CPSC 304 Project Cover Page

Milestone #: **4**

Date: **11/29/2024**

Group Number: **35**

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Pearl (Jin Ju) Park	24181646	q1i7x	jinjpark@student.ubc.ca
Zayan Sheikh	23414329	m1g3t	zayans@student.ubc.ca
Daksh Mathur	45359395	w5i1d	dmathu01@student.ubc.ca

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)
In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia.

Repository Link:
https://github.students.cs.ubc.ca/CPSC304-2024W-T1/project_m1g3t_q1i7x_w5i1d

FIGMA Prototype:
https://www.figma.com/design/Wn7FzRrzTkFltW6eu2CgXn/Untitled?node-id=0-1&t=QgJ4XxDyBR0bPiJm-1

Description of Project
Our application is designed to make food at UBC-affiliated restaurants more affordable and satisfying for UBC students. By storing data about different restaurants and coupons, we enable students to buy tastier and more affordable food on campus.

Students can:
1. Create an order
2. View coupons offered by UBC-affiliated restaurants
3. Provide feedback about these restaurants

Students can place orders given that they have a valid account that is linked to a UBC student number, to ensure only UBC students can use this app. They can select the restaurant, branch (by address), coupon, food, and quantity for their order, and also specify a payment method (e.g., Cash, Credit, Debit). There is also “restaurants of the day” table shown on the order page, and anyone who has visited all of the restaurants of the day gets their name on a “leaderboard” type table, to encourage students to try new places. Finally, users can view their total payments, where they can see the total costs for every order they’ve made, summing up all of the foods with the given quantities, delivery charges, etc.

Coupons are available on a first-come, first-served basis, as their availability is determined by the number of uses remaining. Each coupon used within an order decreases its remaining uses by one, until no uses are left and the coupon is deleted from the table. In the coupon page, students are able to view the coupons that are offered from each restaurant and search for specific coupons available, project attributes of coupons that they wish to see and see good deal coupons offered by each restaurant.

Students can provide feedback on their dining experiences at relevant restaurants. They can submit new feedback, which includes their account ID, student ID, order date, branch ID, and a rating. Students can also update their existing feedback if needed. Additionally, students can view all feedback associated with their account ID. To help students identify the best restaurants, the application allows them to view the best-rated branch based on average feedback ratings.

Differences from the original

- Removed the promo_code attribute between order and coupon → thought this was unnecessary since anyone can use the provided coupons as long as they buy from that branch where the coupon was offered

SQL Queries

1. **Insert Query:**

   Purpose: To insert new feedback for a branch of a restaurant visited on a specific date
   How: Insert your Account ID, Student ID, Order Date, Branch ID and your feedback rating in the user input fields. Ensure your inputs are valid and all fields are completely filled. Once that is done, click the appropriate button to Add Feedback and the result will be added into the feedback table.
   This query requires accepting inputs, so the code for handling the input query is in the frontend feedback.js in addFeedback(event) line 41.

   Location of frontend code: feedback.js, function name: addFeedback(event), line number: 41.

   Location of backend code: file: feedbackService.js, function name: submitFeedback(accountId, sid, order_date, branchId, rating), line number: 29

2. **Update Query:**

   Purpose: To update a previous feedback for a branch of a restaurant visited on a specific date after a new experience
   How: Enter your Account ID, Student ID, Order Date, Branch ID and your NEW feedback rating in the user input fields. Ensure your inputs are valid and all fields are completely filled. Once that is done, click the appropriate button to Submit Feedback and the result will be updated in the feedback table.
   This query requires accepting and updating inputs, so the code for handling the input query is in the frontend feedback.js in updateFeedback(event) line 65.

   Location of frontend code: feedback.js, function name: updateFeedback(event), line number: 65.

   Location of backend code: file: feedbackService.js, function name: updateFeedback(accountId, sid, order_date, branchId, newRating), line number: 47

3. **Delete Query:**

   Purpose: To delete a past feedback for a branch of a restaurant visited on a specific date
   How: Enter your Account ID, Student ID, Order Date and Branch ID in the user input fields. Ensure your inputs are valid and all fields are completely filled. Once that is done,

click the appropriate button to Delete the Feedback and the result will be deleted from the feedback table.
This query requires accepting and deleting inputs, so the code for handling the input query is in the frontend feedback.js in deleteFeedback(event) line 160.

Location of frontend code: feedback.js, function name: deleteFeedback(event), line number: 160.

Location of backend code: file: feedbackService.js, function name: deleteFeedback(accountId, sid, order_date, branchId), line number: 103

4. **Selection Query:**
Purpose: To select specific coupons to view based on AND, OR or = operations
How: insert the type of
This query requires processing of input, so the code for handling the input/creating the query is in the frontend coupon.js in checkSelectCoupon() line 46

Location of frontend code (where query parsing and sanitization takes place): coupon.js, function name: checkSelectCoupon(), line number: 46

Location of backend code: file: couponService.js, function name: fetchSelectedCoupons(query), line number: 39

5. **Projection Query:**

Purpose: To view all coupons offered by all UBC branches
How: insert the type of columns you want to see in the coupon table and submit to get those columns
This query requires processing of input, so the code for handling the input/creating the query is in the frontend coupon.js in checkProjectCoupon() line 124

Location of frontend code (where query parsing and sanitization takes place): coupon.js line number: 46

Location of backend code: file: couponService.js, function name: projectCoupons(query), line number: 68

6. **Join Query:**

Purpose: Creating a order: to get all the food info associated with the branch

How: button and drop down once the branch (via restaurant name and street_address) has been selected

Location of query code: file: ./orderService.js, function names: getRestaurants(), getCouponBranch(bid), getRestaurantBranch(res_name), line numbers: (248, 260, 279 respectively)

7. **Aggregation With Group By Query:**

Purpose: To find the total cost of each order.
How: Via a button in the order page
Query logic: The consists table will have many tuples with the same order_id for each different food in that tuple, along with a quantity of the food that was added. The food table contains the cost for each food. Therefore, we can use group by on order_id, and then sum the cost of each food under this order_id (and multiply it by the quantity of that food in the order)

Location in code: file: ./orderService.js, line number: 188

Code for Aggregation With Group By Query:

```
SELECT d.order_id,
        b.restaurant_name,
        SUM(f.cost * c.quantity)
    FROM Delivery d, Branch b, Consists_Delivery c, Food f
    WHERE d.branch_id = b.branch_id
    AND d.order_id = c.order_id
    AND c.food_name = f.food_name
    GROUP BY d.order_id, b.restaurant_name

    UNION

    SELECT p.order_id,
        b.restaurant_name,
        SUM(f.cost * c.quantity)
    FROM Pickup p, Branch b, Consists_Pickup c, Food f
    WHERE p.branch_id = b.branch_id
    AND p.order_id = c.order_id
    AND c.food_name = f.food_name
    GROUP BY p.order_id, b.restaurant_name
```

8. **Group By With Having Query:**

Purpose: to find the best coupon deals by branch where the resulting branches displayed from clicking the button reveals branches whose maximum discount percentage coupon has a discount >= 15%

How: in the coupon page where there is a button that reveals the branches with these coupons

Location of frontend code: coupon.js, function name: getGoodDealRestaurant(), line number: 204

Location of backend code: file: couponService.js, function name: retrieveGoodDealRestaurants(), line number: 88

Code for Group By With Having Query:

```
SELECT
   R.name,
   B.branch_id,
   B.street_address,
   MAX(dc_percent)
FROM
   COUPON C,
   BRANCH B,
   RESTAURANT R
WHERE
   R.name = B.restaurant_name
   AND B.branch_id = C.branch_id
GROUP BY
   B.branch_id,
   B.street_address,
   R.name
HAVING
   MAX(C.dc_percent) >= 0.15;
```

9. Nested aggregation with GROUP BY:

Purpose: to find the ID(s) of the best rated branch based on the result of a nested aggregation by GROUP BY query which finds the average ratings of all feedbacks of all branches and then finds the best rated branch based on that

How: in the feedback page, at the bottom, there is a button that reveals the ID(s) of the best rated branch

Location of frontend code: feedback.js, function name: getBestRatedBranch(), line number: 94

Location of backend code: file: feedbackService.js, function name: retrieveGoodDealRestaurants(), line number: 85

Code for Nested aggregation with GROUP BY QUERY:

```
SELECT BRANCH_ID
FROM FEEDBACK_RATING
GROUP BY BRANCH_ID
HAVING
AVG(RATING) = (SELECT MAX(AVG(RATING)) FROM FEEDBACK_RATING
GROUP BY BRANCH_ID);
```

10. Division

Purpose: To find all users who have visited ALL of the restaurants of the day, mainly as a "leaderboard" type reward for visiting all the recommended restaurants.
How: Via a button on the order page

Location in the code: file: ./orderService.js, line number: 149
Code for Division Query:

```
SELECT DISTINCT a.account_id
    FROM Account a
    WHERE NOT EXISTS (
      SELECT 1
      FROM RestaurantOTD r
      WHERE NOT EXISTS (
        SELECT d.order_id
        FROM Delivery d, Branch b
        WHERE d.branch_id = b.branch_id
        AND d.account_id = a.account_id
        AND b.restaurant_name = r.name
```

```
        UNION
        SELECT p.order_id
        FROM Pickup p, Branch b
        WHERE p.branch_id = b.branch_id
        AND p.account_id = a.account_id
        AND b.restaurant_name = r.name
    )
)
```