## Chapter 1

## INTRODUCTION

FOOD++ is a full stack web application that helps users locate all the available hotels and restaurants in a particular region. It lets the users see the details about the restaurant, the cuisines and dishes served by this restaurant and allows them to write reviews or rate a particular restaurant.

The front end of the application is built using JavaScript, HTML5 and CSS3. HTML5 is used to give structure to the web pages of our application. CSS3 is used to give style to the web pages and JavaScript is used to add logic and DOM manipulation to the web pages. Bootstrap and Semantic-UI are two CSS libraries that the application uses for styling. JQuery is a JavaScript library used for DOM manipulation.

The back end of the application is built using node.js, which is a JavaScript runtime. The framework used for building the HTTP server is express.js. EJS is used for templating the web pages. The database of the application is built using MySQL 5.7 and the database is run on the MySQL community server.

## Chapter 2

## PROJECT REQUIREMENTS

### 2.1 Hardware

* Processor : inteli52.4GHz, 64bitprocessor
* Ram : 4GBRAM
* Hard Disk : 50GB
* Networking technology : Ethernet /  Wireless Ethernet

**2.2 Software**

* Operating System : WINDOWS / MAC / LINUX
* Programming Languages : HTML5, CSS3, JavaScript, EJS
* Libraries : Bootstrap, Font-Awesome, Semantic-UI, jQuery
* Database : MySQL
* Server : node.js, Express
* Web browser : Google Chrome, IE8+, Mozilla Firefox

**Chapter 3**

**LITERATURE SURVEY**

**3.1 Current market**

Locating restaurants and viewing the restaurant’s menu and images from home helps the user pick a restaurant he would want to go depending on his food preferences, without actually visiting the restaurant. This way the user is also able to discover restaurants in a location that he never knew were present there before.

Online food ordering and restaurant viewing portals are flourishing in India as people prefer to save time and pick a restaurant with ease, instead of going to every restaurant and exploring it one by one. The most famous online food ordering and restaurant viewing portals in India right now are foodpanda.in, zomato.com and swiggy.com. These portals allow users to select a location they want to dine at or need food to be ordered from. They then select a particular restaurant and view the menus of these particular restaurants. The user then places a home delivery order to these restaurants if he wishes. On placing an order, the online portal sends the order request to the particular restaurant with details about the customer and the delivery.

**3.2 Front-end Technology**

**3.2.1 JavaScript**

JavaScript, often abbreviated as JS, is a high-level, dynamic, weakly typed, prototype-based, multi-paradigm, and interpreted programming language. JavaScript on the front-end is used for DOM manipulation and AJAX.

**3.2.2 HTML5**

HTML5 is the latest version of Hypertext Mark-up Language that is used to define the structure of web pages. Hypertext Mark-up Language is the standard mark-up language for creating web pages and web applications.

**3.2.3 CSS3**

CSS or Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a mark-up language. CSS3 is the latest evolution of the Cascading Style Sheets language and aims at extending CSS2.1.

**3.2.4 JQuery**

JQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. It is free, open-source software using the permissive MIT License.

**3.3 Back-end technology**

**3.3.1 Node.js**

Node.js is an open-source, cross-platform JavaScript runtime environment for executing JavaScript code server-side. Node.js is built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

**3.3.2 Express.js**

Express.js, or simply Express, is a web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs.

**3.3.3 EJS**

EJS is a simple templating language that lets you generate HTML mark-up with plain JavaScript. It can be used with the express view engine to generate the HTML file.

**3.4 Database technology**

**3.4.1 MySQL**

MySQL is an open source relational database management system (RDBMS) based on Structured Query Language (SQL).  MySQL runs on virtually all platforms, including Linux, UNIX, and Windows.

**Chapter 4**

**SYSTEM DESIGN**

**4.1 Database Design**

The data required for this project is organized and stored as tables in a MySQL database. The list of tables in this project are:-

* USER: Information about the users accessing the application.
* RESTAURANT: Information regarding the restaurants that have signed up our application.
* REVIEW: Reviews written by the user for a particular restaurant.
* MENU: Information about all the menus offered by the restaurants.
* MENU\_ITEM: Information about dishes that are present in a particular menu.

The Schema (Fig 4.1) depicts the dependencies among the tables and the Entity-Relation Diagram (Fig 4.2) depicts the relations and their corresponding entities.

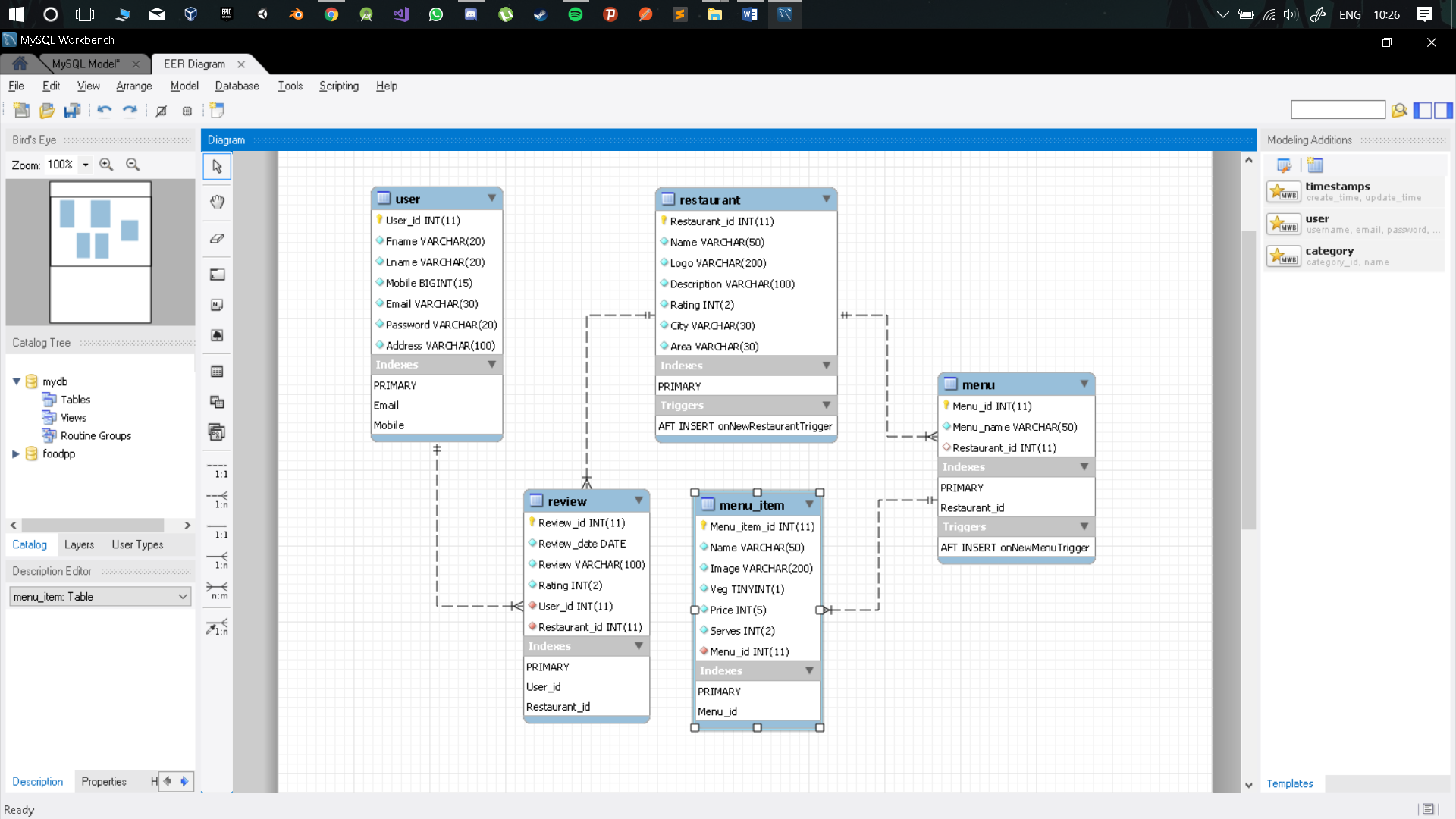


Fig 4.1 Database Schema

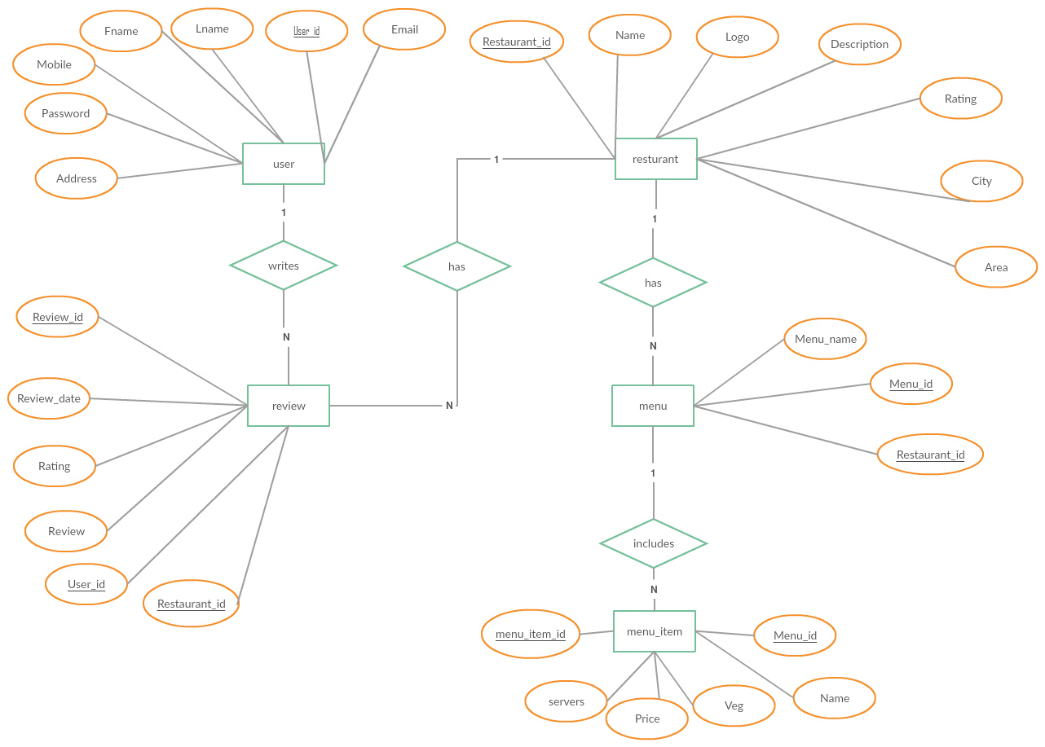


Fig 4.2 Entity – Relationship Diagram

**4.2 Server design**

The server program of our application is built using JavaScript upon the express.js framework and is run on node.js. The server program holds event handles for all the routes in the application to which requests can be sent. The program then processes these requests and responds accordingly. The server has the ability to connect with the MySQL database server using the mysql package. This lets the program perform CRUD operations on the database and supports data persistence throughout the application.

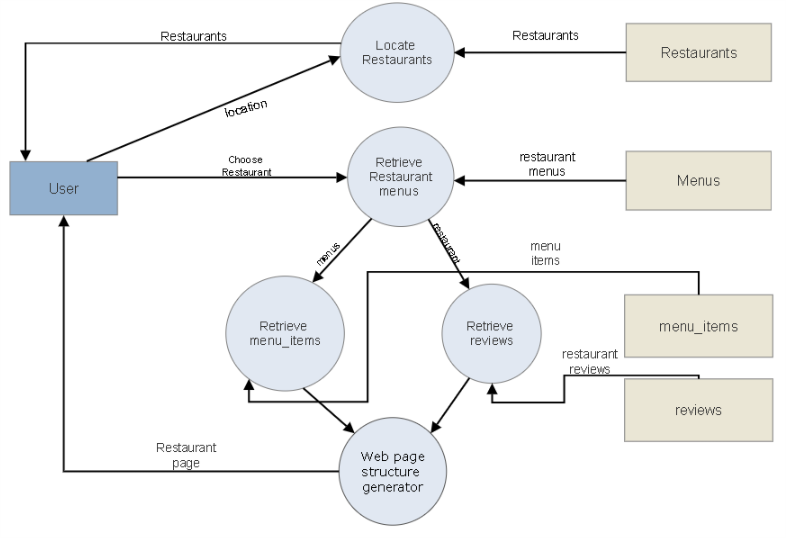


Fig 4.3 Data Flow Diagram

**Chapter 5**

**IMPLEMENTATION**

**5.1 Server**

**5.1.1 Importing required packages**

var express = require("express");

var app = express();

var bodyParser = require("body-parser");

var methodOverride = require("method-override");

var mysql = require("mysql");

var connectionObject = {

host: "localhost", user: 'root',  password: 'password',

database: 'foodpp', port: 3306};

var session = require("express-session");

var pmysql = require('promise-mysql');

var forEach = require('async-foreach').forEach;

**5.1.2 Configuring application**

app.set("view engine", "ejs");

app.use(bodyParser.urlencoded({extended:true}));

app.use(express.static(\_\_dirname+"/public"));

app.use(methodOverride("\_method"));

app.use(session({

secret: "DBMS mini project",

resave: false,

saveUninitialized: false }));

app.use(function (req, res, next) {

res.locals.currentUser = req.session.user;

next(); });

**5.1.3 Index routes**

// ROOT - show index page

app.get("/", function (req, res) {

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err) {

if(err) { console.log(err) } else {

connection.query("SELECT DISTINCT City, Area FROM restaurant",function (err2, locations, fields) {

if (err2) { console.log(err2) } else {

var cities = new Set();

locations.forEach( function(element, index) {

cities.add(element.City); });

let citiesArray = Array.from(cities);

res.render("index",{cities: citiesArray, locations:locations});}});}});});

app.post("/", function (req, res) {

res.redirect("/restaurants/"+req.body.city+"/"+req.body.area+"");});

// LOGIN - show login form

app.get("/login", function (req, res) {

res.render("login");});

//login logic

app.post("/login", function (req, res) {

email = req.body.user.email;

password = req.body.user.password;

var query = "SELECT \* FROM user WHERE email = ?";

var connection = mysql.createConnection(connectionObject);

connection.query(query, [email], function (err, results, fields) {

if (err) { console.log(err); }

else { if(results.length>0){

if(results[0].Password == password){

req.session.user = results[0];

res.redirect("/");

} else {res.send("email does not match password");}

} else {res.send("email does not exist");}}});});

// REGISTER -  show form to register

app.get("/register", function (req, res) {

res.render("register");});

// register logic

app.post("/register", function (req, res) {

fname = req.body.user.fname;lname = req.body.user.lname;

mobile = req.body.user.mobile;email = req.body.user.email;

password = req.body.user.password;address = req.body.user.address;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err) {

if(err){console.log(err)} else {

var values = [[fname, lname, mobile, email, password, address]];

var queryFields = "Fname, Lname, Mobile, Email, Password, Address";

var query = "INSERT INTO user(" + queryFields + ") VALUES ?";

connection.query(query, [values], function (err, result, fields) {

if(err) {console.log(err);} else {console.log("User created");

res.redirect("/login");}});}});});

//   LOGOUT

app.get("/logout", function (req, res) {

delete req.session.user;

res.redirect("/");});

**5.1.4 Restaurant routes**

// INDEX - show all restaurants

app.get("/restaurants/:city/:area", function (req, res) {

var finalRestaurants = [];

var semiFinalRestaurants = [];

var city = req.params.city;

var area = req.params.area;

var location = { city: city, area: area };

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if (err1) { console.log(err1); } else {

var query1 = "SELECT \* FROM restaurant where City = '"+city+"' and Area = '"+area+"'";

connection.query(query1, function (err2, restaurants, fields) {

if (err2) { console.log(err2); } else {

forEach(restaurants, function(restaurant, index) {

var query2 = "SELECT Menu\_name FROM menu WHERE Restaurant\_id = "+restaurant.Restaurant\_id;

connection.query(query2, function (err3, menuNames, index) {

if (err3) { console.log(err3); } else{

restaurant.menuNames = menuNames;

semiFinalRestaurants.push(restaurant);}});

var done = this.async();

setTimeout(done, 50);},

function (notAborted) {

forEach(semiFinalRestaurants, function (restaurant, index) {

var query3 = "SELECT COUNT(\*) FROM review WHERE Restaurant\_id ="+restaurant.Restaurant\_id;

connection.query(query3, function (err4, noOfReviews, index) {

if (err4) { console.log(err4); } else{

restaurant.noOfReviews = noOfReviews[0]["COUNT(\*)"];

finalRestaurants.push(restaurant);}});

var done = this.async();

setTimeout(done, 50);},

function (notAborted2) {

res.render("restaurants/index",{restaurants: finalRestaurants, location: location});});});}});}});});

// NEW - form to create a new restaurant

app.get("/restaurants/new", function (req, res) {

res.render("restaurants/new");});

// CREATE - creates a new restaurants

app.post("/restaurants", function (req, res) {

var Name = req.body.restaurant.Name;

var Logo = req.body.restaurant.Logo;

var Description = req.body.restaurant.Description;

var Rating = req.body.restaurant.Rating;

var City = req.body.restaurant.City;

var Area = req.body.restaurant.Area;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if (err1) { console.log(err1); }

else {

var values = [[Name, Logo, Description, Rating, City, Area]];

var queryFields = "Name, Logo, Description, Rating, City, Area";

var query = "INSERT INTO restaurant(" + queryFields + ") VALUES ?";

connection.query(query, [values], function (err2, result, fields) {

if(err2) { console.log(err2); }

else {

console.log("Restaurant created");

res.redirect("/");}});}});});

// SHOW - shows more information about one restaurants \*

app.get("/restaurants/:id", function (req, res) {

restaurantId = req.params.id;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if (err1) { console.log(err1); }

else {

var query1 = "SELECT \* FROM restaurant where Restaurant\_id = "+restaurantId;

connection.query(query1, function (err2, restaurant, fields1) {

if (err2) { console.log(err2); }

else { restaurant = restaurant[0];

var query2 = "SELECT \* FROM menu WHERE Restaurant\_id = " + restaurant.Restaurant\_id;

connection.query(query2, function (err3, menus, fields2) {

if(err3) { console.log(err3); }

else{

restaurant.menus = menus;

forEach(restaurant.menus, function (menu, index) {

var query3 = "SELECT \* FROM menu\_item WHERE Menu\_id = " + menu.Menu\_id;

connection.query(query3, function (err4, menuItems, fields3) {

if (err4) { console.log(err4) } else {

restaurant.menus[index].menuItems = menuItems;} });

var done = this.async();

setTimeout(done, 50);

}, function (notAborted) {

var query4 = "SELECT r.\*,DATE\_FORMAT(r.Review\_date,'%d/%m/%Y') AS niceDate, u.Fname  FROM review r, USER u WHERE r.Restaurant\_id = "+restaurant.Restaurant\_id+" and r.User\_id = u.User\_id";

connection.query(query4, function (err5, reviews, fields4) {

if (err5) { console.log(err5) } else {

restaurant.reviews = reviews;

res.render("restaurants/show", {restaurant:restaurant});}});});}});}});}});});

**5.1.5 Reviews routes**

// NEW - form to create a new review for the particular restaurant

app.get("/restaurants/:id/reviews/new",isLoggedIn, function (req, res) {

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if (err1) { console.log("1"+err1); }

else {

var query = "SELECT \* FROM restaurant WHERE Restaurant\_id = "+req.params.id;

connection.query(query, function (err2, Restaurants, fields) {

if (err2) { console.log(err2); } else {

if(Restaurants.length<=0){

res.send("restaurant does not exist");

} else {

console.log(Restaurants[0]);

res.render("reviews/new",{restaurant:Restaurants[0], user: req.session.user});}}});}});});

// CREATE - creates a new review

app.post("/restaurants/:id/reviews", isLoggedIn, function (req, res) {

var Review = req.body.review.Review;

var Rating = req.body.review.Rating;

var UserId = req.session.user.User\_id;

var RestaurantId = req.params.id;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if (err1) { console.log(err1); }

else {

var queryFields = "Review\_date, Review, Rating, User\_id, Restaurant\_id";

var query = "INSERT INTO review("+queryFields+") VALUES (CURDATE(), '"+Review+"', "+Rating+", "+UserId+", "+RestaurantId+")";

connection.query(query, function (err2, results, fields) {

if (err2) { console.log(err2); } else {

console.log("review created");

res.redirect("/restaurants/"+RestaurantId);}});}});});

**5.1.6 Menu Routes**

// NEW - form to create a new menu for the particular restaurant

app.get("/restaurants/:id/menus/new", isLoggedIn, function (req, res) {

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if (err1) { console.log(err1); } else {

var query = "SELECT \* FROM restaurant WHERE Restaurant\_id = "+req.params.id;

connection.query(query, function (err2, Restaurants, fields) {

if (err2) { console.log(err2); } else {

if(Restaurants.length<=0){

res.send("restaurant does not exist");} else {

console.log(Restaurants[0]);

res.render("menus/new",{restaurant:Restaurants[0]});}}});}});});

// CREATE - creates a new menu

app.post("/restaurants/:id/menus", isLoggedIn, function (req, res) {

var menuName = req.body.menu.Name;

var RestaurantId = req.params.id;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if (err1) { console.log(err1); } else {

var values = [[menuName, RestaurantId]];

var queryFields = "Menu\_name, Restaurant\_id";

var query = "INSERT INTO menu("+queryFields+") VALUES ?";

connection.query(query, [values], function (err2, results, fields) {

if (err2) { console.log(err2); } else {

console.log("menu created");

res.redirect("/restaurants/"+RestaurantId);}});}});});

**5.1.7 Menu item routes**

// NEW - form to create a new menu\_item for a particular restaurant

app.get("/restaurants/:restaurantId/menus/:menuId/menu\_items/new", isLoggedIn, function (req, res) {

var restaurantId = req.params.restaurantId;

var menuId = req.params.menuId;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if (err1) { console.log(err1); } else {

var query1 = "SELECT \* FROM restaurant WHERE Restaurant\_id = "+restaurantId;

connection.query(query1, function (err2, Restaurants, fields) {

if (err2) { console.log(err2); } else {

if(Restaurants.length<=0){ res.send("restaurant does not exist");

} else { var query2 = "SELECT \* FROM menu WHERE Menu\_id = " + menuId;

connection.query(query2, function (err3, menus, fields) {

if(err3){ console.log(err3); } else { if(menus.length <= 0){

res.send("menu does not exist"); } else {

res.render("menu\_items/new",{restaurant:Restaurants[0], menu:menus[0]});}} });}}});}});});

// CREATE - creates a new menu\_item

app.post("/restaurants/:restaurantId/menus/:menuId/menu\_items", isLoggedIn, function (req, res) {

var restaurantId = req.params.restaurantId;var menuId = req.params.menuId;

var Name = req.body.menuItem.Name;var Image = req.body.menuItem.Image;

var Veg = req.body.menuItem.Veg;var Price = req.body.menuItem.Price;

var Serves = req.body.menuItem.Serves;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

if(err1){ console.log(err1); }else{

var values =[[Name, Image, Veg, Price, Serves, menuId]];

var queryFields = "Name, Image, Veg, Price, Serves, Menu\_id";

var query = "INSERT INTO menu\_item("+queryFields+") VALUES ?";

connection.query(query, [values], function (err2, results, fields) {

if (err2) { console.log(err2); } else {console.log("menu\_item added");

res.redirect("/restaurants/"+restaurantId);}});}});});

**5.1.8 Auth middleware function**

function isLoggedIn (req, res, next) {

if(req.session.user){

return next();} else {res.redirect("/login");}}

**5.1.9 Run server**

app.listen(8080, function () {

console.log("FPP server is running")});

**5.2 Database**

**5.2.1 Creation of tables**

CREATE TABLE user(

User\_id INT AUTO\_INCREMENT PRIMARY KEY,

Fname VARCHAR(20) not null,

Lname VARCHAR(20) not null,

Mobile BIGINT(15) not null,

Email VARCHAR(30) not null,

Password VARCHAR(20) not null,

Address VARCHAR(100) not null,

unique(Email),

unique(Mobile));

CREATE TABLE restaurant(

Restaurant\_id INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(50) not null,

Logo VARCHAR(200) not null,

Description VARCHAR(200) not null,

Rating int(2) not null,

City varchar(30) not null,

Area varchar(30) not null);

CREATE TABLE menu(

Menu\_id INT AUTO\_INCREMENT PRIMARY KEY,

Menu\_name Varchar(50) not null,

Restaurant\_id INT,

FOREIGN KEY(Restaurant\_id) REFERENCES restaurant(Restaurant\_id) ON DELETE CASCADE);

CREATE TABLE menu\_item(

Menu\_item\_id INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(50) not null,

Image varchar(200) not null,

Veg BOOLEAN not null,

Price INT(5) not null,

Serves INT(2) not null,

Menu\_id INT not null,

FOREIGN KEY(Menu\_id) REFERENCES menu(Menu\_id) ON DELETE CASCADE );

CREATE TABLE review(

Review\_id INT AUTO\_INCREMENT PRIMARY KEY,

Review\_date DATE not null,

Review VARCHAR(100) not null,

Rating INT(2) not null,

User\_id INT not null,

Restaurant\_id INT not null,

FOREIGN KEY(User\_id) REFERENCES user(User\_id) ON DELETE CASCADE,

FOREIGN KEY(Restaurant\_id) REFERENCES restaurant(Restaurant\_id) ON DELETE CASCADE);

**5.2.2 Creation of procedures**

delimiter //

drop PROCEDURE IF EXISTS onNewRestaurant//

CREATE PROCEDURE onNewRestaurant( )

BEGIN

  Insert into menu ( Menu\_name, Restaurant\_id) values("Default Menu", (select max(Restaurant\_id) from restaurant));

END//

drop procedure IF EXISTS onNewMenu//

CREATE PROCEDURE onNewMenu()

BEGIN

INSERT into menu\_item(Name, Image, Veg, Price, Serves, Menu\_id)

VALUES("Default Name", "https://cdn.pixabay.com/photo/2017/02/01/10/16/buns-2029399\_960\_720.png", 1, 50, 2, (select max(Menu\_id) from menu));

END//

**5.2.3 Creation of triggers**

drop trigger if EXISTS onNewRestaurantTrigger//

CREATE TRIGGER onNewRestaurantTrigger

AFTER INSERT ON restaurant

FOR EACH ROW

BEGIN

CALL onNewRestaurant();

END//

drop trigger if EXISTS onNewMenuTrigger//

CREATE TRIGGER onNewMenuTrigger

AFTER INSERT ON menu

FOR EACH ROW

BEGIN

CALL onNewMenu();

END//

**Chapter 6**

**TESTING**

**6.1 Unit Testing**

The three units, namely, database unit, views unit and backend Node.js unit are tested individually before integrating them into one single web application.

* The database is tested through a number of DDL (Data Definition Language) and DML (Data Manipulation Language) commands in order to discover inconsistencies that arise regarding prime key constraints and other referential integrity constants.
* The views unit is primarily tested for its frontend functionality that is, for the user interface. It is ensured that the input forms function as they are required. The EJS pages are tested individually to avoid any discrepancies in the layout, format and style of the user interface. After linking the EJS pages to the CSS files to obtain the required style for the web application and the JavaScript files to perform logic on the front end, tests are performed again in order to ensure no inconsistency while linking.
* The Node.js unit is individually tested for proper database connectivity and to discover any errors that might arise while using the express mysql package. Queries are executed using different Statement objects with test data to ensure the right connection is established with the database and the queries yield required output.

After it is ensured that the individual unit work fine and generate the required output, they are integrated into a web application. The data that is received as input through the EJS pages is received by the server program where the request handlers perform the respective actions. The route handlers in the program have an established connection to the database where it does suitable operations on the database using the data as per the functionality selected by the user.

**6.2 Integration Testing**

After the three individual units have been integrated into a single web application, it is essential to check that the application works as a whole. The integration testing is started by testing that all the web pages can be accessed. Smooth transitions from one web page to the correct redirected page is also checked.

The forms are tested with input test data and verified that the input is received and processed by the server program that has established a connection with the database. The test data received as input is used to modify the database through the express - mysql API and SQL commands. This procedure is also tested to ensure correct access, insertion and update of the tables corresponding to the action performed by the user.

**Chapter 7**

**RESULTS**

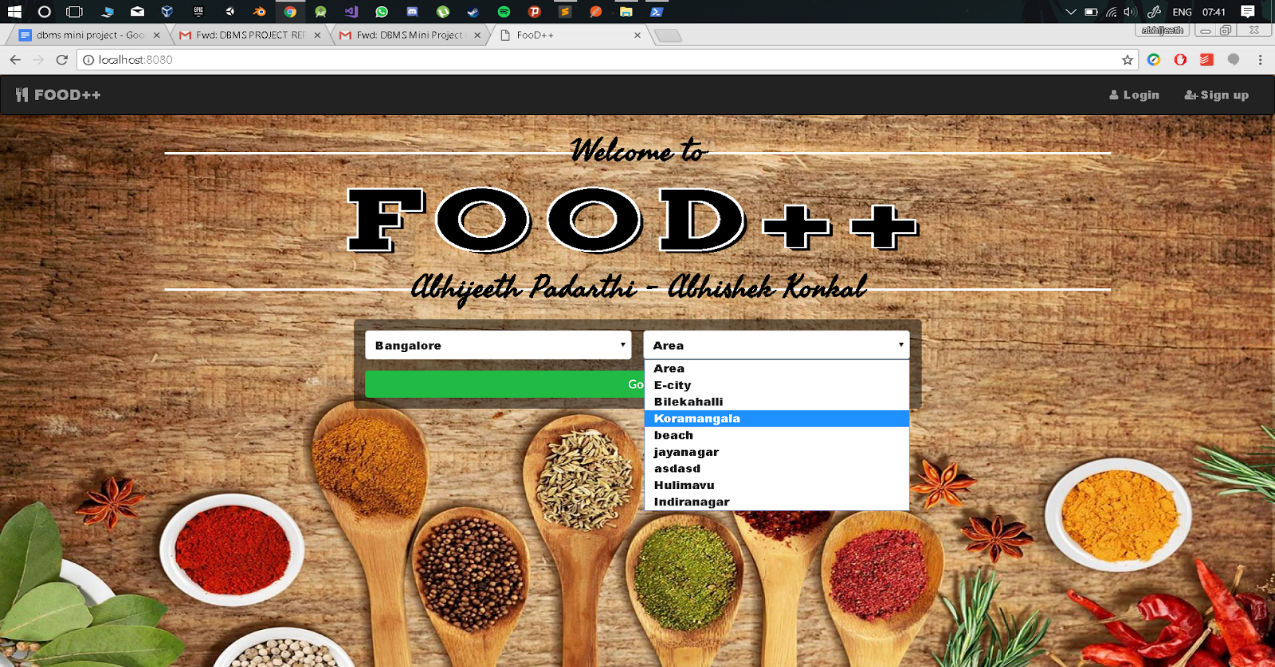


Fig 7.1 Landing page

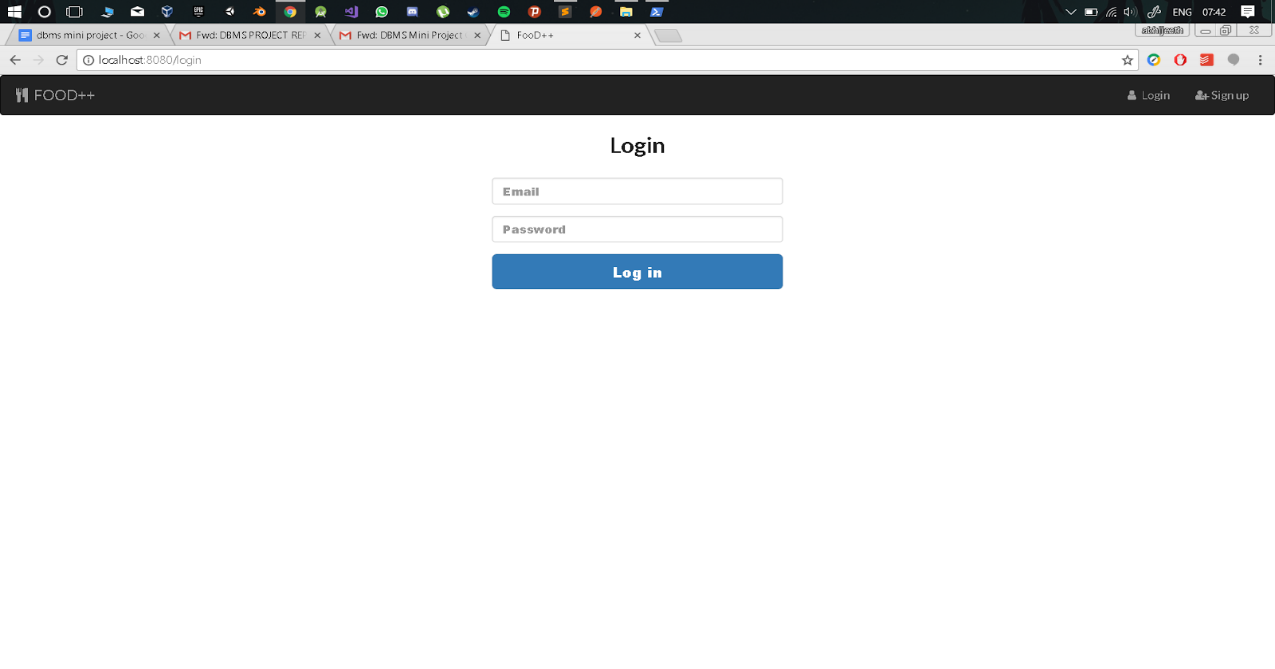


Fig 7.2 Login page

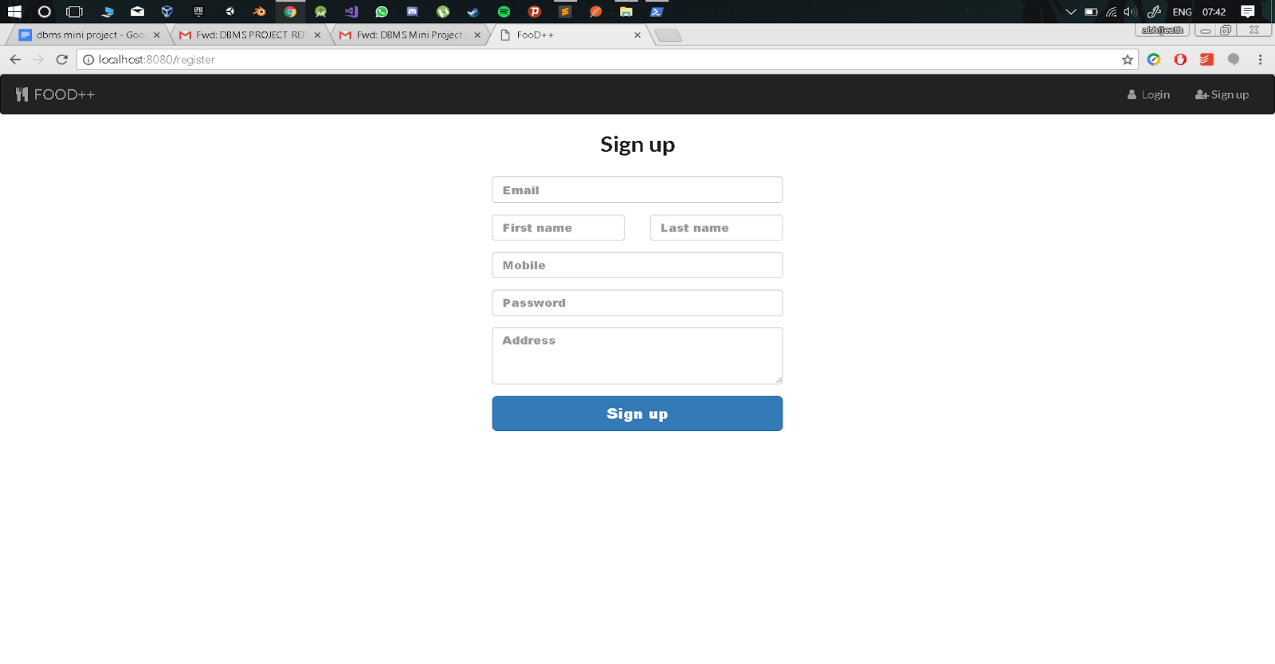


Fig 7.3 Sign up page

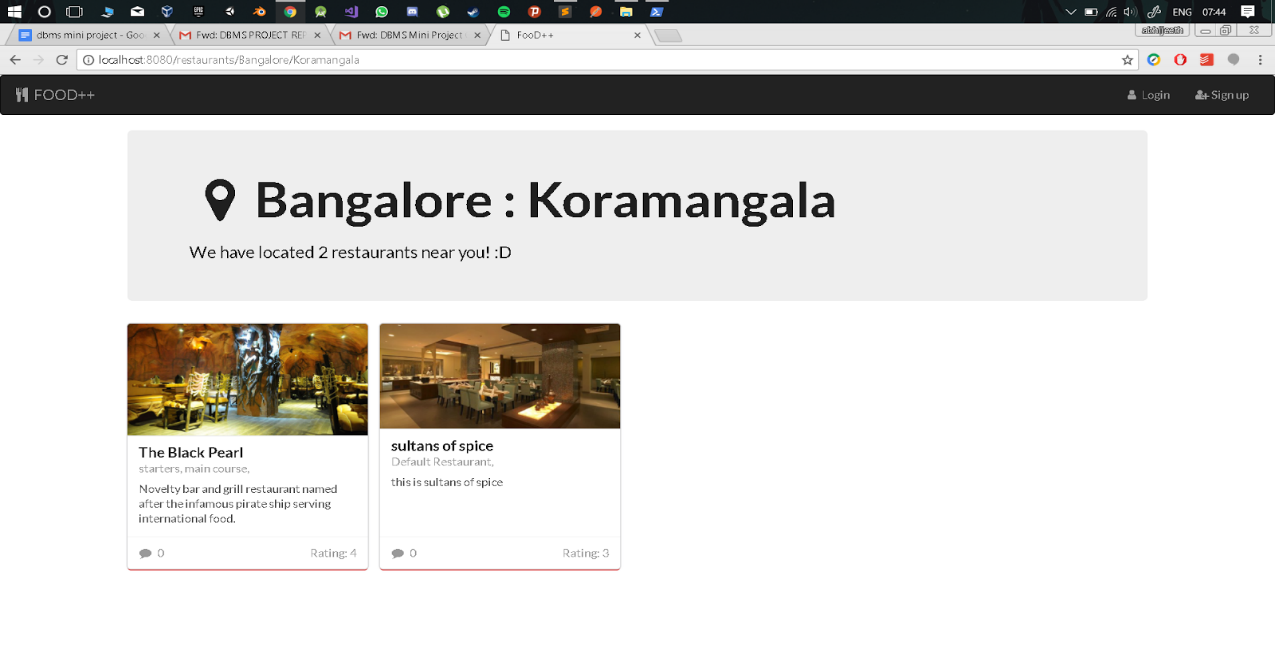


Fig 7.4 Located restaurants page

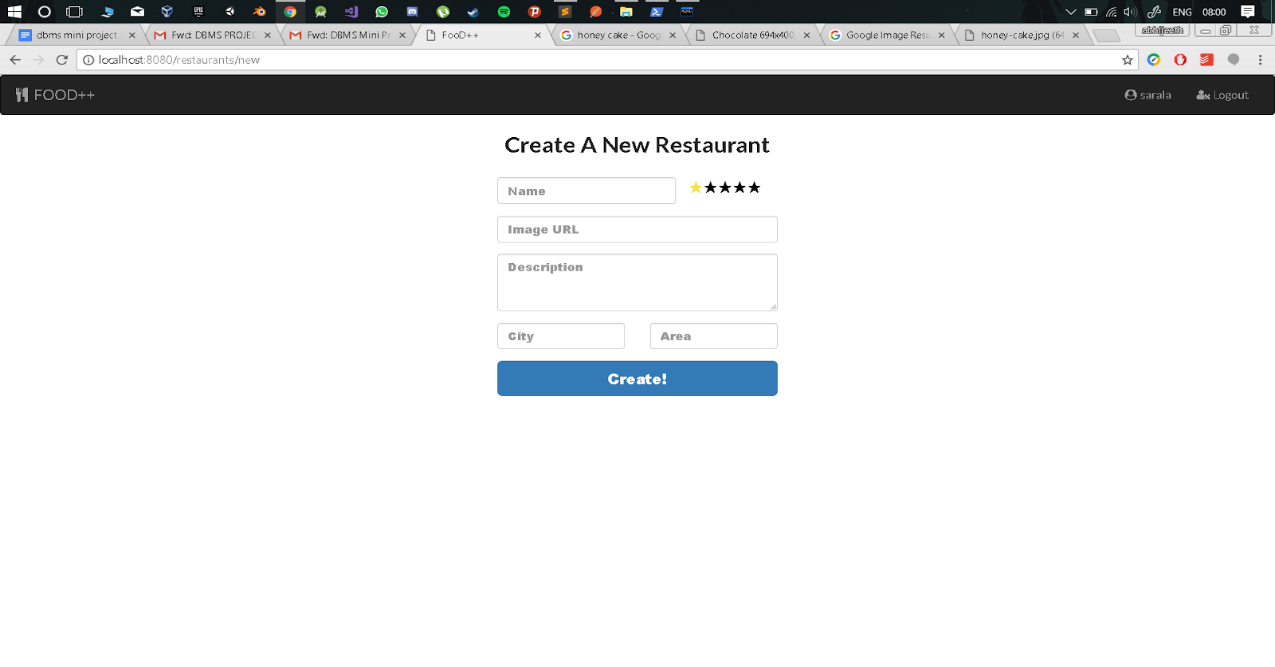


Fig 7.5 Form to create a new restaurant

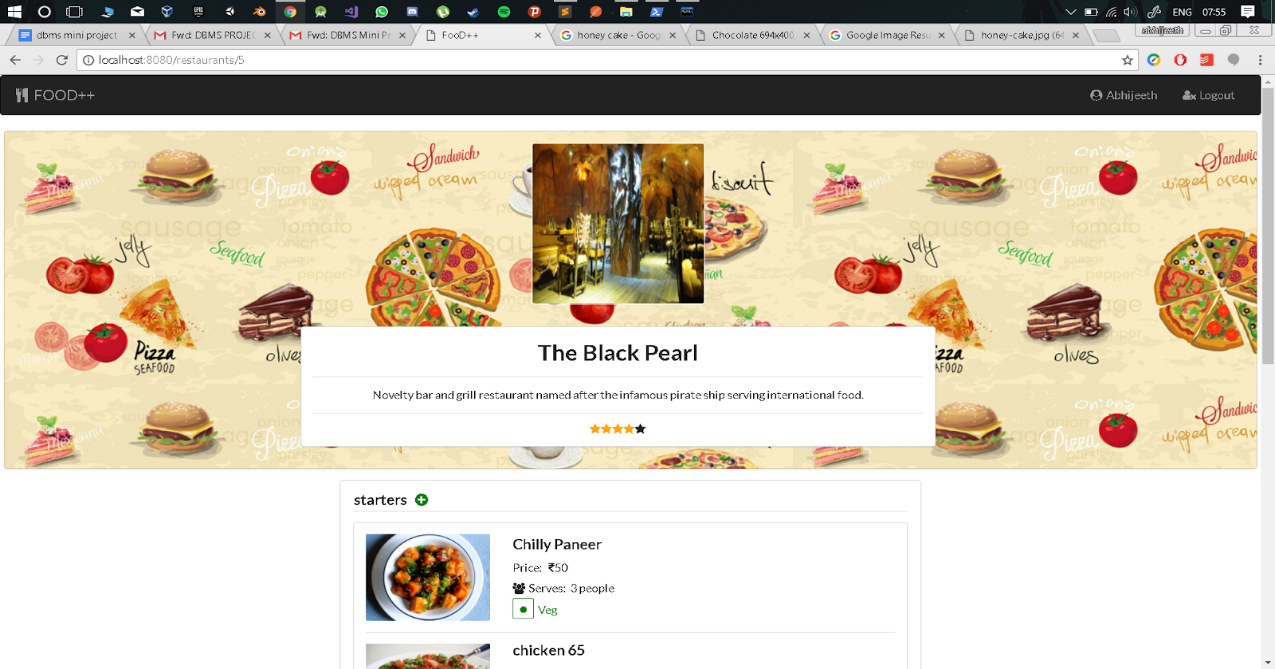


Fig 7.6 Specific restaurant page

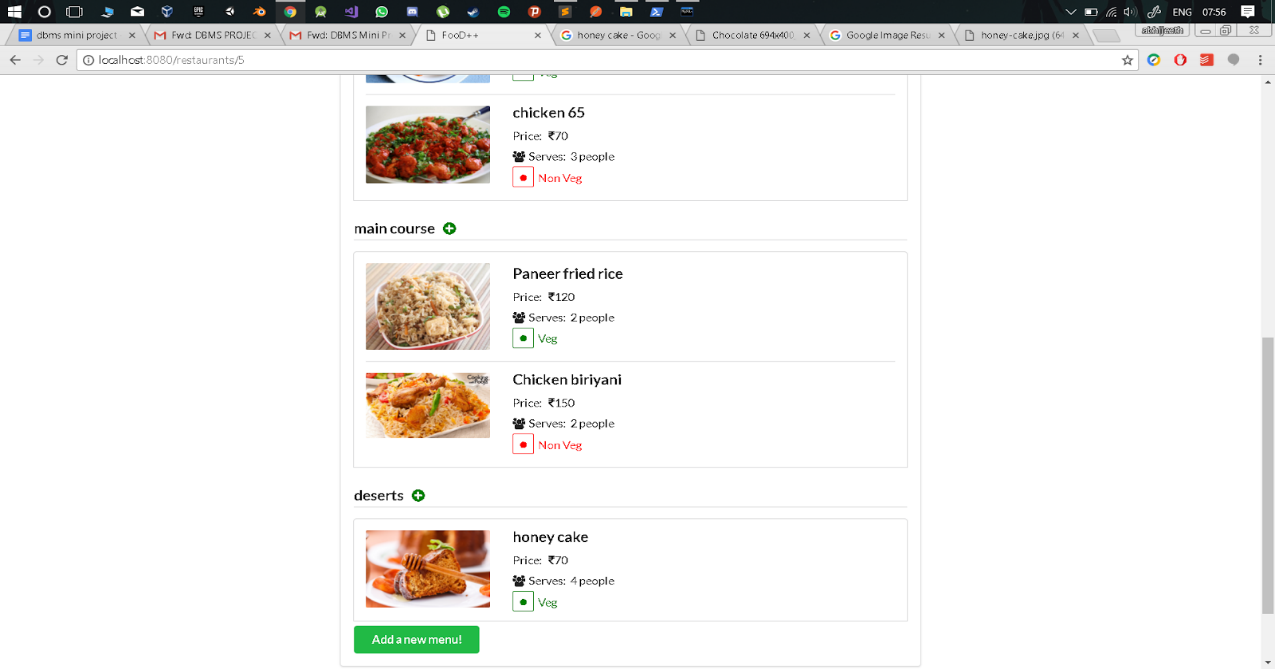


Fig 7.7 Menu of a particular restaurant

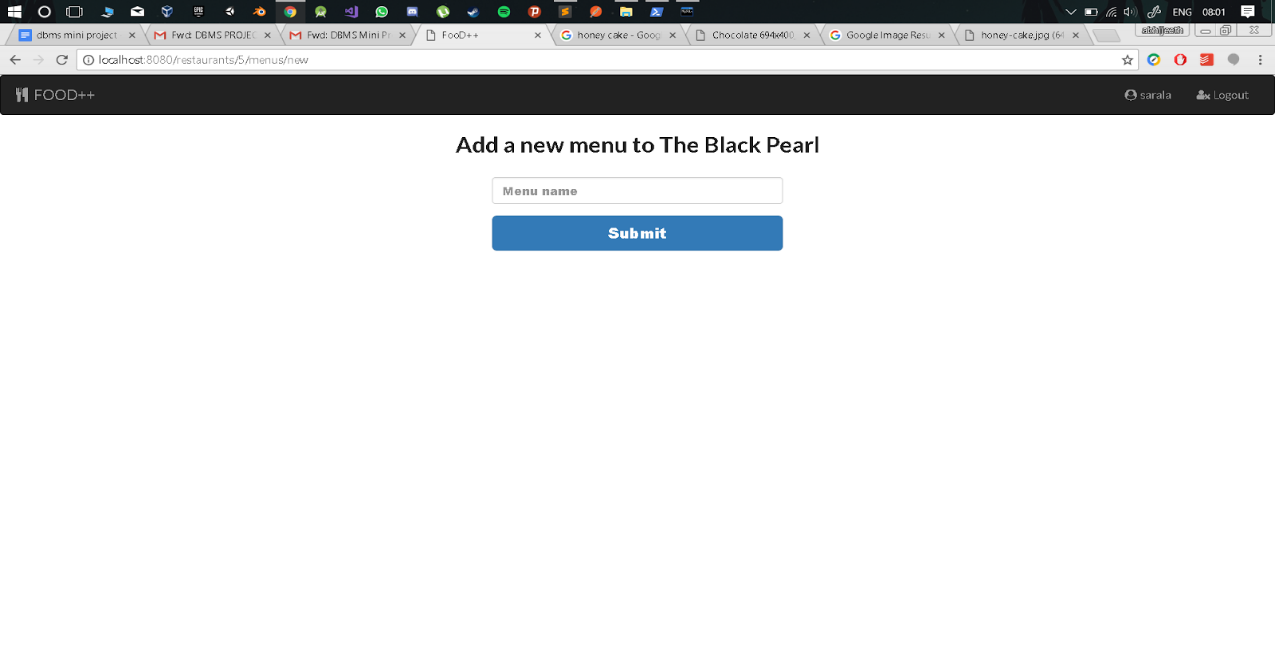


Fig 7.8 Form to add a new menu

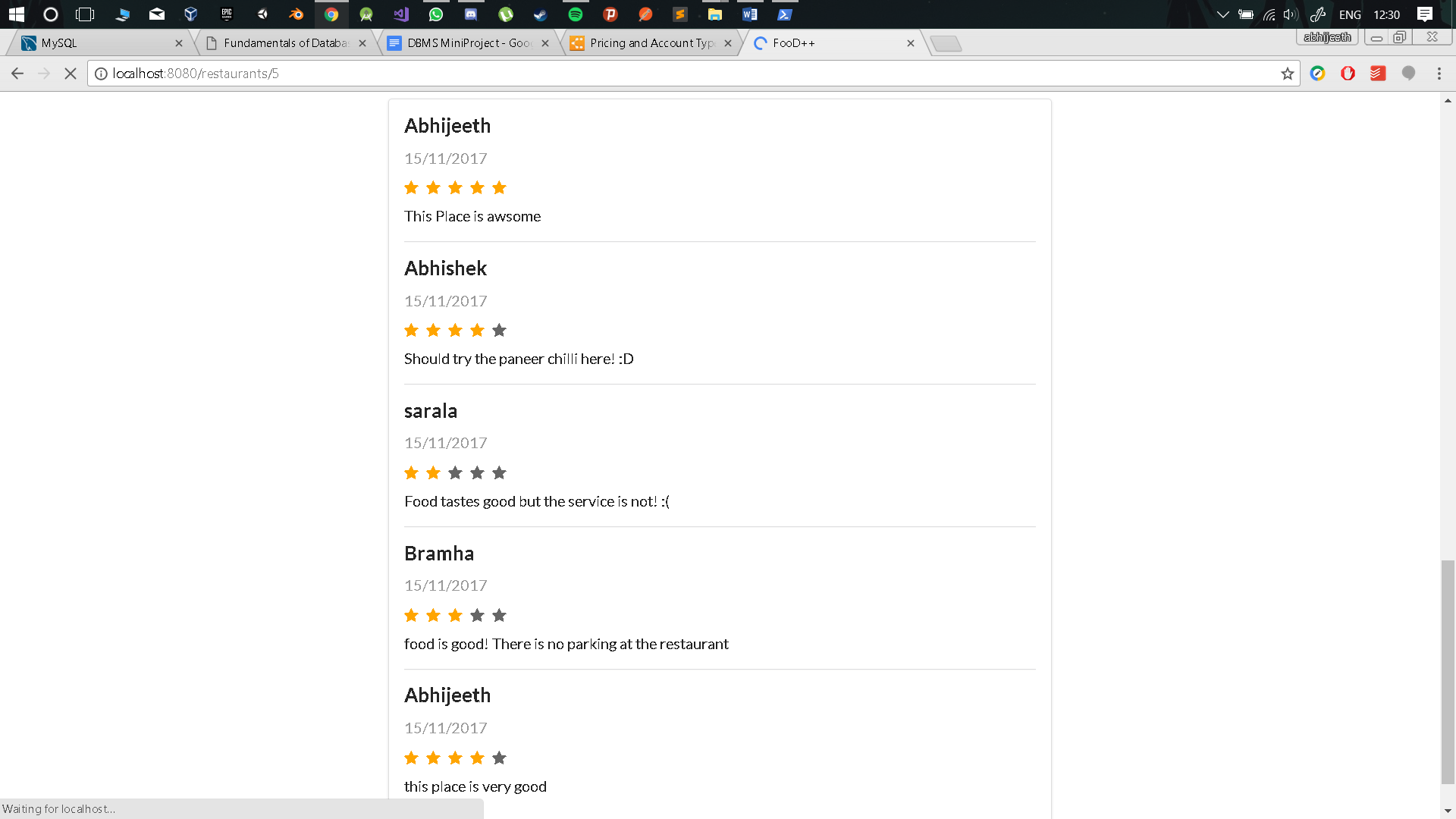


Fig 7.9 Reviews of a specific restaurant

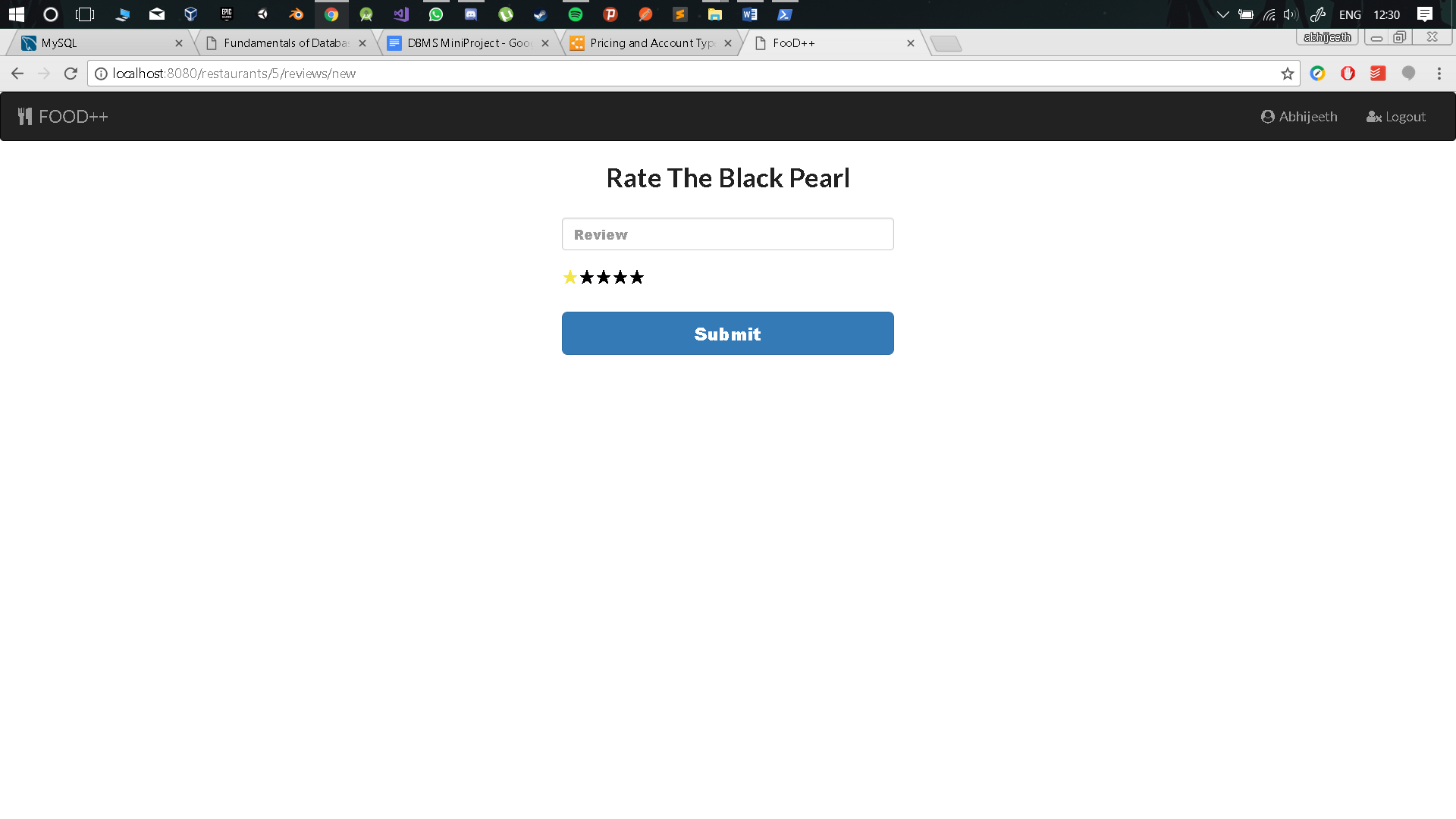


Fig 7.10 Form to add a new review

**CONCLUSION**

Our application currently supports the following functionalities. The customer upon selecting a location will be able to view all the restaurants in that particular location. Once the user has located the restaurants, he/she can click on one of the restaurants to view the Address of the restaurant, cuisines offered by the restaurant and also all dishes served at the particular restaurant. The user also has the ability to see reviews written about that restaurant by other users and also enter his/her own reviews. If the user is an admin then the user has the ability to create restaurants of his own and enter menus and dishes for these particular restaurants.

In the future we plan on building a system that connects our application with the restaurant's ordering systems so that the users can place food orders to the restaurant directly through our application. We also plan on implementing privilege levels for users such as customer, admin, etc. so that customers only have the ability to view the restaurant details, write reviews and order food, while the admins have the ability to add restaurants, menus and menu items.

**REFERENCE**

[1] https://nodejs.org/en/

[2] https://expressjs.com/

[3] https://developer.mozilla.org/en-US/

[4] https://www.foodpanda.in/

[5] https://www.zomato.com/bangalore

[6] https://www.swiggy.com/bangalore

[7] https://www.wikipedia.org/

[8] https://www.tutorialspoint.com/mysql/