## Chapter 1

## INTRODUCTION

LOGGED-INN is a full stack web application that helps users locate all the available hotels and lodges for accommodation in a particular region. It lets the users see the details about the accommodation, the rooms and suites offered by this accommodation and allows them to write reviews or rate a particular hotel/lodge.

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. JavaScript is also used to interact with the Google Maps JavaScript API to perform location services based operations. 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
* APIs : Google Maps JavaScript API
* Database : MySQL
* Server : node.js, Express
* Web browser : Google Chrome, IE8+, Mozilla Firefox

**Chapter 3**

**LITERATURE SURVEY**

**3.1 Current market**

Locating accommodations and viewing the hotel’s/lodge’s rooms and suites from home helps the user pick a accommodation he would want to spend time at depending on his comfort, requirements in terms of capacity, food preferences, time required for the accommodation to be reserved without actually visiting the accommodation. This way the user is also able to discover accommodations in a location that he never knew were present there before and make plans of travel or stay beforehand.

Online hotel booking and viewing portals are flourishing in India as people prefer to save time and pick a hotel with ease, instead of booking a room or a suite at the hotel itself. The most famous online hotel booking and viewing portals in India right now are trivago.in, goibibo.com and makemytrip.com. These portals allow users to select a location they want to spend at. They then select a particular hotel/lodge in that location and view the rooms and suites present at these particular hotels. The user then books a room for a particular duration. On performing a booking, the online portal sends the booking request to the particular hotel with details about the customer and the duration.

**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.

**3.2.4 Google Maps JavaScript API**

The Maps JavaScript API lets you customize maps with your own content and imagery for display on web pages and mobile devices.

**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.

**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.
* HOTEL: Information regarding the hotels that have signed up on our application.
* REVIEW: Reviews written by the user for a particular hotel.
* CATEGORY: Information about all the room categories offered by the hotels.
* ROOM: Information about a room that is present in a particular category.

The Schema (Fig 4.1) depicts the dependencies among the tables.

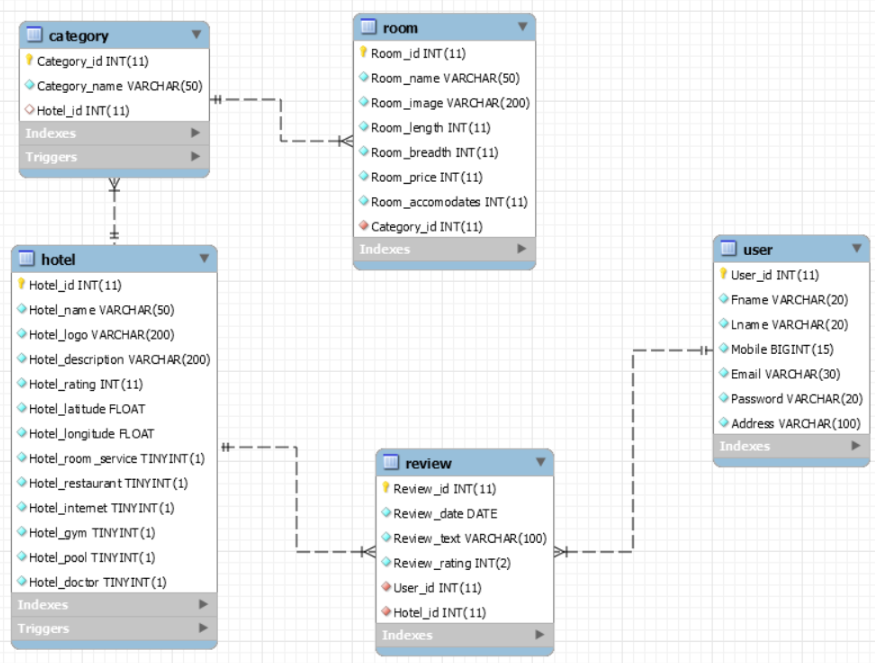


Fig 4.1 Database Schema

**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 handlers 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.

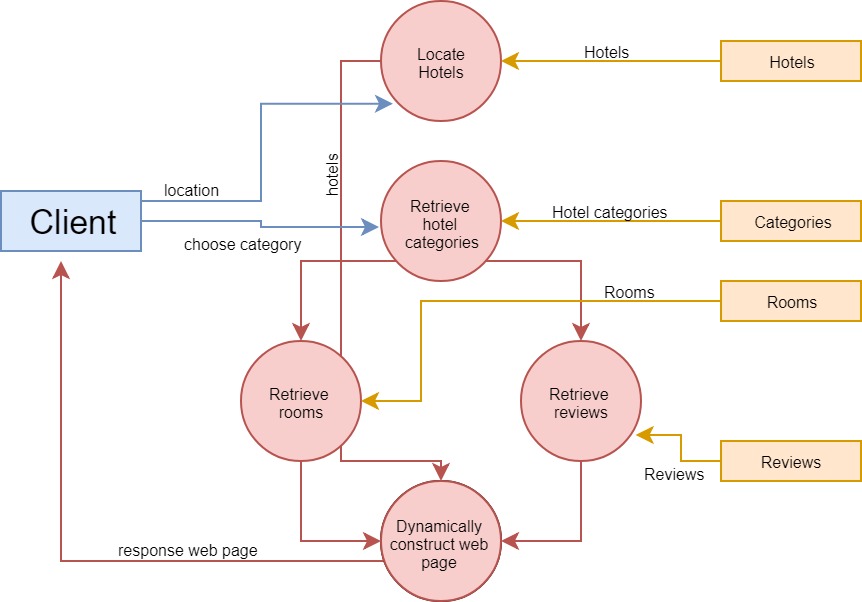


Fig 4.2 Data Flow Diagram

**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: 'sssss', database: 'loggedinn', port: 3306 };

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

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: "WEB 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) { res.render("index"); });

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

res.redirect("/hotels/"+req.body.lat+"/"+req.body.lng); });

//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 { connection.end(); res.send("email does not match password");

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

// 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");

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

// LOGOUT

app.get("/logout", function (req, res) {delete req.session.user; res.redirect("/"); });

**5.1.4 Hotel routes**

// INDEX - show all hotels

app.get("/hotels/:lat/:lng", function (req, res) {

var finalHotels = []; var semiFinalHotels = [];

var lat = Number(req.params.lat); var lng = Number(req.params.lng);

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

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

var query1 = "select \* from hotel where Hotel\_latitude > " + (lat - 0.05) +" and Hotel\_latitude < "+(lat+0.05);

query1 += " and Hotel\_longitude > " + (lng - 0.05) + " and Hotel\_longitude < "+(lng + 0.05);

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

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

forEach(hotels, function(hotel, index) {

var query2 = "SELECT Category\_name FROM category WHERE Hotel\_id = "+hotel.Hotel\_id;

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

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

hotel.categoryNames = categoryNames; semiFinalHotels.push(hotel);}});

var done = this.async();setTimeout(done, 50);},

function (notAborted) { forEach(semiFinalHotels, function (hotel, index) {

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

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

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

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

finalHotels.push(hotel);}});

var done = this.async();setTimeout(done, 50);},

function (notAborted2) {

console.log(JSON.stringify(finalHotels));

connection.end();

res.render("hotels/index",{hotels: finalHotels}); });});}});}});});

// NEW - form to create a new hotel

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

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

// CREATE - creates a new hotel

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

var Name = req.body.hotel.Name; var Logo = req.body.hotel.Logo;

var Description = req.body.hotel.Description;

var Rating = req.body.hotel.Rating;

var lat = req.body.hotel.Lat; var lng = req.body.hotel.Lng;

var roomService = req.body.hotel.RoomService;

var restaurant = req.body.hotel.Restaurant;

var internet = req.body.hotel.Internet; var gym = req.body.hotel.Gym;

var pool = req.body.hotel.Pool; var doctor = req.body.hotel.Doctor;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

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

var values = [[Name, Logo, Description, Rating, lat, lng, roomService, restaurant, internet, gym, pool, doctor]];

var queryFields = "Hotel\_name, Hotel\_logo, Hotel\_description, Hotel\_rating, Hotel\_latitude, Hotel\_longitude, Hotel\_room\_service, Hotel\_restaurant, Hotel\_internet, Hotel\_gym, Hotel\_pool, Hotel\_doctor";

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

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

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

else {console.log("Hotel Added"); connection.end(); res.redirect("/");

}});}});});

// SHOW - shows more information about one hotel

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

hotelId = req.params.id;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

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

var query1 = "SELECT \* FROM hotel where Hotel\_id = "+hotelId;

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

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

else { hotel = hotel[0];

var query2 = "SELECT \* FROM category WHERE Hotel\_id = " + hotel.Hotel\_id;

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

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

else{ hotel.categories = categories;

forEach(hotel.categories, function (category, index) {

var query3 = "SELECT \* FROM room WHERE Category\_id = " + category.Category\_id;

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

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

hotel.categories[index].rooms = rooms;}});

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.Hotel\_id = "+hotel.Hotel\_id+" and r.User\_id = u.User\_id";

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

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

hotel.reviews = reviews; console.log(JSON.stringify(hotel));

connection.end();res.render("hotels/show", {hotel:hotel});

}});});}});}});}});});

**5.1.5 Reviews routes**

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

app.get("/hotels/: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 hotel WHERE Hotel\_id = "+req.params.id;

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

if (err2) { console.log(err2); } else {if(hotels.length<=0){

res.send("hotel does not exist");} else {connection.end();

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

}}});}}); });

// CREATE - creates a new review

app.post("/hotels/: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 hotelId = req.params.id;

var connection = mysql.createConnection(connectionObject);

connection.connect(function (err1) {

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

var queryFields = "Review\_date, Review\_text, Review\_rating, User\_id, Hotel\_id"; var query = "INSERT INTO review("+queryFields+") VALUES (CURDATE(), '"+Review+"', "+Rating+", "+UserId+", "+hotelId+")";

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

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

console.log("review created"); connection.end();

res.redirect("/hotels/"+hotelId); }});}});});

**5.1.6 Auth middleware function**

function isLoggedIn (req, res, next) {

if(req.session.user){ return next();} else {res.redirect("/login");}}

**5.1.7 Run server**

app.listen(8080, function () { console.log("server is running")});

**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**

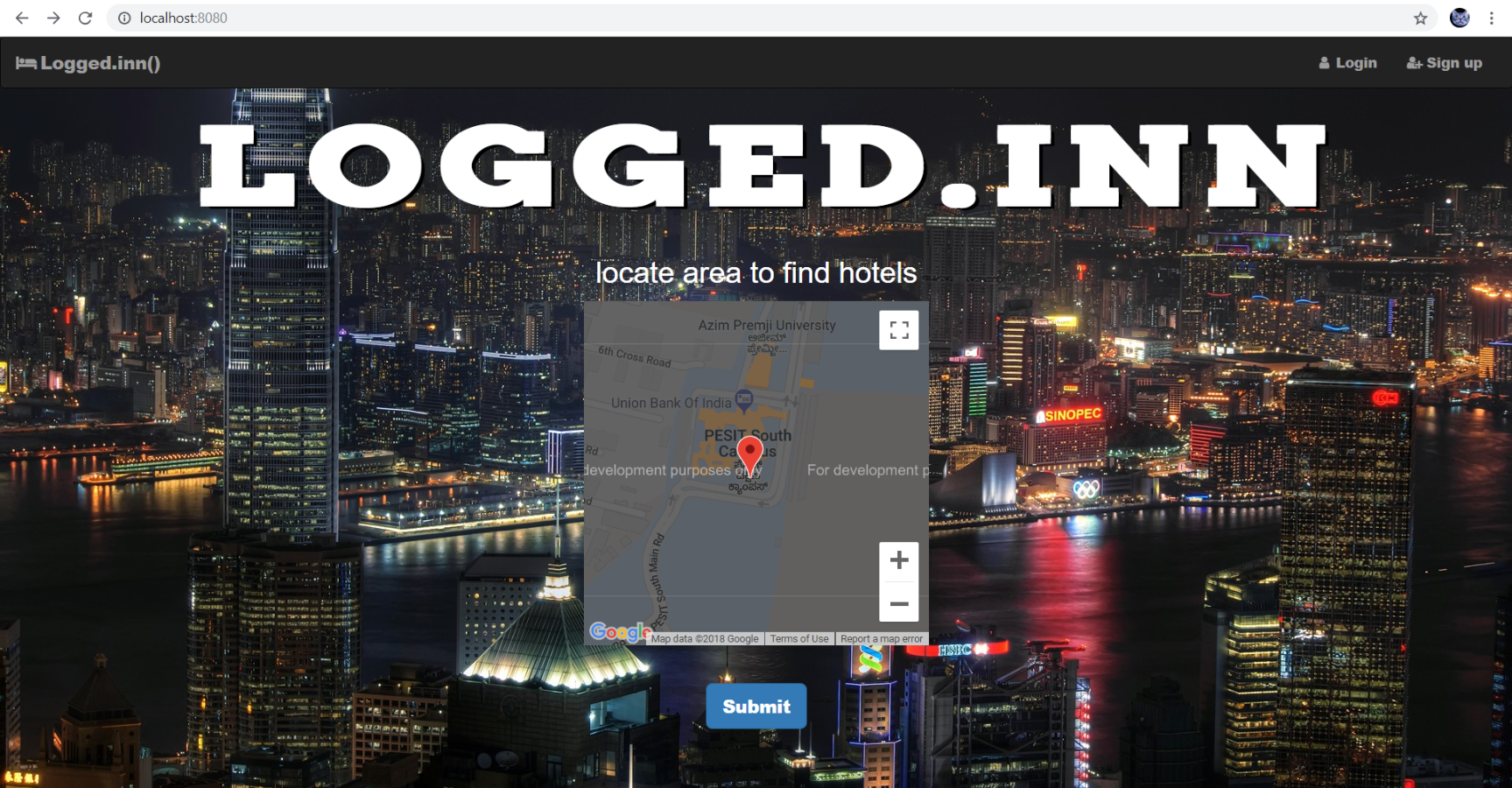


Fig 7.1 Landing page

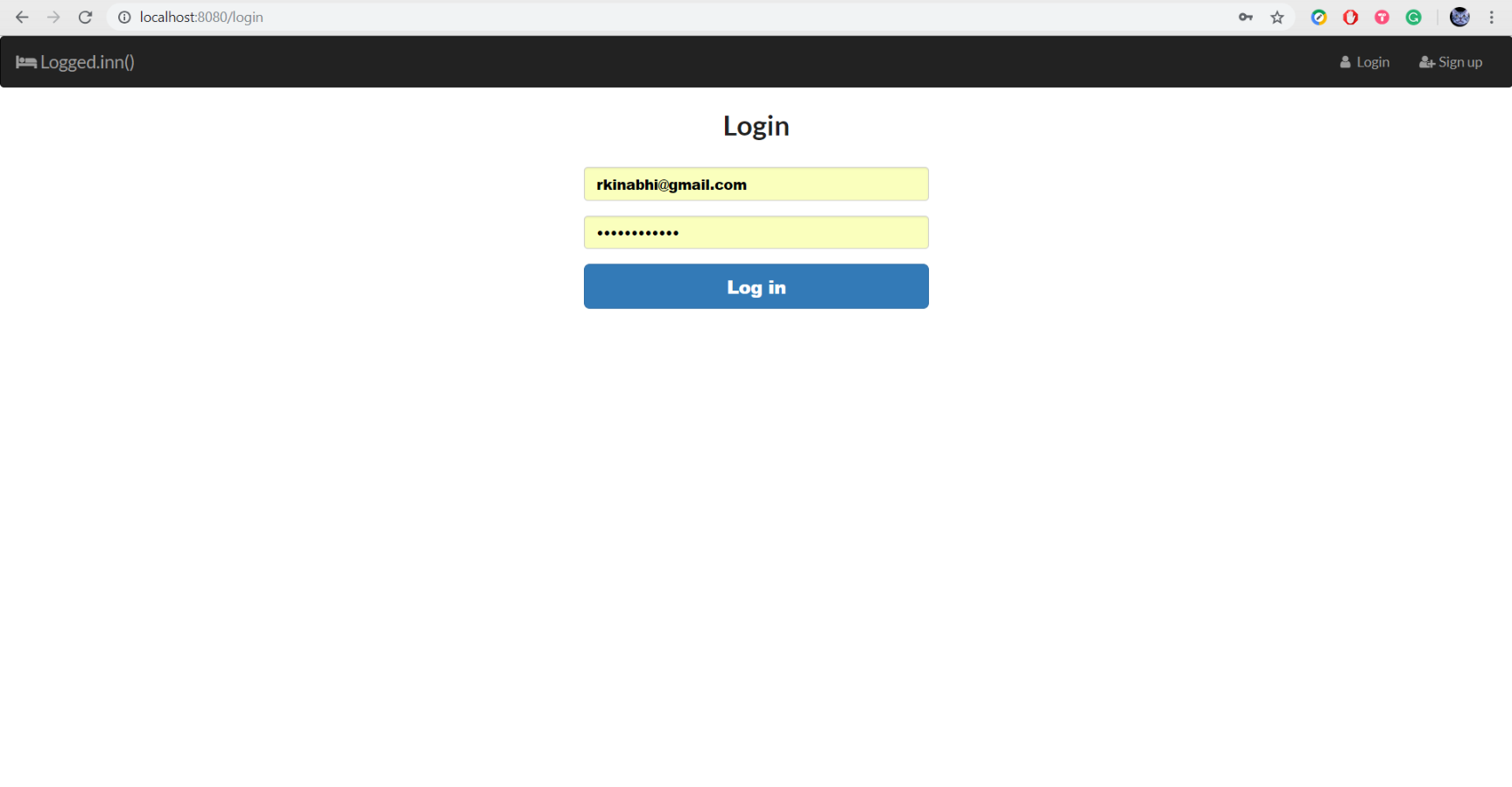


Fig 7.2 Login page

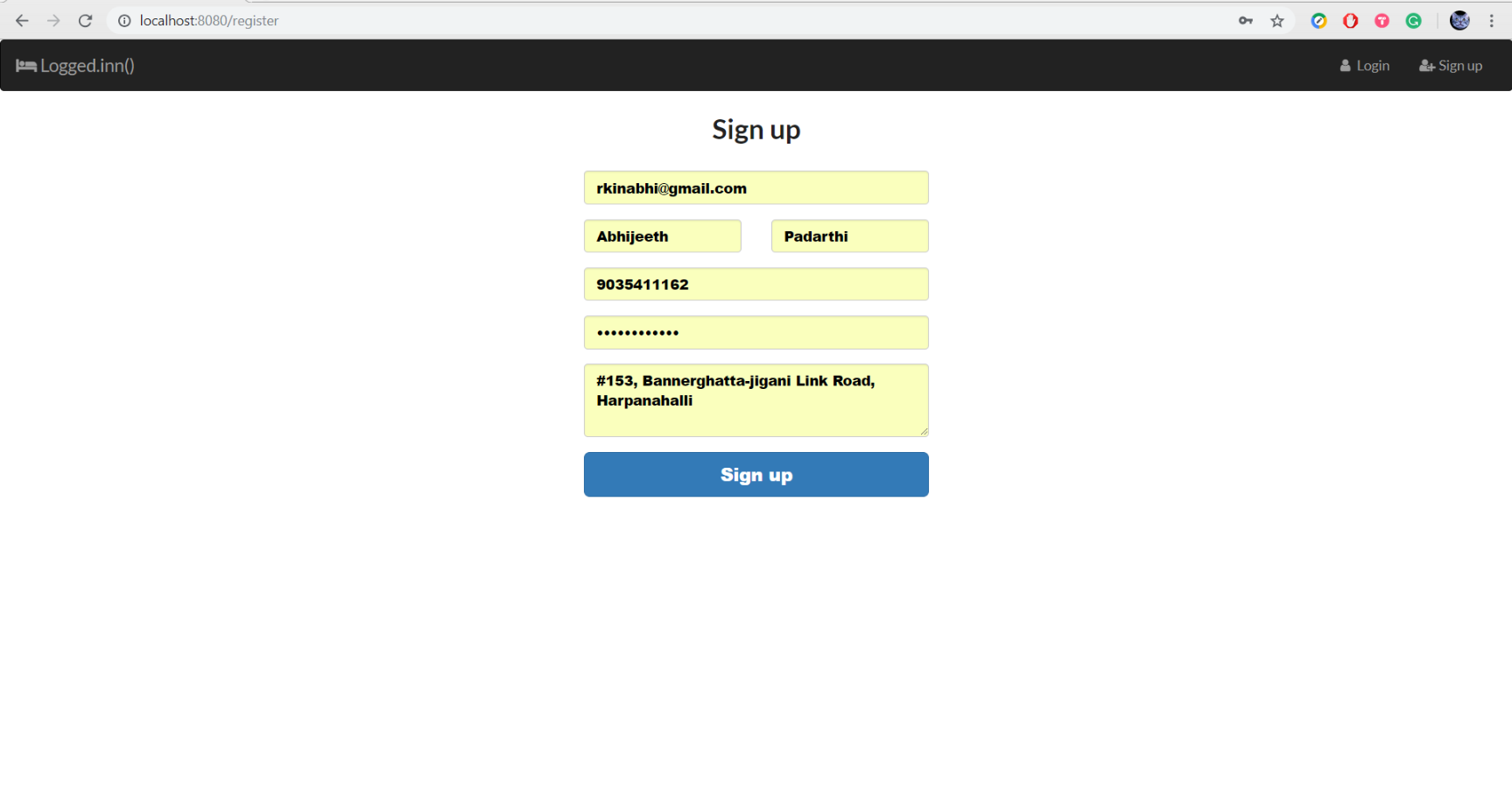


Fig 7.3 Sign up page

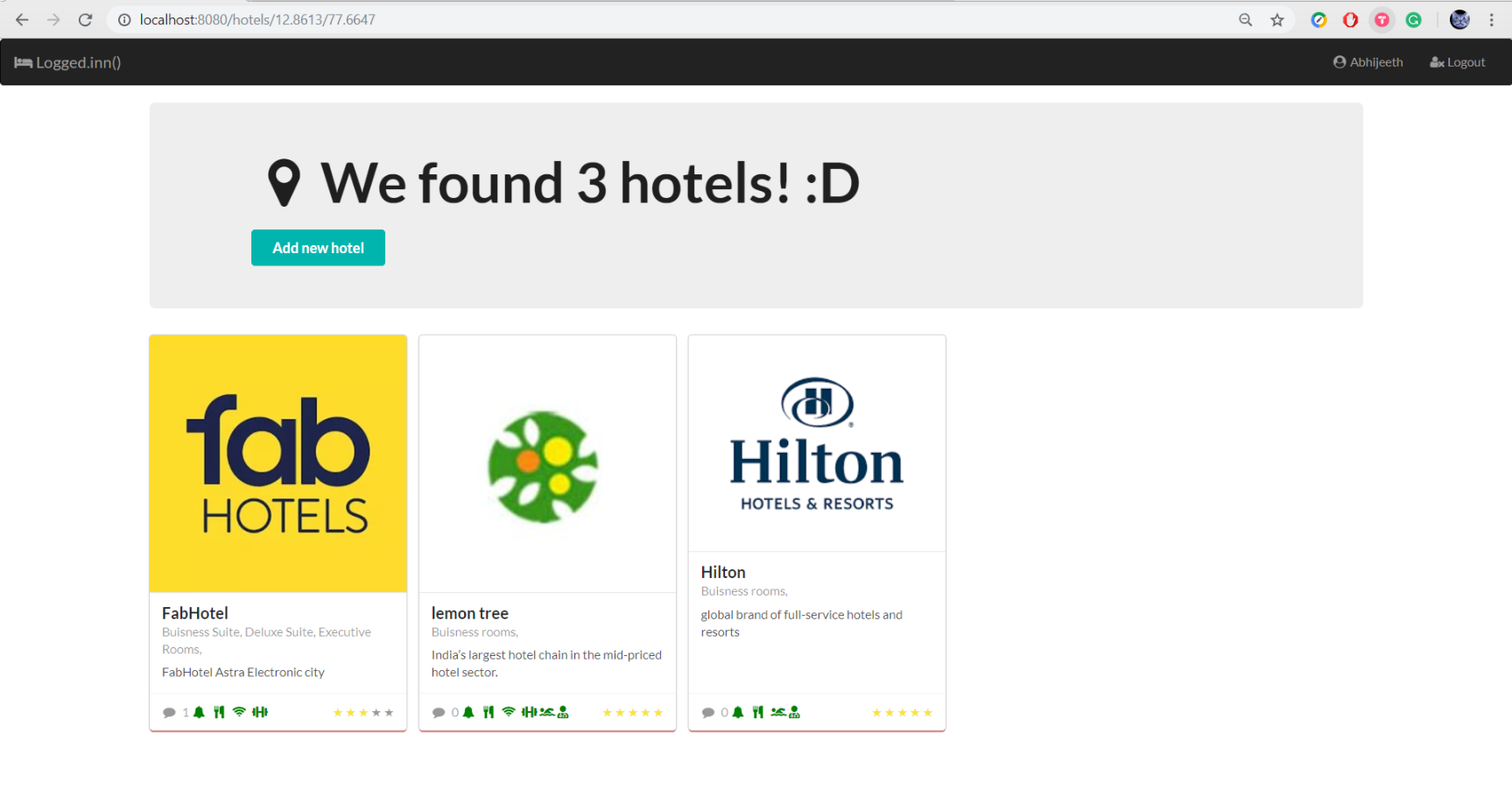


Fig 7.4 Located hotels page

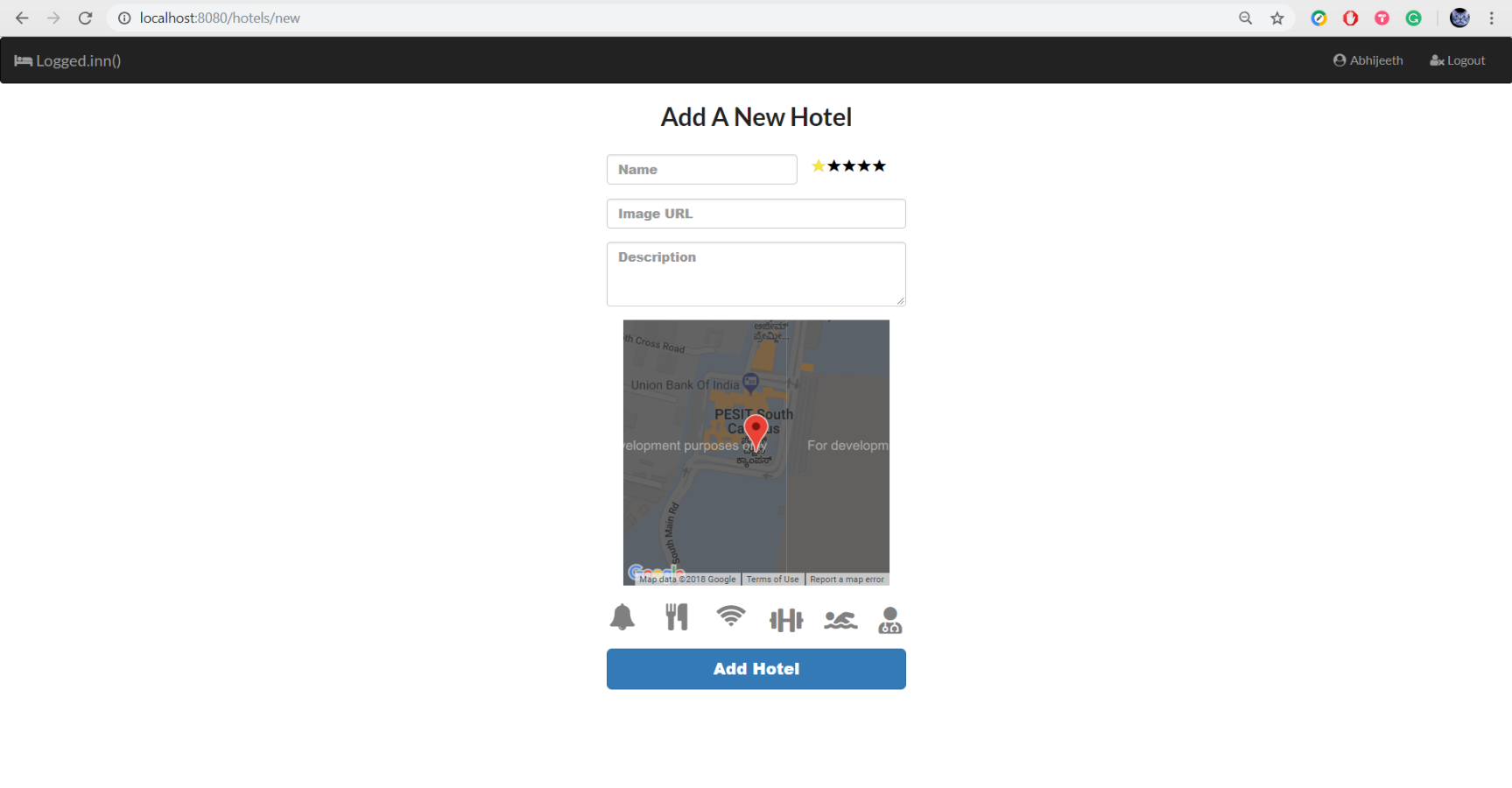


Fig 7.5 Form to create a new hotel

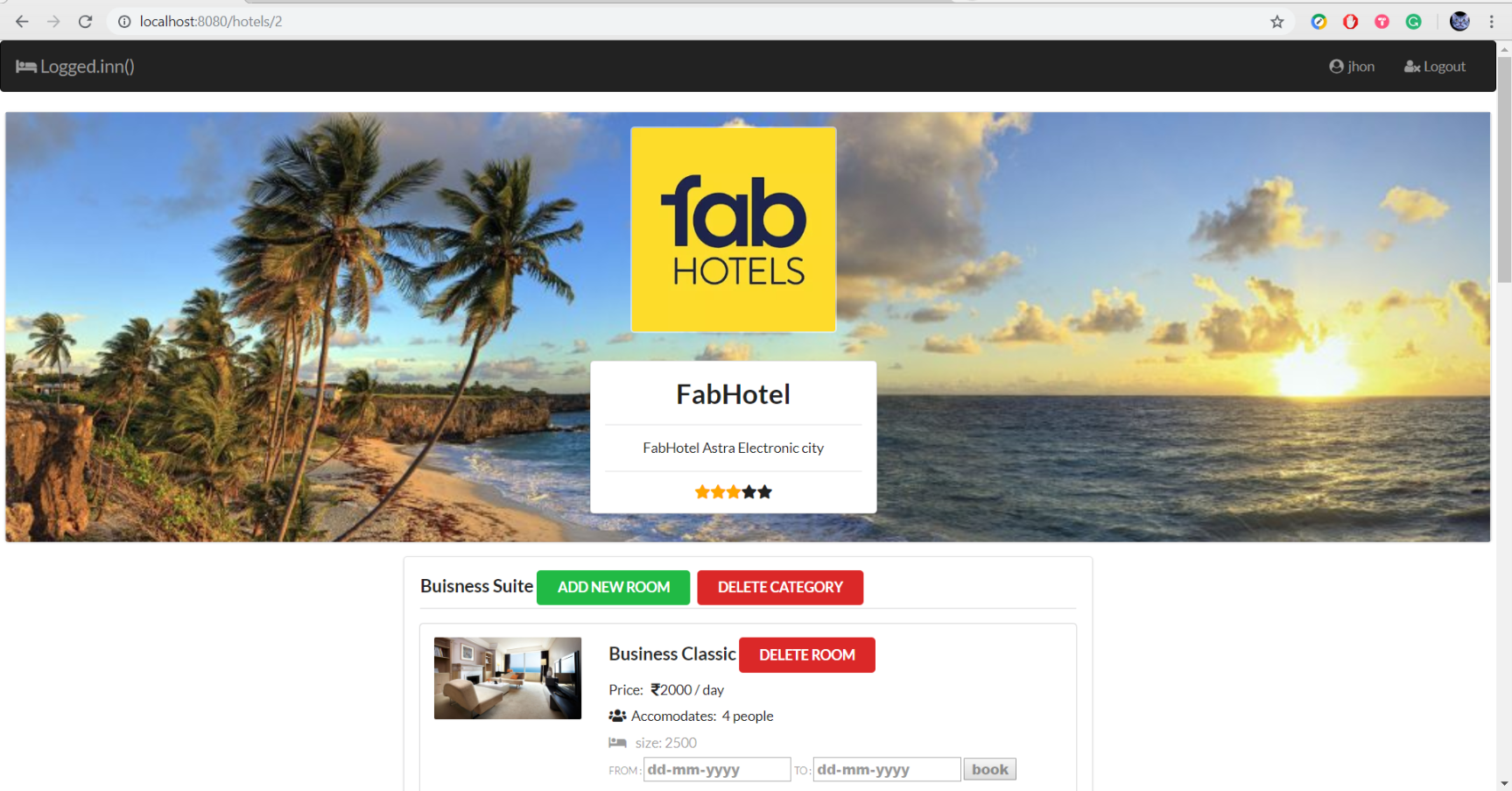


Fig 7.6 Specific hotel page

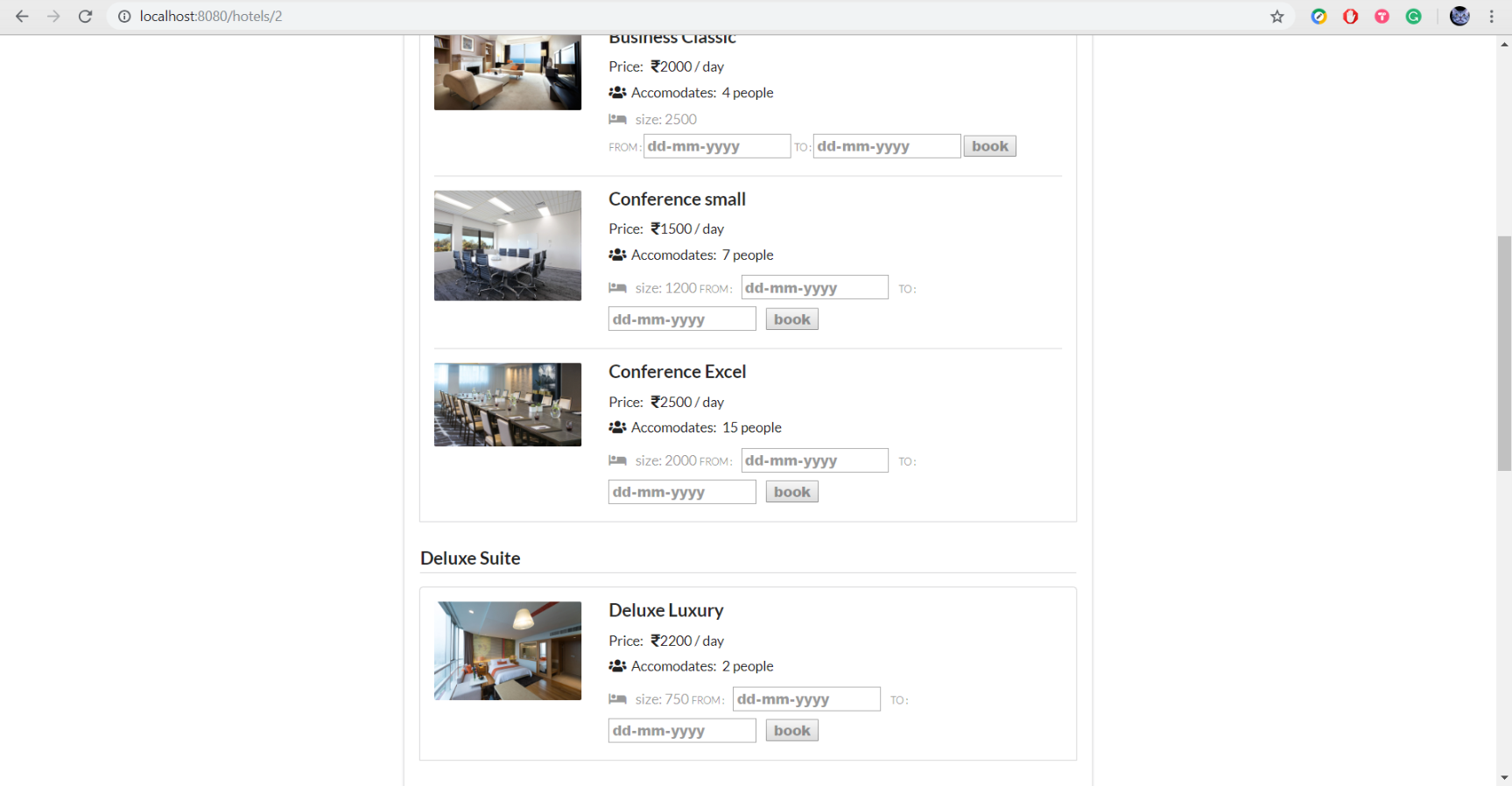


Fig 7.7 Rooms of a particular hotel

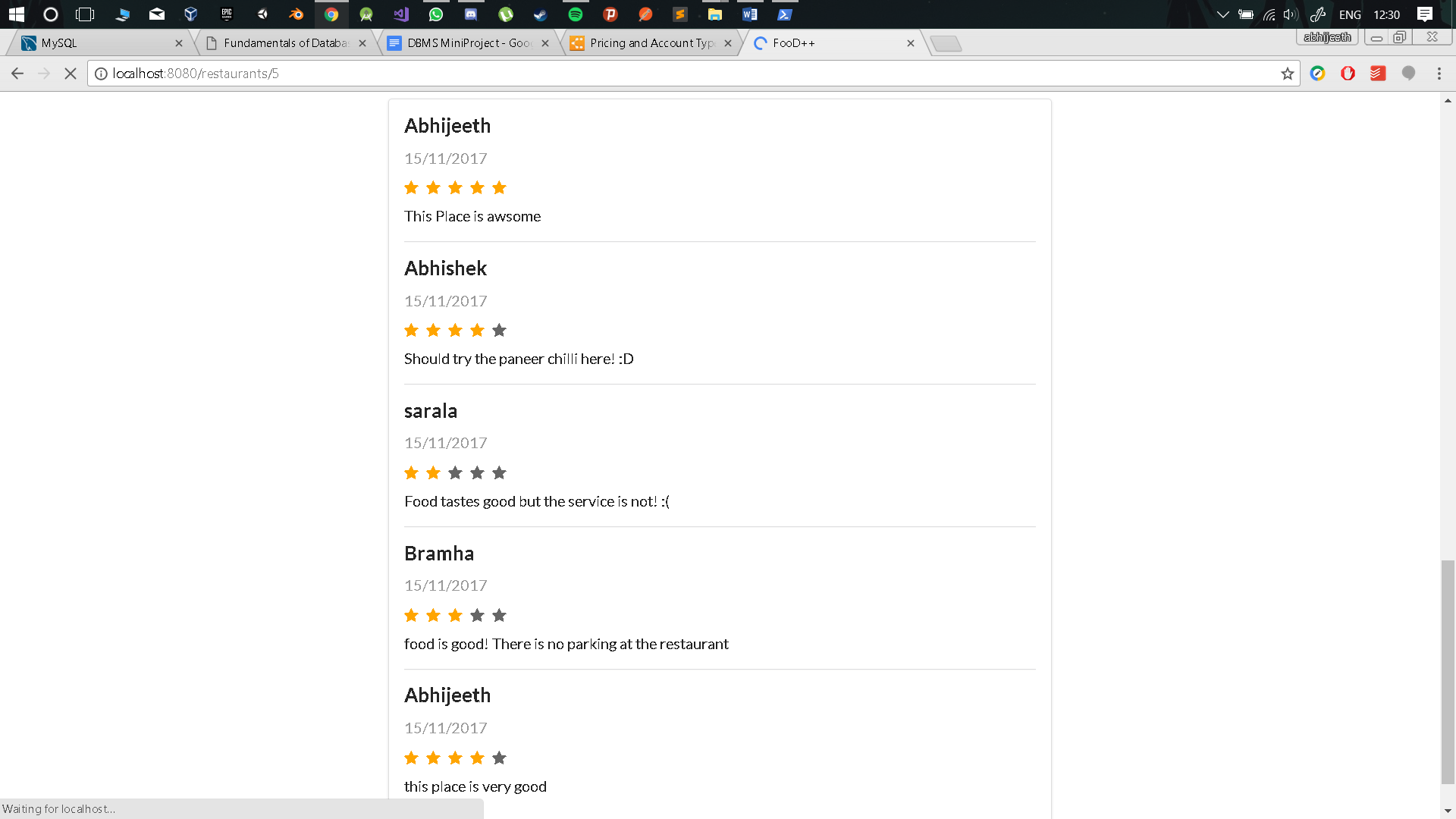


Fig 7.8 Reviews of a specific hotel

**CONCLUSION**

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

**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/>

[9] https://developers.google.com/maps/documentation/javascript/tutorial