COM6504 Intelligent Web

Junshan Liu, Zhifang Zhu, Xixi Ye

1. Introduction

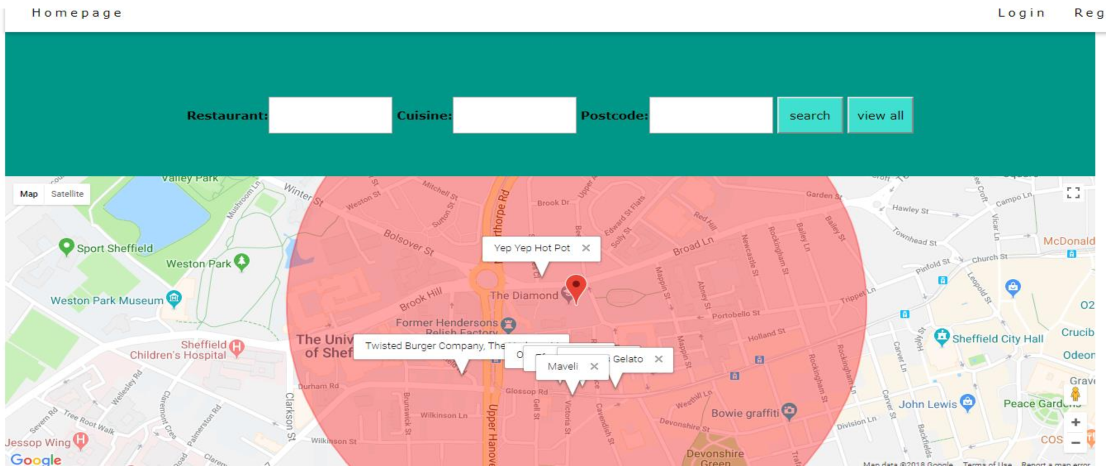
The project aims to develop a website and a progressive app for finding restaurants and for reviewing/rate them. The users can also create new pages for restaurants. This report provides a concise statement to the solutions because of the limitation on pages. All the functions required in the assignment are achieved successfully except the progressive web app part.

2. Searching for restaurants and displaying results

This part can achieve the function of searching restaurants by inputting some keywords through a form. Those terms like type of cuisine and postcode are in AND conditions. The search engine asks restaurant name as required and meanwhile if other terms are detected being filled, it also needs be fitted as AND conditions. The ideal result is to return a list of restaurants most relevant to the query. And restaurants in a defined radius of 500 meters are also shown on a map within the web page.

2.1 Solution

The homepage contains the search function and a map showing restaurants location marks within defined radius of 500 meters around the user location:



A form is used to search expected restaurants. A list of restaurants relevant to the query will be presented with Ajax while searching with certain terms. The advantage of Ajax is that it can update part of web page without loading a new web page. The result page of searched restaurant list is presented in the appendix part. The restaurants information including restaurant name, cuisine type, address and rating is stored and communicated in JSON format.

```
function sendAjaxQuery(url, data) {
    $.ajax({
        url: url ,
        data: data,
        dataType: 'json',
        type: 'POST',
        success: function (dataR) {
            // no need to JSON parse the result, as we are using
            // dataType:json, so JQuery knows it and unpacks the
            // object for us before returning it

            var obj= dataR;
            //var obj = JSON.parse(dataR);
            // in order to have the object printed by alert
            // we need to JSON stringify the object

            document.getElementById('result_name').innerHTML='name: '+obj.name;
                //"<a href=restaurant?name="+obj.name+">name: "+obj.name+"</a>";

            document.getElementById('result_cuisine').innerHTML= 'cuisine: '+obj.cuisine;
            document.getElementById('result_postcode').innerHTML= 'postcode: '+obj.postcode;
            document.getElementById('result_address').innerHTML= 'address: '+obj.address;
            document.getElementById('result_description').innerHTML= 'description: '+obj.description;
        },
{ id: 5b11815329e3e64bb0b0980b,                description: 'this is Sundaes Gelato',
  name: 'Sundaes Gelato',                      reviews: '    ',
  cuisine: 'Dessert',                          rating: 2.88,
  postcode: 'S10 2HS',                         longitude: -1.480883,
  address: '262 Glossop Rd',                   latitude: 53.380159
```

2.2 Issues

With the design of search engine, a challenge faced is how to detect the terms, which furthermore decide which SQL query to be used and visit the database.

With the design of Google map, since it works as a third-party-API, some troubles about adaptability like location request, postcode referring are concerned during design.

2.3 Requirements

Nearly all the requirements for this part has been achieved successfully. It allows users to input a series of terms and a list of restaurants most relevant to the query will be returned. A Google map is added to show all the restaurants stored in the database in a defined radius of 500 meters starting from the user location.

2.4 Limitations

The radius on the map is set as a default value instead of defined by users.

The search engine can not support a query which does not contain restaurant name.

Once we tried to add web link to the search result in order to apply the function of accessing the restaurant webpage directly from search, as follows:

```
document.getElementById('result_name').innerHTML="<a href=restaurant?name="+obj.name+">name:
"+obj.name+"</a>";
```

But in practice we face trouble that the link can not translate space ' ' to '&20;' directly and we have no idea how to translate it too. So the idea is cancelled.

3 The restaurant page

3.1 Solution

The searched restaurants list allows accessing the restaurant page containing all the relevant information. Because of the paper limitation, the template restaurant page is shown in the appendix.

In the website of all restaurant list, we create the link to individual restaurant page,

```
<a href="restaurant?name=<%=obj[j].name%>"><%= obj[j].name %></a>
```

And access res.render(), it delivers relevant information queried from the database to the restaurant page (where variance *obj* refers to relevant restaurant information).

```
router.get('/restaurant', function(req, res, next) {
                    ……
        res.render('restaurant', {obj:character});
});
```

In order to use *obj* received from server in further JavaScript, we design a small trick in the static webpage, in html—`<div id="object" visibility: hidden><%=JSON.stringify(obj)%></div>`

in javascript—`var obj={};obj=JSON.parse($("#object").html());`

with this trick, we manage to deliver data from server to javascript, and in the webpage we present every restaurant data. To deserve to be mentioned, the rating level is calculated by

```
rating=( (rating* rate_count+new_rate)/(rate_count+1) ).toPrecision(3);
```

3.2 Issues

In this part, the first challenge we met is how to recognize the number of restaurants in the list in order to allocate related <div> part for them to present. Since poor experience we have in HTML coding before, we tried several methods and finally find the for loop syntax adopted by HTML.
Further, we face the trouble mentioned in the previous part that in javascript unable to use data directly received from res.render(). And the solution of this is also shown in previous part.

3.3 Requirements

We succeed from querying the database, delivering relevant data and presenting in the individual pages.

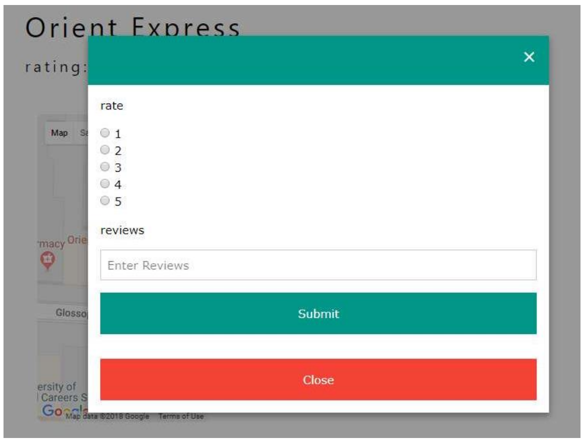
3.4 Limitations

While trying to upload images of restaurant, we face difficulties, and fail to find the solution to save a local image into correct folder thus unable to present pictures in the webpages.

4 Allowing restaurant reviews

4.1 Solution

Users can add ratings and reviews to page. These will be uploaded to the server and be seen by other users as well on the restaurant page.



Rating selection is designed in type= ″ratio″ and in order to transfer precise information, we design a hidden input as:

```
<input type="hidden" name="id" value=<%=obj.id%> />
```

Where value is the restaurant unique id allocated by database.
With submit button, the form sends a POST method with rating and reviews. Later it is stored in database and callback to refresh the restaurant webpage.

4.2 Issues

In this part, every progress works as expected, with few problems. The reviews are POSTed by form and in routes/index.js function in controllers/characters.js is invoked to handle the database update. And it reloads the restaurant webpage.

4.3 Requirements

Adding ratings and reviews to page is achieved successfully but uploading photos is missing to be implemented.

4.4 Limitations

Two limitations are to be mentioned. First, ratings are calculated with precision (3), and with a range of 1 to 5. Second, reviews are stored in mere one item in database.

5 Creating a new restaurant page

5.1 Solution

Users can add a new restaurant after input the basic restaurant information including name, cuisine, address, postcode, latitude and longitude. All filled information are POSTed by form and use insert() function to save in the database. As database model demands, primary information are must (type= "required"), and initial rating is set as 3, rate_count is set as 0.

5.2 Issues

After filling the form and POSTing data to server, we design to store data in the database, and further do not give general response back to form. Instead of it, we choose to redirect the page as soon as submit.

```
<input type="submit" value="submit" onclick="javascript:window.location='list';">
```

5.3 Requirements

The requirements in this part contain the form to POST and the structure need fit database schema.

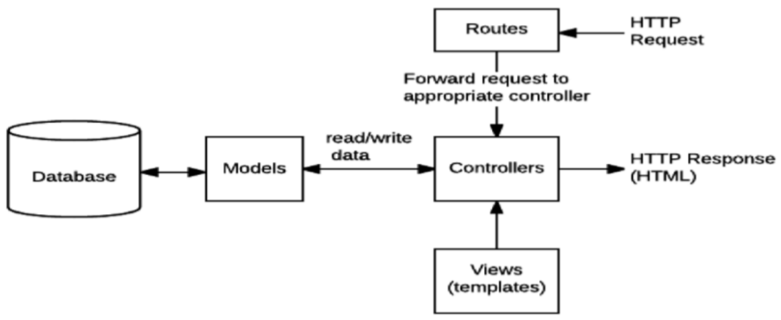
5.4 Limitations

This part exploits a serious defect in our website as user need to fill in the longitude and latitude of restaurant, thus it can be used in Google Map marker. However, it is obviously not a nature way to add a restaurant.

6 The Server Architecture

6.1 Solution

The server architecture is shown as below:



In order to explain this part, we use a simple request to demonstrate. With the client request /index webpage, the request is sent to routes and find a response route.get('/index'). With function invoked inside, which calls controllers to handle database event- use models to create query and access database. In response, the server replies res.render() and static views of webpage to client.

6.2 Issues

In the part of database, we need to create connection in /databases, create data schema in /models, initialize data and invoke functions in /controllers. Especially to be mentioned, the query between database and server ought to be set in JSON format.

6.3 Requirements

Database requires normalized query to access and JSON is prefect designed to use to communicate between web and database.

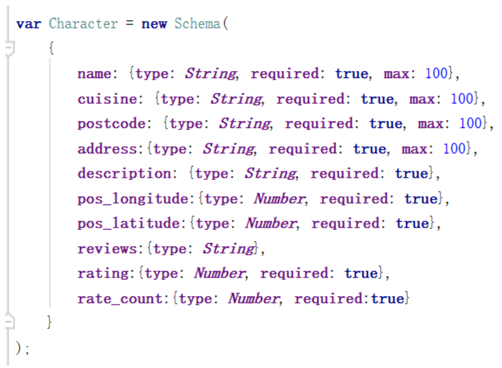
6.4 Limitations

In this design, we use database with an initial condition that drop all data in the database at first. Then we create 6 items in restaurant database and one user.

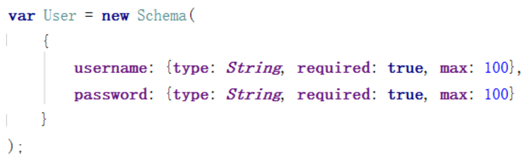
7 The Database

7.1 Solution

MongoDB is used for database in this project. The models/characters.js and models/users.js contain the structure of individual database as follows:

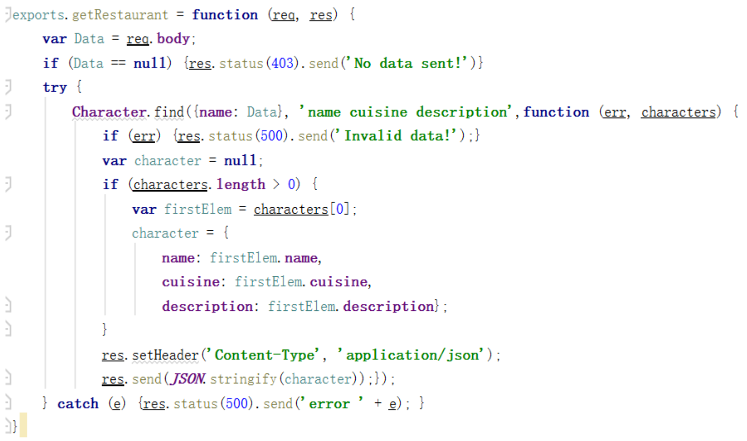


```
var Character = new Schema(
    {
        name: {type: String, required: true, max: 100},
        cuisine: {type: String, required: true, max: 100},
        postcode: {type: String, required: true, max: 100},
        address:{type: String, required: true, max: 100},
        description: {type: String, required: true},
        pos_longitude:{type: Number, required: true},
        pos_latitude:{type: Number, required: true},
        reviews:{type: String},
        rating:{type: Number, required: true},
        rate_count:{type: Number, required:true}
    }
);
```



```
var User = new Schema(
    {
        username: {type: String, required: true, max: 100},
        password: {type: String, required: true, max: 100}
    }
);
```

A template Mongo query using name to return a list of restaurant information as follows:



```
exports.getRestaurant = function (req, res) {
    var Data = req.body;
    if (Data == null) {res.status(403).send('No data sent!')}
    try {
        Character.find({name: Data}, 'name cuisine description',function (err, characters) {
            if (err) {res.status(500).send('Invalid data!');}
            var character = null;
            if (characters.length > 0) {
                var firstElem = characters[0];
                character = {
                    name: firstElem.name,
                    cuisine: firstElem.cuisine,
                    description: firstElem.description};
            }
            res.setHeader('Content-Type', 'application/json');
            res.send(JSON.stringify(character));});
    } catch (e) {res.status(500).send('error ' + e); }
}
```

7.2 Issues

With the function demands, the design need two schema respectively to store restaurant and user information, thus it introduce some troubles. But finally, we solve every part.

7.3 Requirements

It satisfies the requirements that the database of restaurants is implemented using MongoDB and contain all the information relevant to each restaurant.

7.4 Limitations

As software required, there should be several command instruments to deal before start MongoDB. (make directory and start mongo as lecture slides stating)

8 A Progressive Web App

During the progress of design, we find difficulties in creating a progressive web app, having few idea of implement related .js file, so we fail in this part.

9 Quality of the solution

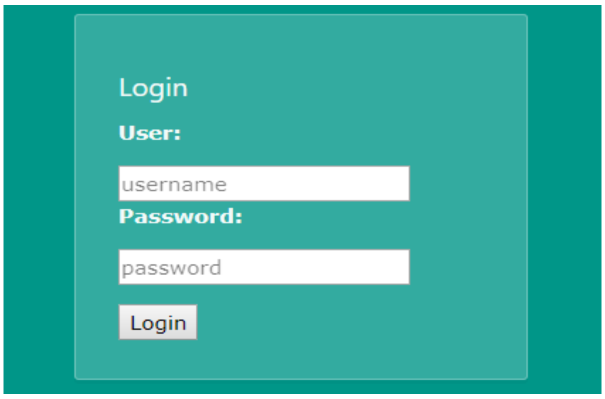
During the progress of design, we manage to use all techniques taught in course and try to design the whole server and client easy to use and to understand. However, there are still some defects like few css design, possible unhandled error response.

10 Conclusions

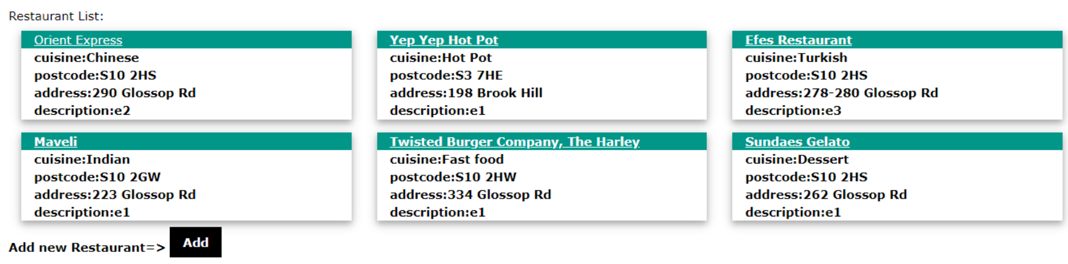
This project has achieved most of the requirements for the assignment successfully using the techniques taught in lectures and labs. The registration and login sections have also been implemented. All the members of the group contributed equally to the assignment solution. The server runs based on NodeJS environment.

Appendix

Login page:



The searched restaurant list:

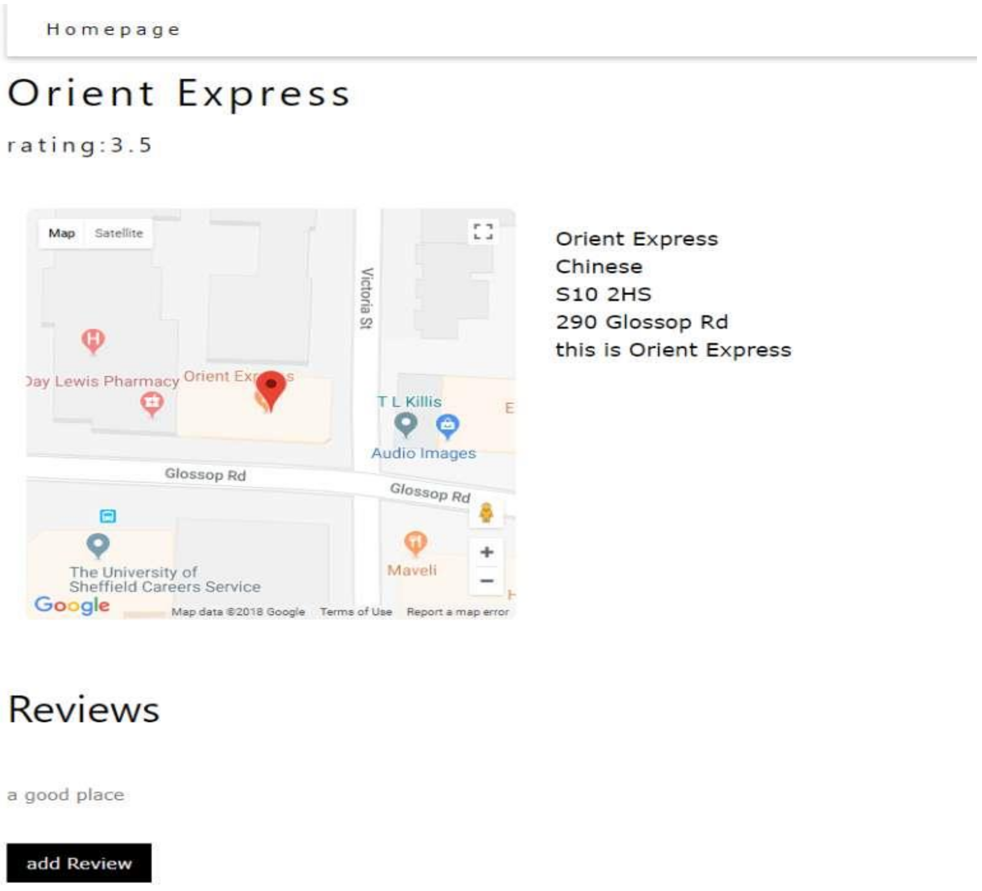


Restaurant List:

Orient Express cuisine:Chinese postcode:S10 2HS address:290 Glossop Rd description:e2	Yep Yep Hot Pot cuisine:Hot Pot postcode:S3 7HE address:198 Brook Hill description:e1	Efes Restaurant cuisine:Turkish postcode:S10 2HS address:278-280 Glossop Rd description:e3
Maveli cuisine:Indian postcode:S10 2GW address:223 Glossop Rd description:e1	Twisted Burger Company, The Harley cuisine:Fast food postcode:S10 2HW address:334 Glossop Rd description:e1	Sundaes Gelato cuisine:Dessert postcode:S10 2HS address:262 Glossop Rd description:e1

Add new Restaurant=> Add

The template restaurant page for Yep Yep Hot Pot:



Homepage

Orient Express

rating:3.5

Orient Express
Chinese
S10 2HS
290 Glossop Rd
this is Orient Express

Reviews

a good place

add Review