1 Introduction and motivation

As data storage getting cheaper and cheaper in recent years, a new kind of app, labelled "Finder" or "Finding App", has been exploding in popularity. It serves as its label suggested, in finding places of interest for the users. Out of all the "Finder" apps a particular type stands out, called "Restaurant Finder" or "Dining-out Finder" or simply just "Foodie".

A "Foodie" app helps the user to find restaurants in a twofold way: finding unspecified restaurants nearby using external GPS data or finding user specified restaurants from the database. The app also provides a platform to showcase dishes the restaurants have to offer and to integrate user ratings of the restaurants.

Among all the mature "Foodie" apps, including Yelp, Foodspotting, Urbanspoon, OpenTable, we find them focusing on a higher level. For example users always search for a restaurant or a cuisine, and all the displayed ratings are for the restaurants. It struck on us that none of them have focused on what people go to the restaurants for, the food itself. Indeed you can see other customers' opinion on some dishes, but only through the comment section which is structurally abstruse and restricted. And when you search for a specific dish, you cannot judge which one of the return restaurant offers the best, simply because restaurants' ratings contain lots of other considerations and not only about this dish.

To give it a more lively illustration, we can safely say, sometimes people wouldn't mind the environments in exchange for fabulous food. Having a low rating because of minor hygienic issues or bad attitude from waiters really should separate from the rating for food; and having a high rating does not necessarily mean this restaurant does every dish right.

Apart from the above, we found that the "Foodie" apps doesn't dynamically tailor the results to the preferences of users. That saying, all the data are static and thus the same to everyone. We want to make Food-a-pedia more personalized, so that ratings would bear more meaning and weight.

We also wanted to give this app a more informative feeling, hence the name, by also focusing on a more basic level, the ingredients, and the relationship between dishes and ingredients. This way, users can see if certain dish contains some allergens or they can search for dishes containing ingredients they love. This is something "Foodie" apps don't offer.

Food-a-pedia takes on the idea of "Foodie" from a different angle and extends it to a different direction. It's a web-based database application about food. Using Food-a-pedia, users will be able to find the best restaurants offering their favorite dishes and ingredients, and also get to know more about these food.

This report will describe the solution we propose to engage problems raised in section 2. We will also dissect the section into several subsections to talk in detail about the implementation of the solution and the work we have done. In section 3 we will talk a little about UI and the performance of Food-a-pedia.

2 Solution Domain

To fully engage the problems raised above, we introduced the following functionalities.

1. In Food-a-pedia ratings are given on different dishes the restaurants offer instead of restaurants themselves. Users search for a specific dish, and he can get all the restaurants provided that dish sorted by rating.
2. To spice things up we integrated a dynamic rating system which will change the ratings of dishes according to currently logged-in user's preference. This preference is computed through how this user agrees with other users' ratings. Thus a user that has disparity in taste between him and another one will have little impact on the other's search results. Users can toggle on or off this functionality.
3. Users can search for a specific food genre and get a list of restaurants that provide food of this genre.
4. Users are allowed to search for the name of an ingredient and get a list of dishes containing this ingredient. For instance, for a query of "chicken", the result list should be "Kung Pao chicken", "General Tso's chicken", "Lemon chicken" etc.
5. Users also can search for a dish and get a list of ingredients in it. For example, for a query of "Fish Taco", the result would be "Flour", "Lettuce", "Salsa" and "Pollock".
6. We also included the generic "Foodie" function of searching for restaurants, alongside searching for what the restaurants offer.

The Food-a-pedia architecture features three parts. A front-end user interface, a data access object level and a back-end database. The database is done in MySQL. The front-end web UI is done using HTML. Data access and exchanges are processed by Java and JSP. We used JDBC for connection between Java and MySQL.

To implement the functionalities mentions, the first step is to design the database.

2.1 Database Design

Intuitively, Food-a-pedia has four entity sets, namingly Restaurant, Dish, Ingredient and Customer. Each set contains ID as their primary keys plus information about the entities. To accompany the entities we created several relationships:

- Dish and Ingredient are linked by "Cuisine" relationship. Thus we can query dishes for ingredient or vice versa.
- There is a ternary relationship "Rating" connecting Dish, Restaurant and Customer. It records every rating customers give to a dish at a restaurant.
- A relationship "Recommend" self-refers on Customer. Combine this relationship and Rating we can compute the similarity in taste and thus get the personalized ratings.

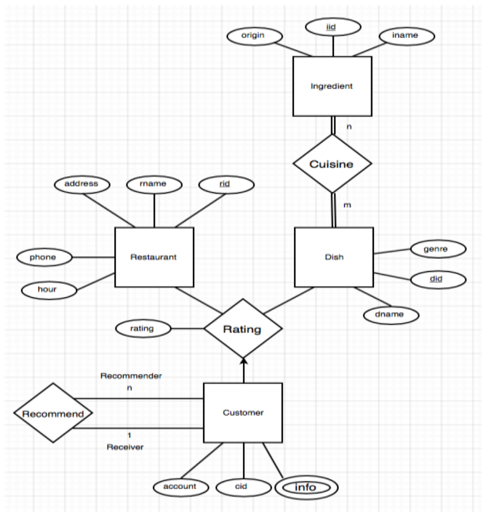


Fig.2.1.1 The corresponding E-R diagram

The Recommend relationship requires dynamically creating
and dropping tables of data. So we decided to not create a
Table for it, instead, we generate the data temporarily in Java.
Do a BCNF decomposition on the relation
R(rid, did, cid, iid, cinfo, account, rname, rinfo,
dname, genre, iname, origin, rating) with the
functional dependencies

- rid->rname, rinfo
- cid->cinfo,account
- did->dname, genre
- iid->iname, origin
- cid, did, rid->rating

In order to make the tables more compact we created Cuisine to handle the many-to-many Dish-Ingredient relationship.

Result relations are as following:

- Customer(cid, cinfo, account)
- Restaurant(rid, rname, rinfo)
- Dish(did, dname, genre)
- Ingredient(iid, iname, origin)
- Cuisine(did, iid)
- Rating(cid, rid, did, rating).

2.2 Refine the Design

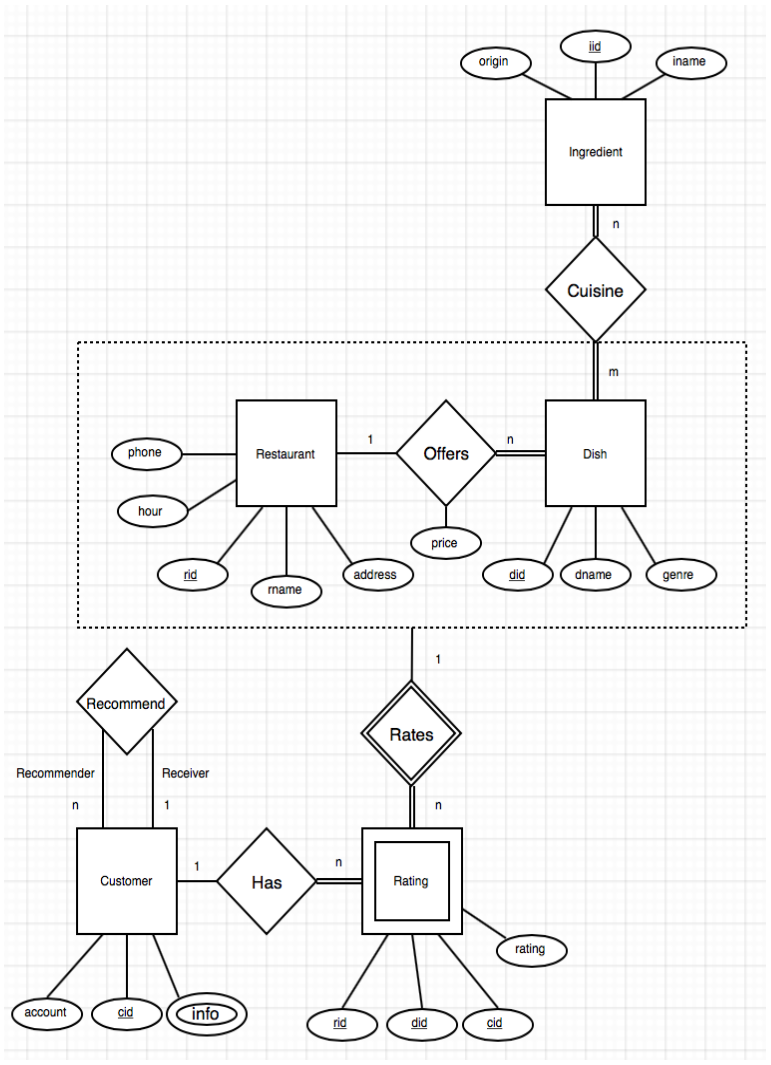
Now we have the database, it’s necessary to check the correctness and robustness it. We want to check if this design can fully implement the required functionalities and, for possible future upgrades, add additional ones.

Although already in the BCNF, the problem here is that
Problem: The ternary relationship Rating cannot independently illustrate

the binary relationship between Restaurant and Dish. A combination of
Restaurant and Dish cannot exist if there’s no customer rating it. Thus
we may not be able to query a restaurant’s full menu.

In order to solve this, we
Break down the ternary relationship into binary relationships:
. To preserve the integrity of Restaurant-Dish, we make them into an aggregation. These two entities are connected by an “Offers” relationship, thus we can add additional information like price to the combination. Each combination also is represented by an offer_id. Finally by turning Rating from a relationship to a weak entity, and link it to Customer and the aggregated entity we are able to decompose the ternary relationship.

- Customer(<u>cid,</u> cinfo, account)
- Restaurant(<u>rid</u>, rname, rinfo)
- Dish(<u>did</u>, dname, genre)
- Offer(<u>oid</u>, rid, did, price)
- Ingredient(<u>iid</u>, iname, origin)
- Cuisine(did, iid)
- Rating(<u>cid</u>, <u>oid</u>, rating).



And the corresponding database schema we created is

```
CREATE TABLE `Restaurant`
(
   `R_Id` INT NOT NULL AUTO_INCREMENT,
   `Name` NVARCHAR(80) NOT NULL,
   `Address` NVARCHAR(70) NOT NULL,
   `PhoneNo` NVARCHAR(24) NOT NULL,
   `Dilivery` NVARCHAR(20),
   `Parking` NVARCHAR(40),
                  `Wi-Fi` NVARCHAR(20),
   `Website` NVARCHAR(70),
   `MonToFri_OpenHourStart` NVARCHAR(20) NOT NULL,
   `MonToFri_OpenHourEnd` NVARCHAR(20) NOT NULL,
   `SatToSun_OpenHourStart` NVARCHAR(20) NOT NULL,
   `SatToSun_OpenHourEnd` NVARCHAR(20) NOT NULL,
    CONSTRAINT `PK_Restaurant` PRIMARY KEY  (`R_Id`)
);


CREATE TABLE `Dish`
(
   `Dish_Id` INT NOT NULL AUTO_INCREMENT,
   `Name` NVARCHAR(120),
   `Genre` NVARCHAR(40),
    CONSTRAINT `PK_Dish` PRIMARY KEY  (`Dish_Id`)
);


CREATE TABLE `Customer`
```

```
(
   `Customer_Id` INT NOT NULL AUTO_INCREMENT,
   `CustomerAccount` NVARCHAR(70) NOT NULL,
   `FirstName` NVARCHAR(40),
   `LastName` NVARCHAR(20),
   `Address` NVARCHAR(70),
   `City` NVARCHAR(40),
   `State` NVARCHAR(40),
   `Country` NVARCHAR(40),
   `PostalCode` NVARCHAR(10),
   `Phone` NVARCHAR(24),
   `Email` NVARCHAR(60) NOT NULL,
   CONSTRAINT `PK_Customer` PRIMARY KEY  (`Customer_Id`)
);
```

Fig.2.2.1 The final diagram

```
CREATE TABLE `Ingredient`
(
   `Ingredient_Id` INT NOT NULL AUTO_INCREMENT,
   `Name` NVARCHAR(40),
   `Origins` NVARCHAR(40),
    CONSTRAINT `PK_Ingredient` PRIMARY KEY  (`Ingredient_Id`)
);
CREATE TABLE `Rating`
(
   `O_Id` INT NOT NULL,
   `Customer_Id` INT NOT NULL,
   `CustomerRate` INT NOT NULL,
    CHECK (CustomerRate > 0 AND CustomerRate < 5),
    CONSTRAINT `PK_Rating` PRIMARY KEY  (`Customer_Id`,`O_Id`)
   );
CREATE TABLE `Offer`
(
   `O_Id` INT NOT NULL AUTO_INCREMENT,
   `Dish_Id` INT NOT NULL,
   `R_Id` INT NOT NULL,
   `Price` FLOAT NOT NULL,
    CONSTRAINT `PK_Offer` PRIMARY KEY  (`O_Id`)
);
CREATE TABLE `Cuisine`
(
   `Dish_Id` INT NOT NULL,
   `Ingredient_Id` INT NOT NULL,
    CONSTRAINT `PK_Cuisine` PRIMARY KEY  (`Dish_Id`,`Ingredient_Id`)
);
/*******************************************************************************
  Create Foreign Keys
********************************************************************************/
ALTER TABLE `Rating` ADD CONSTRAINT `FK_Offer_Id`
   FOREIGN KEY (`O_Id`) REFERENCES `Offer` (`O_Id`);


ALTER TABLE `Rating` ADD CONSTRAINT `FK_Customer_Id`
   FOREIGN KEY (`Customer_Id`) REFERENCES `Customer` (`Customer_Id`);


ALTER TABLE `Offer` ADD CONSTRAINT `FK_Dish_Id2`
   FOREIGN KEY (`Dish_Id`) REFERENCES `Dish` (`Dish_Id`);


ALTER TABLE `Offer` ADD CONSTRAINT `FK_R_Id2`
   FOREIGN KEY (`R_Id`) REFERENCES `Restaurant` (`R_Id`);


ALTER TABLE `Cuisine` ADD CONSTRAINT `FK_Dish_Id3`
   FOREIGN KEY (`Dish_Id`) REFERENCES `Dish` (`Dish_Id`);


ALTER TABLE `Cuisine` ADD CONSTRAINT `FK_Ingredient_Id`
   FOREIGN KEY (`Ingredient_Id`) REFERENCES `Ingredient` (`Ingredient_Id`);
```

One thing we have learned from creating the tables is that
The foreign keys are created using ALTER clause
after the creation of tables:

Because if foreign keys are included in the CREATE clause, it cannot compile due to concurrency issues. That saying, the tables are created spontaneously.

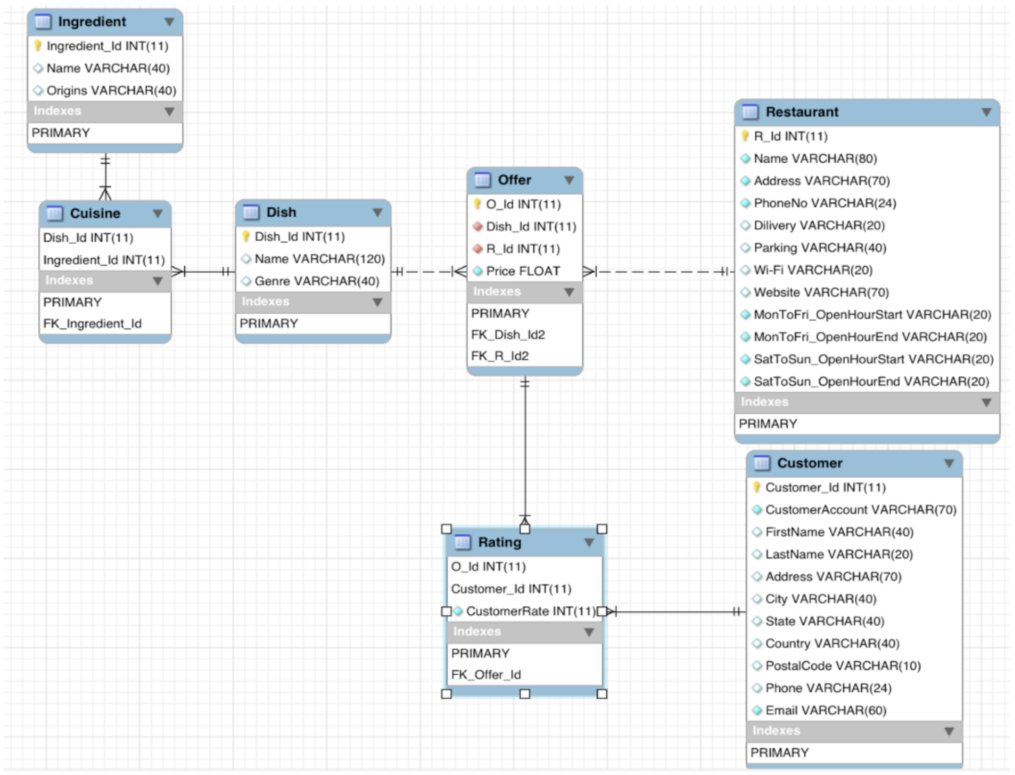


Fig.2.2.2 The schema diagram

All the attributes have self-explanatory names so I won't describe here what they represent.

2.3 Query and algorithm

Now the database is created, the next step is to query the database and fulfill proposed functionalities.

2.3.1 Search for a dish with ratings

Rating table contains all the ratings on what each restaurant has to offer. We search for the name of a dish, and get a list of restaurant with ratings from different users. Then we use GROUP BY and AVG to combine the duplicate restaurants and get the average rating of the dish at restaurant. The query is

```
SELECT re.name, AVG(ra.CustomerRate) as Rating FROM Restaurant re, Rating ra, Dish d, Offer o WHERE d.Dish_Name='input'
and d.Dish_Id=o.Dish_Id and o.R_Id=re.R_Id and ra.O_Id=o.O_Id
GROUP BY re.name
ORDER BY Rating desc;
```

2.3.2 Implementing the personalization system on 3.3.1

To give an illustration of this system, an example is given as: if user A and B both rated dishes 1 and 2 and their ratings are similar, then when A searches for a list containing dishes 3, 4, 5 and 6, and B has rated 4 and 6, these two will have more priorities than 3 and 5. On the contrary, if A and B rate dishes disparately, then B's opinion will weigh much less in A's search results.

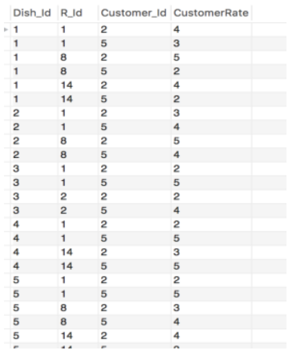
To implement this, after logging into the app, a user will receive a list with a weighted modifier applied based on his preference in comparison to others'. We will calculate the weight point between two users' ratings. The more similar the ratings are, the higher the weight point will be. Then for all the search results we apply weight points from every pair of users and calculate the average. Thus for different users, as long as they have given ratings, the search results will be personalized. Otherwise it will display unweighted results.

User A's relativity with user B is given by
:Relativity=(rA1-rB1+rA2-rB2+...+rAk-rBk)/k where rAkandrBk are ratings towards offer k from user A and B respectively.

For every other user, we compute their relativities with the currently logged-in user to get a table of relativities values. Then when the user search for a dish with ratings, we apply these values on their respective users' ratings, and get the average to calculate new, weighted ratings.
:Weighted Rating=r1*Relativity1+r2*Relativity2+...+rk*Relativityk/1kRelativityi where rkis the user k's rating of this dish.

Following this logic, we use the query below to get all the dishes that userA and userB have both rated.

```
SELECT tag.* FROM(
SELECT r.* FROM rating r WHERE r.Dish_Id in (SELECT r2.Dish_Id FROM Rating r2 WHERE
r2.Customer_Id = 'userA') and r.R_Id in (SELECT r3.R_Id FROM Rating r3 WHERE
r3.Customer_Id = 'userA') and (r.Customer_Id = 'userA' or r.Customer_Id = 'userB'))as tag
INNER JOIN
(SELECT r.Dish_Id, r.R_Id FROM rating r WHERE r.Dish_Id in (SELECT r2.Dish_Id FROM Rating r2 WHERE
r2.Customer_Id = 'userA') and r.R_Id in (SELECT r3.R_Id FROM Rating r3 WHERE
r3.Customer_Id = 'userA') and (r.Customer_Id = 'userA' or r.Customer_Id = 'userB')
GROUP BY Dish_Id, R_Id
HAVING Count(*) > 1) as gag
ON tag.Dish_Id=gag.Dish_Id and tag.R_Id=gag.R_Id;
```



Dish_Id	R_Id	Customer_Id	CustomerRate
1	1	2	4
1	1	5	3
1	8	2	5
1	8	5	2
1	14	2	4
1	14	5	2
2	1	2	3
2	1	5	4
2	8	2	5
2	8	5	4
3	1	2	2
3	1	5	5
3	2	2	2
3	2	5	4
4	1	2	2
4	1	5	5
4	14	2	3
4	14	5	5
5	1	2	2
5	1	5	5
5	8	2	3
5	8	5	4
5	14	2	4

Fig.2.3.2.1 An example result

As the result we can get the relativity between them using equation .

Then by the query:

```
SELECT re.name, ra.Customer_Id, ra.CustomerRate
FROM Restaurant re, Rating ra, Dish d
WHERE d.Name='input' and d.Dish_Id=ra.Dish_Id and ra.R_Id=re.R_Id
ORDER BY re.name;
```

we are able to get a list of restaurants providing a specific dish with all the ratings from different users. Using equation we will be able to get a table of weighted ratings of each restaurant's offer.

[Some computations and creation of tables are done in Java instead of SQL, because the relativity table is different for each user, thus it is more efficient to do it as a temp variable. See the GetRelation.Java, BuildRelation.Java and DishRating.Java for more algorithmic reference.]

2.3.3 Search for a restaurant

The table Restaurant represents the entities of restaurants. To search for a restaurant, use

```
SELECT * FROM Restaurant r, WHERE r.name = 'input';
```

2.3.4 See what a restaurant has to offer

Offer table contains all the dishes at each restaurant, and is linked to Restaurant and Dish. To see the menu of a restaurant, use

```
SELECT d.name as Menu
FROM Restaurant re, Offer o, Dish d
WHERE re.name = 'input' and d.Dish_Id=o.Dish_Id and o.R_Id=re.R_Id;
```

2.3.5 Search for restaurants providing a genre of food

Dish table contains the genre information, and is linked to Restaurant with Offer table. To search for a list providing a specific genre of food, use

```
SELECT DISTINCT re.*
FROM Restaurant re, Offer o, Dish d
WHERE d.genre = 'Fast food' and d.Dish_Id=o.Dish_Id and o.R_Id=re.R_Id;
```

2.3.6 See what's in the dish

Cuisine table contains a many-to-many relationship on Dish and Ingredient. To search for ingredients a dish has, use

```
SELECT i.name, i.Origins
FROM Dish d, Cuisine c, Ingredient i
WHERE d.name='Flavored pork' and d.Dish_Id = c.Dish_Id
        and c.Ingredient_Id = i.Ingredient_Id;
```

2.3.7 See what dish has the ingredient

Similarly, to search for dishes containing an ingredient, use

```
SELECT d.name
FROM Dish d, Cuisine c, Ingredient i
WHERE i.name='Lettuce' and d.Dish_Id = c.Dish_Id
        and c.Ingredient_Id = i.Ingredient_Id;
```

3 UI & Performance

UI is constructed using HTML. As a large portion of Food-a-pedia is focusing on the concept of different user having different result. We will have a user login page as shown in Fig.3.1.

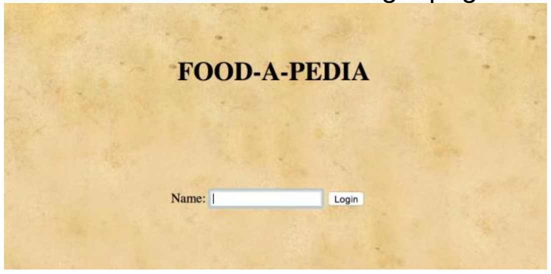


Fig.3.1 Login page

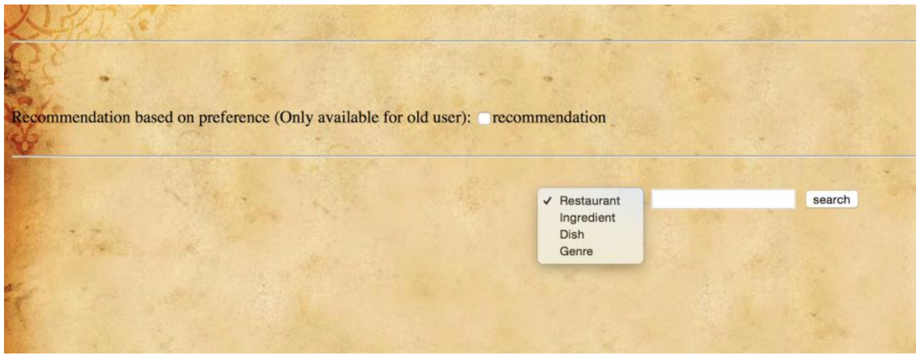
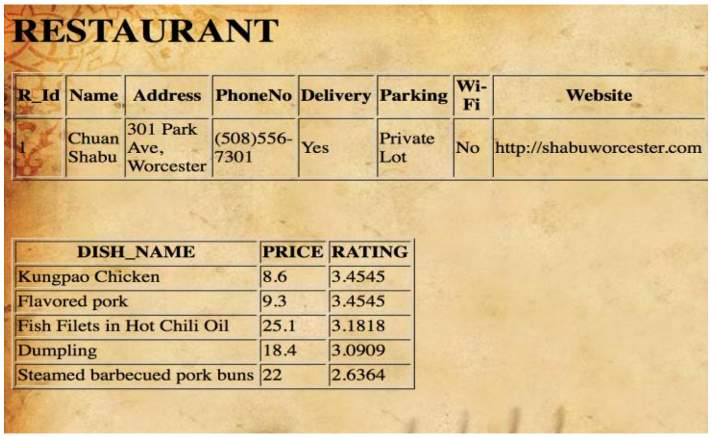

Recommendation based on preference (Only available for old user): ☐ recommendation

Fig.3.2 Search Page

Fig.3.2 is the search page. Customers can search by four different categories: Restaurant, Ingredient, Dish and Genre. Customers existing in the database can search dishes and get results based on preference of other customers instead of general average ratings.

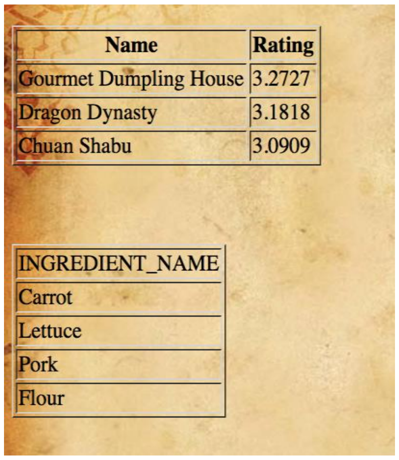
We will show the results of all the proposed functionalities below.


RESTAURANT

R_Id	Name	Address	PhoneNo	Delivery	Parking	Wi-Fi	Website
1	Chuan Shabu	301 Park Ave, Worcester	(508)556-7301	Yes	Private Lot	No	http://shabuworcester.com

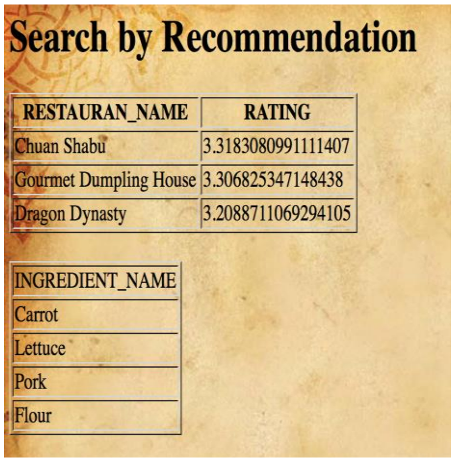
DISH_NAME	PRICE	RATING
Kungpao Chicken	8.6	3.4545
Flavored pork	9.3	3.4545
Fish Filets in Hot Chili Oil	25.1	3.1818
Dumpling	18.4	3.0909
Steamed barbecued pork buns	22	2.6364

Fig.3.3 Searching restaurant name "Chuan Shabu" returns detailed information and dish menu



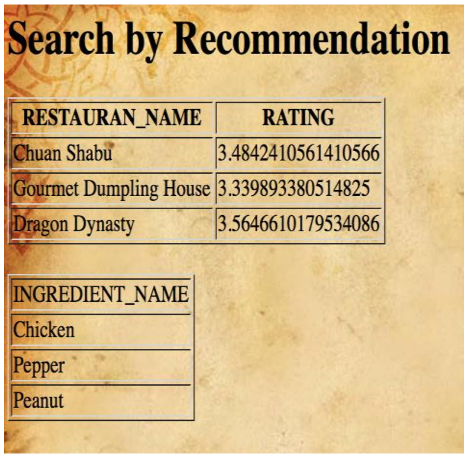
Name	Rating
Gourmet Dumpling House	3.2727
Dragon Dynasty	3.1818
Chuan Shabu	3.0909

INGREDIENT_NAME
Carrot
Lettuce
Pork
Flour


Search by Recommendation

RESTAURAN_NAME	RATING
Chuan Shabu	3.3183080991111407
Gourmet Dumpling House	3.306825347148438
Dragon Dynasty	3.2088711069294105

INGREDIENT_NAME
Carrot
Lettuce
Pork
Flour


Search by Recommendation

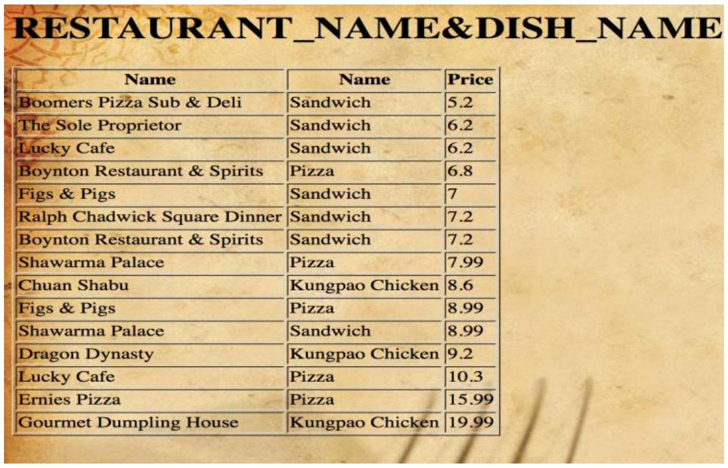
RESTAURAN_NAME	RATING
Chuan Shabu	3.4842410561410566
Gourmet Dumpling House	3.339893380514825
Dragon Dynasty	3.5646610179534086

INGREDIENT_NAME
Chicken
Pepper
Peanut

(a) (b) (c)

Fig.3.4 Searching "Kungpao Chicken" returns restaurant name and rating (a) by general average ratings (b) by accordingly weighted ratings by relativities (c) weighted ratings of another user

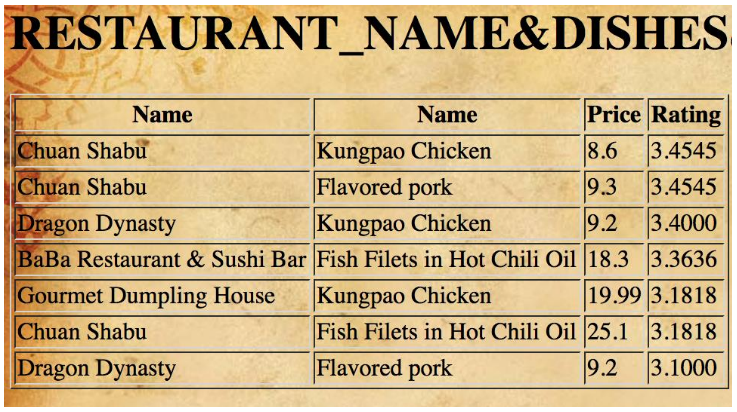
We can see from Fig.3.4, that for the current user, the ranks of restaurants by general average ratings is different from that of weighted ratings, and every user will likely have a different result ranks. Searching for a dish also gives the user a list of ingredients in this dish.



RESTAURANT_NAME&DISH_NAME

Name	Name	Price
Boomers Pizza Sub & Deli	Sandwich	5.2
The Sole Proprietor	Sandwich	6.2
Lucky Cafe	Sandwich	6.2
Boynton Restaurant & Spirits	Pizza	6.8
Figs & Pigs	Sandwich	7
Ralph Chadwick Square Dinner	Sandwich	7.2
Boynton Restaurant & Spirits	Sandwich	7.2
Shawarma Palace	Pizza	7.99
Chuan Shabu	Kungpao Chicken	8.6
Figs & Pigs	Pizza	8.99
Shawarma Palace	Sandwich	8.99
Dragon Dynasty	Kungpao Chicken	9.2
Lucky Cafe	Pizza	10.3
Ernies Pizza	Pizza	15.99
Gourmet Dumpling House	Kungpao Chicken	19.99

Fig.3.5 Searching ingredient name "Chicken" returns dishes contain this ingredient and dishes prices in different restaurants



RESTAURANT_NAME&DISHES

Name	Name	Price	Rating
Chuan Shabu	Kungpao Chicken	8.6	3.4545
Chuan Shabu	Flavored pork	9.3	3.4545
Dragon Dynasty	Kungpao Chicken	9.2	3.4000
BaBa Restaurant & Sushi Bar	Fish Filets in Hot Chili Oil	18.3	3.3636
Gourmet Dumpling House	Kungpao Chicken	19.99	3.1818
Chuan Shabu	Fish Filets in Hot Chili Oil	25.1	3.1818
Dragon Dynasty	Flavored pork	9.2	3.1000

Fig.3.6 Searching genre name "Chuan Food" returns dishes belong this genre and dishes' prices and general average ratings in different restaurants

4 Conclusion

Food-a-pedia addresses the problems that general "Foodie" apps have, by focusing on a more basic level. The outcome of the searches show that Food-a-pedia has achieved perfectly the functionalities and features that we had planned for. That we are able to implement the personalized rating. Moreover, it provides more options for customers such as covering the relationship between a dish and its ingredients.