MealManager

Team-2
Chaithra Lakshmi Sathyanarayana
Thaijasa Badrinath Vijendranath
Vineela Velicheti

INTRODUCTION

- Interface between restaurants, dieticians and customers
- Customers:
  - People looking for meal subscriptions - breakfast, lunch and dinner
  - Option to enroll for a personalized diet plan planned by a affiliated dietician of their choosing
- Restaurants:
  - Orders details known in advance
  - Can offer food for lower rates and still profit
- Dieticians:
  - Can work at the hours of his convenience
- Admin: Registers restaurant and dieticians users
- NoSQL MongoDB is used for storing contact form data

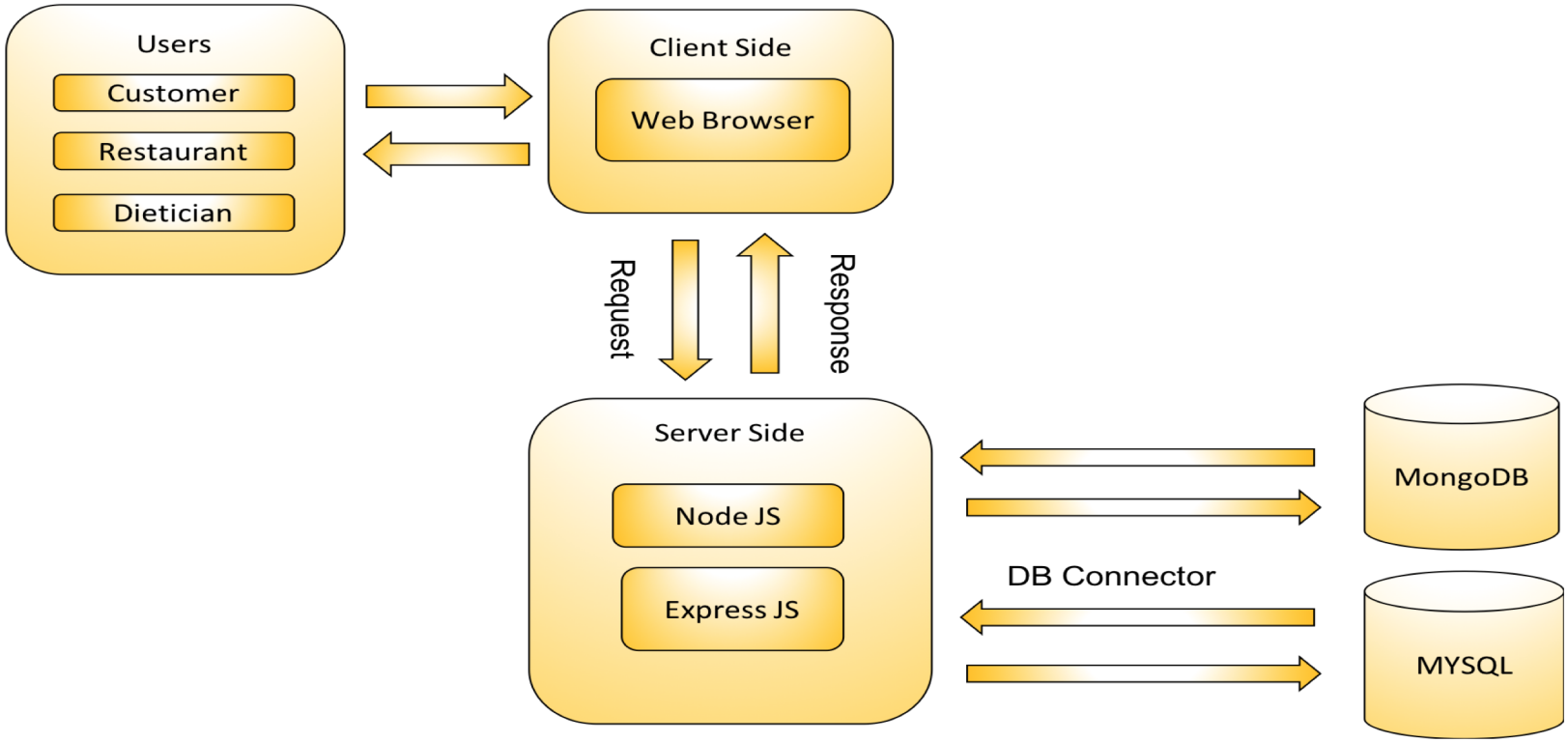
CUSTOMER

Customer without diet plan	Customer with a diet plan
• Register to the application (Enter details and make payment) • Login to the application with credentials • Select the meals based on meal plan • Place the order for selected meal and specify pick-up time • Rate items • Logout of the application	• Register to the application (Enter details and make payment) • Login to the application • Select the meals based on meal plan and diet plan • Place the order for selected meal and specify pick-up time • View and update diet plan progress • Rate items • Rate dieticians • Logout of the application

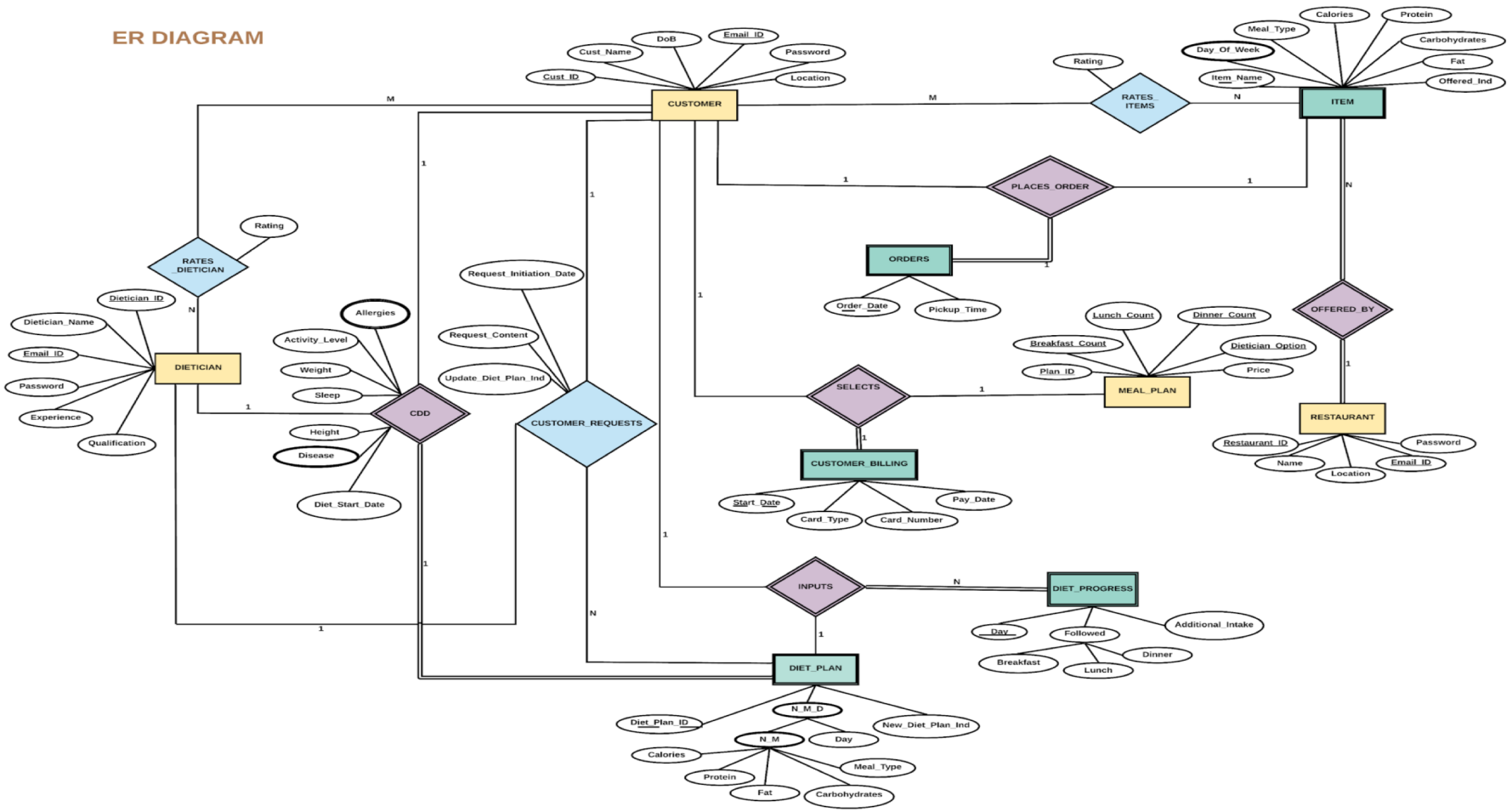
RESTAURANT & DIETICIAN

RESTAURANT	DIETICIAN
• Get registered to the application • Login in to the application with credentials • Add items with nutritional information to the menu • Receive and view item orders from customers • Update/delete items from menu • Generate sales report for a specific time period • Log out of the application	• Get registered to the application • Login in to the application with credentials • Receive diet plan requests from customer • Formulate diet plan for the requests and update them in the application in a time frame of 0 to 2 days • View enrolled customer's diet progress • Cater to change requests orders from the customer • Logout of the application

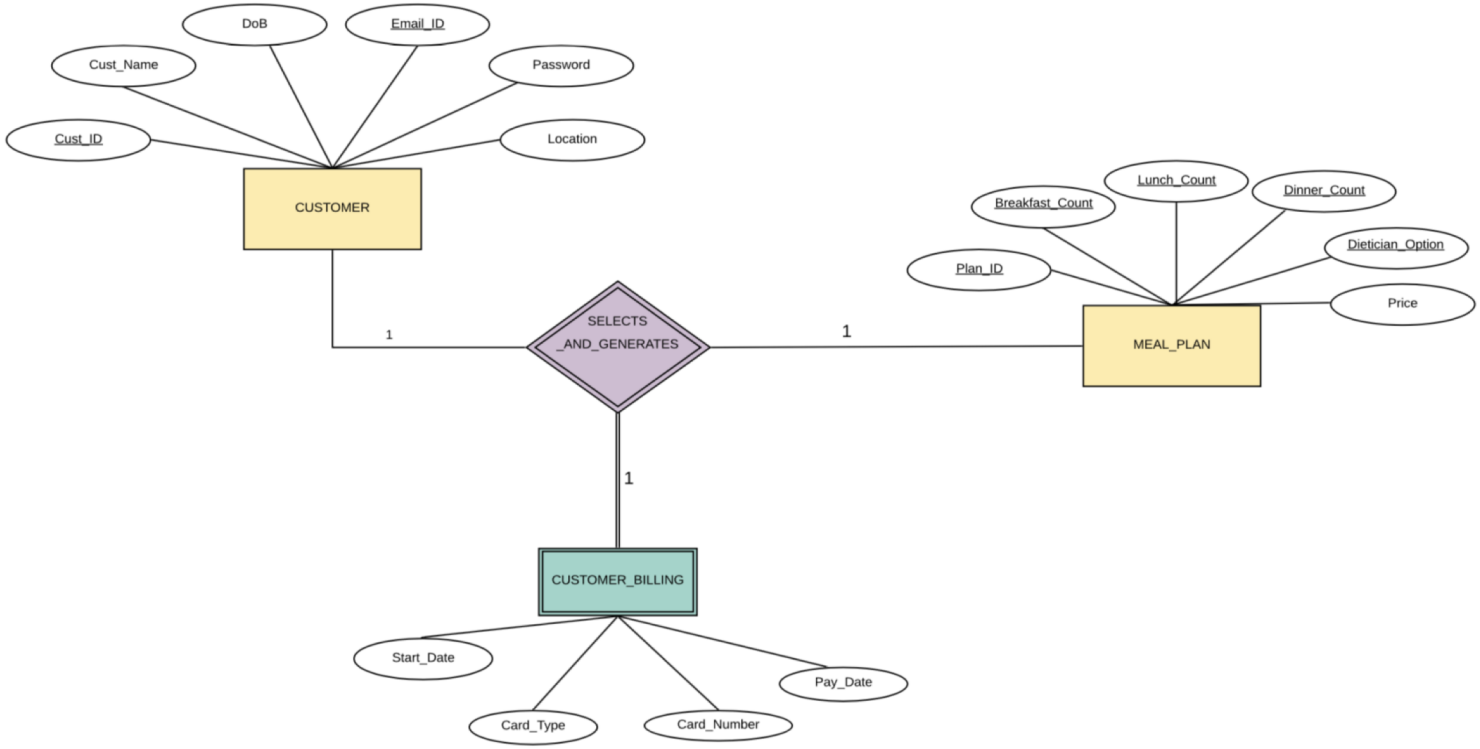
ARCHITECTURE



ER DIAGRAM

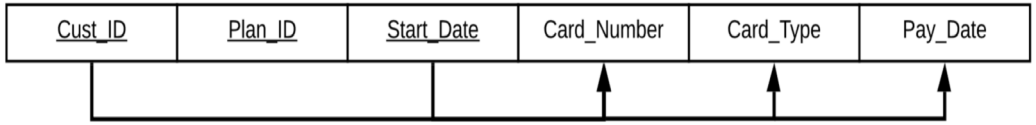

CUSTOMER
Cust_ID
Cust_Name
DoB
Email_ID
Password
Location
RATES_ITEMS
Rating
ITEM
Item_Name
Meal_Type
Day_Of_Week
Calories
Protein
Carbohydrates
Fat
Offered_Ind
PLACES_ORDER
ORDERS
Order_Date
Pickup_Time
RATES_DIETICIAN
Rating
DIETICIAN
Dietician_ID
Dietician_Name
Email_ID
Password
Experience
Qualification
CDD
Allergies
Activity_Level
Weight
Sleep
Height
Disease
Diet_Start_Date
CUSTOMER_REQUESTS
Request_Initiation_Date
Request_Content
Update_Diet_Plan_Ind
SELECTS
MEAL_PLAN
Plan_ID
Lunch_Count
Dinner_Count
Breakfast_Count
Dietician_Option
Price
OFFERED_BY
RESTAURANT
Restaurant_ID
Name
Location
Email_ID
Password
CUSTOMER_BILLING
Start_Date
Card_Type
Card_Number
Pay_Date
INPUTS
DIET_PROGRESS
Day
Followed
Breakfast
Lunch
Dinner
Additional_Intake
DIET_PLAN
Diet_Plan_ID
N_M_D
N_M
Day
Calories
Protein
Fat
Carbohydrates
Meal_Type
New_Diet_Plan_Ind

ER DIAGRAM


DoB
Email_ID
Cust_Name
Password
Cust_ID
Location
CUSTOMER
Lunch_Count
Dinner_Count
Breakfast_Count
Dietician_Option
Plan_ID
Price
SELECTS
_AND_GENERATES
MEAL_PLAN
1
1
1
CUSTOMER_BILLING
Start_Date
Pay_Date
Card_Type
Card_Number

NORMALIZATION

Customer_Billing



Functional Dependencies:

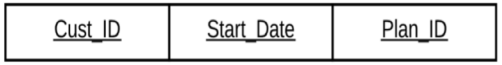
Cust_ID, Start_Date → Card_Number

Cust_ID, Start_Date → Card_Type

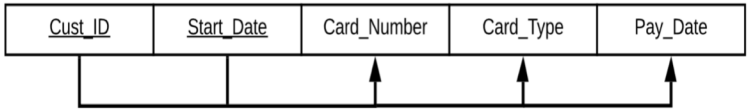
Cust_ID, Start_Date → Pay_Date

After 2NF:

Customer_Meal_Plan



Customer_Billing



DB OBJECTS

DB Object	Total Count
Table	18
Index	1
Trigger	2
Views	20
Stored Procedures	20

VIEW

```
 -- To get Customer Order History
CREATE OR REPLACE VIEW c_order_history_view AS
SELECT o.cust_id,r.name,r.restaurant_id, o.item_name, meal_type, day_of_week, order_date,pickup_time
FROM orders as o
JOIN item as i
ON i.restaurant_id = o.restaurant_id AND i.item_name = o.item_name
JOIN restaurant_item_day as rid
ON rid.restaurant_id = i.restaurant_id AND rid.item_name = i.item_name
AND rid.day_of_week = DAYNAME(ADDDATE(o.order_date,INTERVAL 1 DAY))
JOIN restaurant as r
ON r.restaurant_id = o.restaurant_id
ORDER BY order_date DESC, pickup_time DESC;
```

STORED PROCEDURE

```
375
376     DELIMITER $$
377     CREATE PROCEDURE c_update_item_rating
378     (IN p_cust_id int,IN p_restaurant_id int, IN p_item_name varchar(30), IN p_rating int, OUT v_chck int )
379     BEGIN
380     DECLARE `_rollback` BOOL DEFAULT 0;
381     DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET `_rollback` = 1;
382     START TRANSACTION;
383     SELECT COUNT(*) into v_chck
384     FROM item_ratings
385     WHERE cust_id = p_cust_id AND restaurant_id = p_restaurant_id AND item_name= p_item_name;
386     If v_chck != 1 THEN
387         INSERT INTO item_ratings values(p_cust_id,p_restaurant_id,p_item_name,p_rating);
388     ELSE
389         UPDATE item_ratings
390         SET rating = p_rating
391         WHERE cust_id = p_cust_id AND restaurant_id = p_restaurant_id AND item_name= p_item_name;
392     END IF;
393     IF `_rollback` THEN ROLLBACK;
394     ELSE COMMIT;
395     END IF;
396     END;
397     $$
398     DELIMITER ;
```

TRIGGER

Item

Restauarnt_ID	Item_Name	Meal_Type	Calories	Protein	Carbohydrates	Fat	Offered_Ind

Restaurant_Item_Day

Restauarnt_ID	Item_Name	Day_Of_Week

Purpose:

- If the restaurant wants to stop offering the item, we cannot delete the item from item table due to the necessity to maintain order history. The offered_ind helps in determining if the item is being offered on any day or not.
- If the restaurant is not offering the item currently on any day, and decides that it wants to offer the item, then records will be inserted in restaurant_item_day table.
- This trigger updates offered_ind to Y from N in item table.
- This logic needs to be handled whether the change in restaurant_item_day table happens from UI or from database. To make sure, the logic is always implemented, trigger is used instead of stored procedure.

TRIGGER

```
DELIMITER $$
CREATE TRIGGER update_offered_ind_for_delete
after delete
ON restaurant_item_day
FOR EACH ROW
BEGIN
    DECLARE v_count INT;
    SELECT count(*) into v_count from restaurant_item_day
    WHERE restaurant_id = OLD.restaurant_id AND item_name = OLD.item_name;
    if v_count>0 then
        UPDATE Item
        SET Offered_Ind = 'Y'
        WHERE restaurant_id = OLD.restaurant_id AND item_name = OLD.item_name;
    elseif v_count=0 then
        UPDATE Item
        SET Offered_Ind = 'N'
        WHERE restaurant_id = OLD.restaurant_id AND item_name = OLD.item_name;
    end if;
END;
$$
DELIMITER ;
```

NoSQL: MONGODB for Contact Form

```
> db.contactform.find().pretty();
{
        "_id" : ObjectId("5ccfb7dda87c6fcdaccf2283"),
        "name" : "mongo1@gmail.com",
        "message" : [
                [
                        "This is message1!",
                        "5/5/2019, 9:28:13 PM"
                ],
                [
                        "This is message2",
                        "5/5/2019, 9:28:31 PM"
                ],
                [
                        "Please contact me on 98789373 on sundays between 9am and 10am.",
                        "5/5/2019, 9:28:49 PM"
                ]
        ],
        "update_timestamp" : "5/5/2019, 9:28:49 PM"
}
{
        "_id" : ObjectId("5ccfb839a87c6fcdaccf2324"),
        "name" : "mongo2@gmail.com",
        "message" : [
                [
                        "Hello, \r\nI would like to discuss business opportunities. Please call me on monday.",
                        "5/5/2019, 9:29:45 PM"
                ]
        ],
        "update_timestamp" : "5/5/2019, 9:29:45 PM"
}
>
```

```
3335
3336
3337  function func_updatecontactform(request,result)
3338  {
3339     var contact_email_id = request.body.email_id;
3340     var contact_message  = request.body.contact_message;
3341    var MongoClient = require('mongodb').MongoClient;
3342    var url = "mongodb://localhost:27017/";
3343    MongoClient.connect(url, { useNewUrlParser: true }, function(err, db) {
3344      if (err) throw err;
3345      var dbo = db.db("mealmanager");
3346      var contact_message_time=new Date().toLocaleString();
3347      console.log(contact_message_time);
3348      dbo.collection("contactform").updateOne({name:contact_email_id},
3349    {  $addToSet: {message:[ contact_message,contact_message_time]}   , $set: { update_timestamp: contact_message_time} },
3350      { upsert: true }, function(err, res) {
3351        if (err) {
3352          throw err;}
3353          else{
3354            log.info('Sending success message for contact form');
3355
3356            result.render('contactform',{cust_message:"Your message has been conveyed. You can add more messages with the same contact id."});
3357
3358          }
3359        //console.log("Number of documents inserted: " + res.insertedCount);
3360        });
3361
3362    });
3363
3364
3365   }
3366  module.exports.update_contactform = function(request, result)
3367  {
3368    log.info('Recieved request to update contact form');
3369    func_updatecontactform(request,result);
3370    console.log(request.body);
3371
3372  };
```

Demo Time!

RETROSPECTIVE

What we did well?

- Implementation and integration from frontend to backend of the application
- Teamwork and communication

What we learned new?

- Understand forming complex SQL stored procedures
- Using session concepts in application and generating logs during application run

What we can improve on?

- Gathering and understanding business requirements

FUTURE WORK

- Add images for items in menu
- To increase publicity for restaurants, we can show top 3 restaurants for every month.
- Give an option for all users to upload a profile pic.

Thank You....!!!