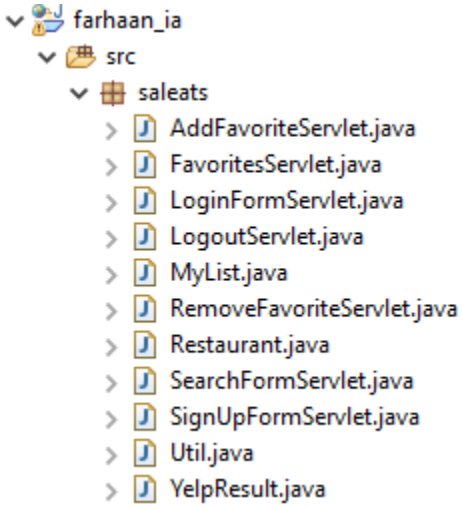
Criteria C: Development

Classes:



Techniques used:

- Connecting certain threads of the java program to a mySQL database.
- A linked list was implemented for viewing restaurants on the front end- *myList*.
- Global methods and variables were used
- Use of encapsulation
- Specified data in files were searched
- Use of inheritance
- Data was deleted from a sequential file without reading the complete file into RAM
- Data was inserted into a sequential file without reading the complete file into RAM

Restaurant class:

This class is where the Restaurant class is defined. This restaurant class is instantiated in all of the other pages when it is called upon. When any Restaurant object is created, it will include the name, coordinate, categories and all these other properties that are shown below. Other classes can call the restaurant class and its properties.

```
public class Restaurant {
    public Restaurant(String restaurant, String address,
            String img_url, String url, String phone,
            String cuisine, String price, double rating) {
        this.name = restaurant;
        this.address= address;
        this.image_url= img_url;
        this.new_url= url;
        this.phone= phone;
        this.price= price;
        this.rating = rating;
        this.cuisine = cuisine;
    }

    public String name;
    public Coordinate coordinates;
    public class Coordinate {
        public double latitude;
        public double longitude;
    }
    public Category [] categories;
    public class Category {
        public String alias;
        public String title;
    }

        public int review_count;
        public String alias;
        public String price;
        public String phone;
        public String url;
        public String image_url;
        public double rating;
        public double distance;
        public Location location;
        public class Location {
            public String city;
            public String address1;
            public String state;
            public String zip_code;
        }
        transient public String address;
        transient public String new_url;
        transient public String cuisine;

        void format() {
            this.address = location.address1+" "+location.city+" "+location.state+" "+location.zip_code;
            this.new_url = "https://www.yelp.com/biz/"+alias;
            this.cuisine = categories[0].title;
        }
    }
```

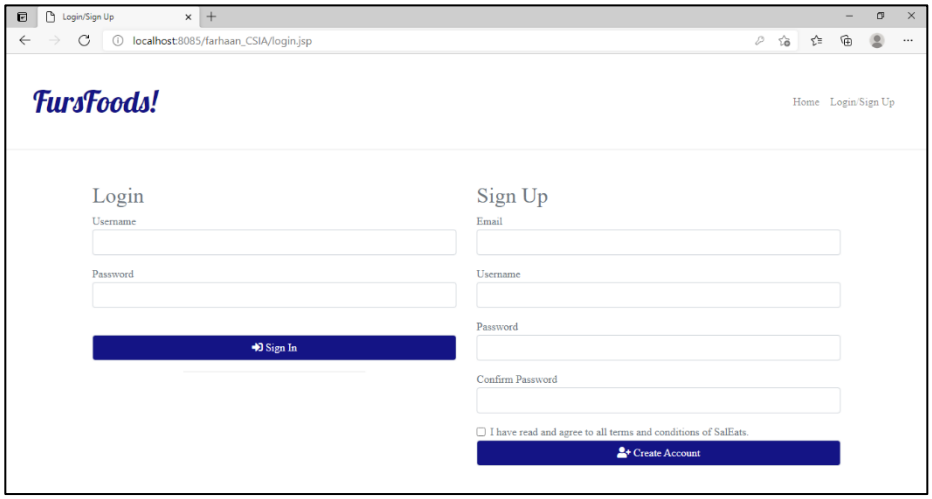
This is the constructor of the class which initializes the properties of the class instance.

The Coordinate and Category are encapsulated classes. coordinates and categories are properties of the *Restaurant class*.

These are properties of the class as well. The Location is an encapsulated class as well.

Login/Sign-Up Page:

This page is where the Login, Logout and Sign-up mechanisms are performed. Throughout this section, various classes and methods are involved to allow for its functionalities.



Sign-Up

The code below allows for users to sign up their account. This is a part of the *SignUpFormServlet class* that is executed on actions in the above UI.

```
public class SignUpFormServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String next = "/login.jsp";
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String email = request.getParameter("email");

        Util.dbconnect();
        if(Util.getUserID(username) != -1) {
            System.out.println(Util.getUserID(username));
            request.setAttribute("exists_error", "User already exists");
        }else if(Util.getEmail(email) != null) {
            request.setAttribute("email_exists_error", "Email already exists");
        } else {
            Util.addUser(username, password, email);
            HttpSession session = request.getSession();
            session.setAttribute("username", username);
            next = "/index.jsp";
        }

        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(next);
        dispatcher.forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Throughout this *SignUpFormServlet class*, there were various methods utilized. To understand this sign-up mechanism, the following methods must be understood. These methods were initialized in the *Util class* and then called in the *SignUpFormServlet class*. The methods are as follows:

addUser()

The *addUser()* method adds a new user to the local database upon input. The parameters of this method include the username, password, and email.

```
public static void addUser(String username, String password, String email) {
    try {
        PreparedStatement ps = conn.prepareStatement("INSERT INTO User (username,passkey,email) VALUES (?,?,?)");
        ps.setString(1, username);
        ps.setString(2, password);
        ps.setString(3, email);
        ps.execute();
        ps.close();
    } catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
    }
}
```

getEmail()

This *getEmail()* method gets the email id from the database.

```
public static String getEmail(String email) {
    try {
        PreparedStatement ps = conn.prepareStatement("SELECT email FROM User WHERE email=?");
        ps.setString(1, email);
        ResultSet rs = ps.executeQuery();
        if(rs.next()) {
            String e = rs.getString("email");
            ps.close();
            rs.close();
            return e;
        }
        ps.close();
        rs.close();
    } catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
    }
    return null;
}
```

dbconnect()

The *dbconnect()* method allows for the code to connect to a local mySQL database. This database contains user profile information, and user favorites information.

```
public static void dbconnect() {
    if(conn != null) return;
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        conn = DriverManager.getConnection(CREDENTIALS_STRING);
    } catch (ClassNotFoundException | SQLException e) {
        e.printStackTrace();
    }
}
```

getUserID()

The *getUserID()* method gets the Username from the local mySQL database.

```
public static int getUserID(String username) {
    try {
        PreparedStatement ps = conn.prepareStatement("SELECT userID FROM User WHERE username=?");
        ps.setString(1, username);
        ResultSet rs = ps.executeQuery();
        if(rs.next()) {
            int userID = rs.getInt("userID");
            ps.close();
            rs.close();
            return userID;
        }
        ps.close();
        rs.close();
    } catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
    }
    return -1;
}
```

getPassword() The *getPassword()* method gets the password from the local mySQL database.	

```
public static String getPassword(int userID) {
    try {
        PreparedStatement ps = conn.prepareStatement("SELECT passkey FROM User WHERE userID=?");
        ps.setString(1, Integer.toString(userID));
        ResultSet rs = ps.executeQuery();
        if(rs.next()) {
            String password = rs.getString("passkey");
            ps.close();
            rs.close();
            return password;
        }
        ps.close();
        rs.close();
    } catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
    }
    return null;
}
```

With these methods, the program is able to add user inputted information to the mySQL database while checking for previously inputted information. With an if/else statement, once the user enters the username, password, and email information in the UI, the code checks whether the information is new, or if it already exists in the mySQL database. This is done by connecting to the database with *dbconnect(),* then verifying the information retrieved from the database with *getEmail()*, *getUserID(),* and *getPassword().* After which, if the information is not previously entered, the code adds a new user to the database using the *addUser()* method.

```
Util.dbconnect();
if(Util.getUserID(username) != -1) {
    System.out.println(Util.getUserID(username));
    request.setAttribute("exists_error", "User already exists");
}else if(Util.getEmail(email) != null) {
    request.setAttribute("email_exists_error", "Email already exists");
} else {
    Util.addUser(username, password, email);
    HttpSession session = request.getSession();
    session.setAttribute("username", username);
    next = "/index.jsp";
}
```

Login

The code below allows for users to login to their account. This is a part of the *LoginFormServlet class* that is executed on actions in the above UI.

```
public class LoginFormServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String next = "/login.jsp";
        String username = request.getParameter("loguser");
        String password = request.getParameter("logpass");

        Util.dbconnect();
        int id = Util.getUserID(username);
        if(id != -1) {
            if(Util.getPassword(id).equals(password)) {
                HttpSession session = request.getSession();
                session.setAttribute("username", username);
                next = "/index.jsp";
            } else {
                request.setAttribute("logpass_error", "Password is incorrect");
            }
        }else {
            request.setAttribute("loguser_error", "User doesn't exist");
        }

        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(next);
        dispatcher.forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Throughout this *LoginFormServlet class*, there were various methods utilized. To understand this login mechanism, the following methods must be known. These methods were initialized in the *Util class* and then called in the *LoginFormServlet class*. The methods below have been previously explained. These methods are as follows:

1) dbconnect() [previously explained]
2) getPassword() [previously explained]
3) getUserID() [previously explained]

With these methods, the program is able to verify the stored user information in the mySQL database and the user input. The *dbconnect()* connects the program to the database, so that the *getUserID()* and *getPassword()* can get the information from this database. With this, using an if/else statement, the login information is verified with user input in this class. This allows users to login.

```
if(id != -1) {
    if(Util.getPassword(id).equals(password)) {
        HttpSession session = request.getSession();
        session.setAttribute("username", username);
        next = "/index.jsp";
    } else {
        request.setAttribute("logpass_error", "Password is incorrect");
    }
}else {
    request.setAttribute("loguser_error", "User doesn't exist");
}
```

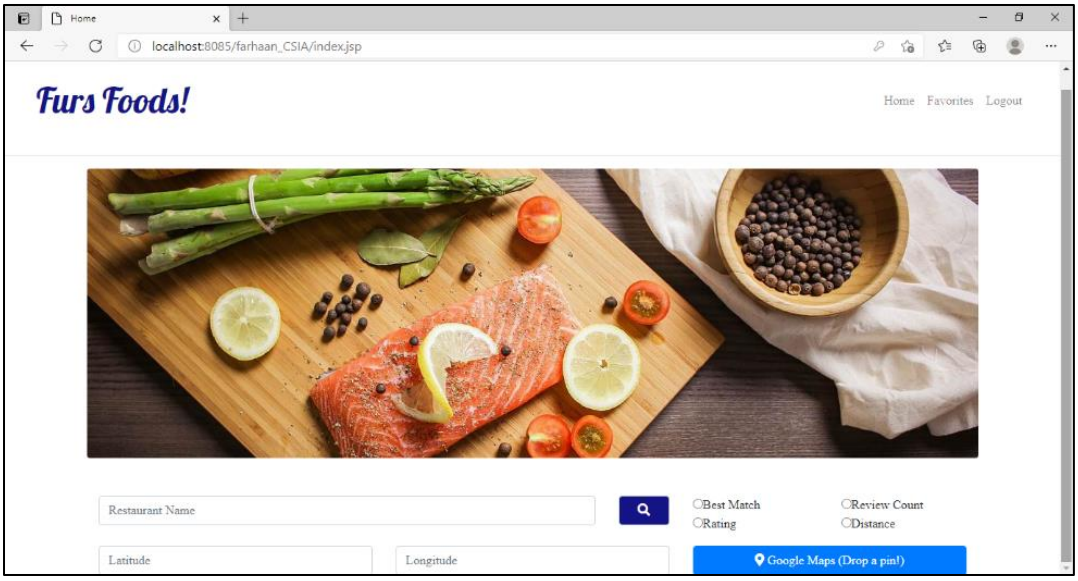
Logout

This part of the code is from the *LogoutFormServlet class*. To logout, the program is invalidating the HTTP session. This allows for the product to logout from the users account.

```
public class LogoutServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        request.getSession().invalidate();
        response.sendRedirect("./index.jsp");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

Home Page:

The home page includes most of the functionality of this product. This page includes the search functionality, the sorting options, and the map shortcut.



Sorting options

As seen below, this is the HTML layout for the sorting radio buttons. This is a snippet from the details.HTML page of the above UI.

```
        <div class="col"><input type="radio" name="order" value="best_match">Best Match</div>
        <div class="col"><input type="radio" name="order" value="review_count">Review Count</div>
    </div>
    <div class="row">
        <div class="col"><input type="radio" name="order" value="rating">Rating</div>
        <div class="col"><input type="radio" name="order" value="distance">Distance</div>
```

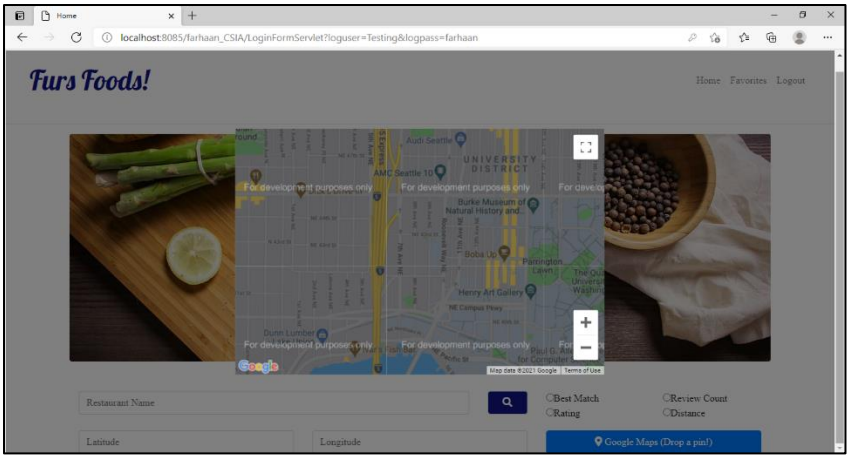
The logic behind these buttons is as explained:

Whenever someone presses the search button, the *doGet()* method from the *SearchFormServlet class* will get executed. As shown below, the program will get the parameters (like longitude, latitude, or order) that were inputted by the user on this page.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String next = "/search_result.jsp";
    String restaurant = request.getParameter("restaurant");
    String latitude = request.getParameter("latitude");
    String longitude = request.getParameter("longitude");
    String order = request.getParameter("order");
    if(order == null || order.isEmpty()) order ="best_match";

    ArrayList<Restaurant> restaurants = getRestaurants(restaurant, latitude, longitude, order);
    request.setAttribute("search_results", restaurants);

    RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(next);
    dispatcher.forward(request, response);
}
```

Map Pop-up Page

This page is another functionality under the main page. It displays the map for the user to pin-point their location to find nearby restaurants.



As shown below, this is part of the program from the *display.jsp* file, wherein, this function displays the map pop-up and captures the latitude and longitude information's from the users click and stores it on the home page. This information is then used when searching for restaurants in the area.

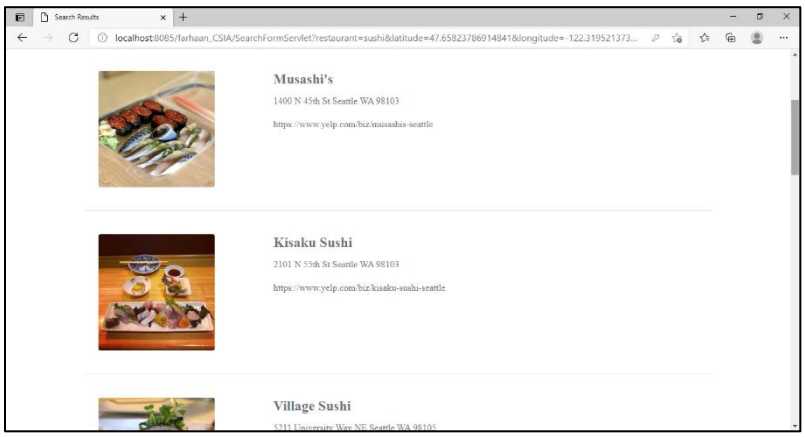
```
//Display Map
function showMap() {
    document.getElementById("overlay").style.display = "block";
    var latLng = {lat: 47.658657, lng:-122.319457}; //{lat: 34.02116, lng: -118.287132}; //
    var map = new google.maps.Map(document.getElementById('map'), {
        center: latLng,
        zoom: 15
    });

    map.addListener('click', function(e) {
        document.searchform.latitude.value = e.latLng.lat();
        document.searchform.longitude.value = e.latLng.lng();
        document.getElementById("overlay").style.display = "none";
    });
}
```

Search mechanism

Based on the user input, the *getRestaurants()* method calls the Yelp API to get the list of restaurants meeting the user input criteria (latitude, longitude, name, order, etc.). The list of restaurants that are returned are displayed on the search page.

```
private static ArrayList<Restaurant> getRestaurants(String name, String latitude, String longitude, String order) {
    //Get restaurant info using Yelp API
    YelpResult result = null;
    try {
        //Establish connection to Yelp API
        URL url = new URL("https://api.yelp.com/v3/businesses/search?term="+name.replaceAll(" ", "%20")
        +"&latitude="+latitude+"&longitude="+longitude+"&limit=10&sort_by="+order);
        HttpURLConnection yelpcon = (HttpURLConnection) url.openConnection();
        String API_KEY = "FUuIYFGeWlR6Ol9sYQER7vNMUi2V6Zm2sQC4Wopdx8btlrYZJcx"
                + "__bhPmF5E_UtXXLu17d7hK6W-wRkjyy-JLbqBkEDmIJOwEntGjSUH7Uqg5VDOqZFDTQfRz0R-XnYx";
        //Send GET Response
        yelpcon.setRequestMethod("GET");
        yelpcon.addRequestProperty("Authorization", "Bearer "+ API_KEY);
        //Parse JSON into restaurant directory
        BufferedReader br = new BufferedReader(new InputStreamReader(yelpcon.getInputStream()));
        Gson gson = new Gson();
        result = gson.fromJson(br, YelpResult.class);
    } catch (IOException ioe) {
        System.out.println("ioe in getschedule: " + ioe.getMessage());
    }catch(JsonSyntaxException jse) {
        System.out.println("jse in getschedule: " + jse.getMessage());
    }
    return result.format();
}
```

This is the search page that displays the returned list of restaurants:



This search page displayed above is from the *search_result.jsp* file. As shown below, this is the HTML behind this page:

```
<!-- Search Results -->
<div class="container mt-4">
    <div class="row mb-3">
        <h2 class="text-secondary">Results for "<%= restaurant %>"</h2>
    </div>
    <hr/>
    <%
    ArrayList<Restaurant> data = (ArrayList<Restaurant>)request.getAttribute("search_results");
    MyList restaurants = new MyList(data);
    if(restaurants.isEmpty()){
    %>
    <div class="container text-secondary">No Results were found!</div>
    <%
    } else {
        int len = restaurants.size();
        for(int i = 0; i < len; i++){
            Restaurant r = restaurants.get(i);
    %>
    <div class="row text-secondary">
        <div class="col-3 m-4">
            <a href="./details.jsp?name=<%=r.name%>&address=<%=r.address%>&url=<%=r.new_url%>&img_url=<%=r.image_url%>
            &phone=<%=r.phone%>&cuisine=<%=r.cuisine%>&rating=<%=r.rating%>&price=<%=r.price%>">
                <img src="<%= r.image_url%>" alt="<%=r.name%>" class="img-fluid rounded img">
            </a>
        </div>
```

The results from the *getRestaurants()* method are move to the implemention of the linked list, *MyList*. The code above basically iterates through the restaurant list and populates the HTML display of the page. If the restaurant list is empty, it displays, “No Results were found!”.

As seen here, when a random string of characters are entered into the search bar, the page will display “No Results were found!”.

Implementation of the linked list (myList)

Here is the implementation of this linked list class with the appropriate properties and methods:

```
public class MyList {
    Node head;

    static class Node {
        Restaurant data;
        Node next;
        Node(Restaurant r) {
            data = r;
            next = null;
        }
    }

    public MyList() {
        this.head = null;
    }

    public MyList(ArrayList<Restaurant> data) {
        this.head = null;
        for(Restaurant r : data) {
            this.add(r);
        }
    }

   public void add(Restaurant data) {
        Node node = new Node(data);
        node.next = null;

        if (this.head == null) this.head = node;
        else {
            Node trav = this.head;
            while (trav.next != null) trav = trav.next;
            trav.next = node;
        }
   }

   public boolean isEmpty() {
        return this.head == null ? true : false;
   }

   public int size() {
        Node trav = this.head;
        int len = 0;
        while(trav != null) {
            trav = trav.next;
            len++;
        }
        return len;
   }

     public Restaurant get(int i) {
         Node trav = this.head;
         while(trav != null && i > 0) {
             trav = trav.next;
             i--;
         }
         return trav == null ? null : trav.data;
     }

 }
```

Node is an encapsulated class with *data* and *next* as properties. There is a constructor to initialize the properties of this class instance.

These are two constructors for this linked list*, myList*.

This *add()* method adds new elements to the linked list

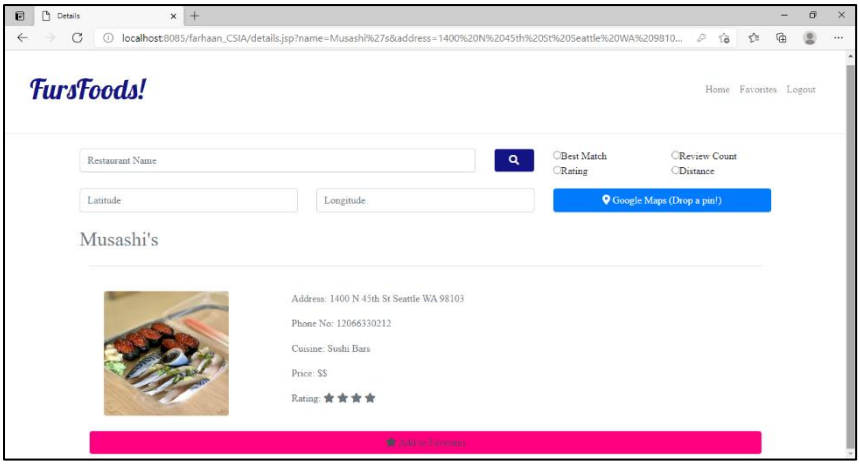
This *isEmpty()* method checks if the list is empty

This *size()* method returns the length of the list.

This *get()* method returns an element from the list

Restaurant Display Page:

After searching for a restaurant, when a user interacts with a specific restaurant, the following page is displayed. The restaurant display page shows the restaurant details for a paticular restaurant and allows the user to remove or add restuarants to their Favorites list.



Firstly, as shown below, the HTML layout displays the restaurant details such as, the address, phone number, cuisine, etc. This is a code snippet from the *details.jsp* HTML page of the above UI.

```
<div class="container mt-4">
    <div class="row">
        <h2 class="text-secondary"><%=name%></h2>
    </div>
    <hr/>
    <div class="row text-secondary">
        <div class="col-3 m-4">
            <a href="<%=url%>"><img src="<%=img_url%>" alt="<%=name%>" class="img-fluid rounded img"></a>
        </div>
        <div class="col-8 mt-4">
            <p class="card-text">Address: <%=address%></p>
            <p class="card-text">Phone No: <%=phone%></p>
            <p class="card-text">Cuisine: <%=cuisine%></p>
            <p class="card-text">Price: <%=price%></p>
            <p class="card-text">Rating:
            <%for(int i = 0; i < Math.floor(Double.parseDouble(rating)); i++) { %>
                <i class="fa fa-star"></i>
            <%}
            if((Double.parseDouble(rating) - Math.floor(Double.parseDouble(rating))) !=0){
            %>
                <i class="fa fa-star-half"></i>
            <%} %>
            </p>
        </div>
    </div>
```

The second functionality of this page is the adding and removing of favorites.

Add Favorites

Whenever someone presses the “Add to Favorites” button, the *doGet()* method from the *AddFavoriteServlet class* will get executed. As shown below, the program will get the parameters (like address, URL, username, phone, etc.) to be displayed on this page.

```
public class AddFavoriteServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        HttpSession session = request.getSession();
        String username = (String)session.getAttribute("username");
        String restaurant = request.getParameter("name");
        String address = request.getParameter("address");
        String img_url = request.getParameter("img_url");
        String url = request.getParameter("url");
        String phone = request.getParameter("phone");
        String cuisine = request.getParameter("cuisine");
        String price = request.getParameter("price");
        double rating = Double.parseDouble(request.getParameter("rating"));

        Util.addRestaurant(username, restaurant, address, img_url, url, phone, cuisine, price, rating);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

Remove Favourite

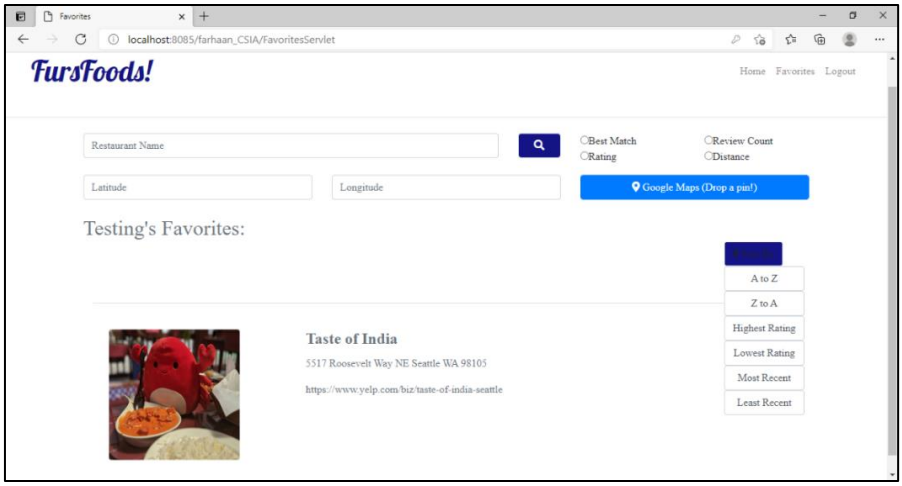
Similar to the add favorite mechanism, the doGet() method from the *RemoveFavoriteServlet* class will get executed. As shown below, the restaurant will be removed from the user's favorite list based on the restaurant name and the username.

```
public class RemoveFavoriteServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String restaurant = request.getParameter("name");
        HttpSession session = request.getSession();
        String username = (String)session.getAttribute("username");

        Util.removeRestaurant(username, restaurant);

        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/FavoritesServlet");
        dispatcher.forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Hence, the functionalities and mechanisms of the restaurant display page were shown.

Favorites Page:

After adding restaurants to your favorites, the restaurants will be displayed on the favorites page. Throughout this section, various classes and methods are involved to allow for its functionality.



This program below is a part of the *FavoritesServlet class* that is executed to allow this favorite mechanism to function.

```
public class FavoritesServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        HttpSession session = request.getSession();
        String username = (String)session.getAttribute("username");
        String opt = request.getParameter("option");
        if(username == null) {
            response.sendRedirect("./index.jsp");
            return;
        }

        Util.dbconnect();
        if(opt == null) {
            ArrayList<Restaurant> favorites = Util.getRestaurants(username);
            session.setAttribute("favorites", favorites);

            RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/favorites.jsp");
            dispatcher.forward(request, response);
        } else {
            int option = Integer.parseInt(opt);
            ArrayList<Restaurant> favorites = Util.getRestaurants(username, option);
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            String code = "";
            if(favorites.isEmpty()){
                code += "<div class=\"container text-secondary\">No Results were found!</div>";
            }else {
                for(Restaurant r: favorites){
                    code += "<div class=\"row text-secondary\"><div class=\"col-3 m-4\">";
                    code += "<a href=\"./details.jsp?name="+r.name+"&address="+r.address+
                            "&url="+r.new_url+"&img_url="+r.image_url+"&phone="+r.phone+
                            "&cuisine="+r.cuisine+"&rating="+r.rating+"&price="+r.price+"\">";
                    code += "<img src=\""+r.image_url+"\" alt=\""+r.name+"\" class=\"img-fluid rounded img\"></a></div>";
                    code += "<div class=\"col-8 mt-4\"><h4 class=\"card-title\"><strong>"+ r.name +"</strong></h4>";
                    code += "<p class=\"card-text\">"+r.address +"</p><p class=\"card-text\">"+r.new_url +"</p></div></div><hr/>";
                }
            }
            out.write(code);
            out.flush();
        }
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

Throughout this *FavoritesServlet class*, there were various methods utilized. To perform this favorite mechanism, the following methods must be understood. These methods were initialized in the *Util class* and then called in the *FavoritesServlet class*. The methods are as follows:

addRestaurant () This *addRestaurant()* method adds a new restaurant and all its properties to the local mySQL database.	

```
public static void addRestaurant(String username, String restaurant, String address,
        String img_url, String url, String phone, String cuisine, String price, double rating) {
    try {
        int userID = getUserID(username);
        if(userID == -1) return;
        PreparedStatement ps = conn.prepareStatement("INSERT INTO Restaurant(userID,restaurant,address,"
                + "img_url,url,phone,cuisine,price,rating) VALUES (?,?,?,?,?,?,?,?,?)");
        ps.setString(1, Integer.toString(userID));
        ps.setString(2, restaurant);
        ps.setString(3, address);
        ps.setString(4, img_url);
        ps.setString(5, url);
        ps.setString(6, phone);
        ps.setString(7, cuisine);
        ps.setString(8, price);
        ps.setString(9, Double.toString(rating));
        ps.execute();
        ps.close();
    } catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
    }
}
```

removeRestaurant ()

This *removeRestaurant ()* method removes the user's restaurant from the local mySQL database.

```
public static void removeRestaurant(String username, String restaurant) {
    try {
        int id = Util.getUserID(username);
        PreparedStatement ps = conn.prepareStatement("DELETE FROM Restaurant"
                + " WHERE userID=? AND restaurant=?");
        ps.setString(1, Integer.toString(id));
        ps.setString(2, restaurant);
        ps.execute();
        ps.close();
    } catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
    }
}
```

getRestaurants()

This *getRestaurants()* method gets the user's *favorites* list from the local mySQL database and returns the *favorites* list as an array.

```
public static ArrayList<Restaurant> getRestaurants(String username){
    try {
        int id = Util.getUserID(username);
        PreparedStatement ps = conn.prepareStatement("SELECT * FROM Restaurant"
                + " WHERE userID=? ORDER BY created_at DESC");
        ps.setString(1, Integer.toString(id));
        ResultSet rs = ps.executeQuery();
        ArrayList<Restaurant> favorites = new ArrayList<Restaurant>();
        while(rs.next()) {
            favorites.add(new Restaurant(rs.getString("restaurant"),
                    rs.getString("address"),
                    rs.getString("img_url"),
                    rs.getString("url"),
                    rs.getString("phone"),
                    rs.getString("cuisine"),
                    rs.getString("price"), rs.getDouble("rating")));
        }
        ps.close();
        rs.close();
        return favorites;
    } catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
    }
    return null;
}
```

getRestaurants(,)

This *getRestaurants()* method gets the user's *favorites* list from the local mySQL database based on the user's sorting option. The method returns the *favorites* list as an array.

```
public static ArrayList<Restaurant> getRestaurants(String username, int option) {
    try {
        int id = Util.getUserID(username);
        String order = null;
        if(option == 1) order = "restaurant ASC";
        else if(option == 2) order = "restaurant DESC";
        else if(option == 3) order = "created_at ASC";
        else if(option == 4) order = "created_at DESC";
        else if(option == 5) order = "rating ASC";
        else order = "rating DESC";
        PreparedStatement ps = conn.prepareStatement("SELECT * FROM Restaurant"
                + " WHERE userID=? ORDER BY "+ order);
        ps.setString(1, Integer.toString(id));
        ResultSet rs = ps.executeQuery();
        ArrayList<Restaurant> favorites = new ArrayList<Restaurant>();
        while(rs.next()) {
            favorites.add(new Restaurant(rs.getString("restaurant"),
                    rs.getString("address"),
                    rs.getString("img_url"), rs.getString("url"),
                    rs.getString("phone"), rs.getString("cuisine"),
                    rs.getString("price"), rs.getDouble("rating")));
        }
        ps.close();
        rs.close();
        return favorites;
    } catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
    }
    return null;
}
}
```

This method gets the users favorite list from the local database and displays it on the Favorites page. By using methods like addRestuarnt, Remove restaurants, getRestaurants(), and getRestaurants(,) , the *FavoritesServlet class* is able to utilize the local mySQL database to create the user's favorites list to be displayed on this page. Hence, we see how this mechanism functions.

Word count: 993