

**KGiSL INSTITUTE OF TECHNOLOGY**

(Approved By AICTE, New Delhi, Affiliate to Anna University

Recognized by UGC, Accredited by NBA(IT)

265, KGISL Campus, Thudiyalur Road, Saravanampatti, Coimbatore-641035**.)**

**DEPARTMENT OF**

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**NAAN MUDHALVAN - INTERNET OF THINGS**

**SMART PARKING**

**NAME:** KEERTHANA.R

**REG NO:** 711721243046

**NM ID:** au711721243046

**TEAM MENTOR:** Mr**.** Mohankumar M

**TEAM EVALUATOR:** Ms. Akilandeeshwari M

**Phase 4: Development Part 2**

**Problem Statement:**

Our challenge is to develop a smart parking solution using IoT technology. We aim to monitor real-time parking space occupancy, offer dynamic parking guidance to users, and seamlessly integrate these features into a mobile app. The ultimate goal is to enhance the efficiency and convenience of public parking services, alleviating the common difficulties of finding available parking spaces in urban areas.

**DESIGNING AN APP FUNCTIONS TO RECEIVE AND DISPLAY PARKING AVAILABILITY DATA RECEIVED FROM THE RASBERRY PI.**

**PROCEDURE:**

**1. Set up the Raspberry Pi:**

Ensure Raspberry Pi is up and running, connected to the internet, and accessible via its IP address or hostname on local network.

**2. Install Node.js and Express.js on the Raspberry Pi:**

If not already installed, install Node.js by following the instructions for your Raspberry Pi's operating system.

**3. Create a Node.js Server:**

Create a new JavaScript file in project directory to define the server.

**4. Install Express.js:**

In project directory, install the Express.js framework.

**5. Start the Server:**

Start the Node.js server on Raspberry Pi.

**6. Create the HTML/JavaScript Web Application:**

Create an HTML file for web application on a separate computer, using a text editor or integrated development environment.

Replace `'raspberry-pi-ip'` with the IP address or hostname of your Raspberry Pi in the HTML/JavaScript code

Save the HTML file and open it in a web browser to ensure the client-side code is working as expected. The parking availability should display "Loading..." initially, and it should update periodically.

**7. Deploy the HTML/JavaScript Web Application:**

Deploy the HTML/JavaScript web application to a web server, a web hosting service, or simply access it from a computer on the same local network as the Raspberry Pi.

**8. Test the Full System:**

Access the web application from your client device and check if it displays real-time parking availability data obtained from your Raspberry Pi. The data should update periodically.

**9. Replace Simulated Data with Real Data:**

Replace the simulated data retrieval logic in the Raspberry Pi server code with actual data source. This could involve sensors, IoT devices, or other methods for tracking parking space availability.

**CODE:**

var createError = require('http-errors');

var express = require('express');

var path = require('path');

var cookieParser = require('cookie-parser');

var logger = require('morgan');

var methodoverride = require('method-override');

var hbs = require('hbs');

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

var connection = require('./models');

var indexRouter = require('./routes/index');

var usersRouter = require('./routes/users');

var carsRouter = require('./routes/cars');

var app = express();

// view engine setup

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'hbs');

// Helpers hbs

hbs.registerHelper('equals', (val1, val2, options) => {

return val1 == val2 ? options.fn(this) : options.inverse(this);

});

app.use(session({

secret: 'parkingsystem',

}));

app.use(logger('dev'));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(methodoverride((req, res, next) => {

if(req.body && typeof req.body == 'object' && req.body.\_method) {

var method = req.body.\_method;

delete req.body.\_method;

return method;

}

}));

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/', indexRouter);

app.use('/users', usersRouter);

app.use('/cars', carsRouter);

// catch 404 and forward to error handler

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

next(createError(404));

});

// error handler

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

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get('env') === 'development' ? err : {};

// render the error page

res.status(err.status || 500);

res.render('error');

});

module.exports = app;

{

"name": "parking",

"version": "0.0.0",

"lockfileVersion": 1,

"requires": true,

"dependencies": {

"accepts": {

"version": "1.3.5",

"resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.5.tgz",

"integrity": "sha1-63d99gEXI6OxTopywIBcjoZ0a9I=",

"requires": {

"mime-types": "2.1.19",

"negotiator": "0.6.1"

}

},

"align-text": {

"version": "0.1.4",

"resolved": "https://registry.npmjs.org/align-text/-/align-text-0.1.4.tgz",

"integrity": "sha1-DNkKVhCT810KmSVsIrcGlDP60Rc=",

"requires": {

"kind-of": "3.2.2",

"longest": "1.0.1",

"repeat-string": "1.6.1"

}

}

{

"name": "parking",

"version": "0.0.0",

"private": true,

"scripts": {

"start": "node ./bin/www"

},

"dependencies": {

"cookie-parser": "~1.4.3",

"debug": "~2.6.9",

"express": "~4.16.0",

"express-session": "^1.15.6",

"hbs": "~4.0.1",

"http-errors": "~1.6.2",

"method-override": "^3.0.0",

"mongoose": "^5.2.4",

"morgan": "~1.9.0"

}

}

var Car = require('./../models/car');

exports.find = (req, res) => {

Car.find({}, (err, cars) => {

if(err) {

return;

}

res.render('cars\_list', {

cars: cars

});

});

}

exports.new = (req, res) => {

res.render('cars\_new');

}

exports.create = (req, res) => {

Car.create(req.body, (err, car) => {

if(err) {

return;

}

res.redirect('/cars');

});

}

exports.edit = (req, res) => {

Car.findById(req.params.id, (err, car) => {

if(err) {

return;

}

res.render('cars\_edit', {

car: car

});

});

}

exports.update = (req, res) => {

Car.update({

\_id: req.params.id

}, req.body, (err, car) => {

if(err) {

return;

}

res.redirect('/cars');

});

}

exports.remove = (req, res) => {

Car.remove({

\_id: req.params.id

}, (err) => {

if(err) {

return;

}

res.redirect('/cars');

});

}

exports.index = (req, res) => {

res.render('index', {

user: req.session.user

});

}

var User = require('./../models/users');

exports.login = (req, res) => {

res.render('login');

}

exports.signin = (req, res) => {

User.findOne({

username: req.body.username,

password: req.body.password

}, (err, user) => {

if(err) {

return;

}

req.session.user = {

username: user.username

}

res.redirect('/');

});

}

exports.register = (req, res) => {

res.render('register');

}

exports.create = (req, res) => {

User.create(req.body, (err, user) => {

if(err) {

return;

}

res.redirect('/users/login');

});

}

**CONCLUSION:**

In conclusion, building a smart parking system using IoT sensors and Raspberry Pi integration is a valuable project that offers solutions to urban parking challenges. By following the step-by-step procedure and code outlined above, we can create a reliable and efficient parking management system.