Project 3
Final Project Report
TEAM: CS 4604-ZELT
Travel Lovers

1. **Overview**

Travel lover is a web-accessible database system application that helps users to plan, record, budget, and review their travel. Key features include that allow users to edit or update profiles; search for scenic spots by place name and filter by specific criteria; tag selected places; review search history and travel history; and support for users to write reviews of visited places. Based on the number of times the place has been searched and marked, and users' reviews of the place, we can make it a recommender system that could send related advertisements to targeted users.

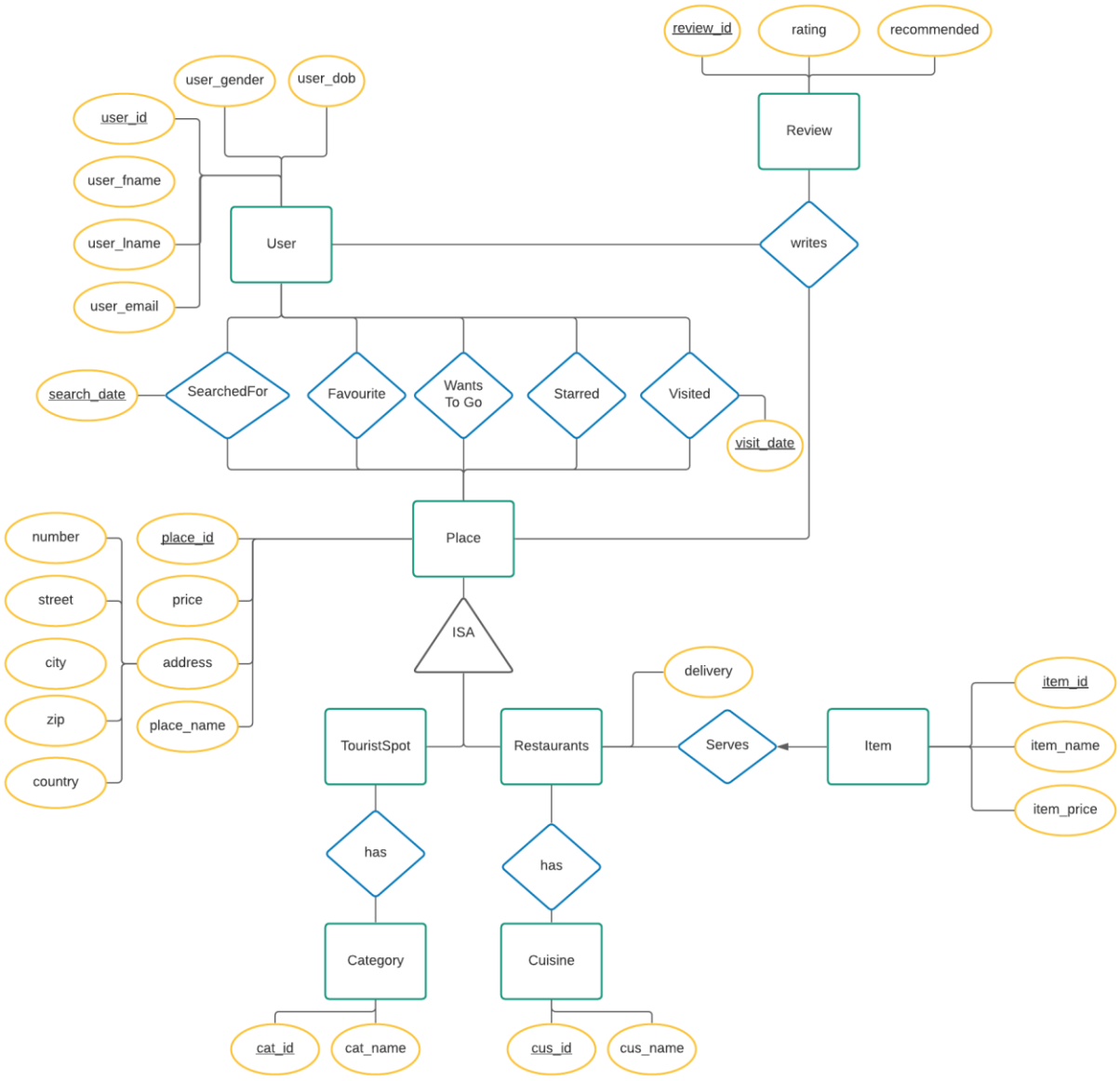
2. **Objectives and accomplishments**

Our objectives included creating a web application capable of holding and accurately modeling our database, along with providing a user-friendly interface that allows for a seamless user experience. Both goals were achieved, with relatively few sacrifices along the way.

Our back end consists of a Node express server written in JavaScript with two primary functions. The first of these is to communicate with a MySQL server that holds all of our data. It accomplishes this, having methods for executing various queries required to run our application. Gathering information on users, reviews, and locations based on user input. The second is to act as the middleman between said MySQL server and the front end of our application. As such those SQL queries are placed into various APIs that our front end can call.

Our front end is based on angular, and when coupled with the back end, it accomplishes all of our goals in terms of user experience. It is inherently very intuitive, with various tabs for searching for locations, search history, favorites, and other useful categories. All in all, we accomplished all of our goals and objectives for this project.

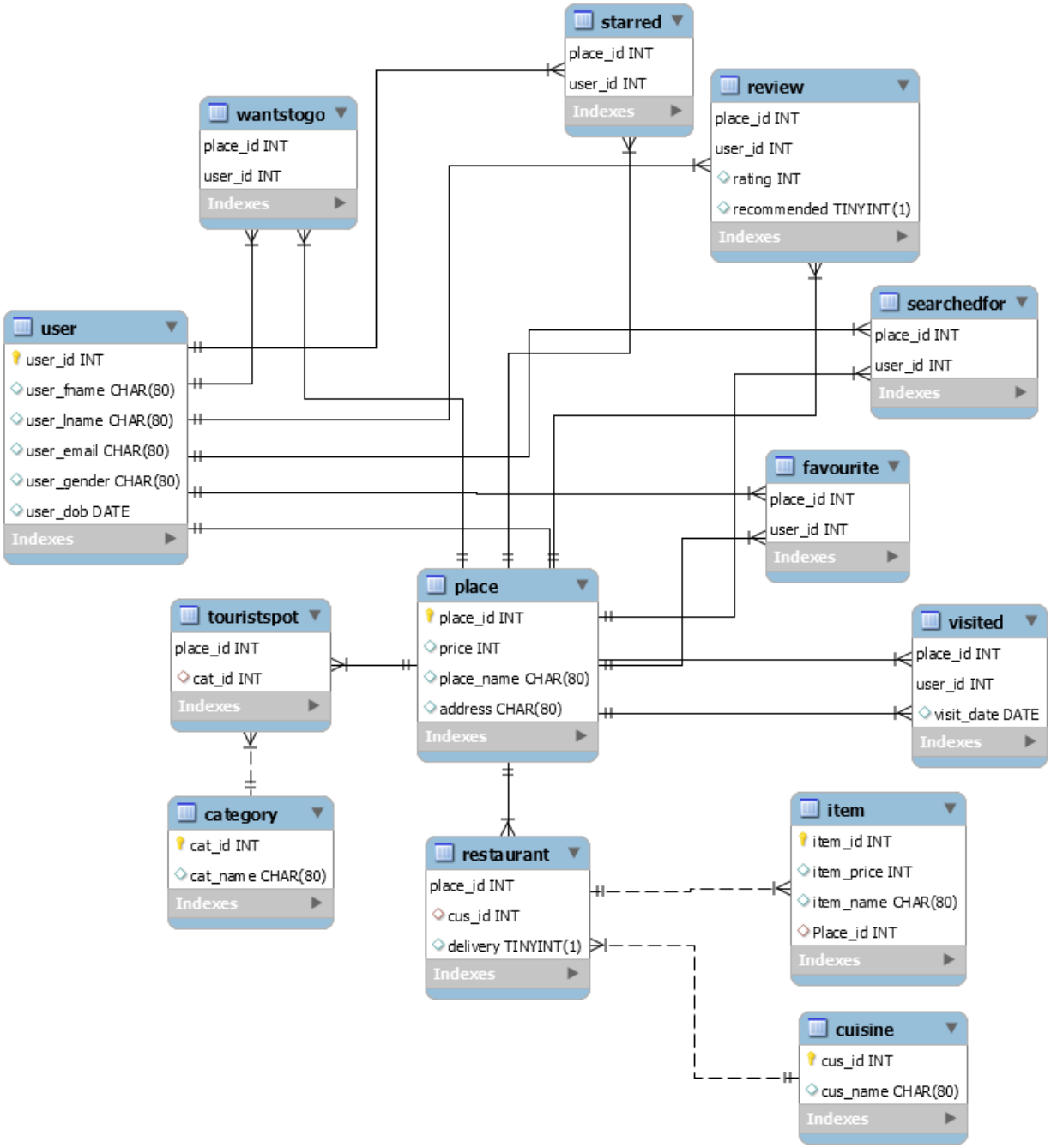
3. E/R diagram



All of the entities participate in many to many relationship; except the relation between Item and Restaurant which has N:1 relationship. All of the requirements were possible to add in the ER diagram.

4. Relational schema

Relational Model Diagram:



Translating the ER diagram into SQL schema:

TouristSpot	CREATE TABLE TouristSpot(place_id int NOT NULL, cat_id int, PRIMARY KEY (place_id), FOREIGN KEY (place_id) REFERENCES Place, FOREIGN KEY (cat_id) REFERENCES Category);
Category	CREATE TABLE Category(cat_id int NOT NULL, cat_name CHAR(20), PRIMARY KEY (cat_id));
Restaurant	CREATE TABLE Restaurant(place_id int NOT NULL, cus_id int, delivery BOOLEAN, PRIMARY KEY (place_id), FOREIGN KEY (place_id) REFERENCES Place, FOREIGN KEY (cat_id) REFERENCES Cuisine);
Cuisine	CREATE TABLE Cuisine(cus_id int NOT NULL, cus_name CHAR(20), PRIMARY KEY (cus_id));
Review	CREATE TABLE Review(place_id int NOT NULL, user_id int NOT NULL, rating int, recommended BOOLEAN, PRIMARY KEY (place_id, user_id), FOREIGN KEY (place_id) REFERENCES Place, FOREIGN KEY (user_id) REFERENCES User);
User	CREATE TABLE User(user_id int NOT NULL, user_fname CHAR(20), user_lname CHAR(20), user_email CHAR(20), user_gender CHAR(20), user_dob date, PRIMARY KEY (user_id));
SearchedFor	CREATE TABLE SearchedFor(

	place_id int NOT NULL, user_id int NOT NULL, PRIMARY KEY (place_id, user_id), FOREIGN KEY (place_id) REFERENCES Place, FOREIGN KEY (user_id) REFERENCES User);
Favourite	CREATE TABLE Favourite(place_id int NOT NULL, user_id int NOT NULL, PRIMARY KEY (place_id, user_id), FOREIGN KEY (place_id) REFERENCES Place, FOREIGN KEY (user_id) REFERENCES User);
WantsToGo	CREATE TABLE WantsToGo(place_id int NOT NULL, user_id int NOT NULL, PRIMARY KEY (place_id, user_id), FOREIGN KEY (place_id) REFERENCES Place, FOREIGN KEY (user_id) REFERENCES User);
Starred	CREATE TABLE Starred(place_id int NOT NULL, user_id int NOT NULL, PRIMARY KEY (place_id, user_id), FOREIGN KEY (place_id) REFERENCES Place, FOREIGN KEY (user_id) REFERENCES User);
Visited	CREATE TABLE Visited(place_id int NOT NULL, user_id int NOT NULL, Visit_date date, PRIMARY KEY (place_id, user_id), FOREIGN KEY (place_id) REFERENCES Place, FOREIGN KEY (user_id) REFERENCES User);
Place	CREATE TABLE Place(place_id int, price int, place_name CHAR(20), address CHAR(20), PRIMARY KEY (place_id));
Item *	CREATE TABLE Item(item_id int, item_price int, item_name CHAR(20),

	Place_id int, FOREIGN KEY (place_id) REFERENCES Restaurant, PRIMARY KEY (item_id));

- All the SQL queries for this project can be found in code/backend/src/events.js.

Normalizing relational schema to BCNF and 3NF:

a) List of non-trivial Functional Dependencies:

FDs in minimal basis:

- user_id -> user_fname, user_lname, user_email, user_gender, user_dob
- review_id -> rating, recommended
- place_id -> price, address, place_name
- cat-id -> cat_name
- cus_id -> cus_name
- item_id -> item_name, item_price
- visit_date -> place_id

These entities can be names as follows for easy naming purpose:

- user_id -> A
- user_fname -> B
- user_lname -> C
- user_email -> D
- user_gender -> E
- user_dob -> Y
- review_id -> F
- rating -> G
- recommended -> H
- place_id -> I
- price -> J
- address -> K
- place_name -> L
- cat_id -> M
- cat_name -> N
- cus_id -> O
- cus_name -> P
- item_id -> Q
- item_name -> R
- item_price -> S
- visit_date -> T

Therefore, the FDs can be written as following:

FDs = {A->BCDEY, F->GH, I-> JKL, M->N, O->P, Q->RS, T->I}

SOME OTHER non-trivial:

FG->H, FH -> G, BFG -> H, BFH ->G, CFG -> H, CG=FH -> G, DFG->H, DFH->G, EFG -> H, EFH -> G, FGY -> H, FHY -> G, ……

But this are duplicates of the FDs in minimal basis; thus, we have decided to work on the FDs that we found from the ER diagram.

b) BCNF decomposition:

Here, R = {ABCDEFGHIJKLMNOPQRSTY}

FDs = {A->BCDEY, F->GH, I-> JKL, M->N, O->P, Q->RS, T->I}

- R is in violation of BCNF. A is not a superkey nor BCDEY ⊆ A. Decompose R into
  - R1: A^+ = {ABCDEY} FD1: { A->BCDEY}
  - R2: A ∪ (R – A^+) = {AFGHIJKLMNOPQRST} FD2: { F->GH, I-> JKL, M->N, O->P, Q->RS, T->I}

❖ R1 is in BCNF.

- R2 is in violation of BCNF[F is not a superkey nor GH ⊆ F]. Decompose R2 into
  - R21: F^+ = {FGH} FD21: { F->GH }
  - R22: F ∪ (R2 – F^+) = {AFIJKLMNOPQRST} FD22: { I-> JKL, M->N, O->P, Q->RS,T->I}

❖ R21 is in BCNF.

- R22 is in violation of BCNF. Decompose R22 into
  - R221: I^+ = {IJKL} F221: { I-> JKL }
  - R222: I ∪ (R22 – I^+) = {AFIMNOPQRST} F222: { M->N, O->P, Q->RS, T->I }

❖ R221 is in BCNF

- R222 is in violation of BCNF. Decompose R222 into:
  - R2221: M^+ = {MN} FD2221: { M->N }
  - R2222: M ∪ (R222 – M^+) = {AFIMOPQRST} FD2222: {O->P, Q->RS, T->I }

- ❖ R2221 is in BCNF.
  - R2222 is in violation of BCNF. Decompose R2222 into:
    - R22221: O^+ = {OP} FD22221: { O->P }
    - R22222: O ∪ (R2222 – O^+) = {AFIMOQRST} FD22222: { Q->RS, T->I }
- ❖ R22221 is in BCNF.
  - R22222 is in violation of BCNF. Decompose R22222 into:
    - R222221: Q^+ = {QRS} FD222221: { Q->RS }
    - R222222: Q ∪ (R22222 – Q^+)= {AFIMOQT} FD222222: {T->I}
- ❖ R222221 and R222222 are both in BCNF.

The final set of BCNF schemas: {ABCDEY}, {FGH}, {IJKL}, {MN}, {OP}, {QRS}, {AFIMOQT}.
= { user_id, user_fname, user_lname, user_email, user_gender, user_dob},
{review_id, rating, recommended},
{place_id, price, address, place_name},
{cat_id, cat_name} , {cus_id, cus_name},
{item_id, item_name, item_price},
{user_id, review_id, place_id, cat_id, cus_id, item_id, visit_date}.

c) 3NF Decomposition:

Minimal Cover: {A->BCDEY, F->GH, I-> JKL, M->N, O->P, Q->RS, T->I}

After decomposition: one table for each FD

- ABCDEY (table 1)
- FGH (table 2)
- IJKL (table 3)
- MN (table 4)
- OP (table 5)
- QRS (table 6)
- TI (table 7)

Here, {user_id, review_id, place_id, cat_id, cus_id, item_id, visit_date} are the keys for original relation.

Therefore, it is in BCNF; since everything is dependent on keys.

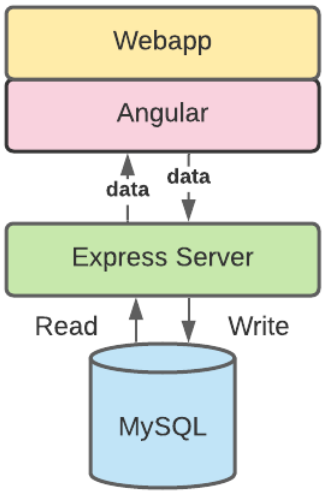
5. Database and dataset

Our database is a simple set of users, reviews, restaurants, tourist spots, and other features. This lets us display all the features of our application, such as showing the average rating of a location, showing the items sold at a restaurant, or storing user information. It was created by hand by us, as we could not find any appropriate examples online that translated well into our formatting, and as such we use the entirety of it in our work. This dataset includes tables on locations, tourist spots, restaurants, users, reviews, favorites, starred locations, locations they want to go to, visited locations, categories, cuisines, search history, favorites, and items.

6. Application

6.1 Application architecture

We used MySQL to create our database and Agular to create the web interface for our database. Express was used to make the middleware between our frontend and backend. The middleware contained the SQL statements needed to provide the required functionality and the web application accessed the middleware using a set of APIs. The overall architecture of our web application is shown in the following diagram:



6.2 Running the application

To run the application, some software needs to be installed first. The required softwares are:

1. NodeJS
2. Express
3. Angular
4. MySQL

Following are the steps to run the application:

- After running the install and load SQL scripts use the following commands on the SQL command line:

```
MySQL> create user 'TravelLover'@'localhost' identified by 'password';

MySQL> grant all on TravelLover.* to 'TravelLover'@'localhost';
```

- Then in the command line:

```
> node src/index.js
```

  Within the directory containing the project to start the back end.

- If this causes an error running:

```
MySQL> ALTER USER 'TravelLover'@'localhost' IDENTIFIED WITH
mysql_native_password BY 'password';
```

- After running the express server:

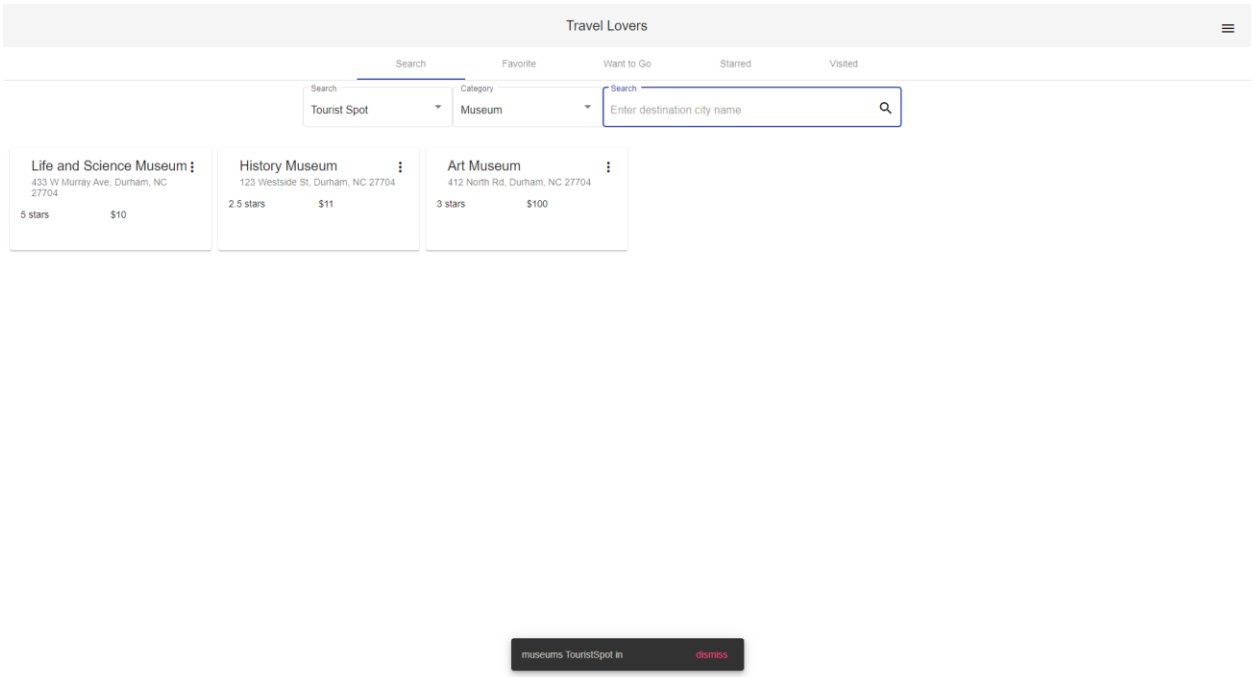
```
> cd frontend

> ng serve
```

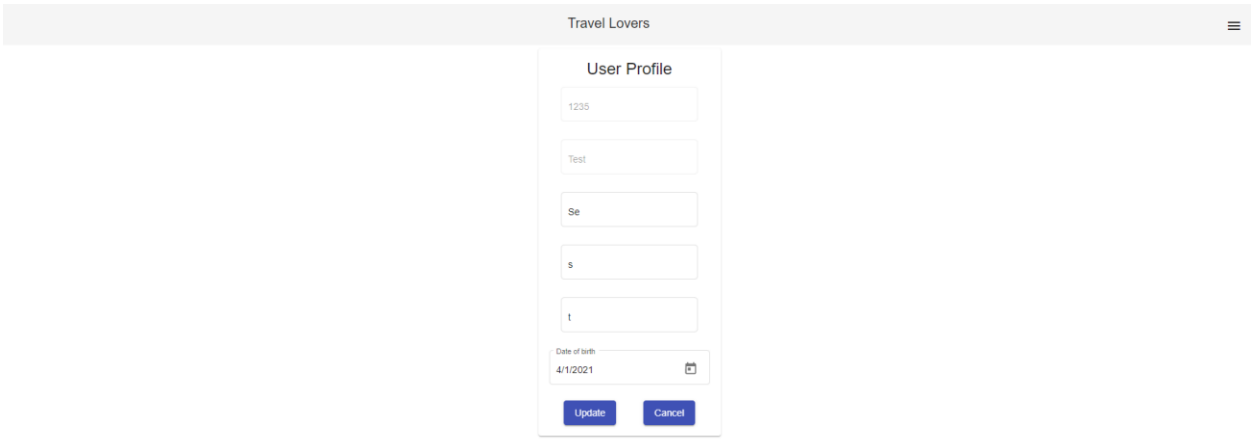
  This will open the web app in localhost:4200

6.3 Testing the application

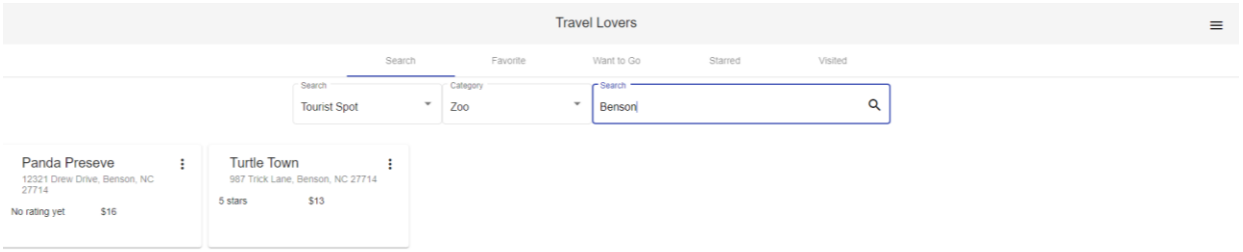
- Users can search for scenic spots.



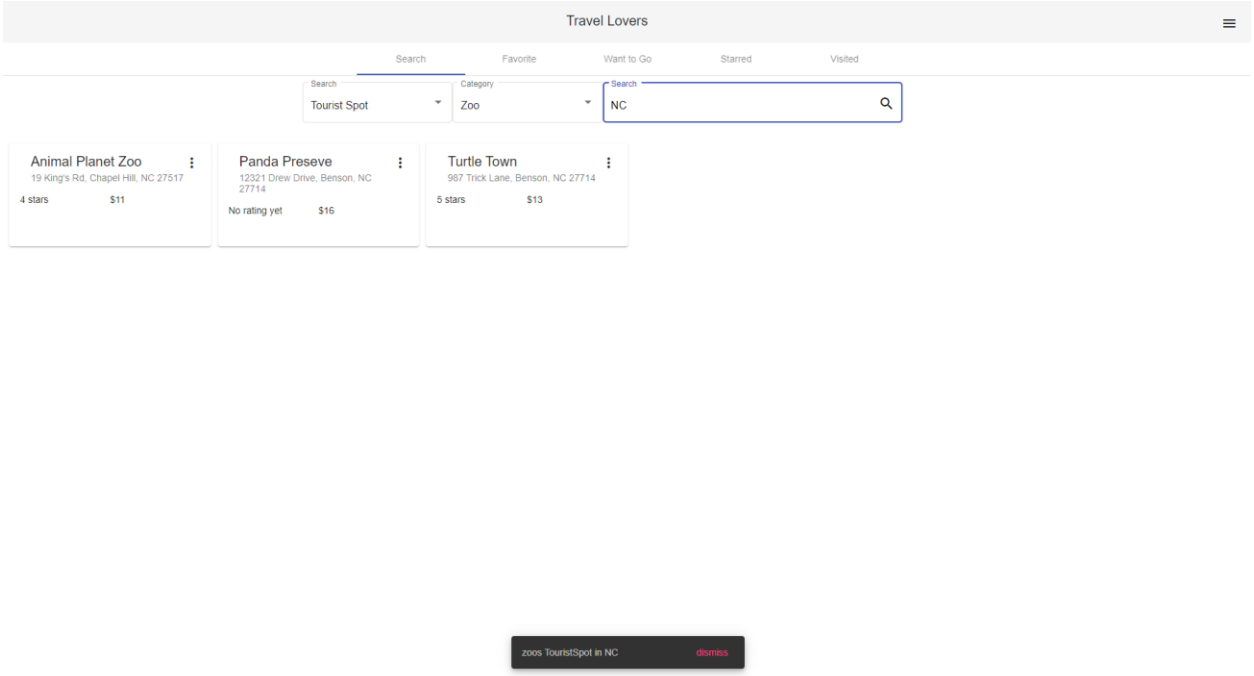
- Users can edit/update the information of their profiles.



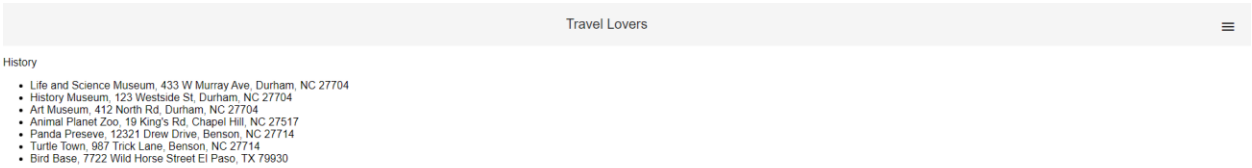
- Users can see a list of scenic spots by a specific city.



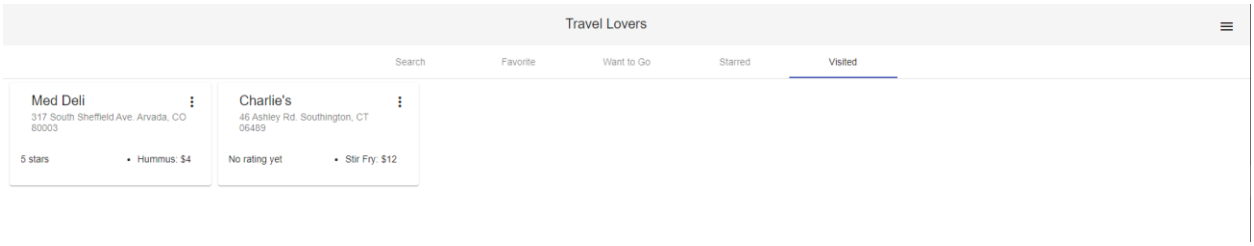
- Users can filter search results by state, genre, etc.



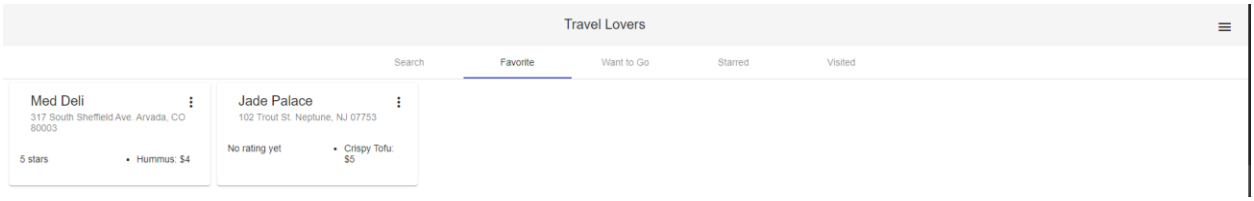
- Users can see search history.

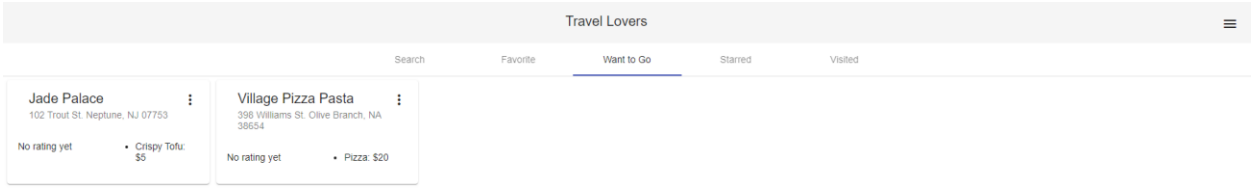


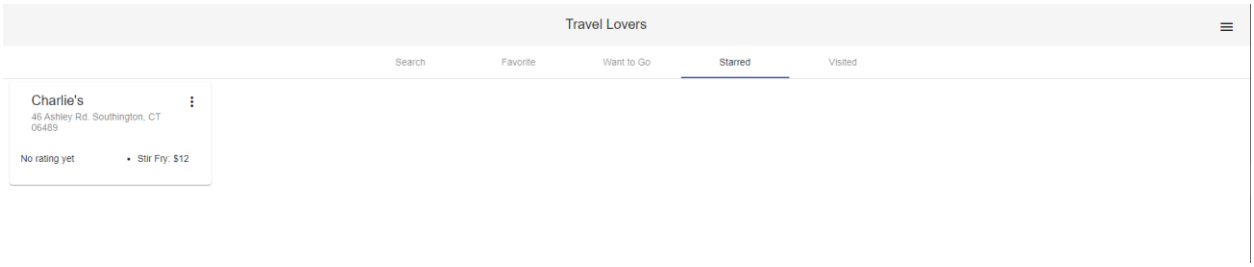
- Users can see his/her travel history.



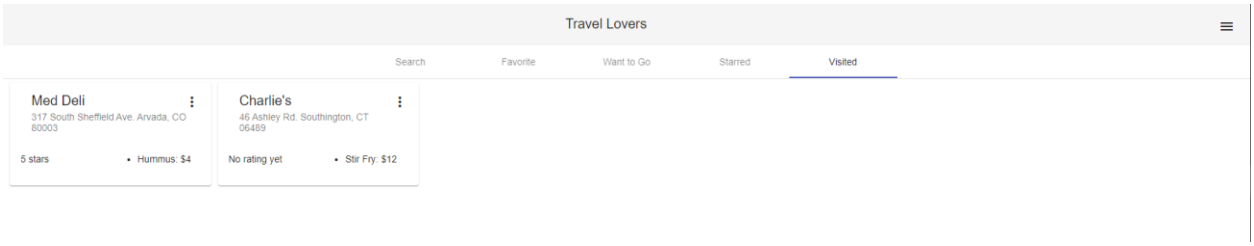
- Users can "favorite", "want to go", "starred places", etc., places.



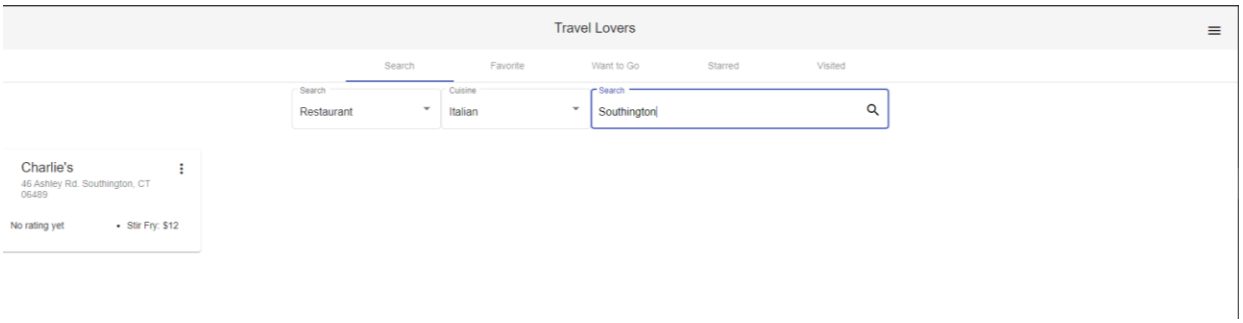




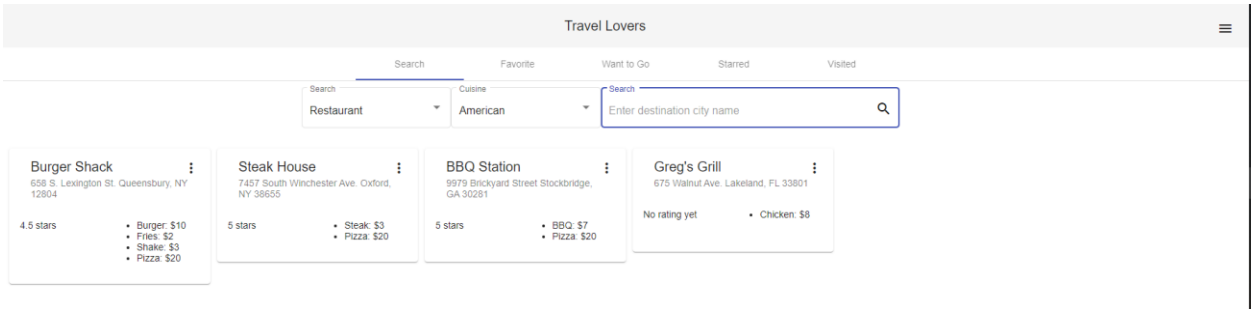
- Users can search if he/she visited this scenic spot already.



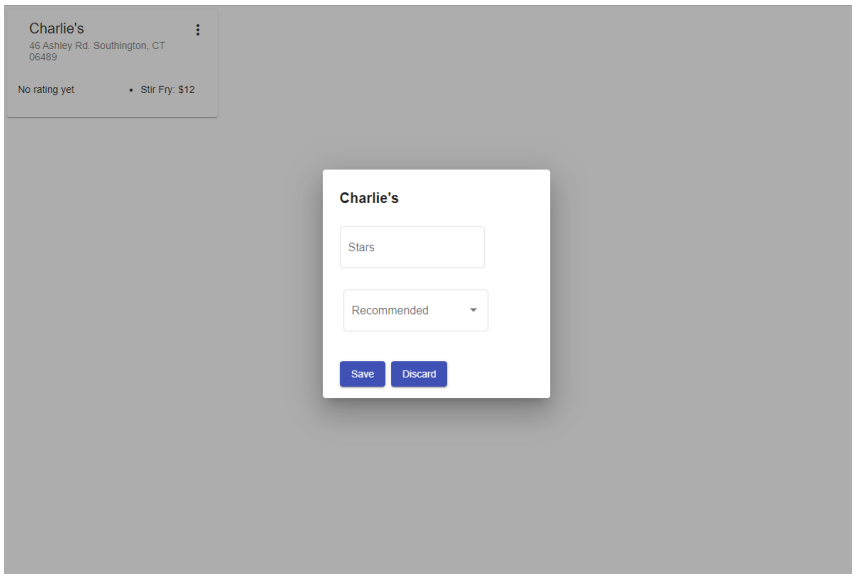
- Users can search for restaurants near a destination.



- Users can search for restaurants for a certain type of cuisine.



- Users can write reviews for the places they visited



7. Conclusion

Travel lover is a database system application formed by the joint collaboration of JavaScript written for the Node Express server back-end and web-accessible interface written angular front-end.

Travel lover uses MySQL to create the database, Agular to create the web interface for the database, and Express to create the middleware between the front-end and the back-end to form the overall architecture of the application.

Travel lover offers features for location search, search history, favorites, and advance search with specific criteria to meet the needs of users to plan, record, budget, and review their travel.

8. Appendices

Team Roles:

- Tahmid Muttaki: worked on Front-end development, managing git repository, project report, and slides.
- Labiba Labanya: worked on managing the git repository, project report, ER diagram, and relational schema refinement.
- Evan Dischinger: worked on the backend, database and dataset, project report, and presentation.
- Zicong Lin: worked on database and dataset, .sql file creation, project report, and delivering the presentation.