

Project Final Report (Due 8/4/2020)

Summer 2020 | CS 157A



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. UserCreatesGroceryList:

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. UserMarksFood:

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. DishHasFood:

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.

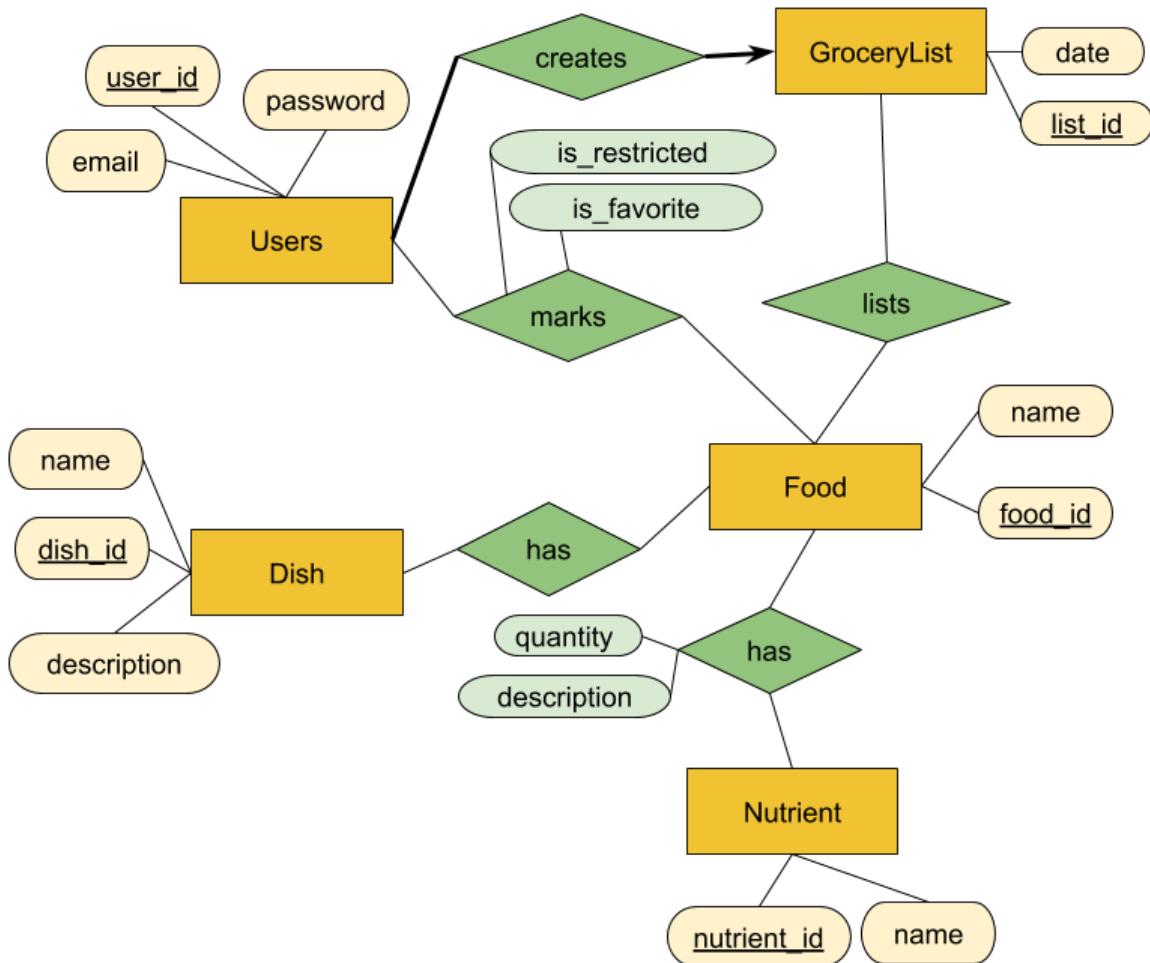
Relationship4. FoodHasNutrient:

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. FoodListsGroceryList:

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. **UserCreatesGroceryList**(user_id, list_id)
- R2. **UserMarksFood**(user_id, food_id, is_restricted, is_favorite)
- R3. **DishHasFood**(dish_id, food_id)
- R4. **FoodHasNutrient**(food_id, nutrient_id, quantity, description)
- R5. **FoodListsGroceryList**(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	sql90
4	testuser4@example.com	f4fopi5
5	testuser5@example.com	fiver
6	testuser6@example.com	qrghbdseed52
7	testuser7@example.com	septem
8	testuser8@example.com	passtest123
9	testuser9@example.com	sjucs157a
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 Cauliflower	Cauliflower 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)

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view is open, showing the 'GNMarket' schema with various tables like 'BodySystem', 'Category', 'Dish', and 'Food'. The 'Food' table is currently selected. The main area displays the 'Food' table's data in a grid. The columns are 'food_id' and 'name'. The data consists of 15 rows:

food_id	name
30001	Asparagus
30002	Broccoli
30003	Carrots
30004	Cauliflower
30005	Celery
30006	Cherries
30007	Grapefruit
30008	Grapes
30009	Kiwis
30010	Lemons / Limes
30011	Bacon / Sausage
30012	Beef
30013	Chicken
30014	Ground beef / Turkey
30015	Ham / Pork

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)

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view is expanded to show the 'GNMarket' schema, which contains tables like BodySystem, Category, Dish, Food, Food_has_Nutrient, Food_in_Category, Food_lists_GroceryList, GroceryList, and Nutrient. The 'Nutrient' table is selected and highlighted in green. Below the tree view, there are tabs for 'Object Info' and 'Session'. The 'Object Info' tab displays the table definition: 'Table: Nutrient' with columns 'nutrient_id' (int PK) and 'name' (varchar(45)). The main workspace is titled 'Query 28' and shows the 'Nutrient' table data in a grid format. The grid has two columns: 'nutrient_id' and 'name'. The data consists of 25 rows, each representing a different nutrient. The first few rows are: 6001 Carbohydrate, 6002 Polyunsaturated fat, 6003 Protein, 6004 Saturated Fat, 6005 Total Dietary Fiber, 6006 Calcium, 6007 Fluoride, 6008 Iron, 6009 Magnesium, 6010 Manganese, 6011 Phosphorus, 6012 Potassium, 6013 Selenium, 6014 Sodium, 6015 Zinc, 6016 Caffeine, 6017 Cholesterol, 6018 Omega-3 Fatty Acid, 6019 Vitamin C, 6020 Vitamin B, 6021 Vitamin D, 6023 Vitamin E, 6024 Water, and 6025 Vitamin A.

nutrient_id	name
6001	Carbohydrate
6002	Polyunsaturated fat
6003	Protein
6004	Saturated Fat
6005	Total Dietary Fiber
6006	Calcium
6007	Fluoride
6008	Iron
6009	Magnesium
6010	Manganese
6011	Phosphorus
6012	Potassium
6013	Selenium
6014	Sodium
6015	Zinc
6016	Caffeine
6017	Cholesterol
6018	Omega-3 Fatty Acid
6019	Vitamin C
6020	Vitamin B
6021	Vitamin D
6023	Vitamin E
6024	Water
6025	Vitamin A

Relationship1. UserCreatesGroceryList(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)

<u>user_id</u>	<u>food_id</u>	<u>is_restricted</u>	<u>is_favorite</u>
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)

<u>dish_id</u>	<u>food_id</u>
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)

The screenshot shows the MySQL Workbench interface with the 'Management' tab selected. In the left sidebar, under the 'GNMarket' schema, the 'Food_has_Nutrient' table is highlighted. The main pane displays the data from the 'Food_has_Nutrient' table in a grid format.

Table: Food_has_Nutrient

Columns:

- food_id** int PK
- nutrient_id** int PK
- quantity** decimal(5,2)
- description** varchar(45)

Result Grid:

food_id	nutrient_id	quantity	description
30001	6001	2.60	asparagus has 2.6g carbohydrates
30001	6003	1.50	asparagus has 1.5g protein
30001	6004	0.10	asparagus has 1.5g fat
30001	6005	1.40	asparagus has 1.4g fiber
30001	6019	0.00	asparagus has 0.38mg vitamin C
30006	6001	19.00	cherries have 19g carbohydrates
30006	6003	1.60	cherries have 1.6g protein
30006	6005	2.50	cherries have 2.5g fiber
30006	6012	0.27	cherries have 268mg potassium
30009	6001	10.00	kiwi has 10g carbohydrates
30009	6004	0.40	kiwi has 0.4g fat
30009	6012	0.22	kiwi has 215mg potassium
30011	6003	2.90	bacon has 2.9g protein
30011	6004	3.50	bacon has 3.5g fat
30011	6012	0.04	bacon has 44mg potassium
30011	6014	0.18	bacon has 178mg sodium
30011	6017	0.01	bacon has 9mg cholesterol

Relationship5. Food_lists_GroceryList(food_id, list_id)

The screenshot shows the MySQL Workbench interface with the 'Management' tab selected. In the left sidebar, under the 'GNMarket' schema, the 'Food_lists_GroceryList' table is highlighted. The main pane displays the data from the 'Food_lists_GroceryList' table in a grid format.

Table: Food_lists_GroceryList

Columns:

- food_id** int PK
- list_id** int PK

Result Grid:

food_id	list_id
30001	12
30002	12
30003	12
30009	12
30013	12
30011	12
30001	13
30008	13
30012	13
30004	13
30003	14
30005	14
30007	14
30010	14
30011	14
30015	14

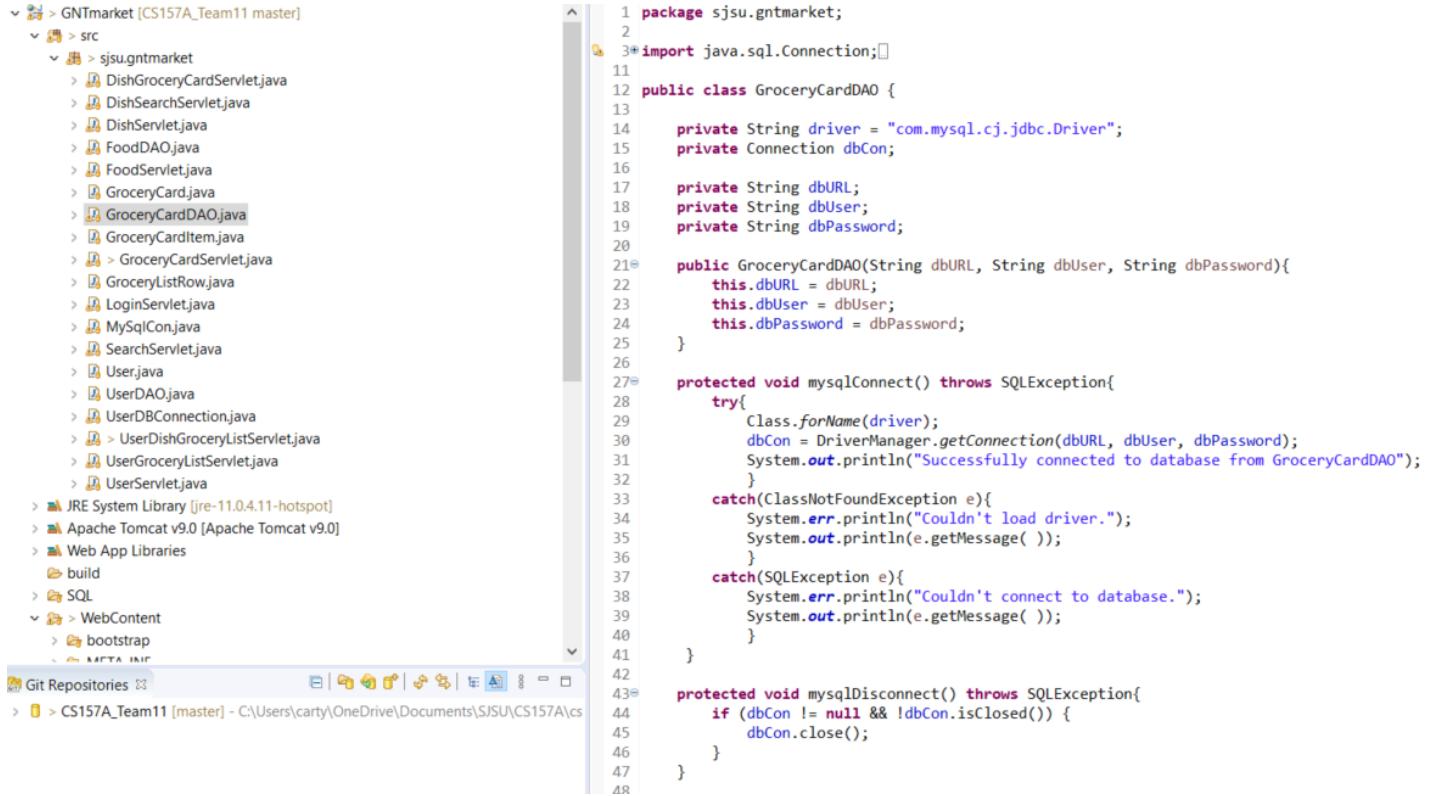
Project Implementation

- 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	<pre> --- createAccount.jsp --- index.jsp --- error.jsp --- login.jsp --- user.jsp </pre>	<pre> --- LoginServlet.java --- User.java --- UserDAO.java --- UserDBConnection.java --- UserDishGroceryListServlet.java --- UserGroceryListServlet.java --- UserServlet.java </pre>
Food	<pre> --- food.jsp --- grocerylist.jsp --- nutrient.jsp </pre>	<pre> --- FoodDAO.java --- FoodServlet.java --- GroceryCard.java --- GroceryCardDAO.java --- GroceryCardItem.java --- GroceryCardServlet.java --- GroceryListRow.java --- SearchServlet.java </pre>
Dish	<pre> --- dish.jsp </pre>	<pre> --- DishGroceryCardServlet.java --- DishSearchServlet.java --- DishServlet.java </pre>

Database Access Object (using GroceryCardDAO.java)



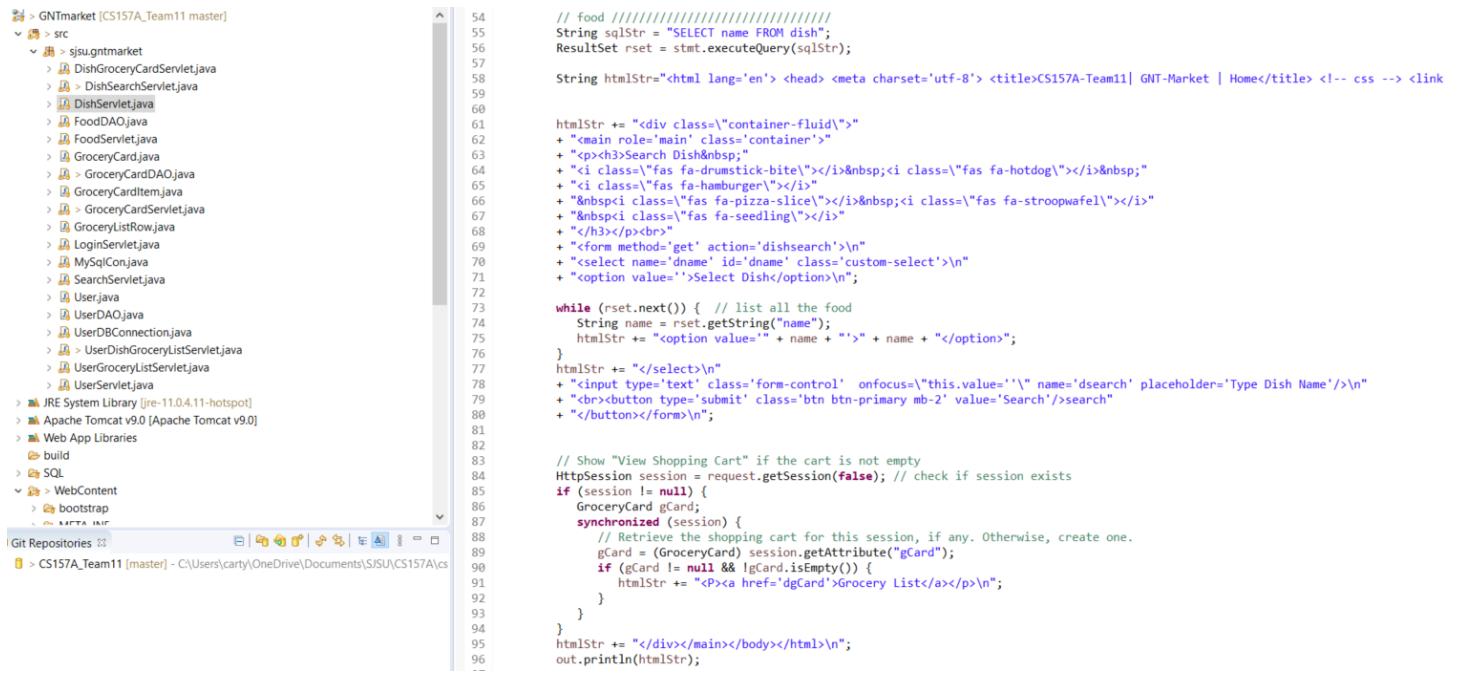
```

1 package sjsu.gntmarket;
2
3 import java.sql.Connection;
4
5 public class GroceryCardDAO {
6
7     private String driver = "com.mysql.cj.jdbc.Driver";
8     private Connection dbCon;
9
10    private String dbURL;
11    private String dbUser;
12    private String dbPassword;
13
14    public GroceryCardDAO(String dbURL, String dbUser, String dbPassword){
15        this.dbURL = dbURL;
16        this.dbUser = dbUser;
17        this.dbPassword = dbPassword;
18    }
19
20    protected void mysqlConnect() throws SQLException{
21        try{
22            Class.forName(driver);
23            dbCon = DriverManager.getConnection(dbURL, dbUser, dbPassword);
24            System.out.println("Successfully connected to database from GroceryCardDAO");
25        } catch(ClassNotFoundException e){
26            System.err.println("Couldn't load driver.");
27            System.out.println(e.getMessage());
28        } catch(SQLException e){
29            System.err.println("Couldn't connect to database.");
30            System.out.println(e.getMessage());
31        }
32    }
33
34    protected void mysqlDisconnect() throws SQLException{
35        if (dbCon != null && !dbCon.isClosed()) {
36            dbCon.close();
37        }
38    }
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96

```

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



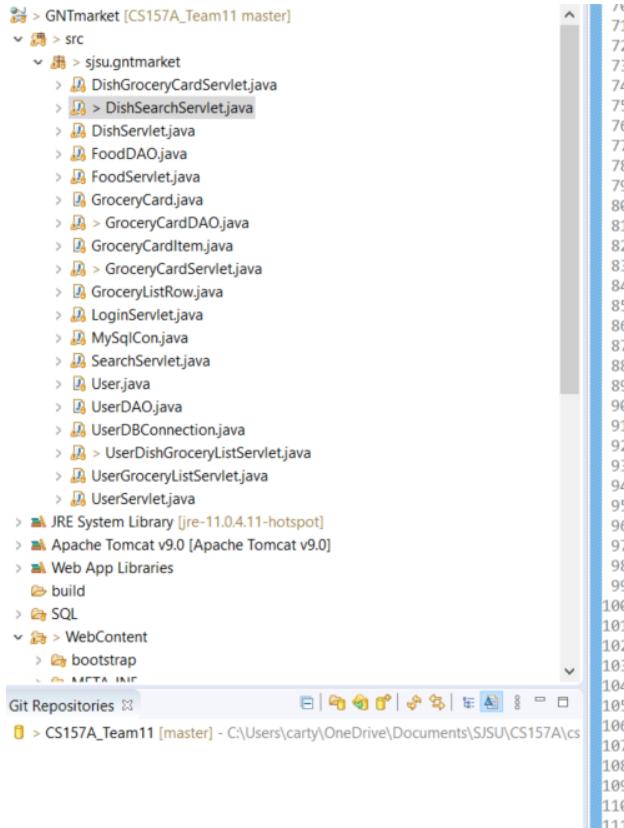
```

1 // GNTMarket [CS157A_Team11 master]
2
3 // src
4
5 // sjsu.gntmarket
6
7 // DishGroceryCardServlet.java
8 // DishSearchServlet.java
9 // DishServlet.java
10 // FoodDAO.java
11 // FoodServlet.java
12 // GroceryCard.java
13 // GroceryCardDAO.java
14 // GroceryCardItem.java
15 // GroceryCardServlet.java
16 // GroceryCardServlet.java
17 // GroceryListRow.java
18 // LoginServlet.java
19 // MySqlCon.java
20 // SearchServlet.java
21 // User.java
22 // UserDAO.java
23 // UserDBConnection.java
24 // UserDishGroceryListServlet.java
25 // UserGroceryListServlet.java
26 // UserServlet.java
27
28 // JRE System Library [jre-11.0.4.11-hotspot]
29 // Apache Tomcat v9.0 [Apache Tomcat v9.0]
30 // Web App Libraries
31 // build
32 // SQL
33 // WebContent
34 // bootstrap
35 // META-INF
36
37 Git Repositories CS157A_Team11 [master] - C:\Users\carty\OneDrive\Documents\SJSU\CS157A\cs
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96

```

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



```

71     String sqlStr = "SELECT * "
72     + "FROM Food f, dish_has_food df, dish d "
73     + "WHERE df.dish_id IN (SELECT dish_id FROM dish WHERE ";
74     if (hasDname)
75         sqlStr += " name = '" + dname + "' ";
76     if (hasDsearch) {
77         if (hasDname)
78             sqlStr += " OR ";
79     }
80     sqlStr += "name LIKE '%" + dsearchW.trim() + "%'";
81 }
82 sqlStr += "AND f.food_id=df.food_id "
83 + "AND df.dish_id=d.dish_id";
84 System.out.println(sqlStr); // for debugging
85 ResultSet rset = stmt.executeQuery(sqlStr);
86
87 if (!rset.next())
88     htmlStr += "<h3>No food found.</h3>\n"
89     + "<p><a href='food'>Back to Select Menu</a></p>\n";
90 } else {
91     // Print the result in an HTML form inside a table
92     htmlStr += "<table border='1' cellpadding='6'>\n"
93     + "<tr>\n"
94     + "<th>Food</th>\n"
95     + "<th>Add</th>\n"
96     + "<th>Delete</th>\n"
97     + "</tr>\n";
98
99     htmlStr += "<form method='get' action='dgCard'>\n";
100    do {
101        String id = rset.getString("food_id");
102        htmlStr += "<tr>\n"
103        + "<td>" + rset.getString("name") + "</td>\n"
104        + "<td><input type='checkbox' name='addF' value='" + id + "'></td>\n"
105        + "<td><input type='checkbox' name='delF' value='" + id + "'></td>\n"
106        + "</tr>\n";
107    } while (rset.next());
108    htmlStr += "</table><br/>\n"
109    + "<p><input type='submit' value='Save'></form></p>\n"
110    + "</main></div></body></html>\n";
111

```

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

```

1 package sjsu.gntmarket;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5 import java.util.List;
6
7 public class GroceryCard {
8
9     private List<GroceryCardItem> gCard;
10
11    // constructor
12    public GroceryCard() {
13        gCard = new ArrayList<GroceryCardItem>();
14    }
15
16    public void add(int id, String name) {
17        Iterator<GroceryCardItem> iter = gCard.iterator();
18        while (iter.hasNext()) {
19            GroceryCardItem item = iter.next();
20            if (item.getId() == id) {
21                // id found, do nothing
22                return;
23            }
24        }
25        // id not found, create a new CartItem
26        gCard.add(new GroceryCardItem(id, name));
27    }
28
29    // Remove
30    public void remove(int id) {
31        Iterator<GroceryCardItem> iter = gCard.iterator();
32        while (iter.hasNext()) {
33            GroceryCardItem item = iter.next();
34            if (item.getId() == id) {
35                gCard.remove(item);
36                return;
37            }
38        }
39    }
40
41    public int size() {
42        return gCard.size();
43    }
44    public boolean isEmpty() {
45        return gCard.size() == 0;
46    }
47
48    public List<GroceryCardItem> getItems() {
49        return gCard;
50    }
}

```

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

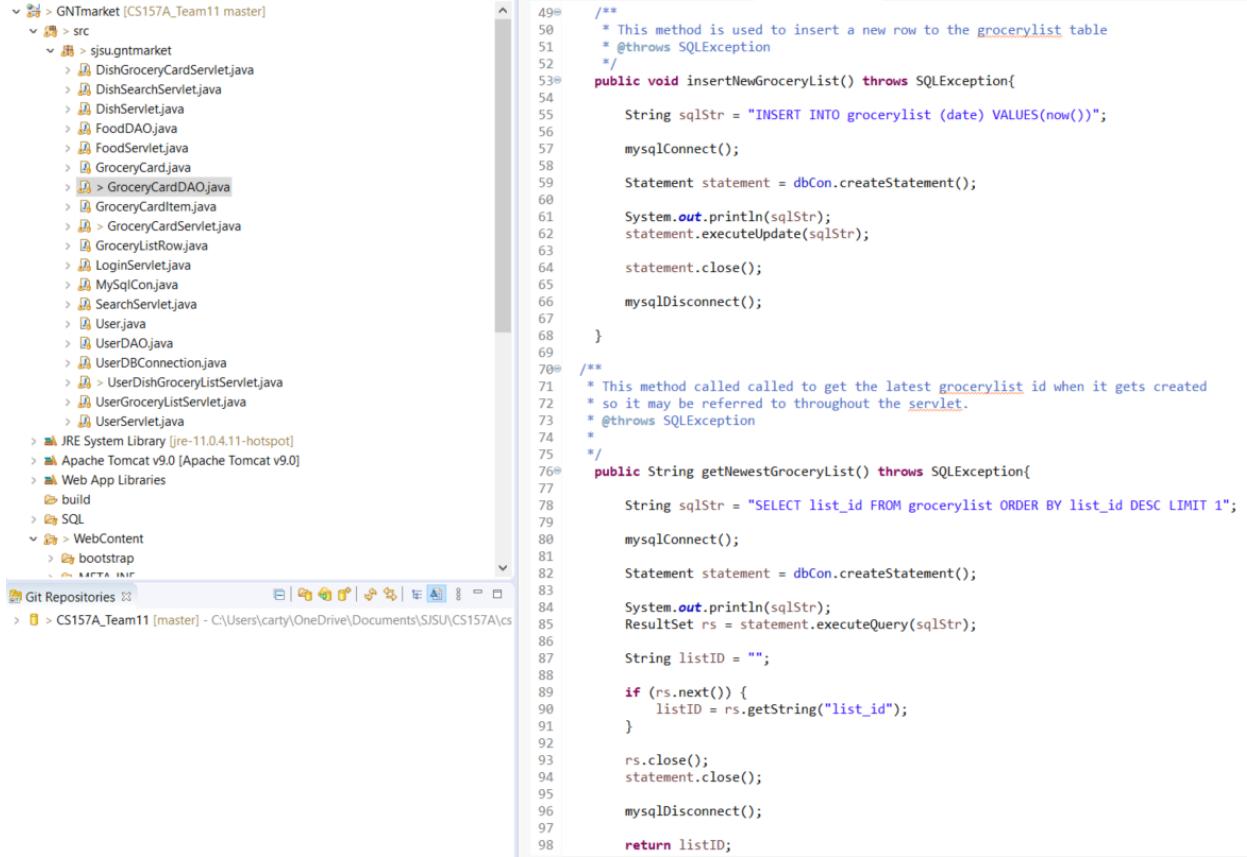
```

1 package sjsu.gntmarket;
2
3 public class GroceryCardItem {
4
5     private int id;
6     private String name;
7
8     // Constructor
9     public GroceryCardItem(int id, String name) {
10         this.id = id;
11         this.name = name;
12     }
13
14     public int getId() {
15         return id;
16     }
17
18     public String getName() {
19         return name;
20     }
21 }

```

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

GroceryCardDAO.java



```

49*   /**
50*    * This method is used to insert a new row to the grocerylist table
51*    * @throws SQLException
52*   */
53*   public void insertNewGroceryList() throws SQLException{
54*
55*       String sqlStr = "INSERT INTO grocerylist (date) VALUES(now())";
56*
57*       mysqlConnect();
58*
59*       Statement statement = dbCon.createStatement();
60*
61*       System.out.println(sqlStr);
62*       statement.executeUpdate(sqlStr);
63*
64*       statement.close();
65*
66*       mysqlDisconnect();
67*
68*   }
69*
70*   /**
71*    * This method called called to get the latest grocerylist id when it gets created
72*    * so it may be referred to throughout the servlet.
73*    * @throws SQLException
74*   *
75*   */
76*   public String getNewestGroceryList() throws SQLException{
77*
78*       String sqlStr = "SELECT list_id FROM grocerylist ORDER BY list_id DESC LIMIT 1";
79*
80*       mysqlConnect();
81*
82*       Statement statement = dbCon.createStatement();
83*
84*       System.out.println(sqlStr);
85*       ResultSet rs = statement.executeQuery(sqlStr);
86*
87*       String listID = "";
88*
89*       if (rs.next()) {
90*           listID = rs.getString("list_id");
91*       }
92*
93*       rs.close();
94*       statement.close();
95*
96*       mysqlDisconnect();
97*
98*       return listID;

```

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.

```

/**
 *
 * 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.

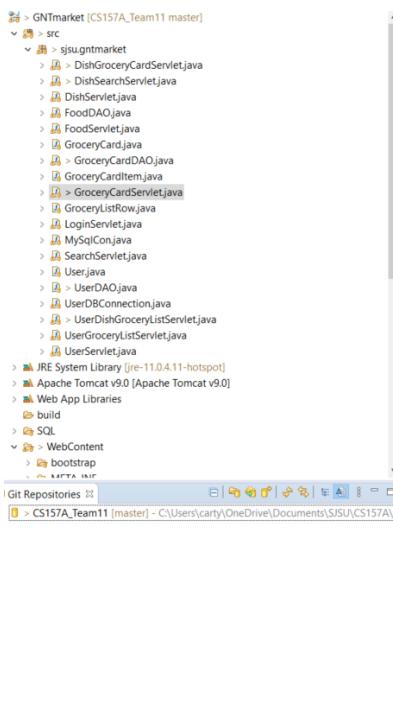
```

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

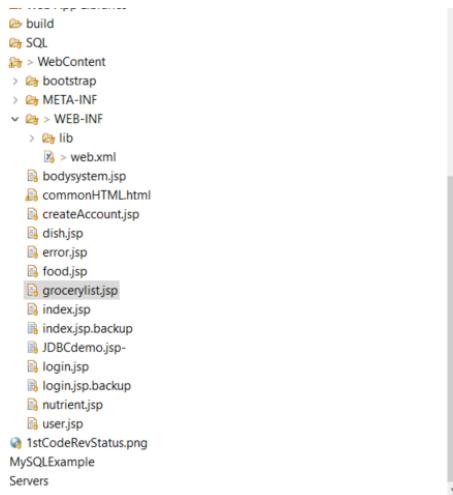
    request.setAttribute("gCard", gCard);
    request.setAttribute("groceryArrList", groceryArrList);

    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



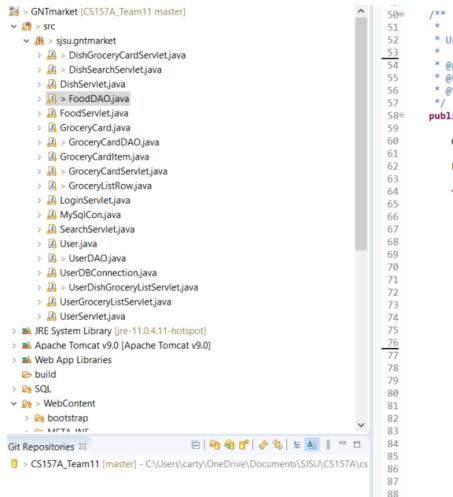
```

76=<main role="main" class="container">
77
78=    <br><c:if test="${currentUser != null}">
79=        <h3><c:out value="${currentUser.name}"></c:out>'s Grocery List
80=        <i class="fas fa-list-alt"></i>
81=    </h3>
82=
83=    </c:if>
84=
85=<table border="1" cellpadding="6">
86=    <tr>
87=        <th>Food</th>
88=        <th>Preference</th>
89=        <th>Nutrient Info</th>
90=    </tr>
91=
92=<:forEach var="food" items="${groceryArrList}">
93=    <tr>
94=        <td><c:out value="${food.name}" /></td>
95=
96=        <td><form method='get' action='gCard'>
97=            <input type='hidden' name='id' value="${food.id}" />
98=            <label style="color:blue;">
99=                <input type='radio' name='todo' value='likeF' />
100=               <font size="+2"><i class="fab fa-gratipay"></i></font></label>
101=
102=               <input type='submit' value='update' />
103=            </form></td>
104=
105=        <td><c:out value="${food.nutrientInfo}" /></td>
106=
107=    </tr>
108=  </:forEach>
109=
110= </table>
111=
112= <p><a href="#">Select More Food</a></p>
113= <br><form method='get' action='userGList'><input type='submit' value='Save' /></form>
114=
115= <:else>
116=     <:catch>
117=         <:when>
118=             </main><!-- /.container -->
119

```

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



```

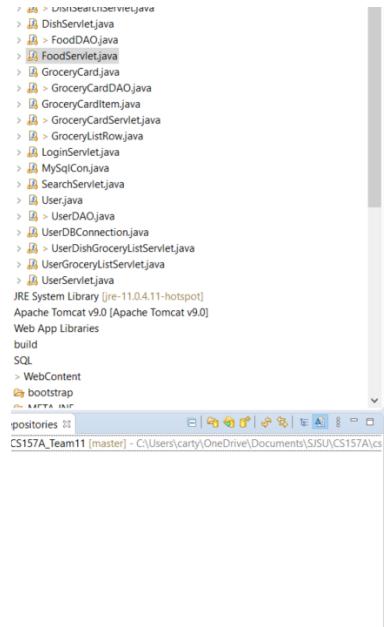
50= /**
51= * Used to get the grocery items and their respective nutrient information to be displayed on the grocery list
52= *
53= * @param gCard
54= * @return
55= * @throws SQLException
56= */
57= public List<GroceryListRow> getGroceryRows(GroceryCard gCard) throws SQLException{
58=     mysqlConnect();
59=     mysqlDisconnect();
60=     List<GroceryListRow> groceryArrList = new ArrayList<GroceryListRow>();
61=     for (GroceryCardItem item : gCard.getItems()) {
62=         int id = item.getId();
63=         String name = item.getName();
64=         String sqlStr2 = "SELECT GROUP_CONCAT(n.name SEPARATOR ', ') AS Nutrients\n";
65=         + "FROM Food_has_Nutrient f NATURAL JOIN Nutrient n\n";
66=         + "WHERE fn.food_id" + id + "\n";
67=         + "AND fn.nutrient_id=n.nutrient_id\n";
68=         + "GROUP BY fn.food_id\n";
69=         Preparedstatement stmt = dbCon.prepareStatement(sqlStr2);
70=         Resultset rset2 = stmt.executeQuery(sqlStr2);
71=         rset2.next(); // Expect only one row in ResultSet
72=         String nutrient_info = rset2.getString("Nutrients");
73=         GroceryListRow gListRow = new GroceryListRow(id, name, nutrient_info);
74=         groceryArrList.add(gListRow);
75=         rset2.close();
76=     }
77=     mysqlDisconnect();
78=     return groceryArrList;
79= }
80=
81=
82=
83=
84=
85=
86=
87=
88=
89=
90=
91=
92=
93=
94=
95=

```

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



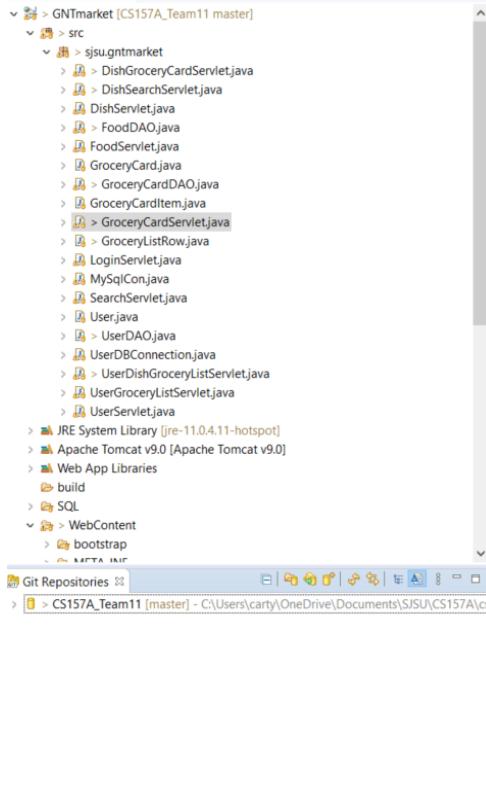
```

56 // Food ///////////////////////////////
57 String sqlStr = "SELECT name FROM Food";
58 ResultSet rset = stmt.executeQuery(sqlStr);
59
60
61 String htmlStr = "<html lang='en'> <head> <meta charset='utf-8'> <title>CS157A-Team11| GNT-Market | Home</title> <!-- css
62
63 htmlStr += "<div class='container-fluid'>
64 + <main role='main' class='container'>
65 + <h2>Search Food<br/>
66 + <i class='fas fa-fish'></i>&nbsp;
67 + <i class='fas fa-cheese'></i>&nbsp;
68 + <i class='fas fa-carrot'></i>
69 + &nbsp;<i class='fas fa-lemon'></i>&nbsp;<i class='fas fa-pepper-hot'></i>
70 + &nbsp;<i class='fas fa-egg'></i>&nbsp;<i class='fas fa-bread-slice'></i>
71 + &nbsp;<i class='fas fa-ice-cream'></i>
72 + </h2><br/>
73 + <form method='get' action='search'>\n"
74 + " <select name='fname' id='fname' class='custom-select'>\n"
75 + " <option value='>Select Food</option>\n";
76
77 while (rset.next()) { // list all the food
78     String name = rset.getString("name");
79     htmlStr += " <option value='"+ name + "'>" + name + "</option>";
80 }
81 htmlStr += "</select>\n"
82 + " <input type='text' class='form-control' onfocus='this.value='\' name='fsearch' placeholder='Type Food Name'/'>\n"
83 + " <br><button type='submit' class='btn btn-primary mb-2' value='Search'>Search</button></form>\n";
84
85
86 // Show "View Shopping Cart" if the cart is not empty
87 HttpSession session = request.getSession(false); // check if session exists
88 if (session != null) {
89     GroceryCard gCard;
90     synchronized (session) {
91         // Retrieve the shopping cart for this session, if any. Otherwise, create one.
92         gCard = (GroceryCard) session.getAttribute("gCard");
93         if (gCard != null && !gCard.isEmpty()) {
94             htmlStr += "<p><a href='gCard?todo=view'>Grocery List</a></p>\n";
95         }
96     }
97 }
98
99 htmlStr += "</main></div></body></html>\n";
out.println(htmlStr);

```

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 `getSelected` is then used as an option to be added to a grocery list.

Marking food as Like or Restriction



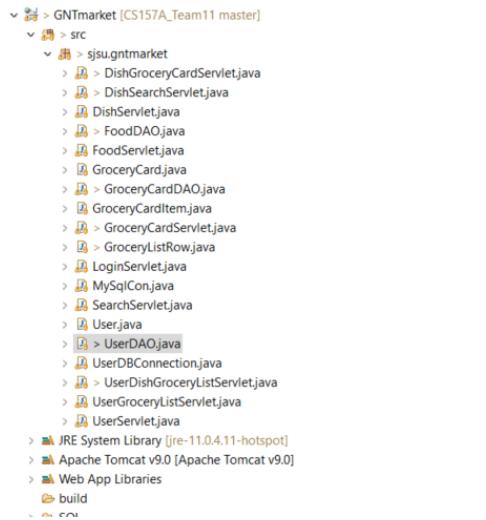
```

121    out.println("<h3>Please Select a Food!</h3></body></html>");
122    return;
123  }
124  for (String id : ids) {
125    strSql = "SELECT * FROM Food WHERE food_id = " + id;
126    System.out.println(strSql); // for debugging
127    rset = stmt.executeQuery(strSql);
128    rset.next(); // Expect only one row in ResultSet
129    String name = rset.getString("name");
130    int idInt = rset.getInt("food_id");
131    if (todo.equals("addF")) {
132      gCard.add(idInt, name);
133    }
134  }
135 } else if (todo.equals("delF")) {
136   String id = request.getParameter("id"); // Only one id for remove case
137   gCard.remove(Integer.parseInt(id));
138
139   gCardID = gCardDAO.getUserGroceryListID(currentUser.getId());
140
141   if(gCardID != 0) {
142     gCardDAO.removeFromGroceryList(gCardID, Integer.parseInt(id));
143   }
144
145 } else if (todo.equals("avoidF")) {
146   String id = request.getParameter("id"); // Only one id for remove case
147   strSql = "REPLACE INTO user_marks_food VALUES (
148     + userID + ", " // user_id
149     + id + ", " // food_id
150     + "1, " // is_restricted
151     + "0); " // is_favorite
152   System.out.println(strSql); // for debugging
153   stmt.executeUpdate(strSql);
154
155   userDao.userMarksFood(userID, id, 1, 0);
156
157 } else if (todo.equals("likeF")) {
158   String id = request.getParameter("id"); // Only one id for remove case
159   strSql = "REPLACE INTO user_marks_food VALUES (
160     + userID + ", " // user_id
161     + id + ", " // food_id
162     + "0, " // is_restricted
163     + "1); " // is_favorite
164   System.out.println(strSql); // for debugging
165   stmt.executeUpdate(strSql);
166
167   userDao.userMarksFood(userID, id, 0, 1);
168
169

```

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



```

116  /**
117   *
118   * This method is used to mark to keep track of what food items that a user has marked as a like or restriction.
119   * Based on what the user has marked as a like or restriction an update is made to the database.
120   *
121   */
122 public void userMarksFood(String userID, String foodID, int isRestricted, int isFavorite) throws SQLException{
123
124   String strSql = "UPDATE user_marks_food "
125   strSql += "SET is_restricted = ? AND is_favorite = ? "
126   strSql += "WHERE user_id = ? AND food_id = ?";
127
128   mysqlConnect();
129
130   PreparedStatement statement = dbCon.prepareStatement(strSql);
131
132   statement.setInt(1, isRestricted);
133   statement.setInt(2, isFavorite);
134   statement.setString(3, userID);
135   statement.setString(4, foodID);
136
137   System.out.println(strSql);
138
139   statement.executeUpdate();
140
141   statement.close();
142
143   mysqlDisconnect();
144
145

```

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.

```
/*
 * 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.

```
/*
 *
 * 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">
    <h3 class="h3 mb-3 font-weight-normal">Create New User</h3>
    <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

File tree:

```

GNTMarket [CS157A_Team11 master]
  +-- src
    +-- sjsu.gntmarket
      +-- DishGroceryCardServlet.java
      +-- DishSearchServlet.java
      +-- DishServlet.java
      +-- FoodDAO.java
      +-- FoodServlet.java
      +-- GroceryCard.java
      +-- GroceryCardDAO.java
      +-- GroceryCardItem.java
      +-- GroceryCardServlet.java
      +-- GroceryListRow.java
      +-- LoginServlet.java
      +-- MySqlCon.java
      +-- SearchServlet.java
      +-- User.java
      +-- UserDAO.java
      +-- UserDBConnection.java
      +-- UserDishGroceryListServlet.java
      +-- UserGroceryListServlet.java
      +-- UserServlet.java
  +-- JRE System Library [jre-11.0.4.11-hotspot]
  +-- Apache Tomcat v9.0 [Apache Tomcat v9.0]
  +-- Web App Libraries
    +-- build
    +-- SQL
  +-- WebContent

```

```

  protected void doGet(HttpServletRequest request, HttpServletResponse response)
                       throws ServletException, IOException {
    String action = request.getServletPath();
    try {
      switch (action) {
        case "/home":
          returnToHome(request, response);
          break;
        case "/new-account":
          showNewUserForm(request, response);
          break;
        case "/insert":
          insertUser(request, response);
          break;
        case "/user-list":
          listUser(request, response);
          break;
        case "/login":
          loginPage(request, response);
          break;
        case "/login-user":
          loginUser(request, response);
          break;
        default:
          System.out.println("Reached Default");
          returnToHome(request, response);
          break;
      }
    }
  }

```

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.

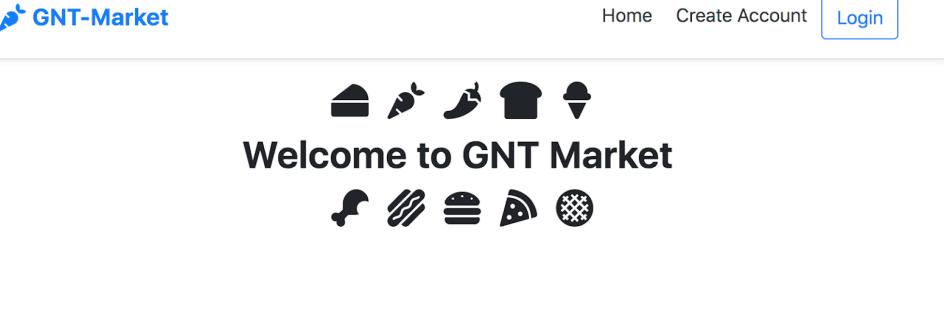
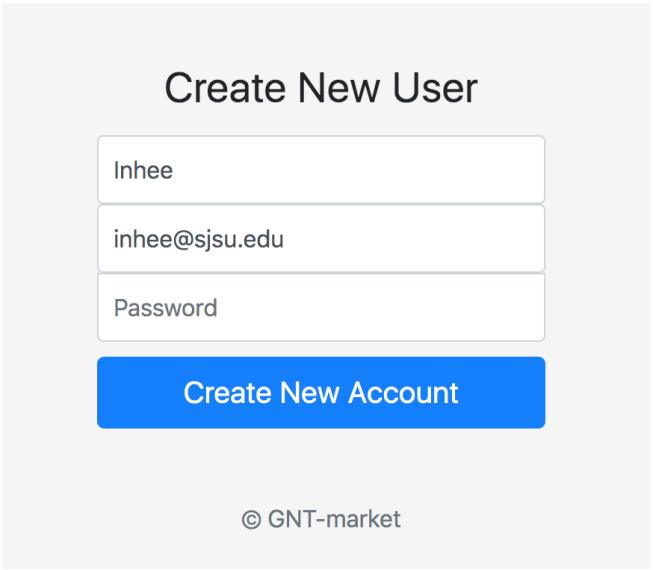
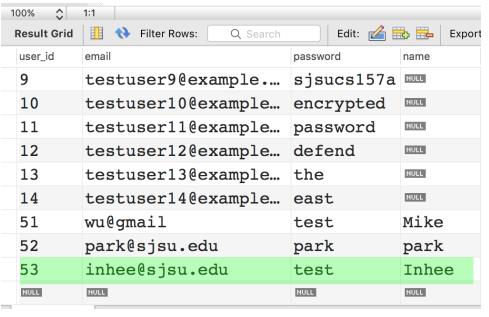
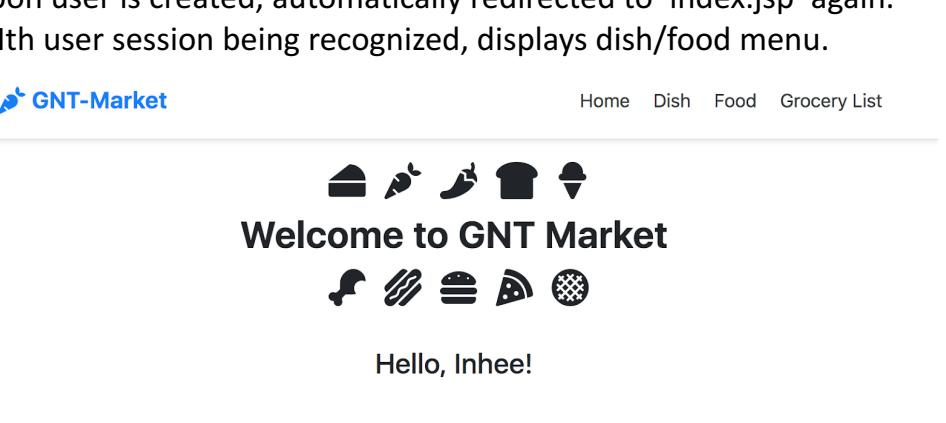
index.jsp

```

50<body>
51<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">
52<h5 class="my-0 mr-md-auto font-weight-normal">
53<a href="#" class="navbar-brand d-flex align-items-center">
54 <i class='fas fa-carrot' style='font-size:24px'></i>
55 <strong>&ampnbspGNT-Market</strong>
56 </a></h5>
57<nav class="my-2 my-md-0 mr-md-3">
58   <a class="p-2 text-dark" href="/GNTmarket/home">Home</a>
59
60  <c:if test="${currentUser != null}">
61    <a class="p-2 text-dark" href="/GNTmarket/dish">Dish</a>
62    <a class="p-2 text-dark" href="/GNTmarket/food">Food</a>
63    <a class="p-2 text-dark" href="/GNTmarket/gCard">Grocery List</a>
64  </c:if>
65
66  <c:if test="${currentUser == null}">
67    <a class="p-2 text-dark" href="/GNTmarket/new-account">Create Account</a>
68    <a class="btn btn-outline-primary" href="/GNTmarket/Login">Login</a>
69  </c:if>
70  <!-- <a class="p-2 text-dark" href="/GNTmarket/user-list">User Test Page</a> -->
71 </nav>
72 </div>
73
74
75<div align="center">
76  <h2>
77    &nbsp;<i class="fas fa-cheese"></i>
78    &nbsp;<i class="fas fa-carrot"></i>
79    &nbsp;<i class="fas fa-pepper-hot"></i>
80    &nbsp;<i class="fas fa-bread-slice"></i>
81    &nbsp;<i class="fas fa-ice-cream"></i>
82  </h2>
83  <h2><strong>Welcome to GNT Market</strong></h2>
84<h2>
85  &nbsp;<i class="fas fa-drumstick-bite"></i>
86  &nbsp;<i class="fas fa-hotdog"></i>
87  &nbsp;<i class="fas fa-hamburger"></i>
88  &nbsp;<i class="fas fa-pizza-slice"></i>
89  &nbsp;<i class="fas fa-stroopwafel"></i>
90 </h2>
91<br><c:if test="${currentUser != null}">
92  <h4>Hello, <:out value="${currentUser.name}"></:out>!</h4>
93 </c:if>
94 </div>
95 </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
<p>Main welcome font page 'index.jsp' with a minimal menu on top for non-login status. Once click "Create Account"</p> 	
<p>Redirected to 'createAccount.jsp' page to recreate an account</p> 	<p>"Users" table is updated</p>  <pre> INSERT INTO users (email, password, name) VALUES (?, ?, ?) </pre>
<p>Upon user is created, automatically redirected to 'index.jsp' again. WIth user session being recognized, displays dish/food menu.</p> 	
<p>Click the "Food"menu, the page is redirected to "FoodServlet.java".</p>	

Search Food 

Select Food

Type Food Name

Search

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

Search Food 

- Select Food
 - Asparagus
 - Broccoli
 - Carrots
 - Cauliflower
 - Celery
 - Cherries
 - Grapefruit
 - Grapes
 - Kiwis
 - Lemons / Limes
 - Bacon / Sausage
 - Beef
 - Chicken
 - Ground beef / Turkey
 - Ham / Pork
 - Cheese
 - Bread
 - Noodle
 - Tomatoes
 - Flour
 - Sugar

“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

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
```

Food Search Result

Food	Add	Delete	Nutrient Info
Broccoli			Carbohydrate, Protein, Saturated Fat

[Select More Food](#)

Food Search Result

Food	Add	Delete	Nutrient Info
Asparagus			Carbohydrate, Protein, Saturated Fat, Total Dietary Fiber, Vitamin C

[Select More Food](#)

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).

Inhee's Grocery List

Food	Preference	Nutrient Info
Broccoli	<input type="radio"/>     <input type="button" value="update"/>	Carbohydrate, Protein, Saturated Fat

[Select More Food](#)

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.

```
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
```

Result Grid			
user_id	food_id	is_restricted	is_favorite
51	5	0	1
51	17	1	0
52	6	1	0
53	27	1	0
54	2	1	0
54	3	1	0
55	2	1	0

UserID: 55

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

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


[Home](#) [Dish](#)

Inhee's Grocery List

Food	Preference	Nutrient Info
Broccoli	<input type="radio"/> <input type="radio"/> update	Carbohydrate, Protein, Saturated Fat
Cheese	<input type="radio"/> <input type="radio"/> update	Carbohydrate
Carrots	<input checked="" type="radio"/> <input type="radio"/> update	Total Dietary Fiber, Vitamin C

[Select More Food](#)
[Save](#)

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.



Final Grocery List

Food	Note
Broccoli	Restricted food
Cheese	
Carrots	Restricted food

[Return to Home](#)
[Return to GroceryList](#)

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

-	--
19	89
23	89
3	90
16	90
3	90
16	90
2	90

Food_lists_GroceryList 1

Users can generate a grocery list from the dish menu: just one dish selection would internally generate all ingredient food lists.

 GNT-Market Home Dish Food Grocery List

Search Dish

Select Dish

Type Dish Name

search

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...

 GNT-Market Home Dish Food Grocery List

Search Dish

- ✓ Select Dish
- Basic Salad
- Hamburger
- Stir-fry Veggies
- Mashed Potatoes
- Vegetable Juice
- Pasta
- Roasted Chicken
- Ham & Eggs
- Carrot Juice
- Grape Juice
- Broccoli Soup
- Cherry Pie
- Lemonade
- Steamed Asparagus
- Beef & Broccoli

SELECT name FROM dish;

dish_id	name	description
1	Basic Salad	Assorted vegetables tossed
2	Hamburger	Basic American Burger
3	Stir-fry Veggies	Assorted Grilled Vegetables
4	Mashed Potatoes	Potatoes 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	Broccoli 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

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.



Food Search Result

Food	Add	Delete
Ground beef / Turkey	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Cheese	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Bread	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Save

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”



Grocery List Preference Check

Food	Preference
Ground beef / Turkey	<input type="radio"/> <input type="radio"/> <input type="button" value="update"/>
Cheese	<input type="radio"/> <input type="radio"/> <input type="button" value="update"/>

[Select More Dish](#)

Save

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
```

 <p>Home Dish Food Grocery List</p> <h3>Search Dish</h3>  <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> Select Dish <input type="text" value="Ham"/> </div> <div style="background-color: #007bff; color: white; padding: 5px; border-radius: 5px; margin-top: 10px;"> search </div> Grocery List	<pre>name LIKE '%Ham%' AND f.food_id=df.food_id AND d.dish_id=df.dish_id)</pre>																					
<p>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.</p>  <h3>Food Search Result</h3>  <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 5px;">Food</th> <th style="padding: 5px;">Add</th> <th style="padding: 5px;">Delete</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">Ground beef / Turkey</td> <td style="padding: 5px;"><input checked="" type="checkbox"/></td> <td style="padding: 5px;"><input type="checkbox"/></td> </tr> <tr> <td style="padding: 5px;">Ham / Pork</td> <td style="padding: 5px;"><input checked="" type="checkbox"/></td> <td style="padding: 5px;"><input type="checkbox"/></td> </tr> <tr> <td style="padding: 5px;">Cheese</td> <td style="padding: 5px;"><input checked="" type="checkbox"/></td> <td style="padding: 5px;"><input type="checkbox"/></td> </tr> <tr> <td style="padding: 5px;">Bread</td> <td style="padding: 5px;"><input checked="" type="checkbox"/></td> <td style="padding: 5px;"><input type="checkbox"/></td> </tr> <tr> <td style="padding: 5px;">Canola Oil</td> <td style="padding: 5px;"><input checked="" type="checkbox"/></td> <td style="padding: 5px;"><input type="checkbox"/></td> </tr> <tr> <td style="padding: 5px;">Eggs</td> <td style="padding: 5px;"><input checked="" type="checkbox"/></td> <td style="padding: 5px;"><input type="checkbox"/></td> </tr> </tbody> </table> <div style="background-color: #007bff; color: white; padding: 5px; border-radius: 5px; margin-top: 10px;"> Save </div>	Food	Add	Delete	Ground beef / Turkey	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Ham / Pork	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Cheese	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Bread	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Canola Oil	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Eggs	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Food	Add	Delete																				
Ground beef / Turkey	<input checked="" type="checkbox"/>	<input type="checkbox"/>																				
Ham / Pork	<input checked="" type="checkbox"/>	<input type="checkbox"/>																				
Cheese	<input checked="" type="checkbox"/>	<input type="checkbox"/>																				
Bread	<input checked="" type="checkbox"/>	<input type="checkbox"/>																				
Canola Oil	<input checked="" type="checkbox"/>	<input type="checkbox"/>																				
Eggs	<input checked="" type="checkbox"/>	<input type="checkbox"/>																				
<p>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.</p>																						

Grocery List Preference Check ✓

Food	Preference
Ground beef / Turkey	<input type="radio"/>  <input type="radio"/> 
Cheese	<input type="radio"/>  <input type="radio"/> 
Ham / Pork	<input type="radio"/>  <input type="radio"/> 
Bread	<input type="radio"/>  <input type="radio"/> 
Canola Oil	<input type="radio"/>  <input type="radio"/> 
Eggs	<input type="radio"/>  <input type="radio"/> 

[Select More Dish](#)

[Save](#)

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.

Final Grocery List 🛒

Food	Note
Ground beef / Turkey	
Cheese	
Ham / Pork	
Bread	
Canola Oil	
Eggs	Restricted food 

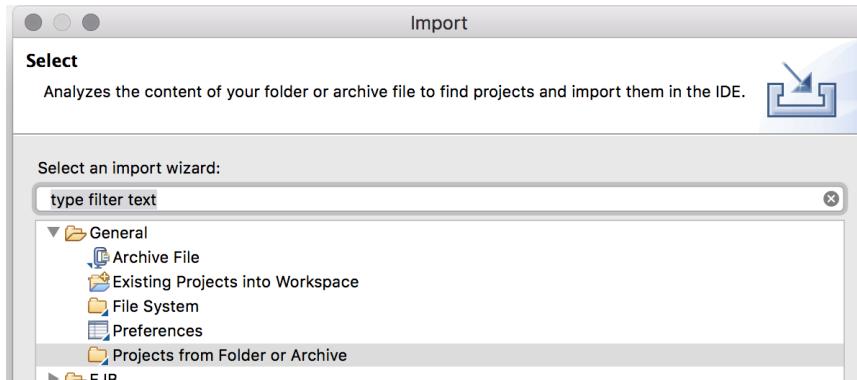
[Return to Home](#)

[Return to GroceryList](#)

Procedures 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. Import Team11/GNTmarket/SQL/gntmarket.sql to your SQL Workbench.

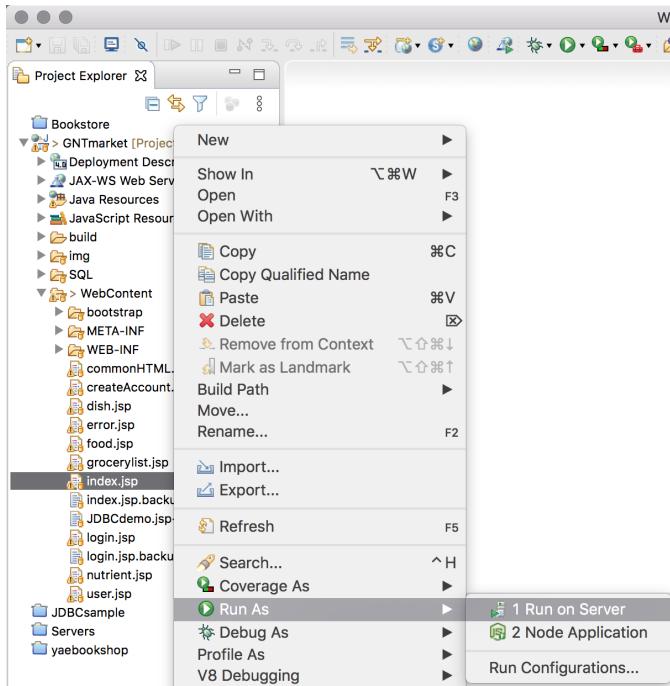
Make sure to change your SQL Workbench credentials by editing WebContent/WEB-INF/web.xml.

```
<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.