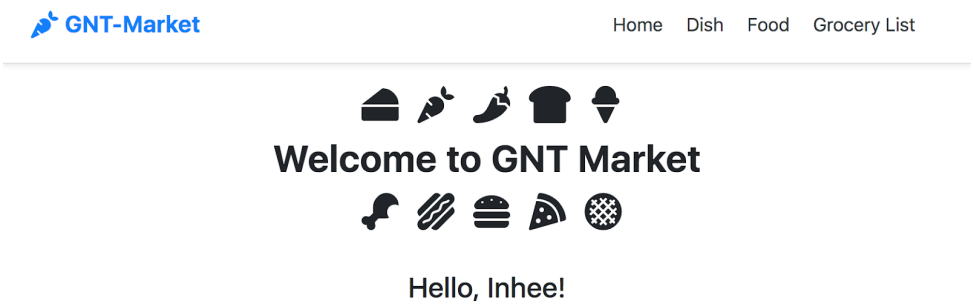
These are additional methods shown that depend on what gets shown in the URL the servlet will direct to a respective method to either make calls to the database such as logging in a user or creating an account, or it will just redirect the user to the home page.

index.jsp

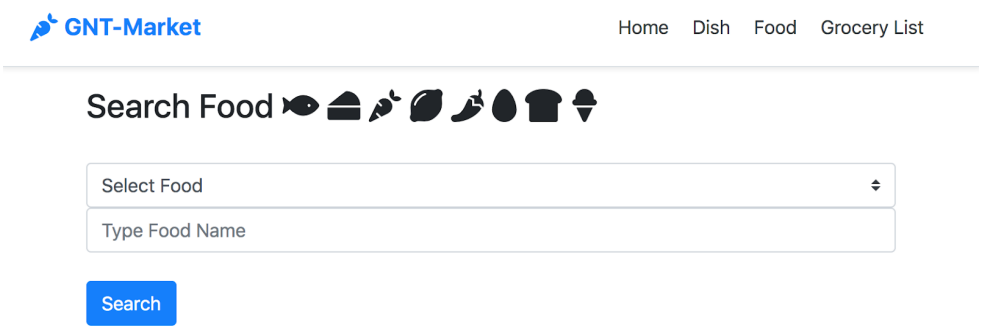
```
<body>
<div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-white border-bottom shadow-sm">
<h5 class="my-0 mr-md-auto font-weight-normal">
<a href="#" class="navbar-brand d-flex align-items-center">
<i class='fas fa-carrot' style='font-size:24px'></i>
<strong> GNT-Market</strong>
</a></h5>
<nav class="my-2 my-md-0 mr-md-3">
    <a class="p-2 text-dark" href="/GNTmarket/home">Home</a>

    <c:if test="${currentUser != null}">
        <a class="p-2 text-dark" href="/GNTmarket/dish">Dish</a>
        <a class="p-2 text-dark" href="/GNTmarket/food">Food</a>
        <a class="p-2 text-dark" href="/GNTmarket/gCard">Grocery List</a>
    </c:if>

    <c:if test="${currentUser == null}">
        <a class="p-2 text-dark" href="/GNTmarket/new-account">Create Account</a>
        <a class="btn btn-outline-primary" href="/GNTmarket/login">Login</a>
    </c:if>
    <!-- <a class="p-2 text-dark" href="/GNTmarket/user-list">User Test Page</a>  -->
  </nav>
</div>


    <div align="center">
        <h2>
          <i class="fas fa-cheese"></i>
          <i class="fas fa-carrot"></i>
          <i class="fas fa-pepper-hot"></i>
          <i class="fas fa-bread-slice"></i>
          <i class="fas fa-ice-cream"></i>
        </h2>
        <h2><strong>Welcome to GNT Market</strong></h2>
        <h2>
          <i class="fas fa-drumstick-bite"></i>
          <i class="fas fa-hotdog"></i>
          <i class="fas fa-hamburger"></i>
          <i class="fas fa-pizza-slice"></i>
          <i class="fas fa-stroopwafel"></i>
        </h2>
        <br><c:if test="${currentUser != null}">
        <h4>Hello, <c:out value="${currentUser.name}"></c:out>!</h4>
        </c:if>
    </div>
</body>
```

The homepage of the web application. Only certain parts of the nav bar will be displayed depending on whether or not the user is logged in.

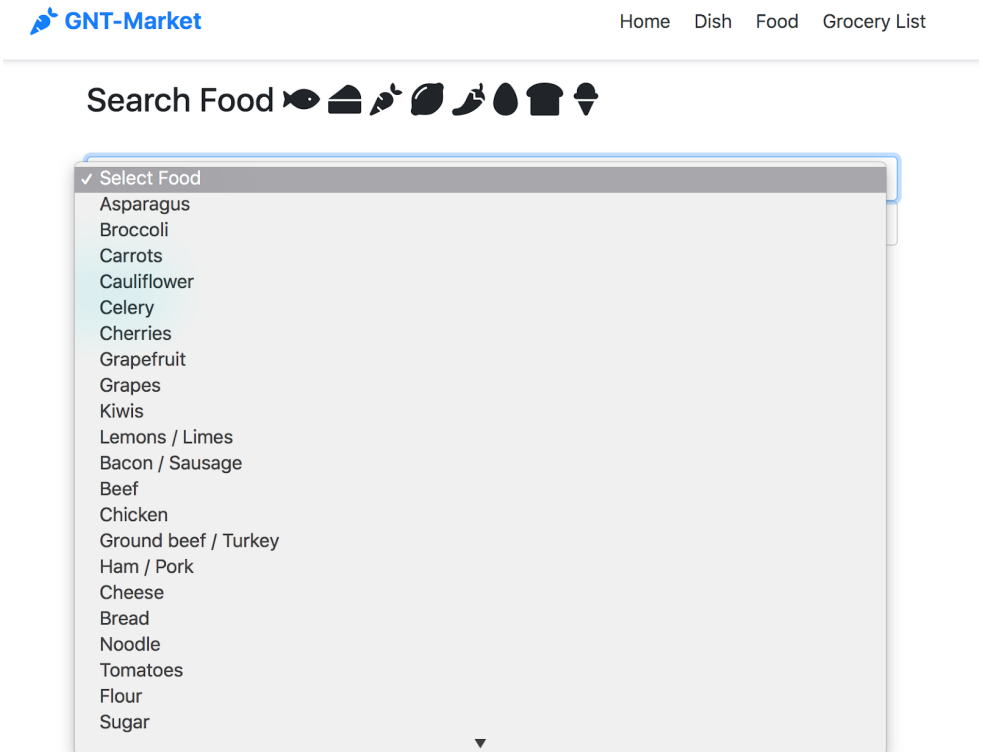
Demonstration of Functions/Features with DB Manipulation

Functions/Features	DB Snapshots
Main welcome font page ‘index.jsp’ with a minimal menu on top for non-login status. Once click “Create Account” 	
Redirected to ‘createAccount.jsp’ page to recreate an account 	“Users” table is updated  INSERT INTO users (email, password, name) VALUES (?, ?, ?)
Upon user is created, automatically redirected to ‘index.jsp’ again. WIth user session being recognized, displays dish/food menu. 	
Click the “Food”menu, the page is redirected to “FoodServlet.java”.	

user_id	email	password	name
9	testuser9@example....	sjsucs157a	NULL
10	testuser10@example...	encrypted	NULL
11	testuser11@example...	password	NULL
12	testuser12@example...	defend	NULL
13	testuser13@example...	the	NULL
14	testuser14@example...	east	NULL
51	wu@gmail	test	Mike
52	park@sjsu.edu	park	park
53	inhee@sjsu.edu	test	Inhee
NULL	NULL	NULL	NULL



Users can select a food item from the pull-down menu, which is fetched from the “Food” table from DB.



“Food” table from DB.

food_id	name
1	Asparagus
2	Broccoli
3	Carrots
4	Cauliflower
5	Celery
6	Cherries
7	Grapefruit
8	Grapes
9	Kiwis
10	Lemons / Limes
11	Bacon / Sausage
12	Beef
13	Chicken
14	Ground beef / Turkey
15	Ham / Pork
16	Cheese
17	Bread
18	Noodle
19	Tomatoes
20	Flour
21	Sugar
22	Lettus

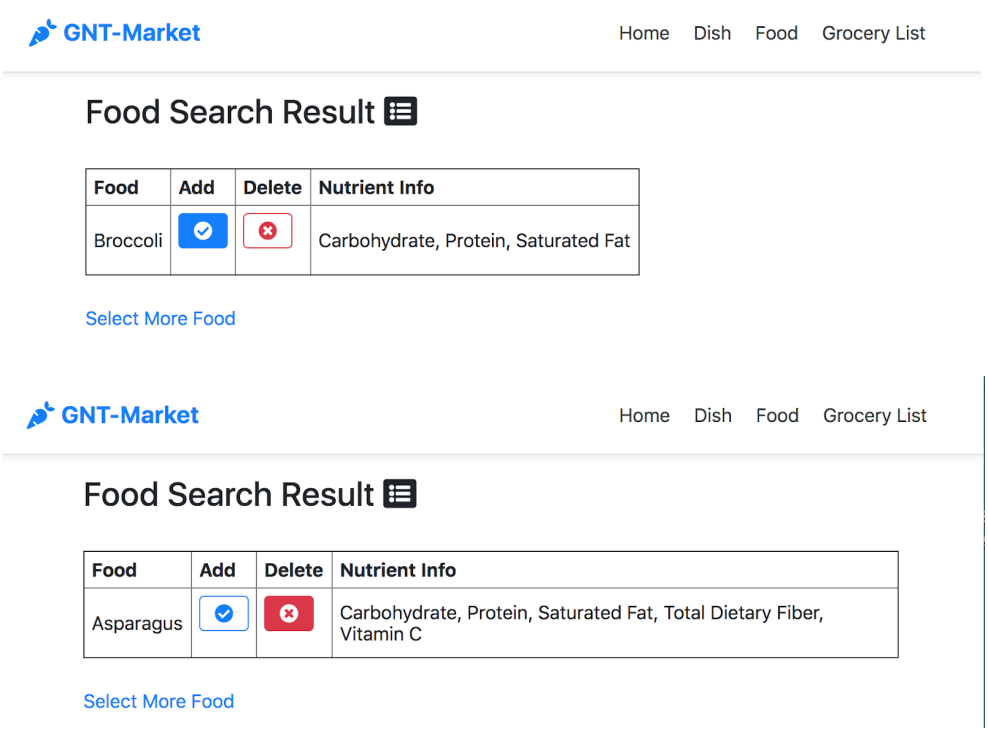
Food 1

SELECT name FROM Food

Upon selection of a food, the page is redirected to ‘SearchServlet.java“. The selected food item is displayed with its corresponding nutrient information, so that users can decide either add to or delete from the grocery list.
Note that this Add/Delete operation does not manipulate DB yet, rather saved in java servlet object.
Users keep searching for more food by clicking the “Select More Food” link, which would redirect to “FoodServlet.java”

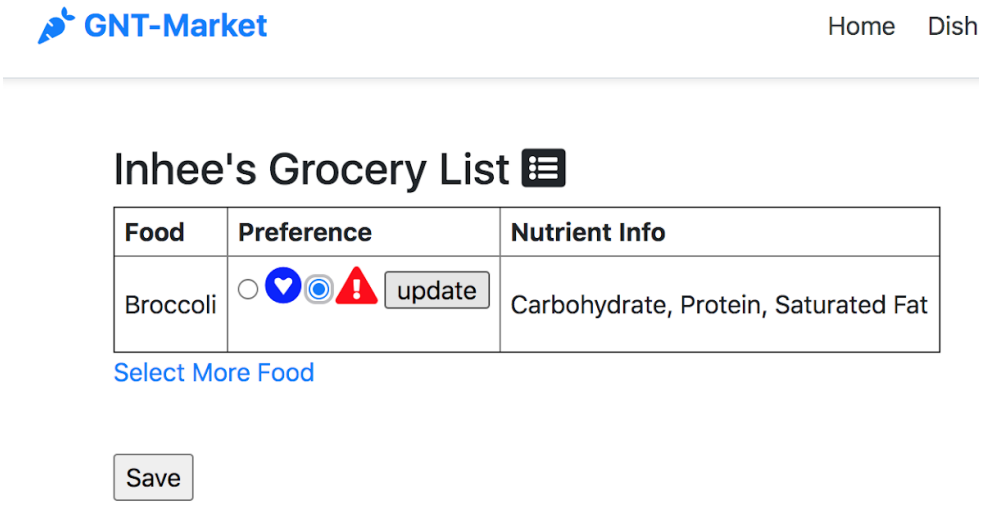
Three tables (Food_has_Nutrient, Food, and Nutrient) are joined to extract nutrient information displayed on the food search result.

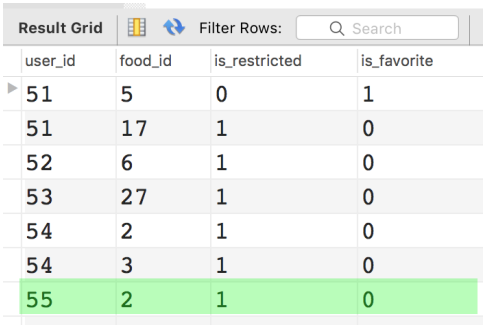
SELECT f.name AS name, f.food_id AS food_id, GROUP_CONCAT(n.name




```
SEPARATOR ', ') AS Nutrients
FROM Food_has_Nutrient fn
NATURAL JOIN  Nutrient n  JOIN
Food f ON f.food_id WHERE
f.name='Broccoli'   AND
fn.food_id=f.food_id GROUP BY
f.food_id
```

Users can mark their dietary preference either “favorite” or “restricted” on each food item by click the radio button followed by update button, which will execute the “REPLACE INTO” sql query (it’s a handy function combining INSERT INTO and UPDATE depending on the existence/absence of the primary keys (user_id, food_id).



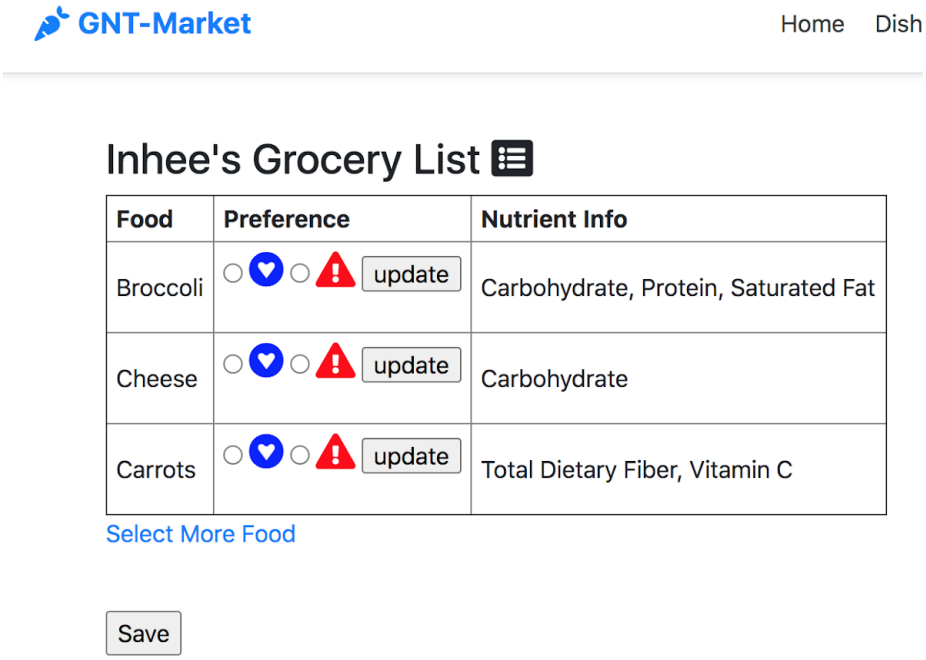


UserID: 55

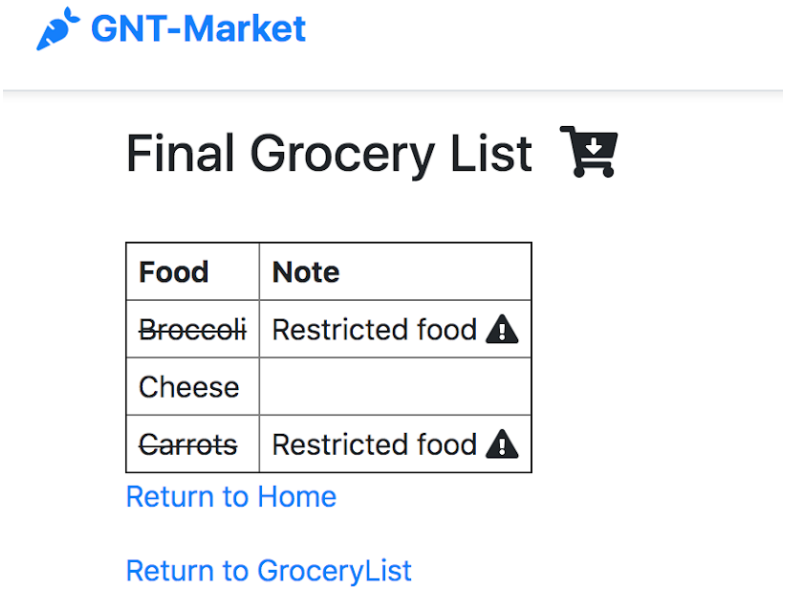
```
REPLACE INTO user_marks_food
VALUES (55, 2, 1, 0)
```

Users keep adding food menus and updating their dietary preference by referring to the provided nutrient information.
Once finished all selection, users can click the “save'' button, which will archive the user selection to the DB. Again a handy “REPLACE INTO” keyword is used instead of “INSERT INTO” or “UPDATE” depending on the existence of primary keys.

```
REPLACE INTO
Food_lists_GroceryList (food_id,
list_id) VALUES (2, 89),(4, 89),(13,
89),(5, 89),(19, 89),(23, 89)
```



Finalized grocery list is redirected to “UserGroceryCardServlet.java”, where 1) creates grocery list in “users_create_grocerylist” table; 2) insert food list to the just created grocery list; also 3) check each food item from the “user_mark_foods” table to extract information to display the restricted food on the grocery list.



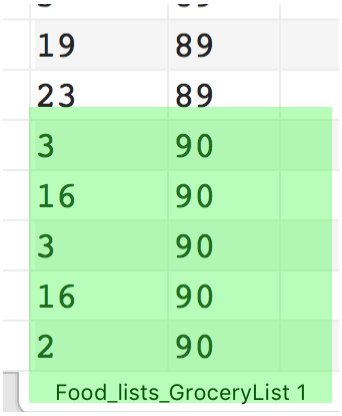
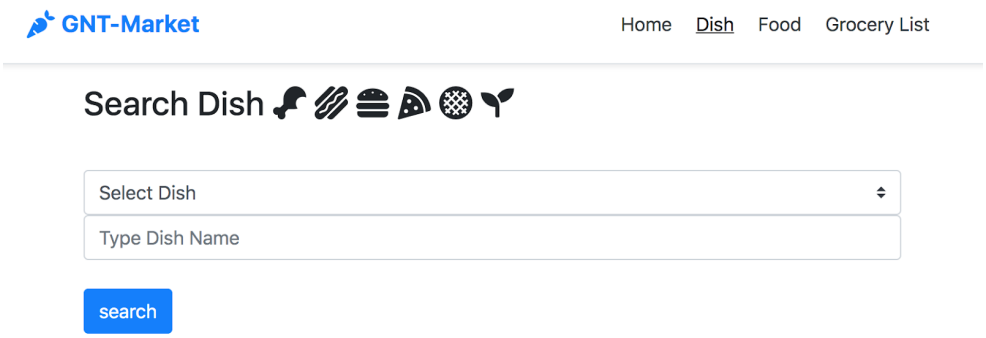
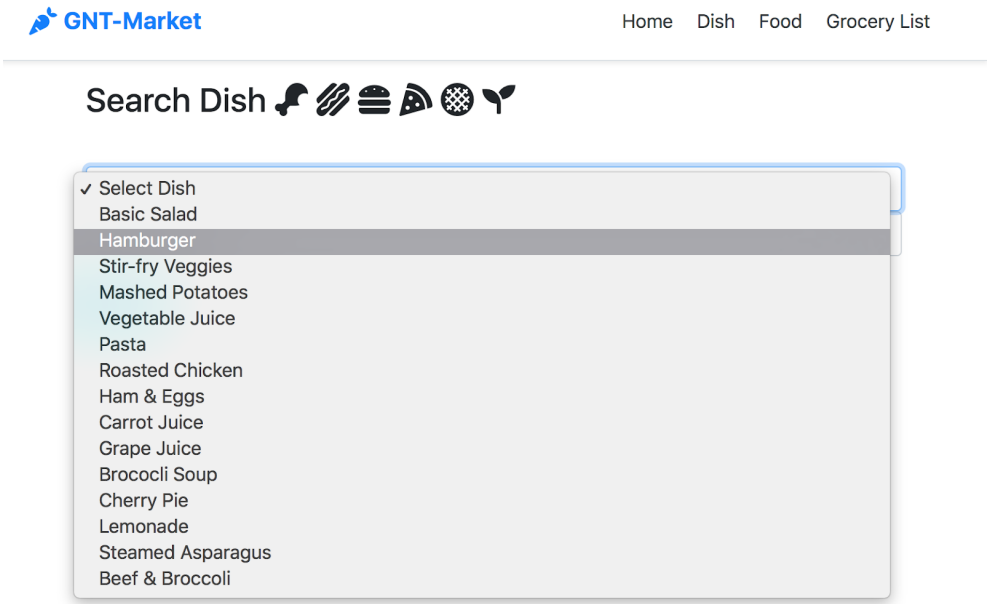
List ID: 90 found for user: 56

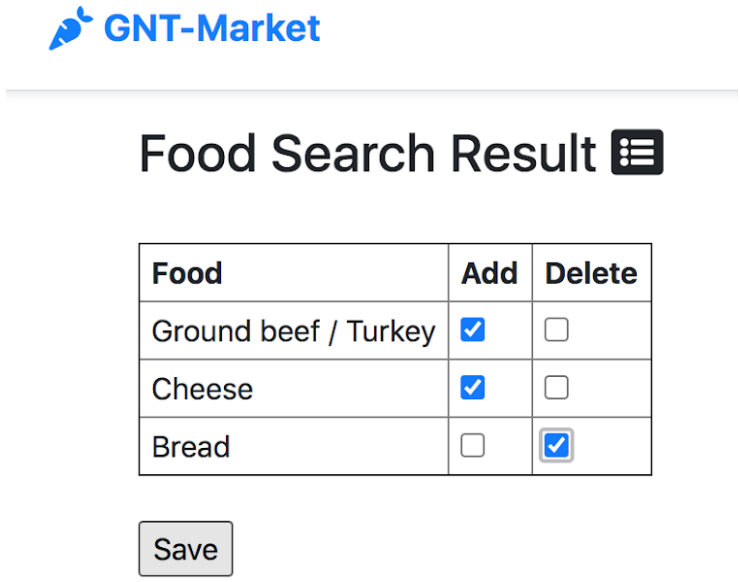
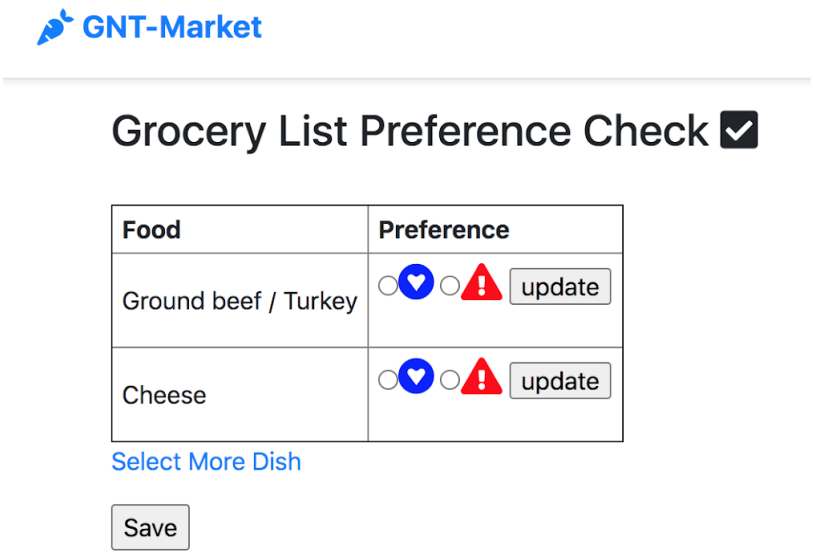
REPLACE INTO user_creates_grocerylist (user_id, list_id) VALUES (56, 90)

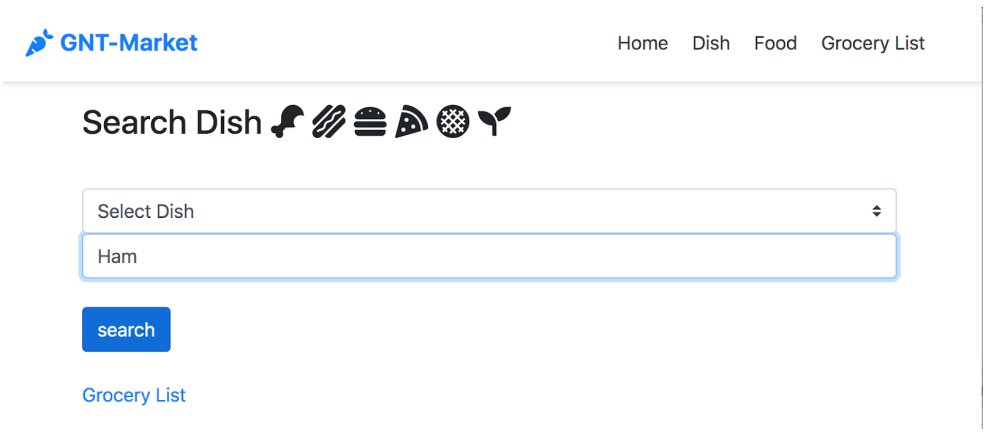
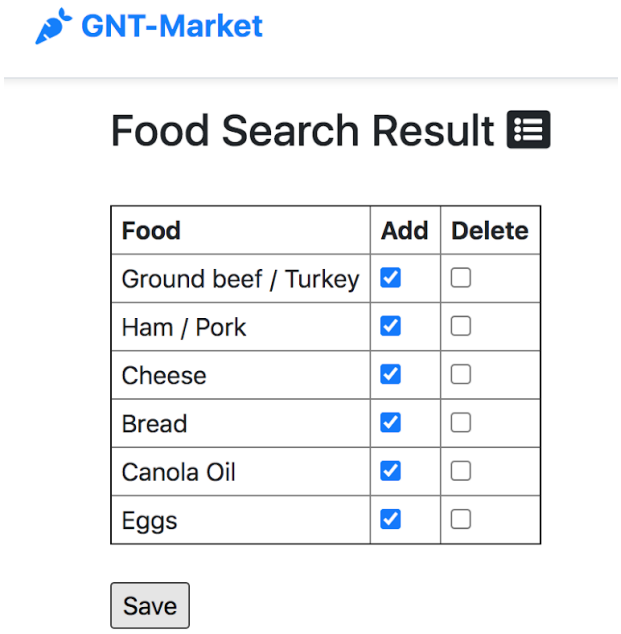
list_id	date
14	2020-07-05 16:13:25
15	2020-07-05 16:13:31
85	2020-08-04 13:55:53
86	2020-08-04 14:11:34
87	2020-08-04 16:09:17
88	2020-08-04 16:14:14
89	2020-08-04 21:24:18
90	2020-08-04 21:28:35

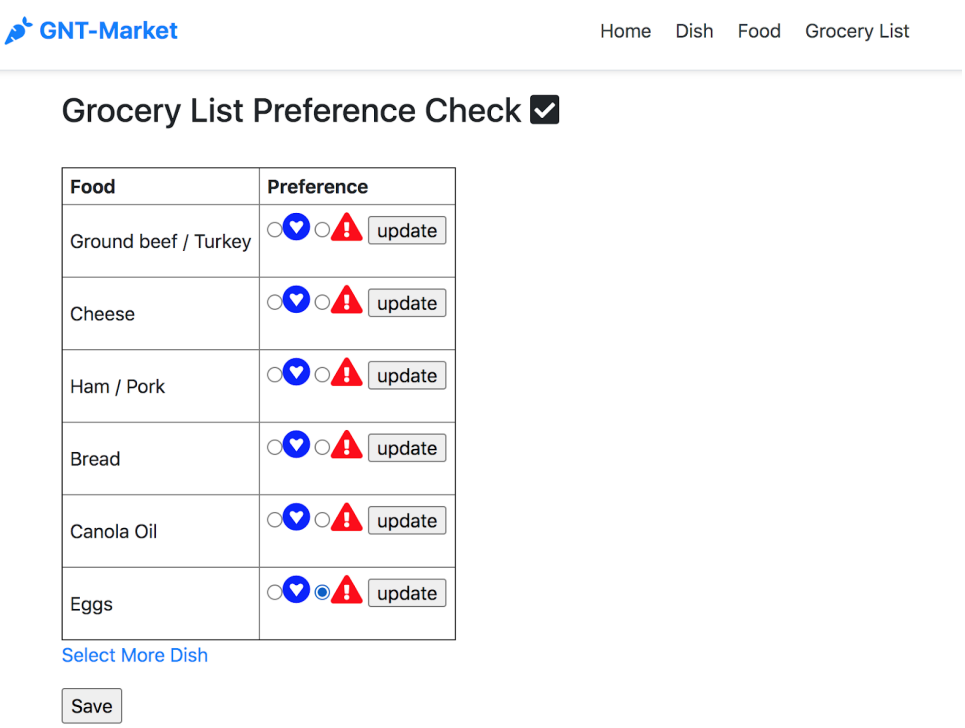
REPLACE INTO Food_lists_GroceryList (food_id, list_id) VALUES (3, 90),(16, 90),(2, 90)

food_id	list_id

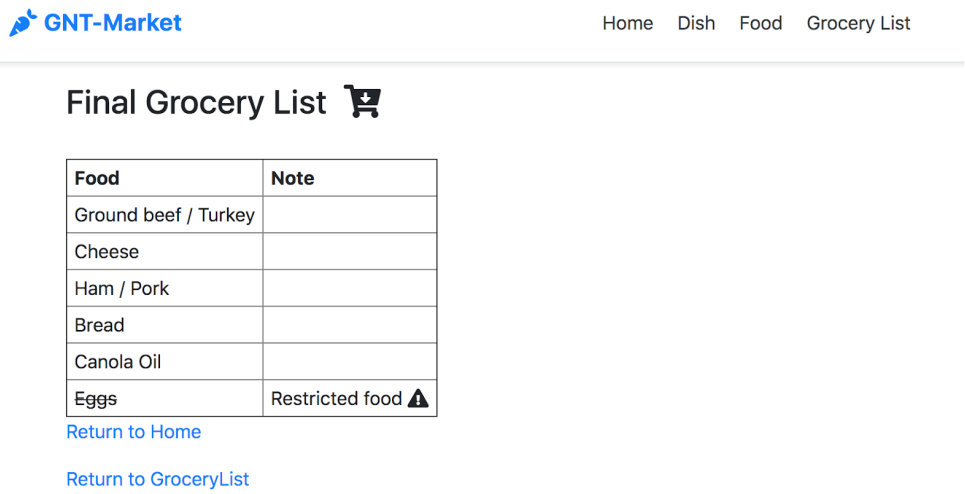
	
Users can generate a grocery list from the dish menu: just one dish selection would internally generate all ingredient food lists. 	
The Dish page is redirected to “DishServlet.java”. All dish items from the “dish” table are fetched and construct a drop-down menu. For example, the user selects a dish menu “Hamburger”, which would generate... 	SELECT name FROM dish; 

The selected dish “Hamburger” lists three ingredient food items. This dish-based search page is redirected to “DishSearchServlet.java”. Users can either add/delete the food items to the grocery list. At this stage, no direct DB manipulation, rather java servlet layer operation to hold food items in the java object. 	
Whether users select food from the “Food” menu or “Dish” menu, an essential activity is to add food items to the grocery list. Thus at this stage, we reuse the “GroceryCardServlet.java” and “groceryList.jsp” 	
Users can also search for a dish by typing the partial word, then it will fetch the “dish” table using that keyword from DB.	SELECT * FROM Food f, dish_has_food df, dish d WHERE df.dish_id IN (SELECT dish_id FROM dish WHERE name = '' OR

<table>
<tr>
<td> 
</td>
<td>name LIKE '%Ham%'AND f.food_id=df.food_id AND d.dish_id=df.dish_id)</td>
</tr>
<tr>
<td>With the “Ham” search word, it would display the ingredient food items from all the dish names containing “Ham”. In this example, Ham & Cheese and Hamburger.
 
</td>
<td></td>
</tr>
<tr>
<td>Similar to the function of “food”, at this stage yet not archiving the selected food to the DB, but store the list in the java servlet object. ONly when clicking the “Save” button on the bottom, the list would be archived into the DB. In this example, users mark Eggs as restricted food.</td>
<td></td>
</tr>
</table>



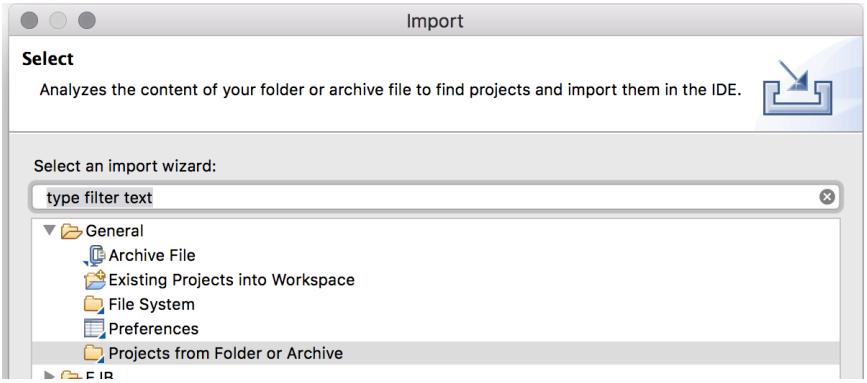
When finalizing the user's grocery list to the DB, 1) create a grocery list ID; 2) insert food_id to grocery list ID; 3) fetching the food_id whose attribution of "is_restricted=1" in the "users_mark_food" table, then displaying the corresponding food with additional information.



sProcedures for Setup and Run the WebApp

1. Assume you have SQL Workbench and TomCat installed on your system.
2. Git clone https://github.com/ipark-CS/CS157A_Team11.git to get source code of GNTmarket, which is in the direction "3rd_WebApp/"; OR obtain the "Team11_GNTmarket.zip".
3. In Eclipse, New Project→ Import from General Files → Import Project from File System or Archive → Select "3rd_WebApp/" (if get it from git clone) or "Team11/GNTmarket/" (if get it

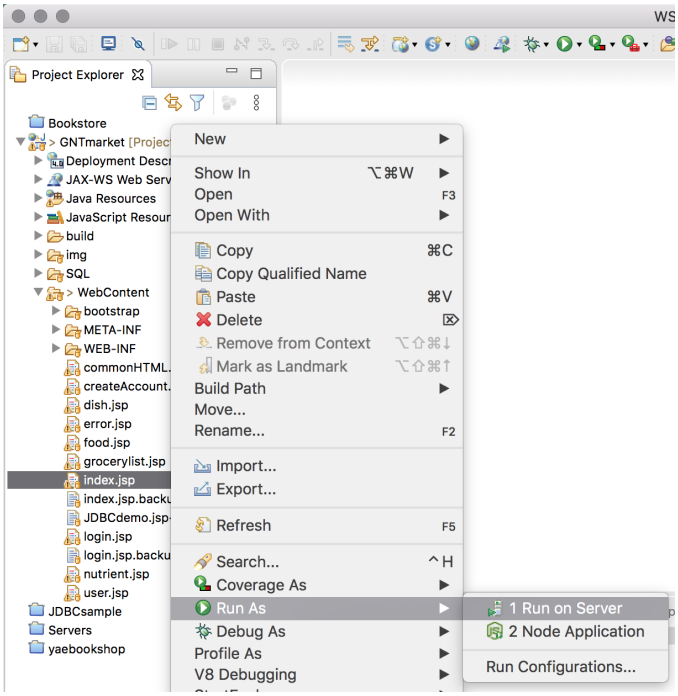
from zip file, firstly unzip it).



4. Impot Team11/GNTmarket/SQL/gntmarket.sql to your SQL Workbench. Make sure to change your SQL Workbench credentials by editing WebContent/WEB-INF/web.xml.

```
<context-param>
  <param-name>dbURL</param-name>
  <param-value>jdbc:mysql://localhost:3306/GNTmarket?serverTimezone=UTC</param-value>
</context-param>

<context-param>
  <param-name>dbUser</param-name>
  <param-value>root</param-value>
</context-param>

<context-param>
  <param-name>dbPassword</param-name>
  <param-value>pass...</param-value>
</context-param>
```

5. The GNTmarket eclipse project was built as a Dynamic Web App, Run on Server by right-click of WebContent/index.jsp.



6. The provided SQL file located in the /3rd_WebApp/SQL folder must also be used to import all the tables used in this application. The “GNTmarket-final.sql” file is our final export for our SQL tables.

Lesson Learned

- Each team member write a short paragraph to describe your hands-on experience in this course project.
- Create a zip file containing an electronic version of the final report in pdf, exported mySQl database files, and all program source code files. (Team#_Project.zip)
- Upload “Team#_Project.zip” file to Canvas by 11:59pm, Tuesday, August 4th, 2020.

Tracy’s Comment:

This has been my first time developing a rather large web application, so there was definitely a big learning curve in understanding how it all works especially with the calls from the application to the database. I have worked with databases before but it was only with queries, this is the first time I was involved with building a database from scratch and incorporating it in a web application. This class has provided me with some good experience in working with a database and seeing how it works in the back-end along with the addition of developing a working web application that makes calls to the database and how it can retrieve and manipulate data. Now I have more confidence in myself that I can be capable of developing more web applications of my own or even possibly as a web developer career.

Inhee’s Comment:

I’ve always been wondering what is “Full Stack Development” and what is “MVC [Model-View-Controller] framework”. Only after completing the webapp project from CS157A, I fully comprehend what they are. From DB backend [Model] via Tomcat web server with Java Servlet [Controller], JSP and front-end using Bootstrap [View], I really enjoyed coding, especially a pair coding with my partner. Now I am comfortable to build another webapp from ideation, ERD, Schema and all the way to WebApp.