Contents

[NodeJS 2](#_Toc83143763)

[Browsers using JS engine 2](#_Toc83143764)

[Node.js Frameworks and Tools 3](#_Toc83143765)

[Module 4](#_Toc83143766)

[Global variable 5](#_Toc83143767)

[Modules and require() 5](#_Toc83143768)

[Using module exporting many properties 5](#_Toc83143769)

[Reading files 6](#_Toc83143770)

[Setting up Server 6](#_Toc83143771)

[Route 8](#_Toc83143772)

[Template engine 8](#_Toc83143773)

[Partials 9](#_Toc83143774)

[Middleware 10](#_Toc83143775)

[Query string 10](#_Toc83143776)

[Template engine form 12](#_Toc83143777)

# NodeJS

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!

A Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.

Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.

When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.

This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.

Node.js has a unique advantage because millions of frontend developers that write JavaScript for the browser are now able to write the server-side code in addition to the client-side code without the need to learn a completely different language.

In Node.js the new ECMAScript standards can be used without problems, as you don't have to wait for all your users to update their browsers - you are in charge of deciding which ECMAScript version to use by changing the Node.js version, and you can also enable specific experimental features by running Node.js with flags.

## Browsers using JS engine

Microsoft IE => Chakra

Firefox => SpiderMonkey

Chrome => V8

Ryan Dahl separated V8 engine along with c++ and run outside browser.

In browser we use

document.getElementById(“”);

but it wont work in Notejs.

But node has its own specific objects like

Fs.readFile();

http.createServer();

# Node.js Frameworks and Tools

Node.js is a low-level platform. In order to make things easy and exciting for developers, thousands of libraries were built upon Node.js by the community.

Many of those established over time as popular options. Here is a non-comprehensive list of the ones worth learning:

* [**AdonisJS**](https://adonisjs.com/): A TypeScript-based fully featured framework highly focused on developer ergonomics, stability, and confidence. Adonis is one of the fastest Node.js web frameworks.
* [**Egg.js**](https://eggjs.org/en/): A framework to build better enterprise frameworks and apps with Node.js & Koa.
* [**Express**](https://expressjs.com/): It provides one of the most simple yet powerful ways to create a web server. Its minimalist approach, unopinionated, focused on the core features of a server, is key to its success.
* [**Fastify**](https://fastify.io/): A web framework highly focused on providing the best developer experience with the least overhead and a powerful plugin architecture. Fastify is one of the fastest Node.js web frameworks.
* [**FeatherJS**](https://feathersjs.com/): Feathers is a lightweight web-framework for creating real-time applications and REST APIs using JavaScript or TypeScript. Build prototypes in minutes and production-ready apps in days.
* [**Gatsby**](https://www.gatsbyjs.com/): A [React](https://reactjs.org/)-based, [GraphQL](https://graphql.org/) powered, static site generator with a very rich ecosystem of plugins and starters.
* [**hapi**](https://hapijs.com/): A rich framework for building applications and services that enables developers to focus on writing reusable application logic instead of spending time building infrastructure.
* [**koa**](http://koajs.com/): It is built by the same team behind Express, aims to be even simpler and smaller, building on top of years of knowledge. The new project born out of the need to create incompatible changes without disrupting the existing community.
* [**Loopback.io**](https://loopback.io/): Makes it easy to build modern applications that require complex integrations.
* [**Meteor**](https://meteor.com/): An incredibly powerful full-stack framework, powering you with an isomorphic approach to building apps with JavaScript, sharing code on the client and the server. Once an off-the-shelf tool that provided everything, now integrates with frontend libs [React](https://reactjs.org/), [Vue](https://vuejs.org/), and [Angular](https://angular.io/). Can be used to create mobile apps as well.
* [**Micro**](https://github.com/zeit/micro): It provides a very lightweight server to create asynchronous HTTP microservices.
* [**NestJS**](https://nestjs.com/): A TypeScript based progressive Node.js framework for building enterprise-grade efficient, reliable and scalable server-side applications.
* [**Next.js**](https://nextjs.org/): [React](https://reactjs.org/) framework that gives you the best developer experience with all the features you need for production: hybrid static & server rendering, TypeScript support, smart bundling, route pre-fetching, and more.
* [**Nx**](https://nx.dev/): A toolkit for full-stack monorepo development using NestJS, Express, [React](https://reactjs.org/), [Angular](https://angular.io/), and more! Nx helps scale your development from one team building one application to many teams collaborating on multiple applications!
* [**Sapper**](https://sapper.svelte.dev/): Sapper is a framework for building web applications of all sizes, with a beautiful development experience and flexible filesystem-based routing. Offers SSR and more!
* [**Socket.io**](https://socket.io/): A real-time communication engine to build network applications.
* [**Strapi**](https://strapi.io/): Strapi is a flexible, open-source Headless CMS that gives developers the freedom to choose their favorite tools and frameworks while also allowing editors to easily manage and distribute their content. By making the admin panel and API extensible through a plugin system, Strapi enables the world's largest companies to accelerate content delivery while building beautiful digital experiences.

Global objects

Eg. These global objects can be used any where.

console.log();

setTimeout();

clearTimeout();

setInterval(() => {

}, interval);

clearInterval();

# Module

App.js

Console.log(module);

C:\Users\pugal\Desktop\Pugal\NodeJs\firstNodeProject>node App.js

Module {

id: '.',

path: 'C:\\Users\\pugal\\Desktop\\Pugal\\NodeJs\\firstNodeProject',

exports: {},

parent: null,

filename: 'C:\\Users\\pugal\\Desktop\\Pugal\\NodeJs\\firstNodeProject\\App.js',

loaded: false,

children: [],

paths: [

'C:\\Users\\pugal\\Desktop\\Pugal\\NodeJs\\firstNodeProject\\node\_modules',

'C:\\Users\\pugal\\Desktop\\Pugal\\NodeJs\\node\_modules',

'C:\\Users\\pugal\\Desktop\\Pugal\\node\_modules',

'C:\\Users\\pugal\\Desktop\\node\_modules',

'C:\\Users\\pugal\\node\_modules',

'C:\\Users\\node\_modules',

'C:\\node\_modules'

]

}

## Global variable

console.log(\_\_dirname);

console.log(\_\_filename);

result

PS C:\Users\pugal\Desktop\Pugal\NodeJs\firstNodeProject> node App.js

C:\Users\pugal\Desktop\Pugal\NodeJs\firstNodeProject

C:\Users\pugal\Desktop\Pugal\NodeJs\firstNodeProject\App.js

## Modules and require()

Count.js

var counter = function(arr) {

    return 'There are '+arr.length+' elements in the array';

}

module.exports = counter;

App.js

var counter = require('./count');

console.log(counter(['Pugal A','Mala P','Tarun P','Sarvesh P']));

Result

PS C:\Users\pugal\Desktop\Pugal\NodeJs\firstNodeProject> node App.js

There are 4 elements in the array

## Using module exporting many properties

**Stuff.js**

var counter = function(arr) {

    return 'There are '+arr.length+' elements in the array';

}

var adder =  function(a, b) {

    return `The sum of number given is ${a+b}`;

}

var pi = 3.142;

module.exports.counter = counter;

module.exports.adder = adder;

module.exports.pi = pi;

**App.js**

var stuff = require('./stuff');

console.log(stuff.counter(['Pugal A','Mala P','Tarun P','Sarvesh P']));

console.log(stuff.adder(5,6));

console.log(stuff.adder(stuff.pi,10));

Three ways to export it

1)

module.exports.counter = counter;

module.exports.adder = adder;

module.exports.pi = pi;

2)

module.exports.adder =  function(a, b) {

    return `The sum of number given is ${a+b}`;

}

3)

module.exports = {

    counter: counter,

    adder: adder,

    pi: pi

};

# Reading files

**App.js**

var fs = require('fs');

fs.readFile('Readme.txt','utf8',function(err,data){Si

    console.log(data);

});

console.log("Data fetched");

**result**

PS C:\Users\pugal\Desktop\Pugal\NodeJs\firstNodeProject> node App.js

Data fetched

This is nodejs tutorial class

Nodejs is a fantastic javascript scripting language.

# Setting up Server

**App.js**

var http = require('http');

var server = http.createServer(function(req, res){

   res.writeHead(200,{'Content-Type':'text/plain'});

   res.end('Hello notejs world');

});

server.listen(3001,'127.0.0.1');

console.log('server up listening port 3001');

but when you send html stream out to browser, you need to change the type to

‘text/html’

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

body {background:skyblue;font-family: Verdana, Geneva, Tahoma, sans-serif;color: #fff;padding: 30px;}

h1{font-size: 48px;text-align: center;}

p{font-size: 16px;text-align: center;}

</style>

</head>

<body>

<h1>This is Nodejs tutorial</h1>

<p>

Nodejs is a fantastic javascript scripting language.

</p>

</body>

</html>

**App.js**

var http = require('http');

var fs = require('fs');

var server = http.createServer(function(req, res){

res.writeHead(200,{'Content-Type':'text/html'});

var readStreem = fs.createReadStream(\_\_dirname+'/index.html','utf8');

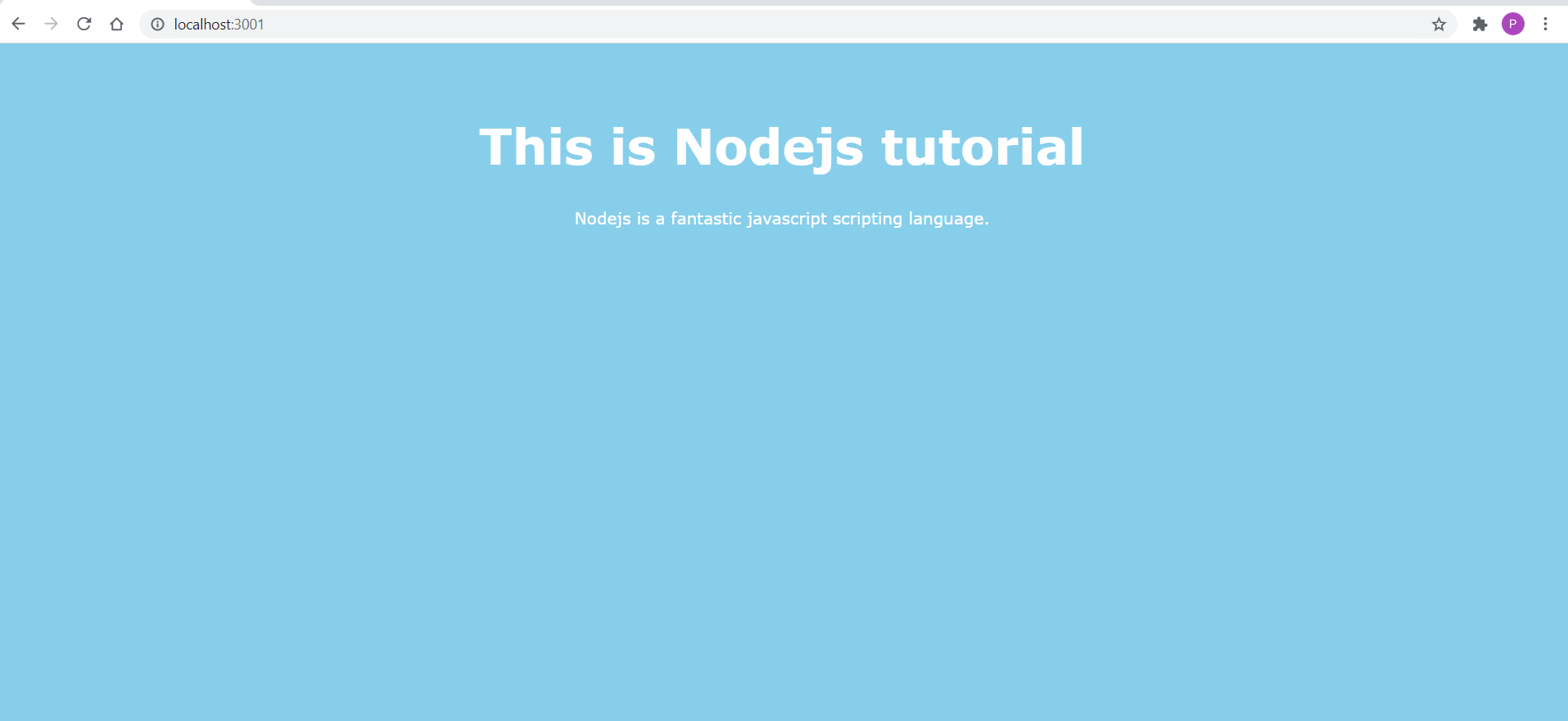
readStreem.pipe(res);

});

server.listen(3001,'127.0.0.1');

console.log('server up listening port 3001');

**Result:**



# Route

var http = require("http");

var fs = require("fs");

var server = http.createServer(function (req, res) {

  if (req.url === "/home" || req.url === "/") {

    res.writeHead(200,{'Content-Type':'text/html'})

    var readStreem = fs.createReadStream(\_\_dirname + "/index.html", "utf8");

    readStreem.pipe(res);

  } else if (req.url === "/contact") {

    res.writeHead(200,{'Content-Type':'text/html'})

    var readStreem = fs.createReadStream(\_\_dirname + "/contact.html", "utf8");

    readStreem.pipe(res);

  } else if (req.url === "/api/customers") {

    var customers = [

      { name: "Pugalendi A", age: 56, city: "Bangalore" },

      { name: "Madhavan", age: 50, city: "Karur" },

      { name: "Shankar N", age: 52, city: "Mumbai" },

    ];

    res.writeHead(200,{'Content-Type':'application/json'})

    res.end(JSON.stringify(customers));

  } else {

    res.writeHead(200,{'Content-Type':'text/html'})

    var readStreem = fs.createReadStream(\_\_dirname + "/404.html", "utf8");

    readStreem.pipe(res);

  }

});

server.listen(3001, "127.0.0.1");

console.log("server up listening port 3001");

# Template engine

ejs is the template engine we can use

1. so c:\projectfolder>npm install ejs
2. include app.set('view engine','ejs'); in the main program
3. create folder \views in the root folder
4. create ejs view in \views folder. Here profile.ejs.

**profile.ejs**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

body {background:skyblue;font-family: Verdana, Geneva, Tahoma, sans-serif;color: #fff;padding: 30px;}

h1{font-size: 48px;text-align: center;}

h2{font-size: 38px;text-align: center;}

p{font-size: 16px;text-align: center;}

ul{text-align: center;}

ul li{list-style-type: none;}

</style>

</head>

<body>

<%- include('partials/nav.ejs') %>

<h1>Profile page of <%= person %></h1>

<p><strong>Age :</strong><%= data.age %></p>

<p><strong>Place :</strong><%= data.place %></p>

<p><strong>Country :</strong><%= data.country %></p>

<h2>Hobbies</h2>

<ul>

<% data.hobbies.forEach(function(item){ %>

<li><%= item %></li>

<% }) %>

</ul>

</body>

</html>

Include folder called /partials inside the views folder

Ejs understands the ejs templates from the **views** folder

## Partials

Include partials nav.ejs in the partials folder

Include the code inside the program profile.ejs (check above)

<%- include('partials/nav.ejs') %>

**nav.ejs**

<nav>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/contact">Contact</a></li>

</ul>

</nav>

**App.js**

Render the view ‘profile’ inside the main program. Ejs understands the ejs file from views folder.

var express = require('express');

var app = express();

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

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

res.sendFile(\_\_dirname+'/index.html');

});

app.get('/contact',function(req, res){

res.sendFile(\_\_dirname+'/contact.html');

});

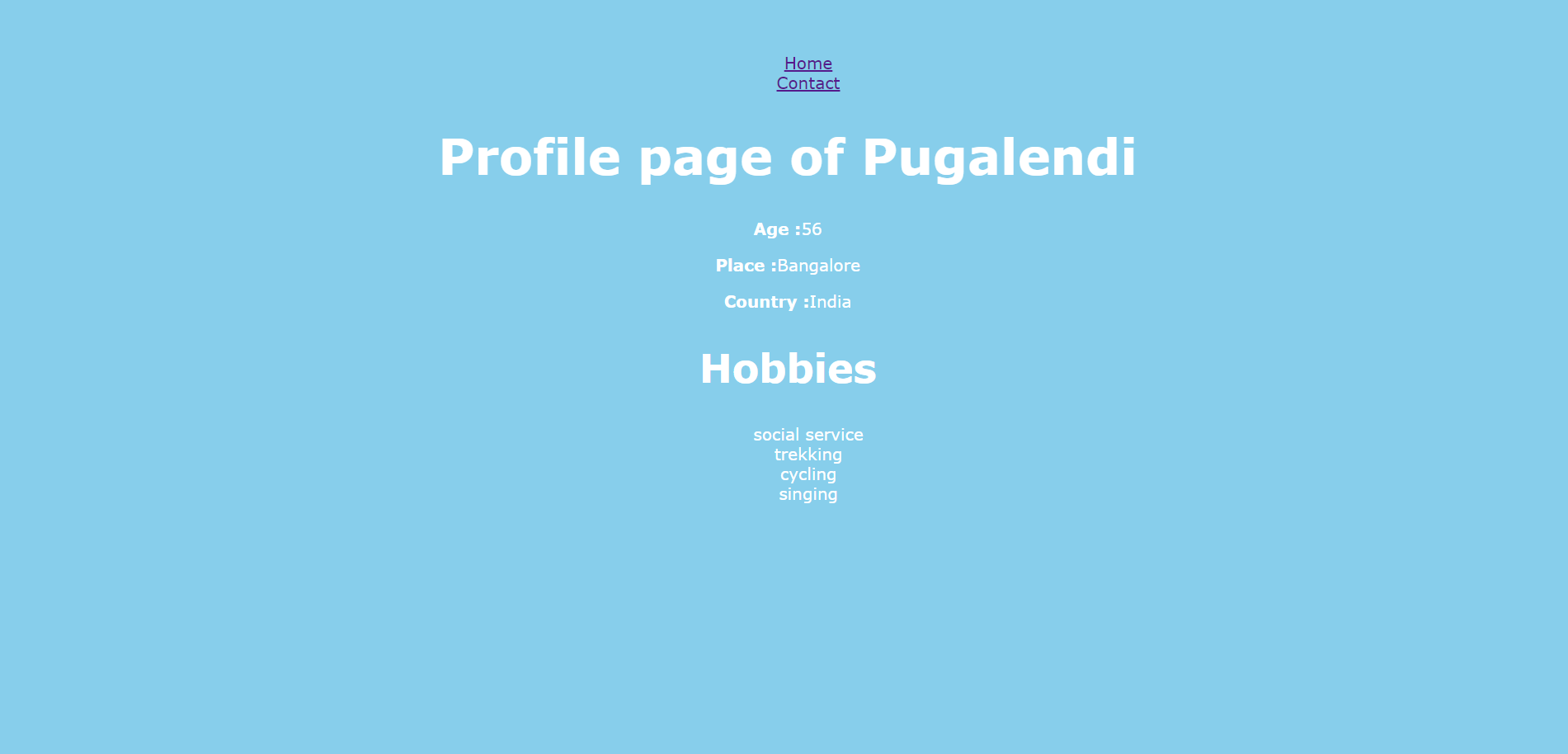
app.get('/profile/:name',function(req, res) {

var data = {age:56, place:'Bangalore', country:'India',hobbies:['social service','trekking','cycling','singing']}

res.render('profile', {person: req.params.name, data: data});

});

app.listen(3001);



## Middleware

The middleware is the pipeline between client and the server. Eg. if we create a folder /asset/ and try to create style.css and use it, it will not work. Because the server cannot access the resource without middleware.

<link href="/assets/style.css" rel="stylesheet" type="text/css"/>

Cannot be accessed directly unless it has to be accessed via middleware.

So the following is the middleware setup by express, the static folder is defined as express.static(‘assets’). The routing is /assets.

app.use('/assets', express.static('assets'));

# Query string

Mysite.com/blog/news?page=2

Page=2

Mysite.come/contact?person=pugal&dept=it&place=bangalore

Parse the request and pull the data .

Simply

console.log(req.query);

express will parse the query string in to object like this,

{ dept: 'marketing', person: 'Pugal' }

This can be rendered along with the contact form.

app.get('/contact',function(req, res){

    console.log(req.query);

    res.render('contact',{qs: req.query});

});

Inside the form we can prefill the value from the backend

        <form id='contact-form' class="contact-form">

            <p>

                <label for='who'>Enter the details to contact</label>

                <input type="text" name='who' value="<%= qs.person %>"/>

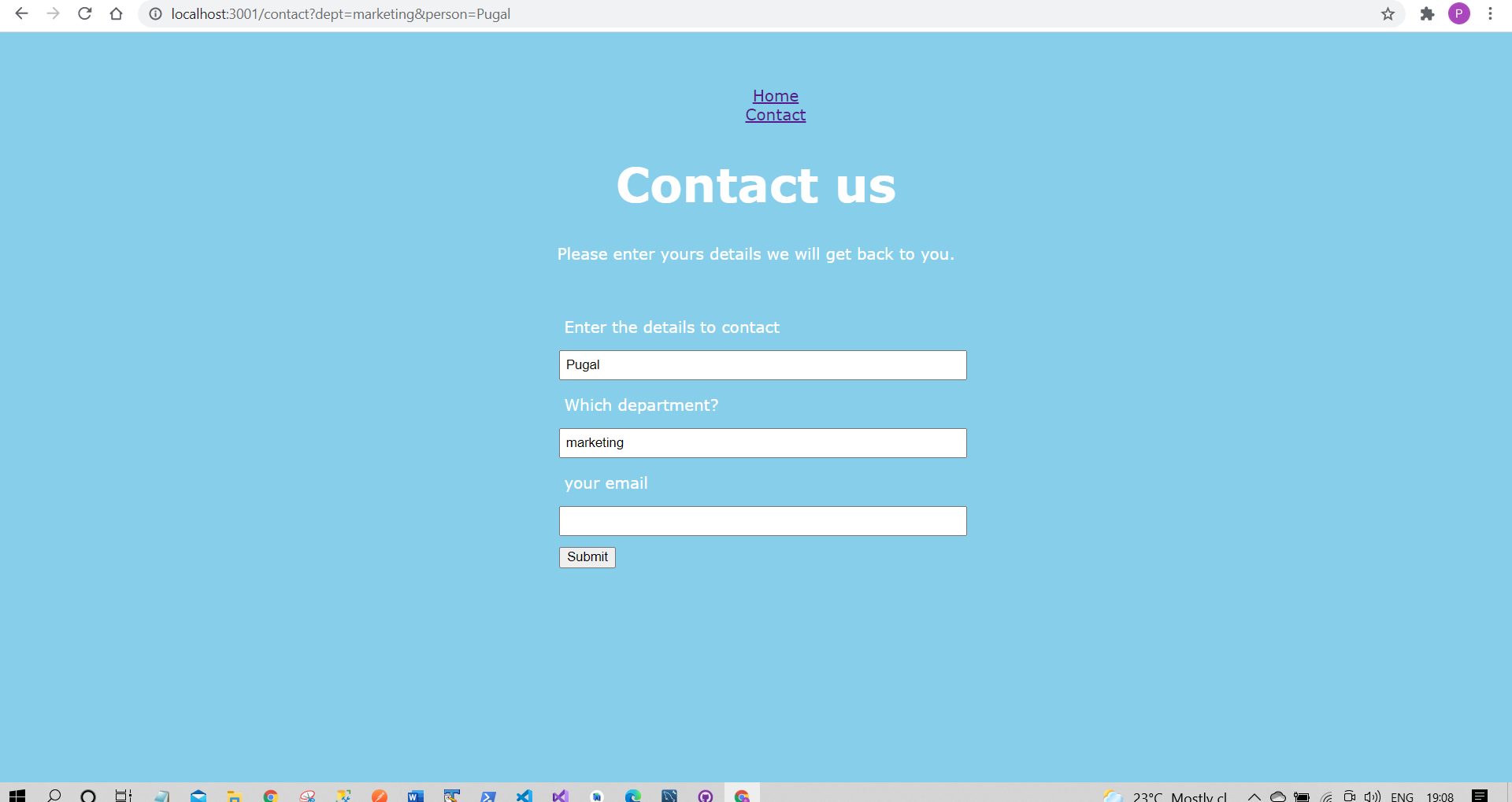
            </p>

            <p>

                <label for='department'>Which department?</label>

                <input type="text" name='department' value="<%= qs.dept %>"/>

            </p>



Handing POST request

1. Install body-parser (npm install body-parser)
2. Include the body-parser property in **App.js**

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

var urlencodedParser = bodyParser.urlencoded({ extended: false });

app.post('/contact',urlencodedParser, function(req, res){

console.log(req.body);

res.render('contact',{qs: {}});

});

## Template engine form

<form id='contact-form' class="contact-form" method="POST" action="/contact">

<p>

<label for='who'>Enter the details to contact</label>

<input type="text" name='who' value="<%= qs.person %>"/>

</p>

<p>

<label for='department'>Which department?</label>

<input type="text" name='department' value="<%= qs.dept %>"/>

</p>

<p>

<label for='department'>your email</label>

<input type="email" name='email'/>

</p>

<p>

<button type="submit">Submit</button>

</p>

</form>

Console log. This property can be passed to the template to show the success message with proper data.

[Object: null prototype] {

who: 'Pugalendi',

department: 'Infrastructure',

email: 'pugal.a@gmail.com'

}