What is Express JS

Express.js is a web framework for Node.js. It is a fast, robust and asynchronous in nature.

Express is a fast, assertive, essential and moderate web framework of Node.js. You can assume express as a layer built on the top of the Node.js that helps manage a server and routes. It provides a robust set of features to develop web and mobile applications.

Advantages

* It can be used to design single-page, multi-page and hybrid web applications.
* It allows to setup middlewares to respond to HTTP Requests.
* It defines a routing table which is used to perform different actions based on HTTP method and URL.
* It allows to dynamically render HTML Pages based on passing arguments to templates.

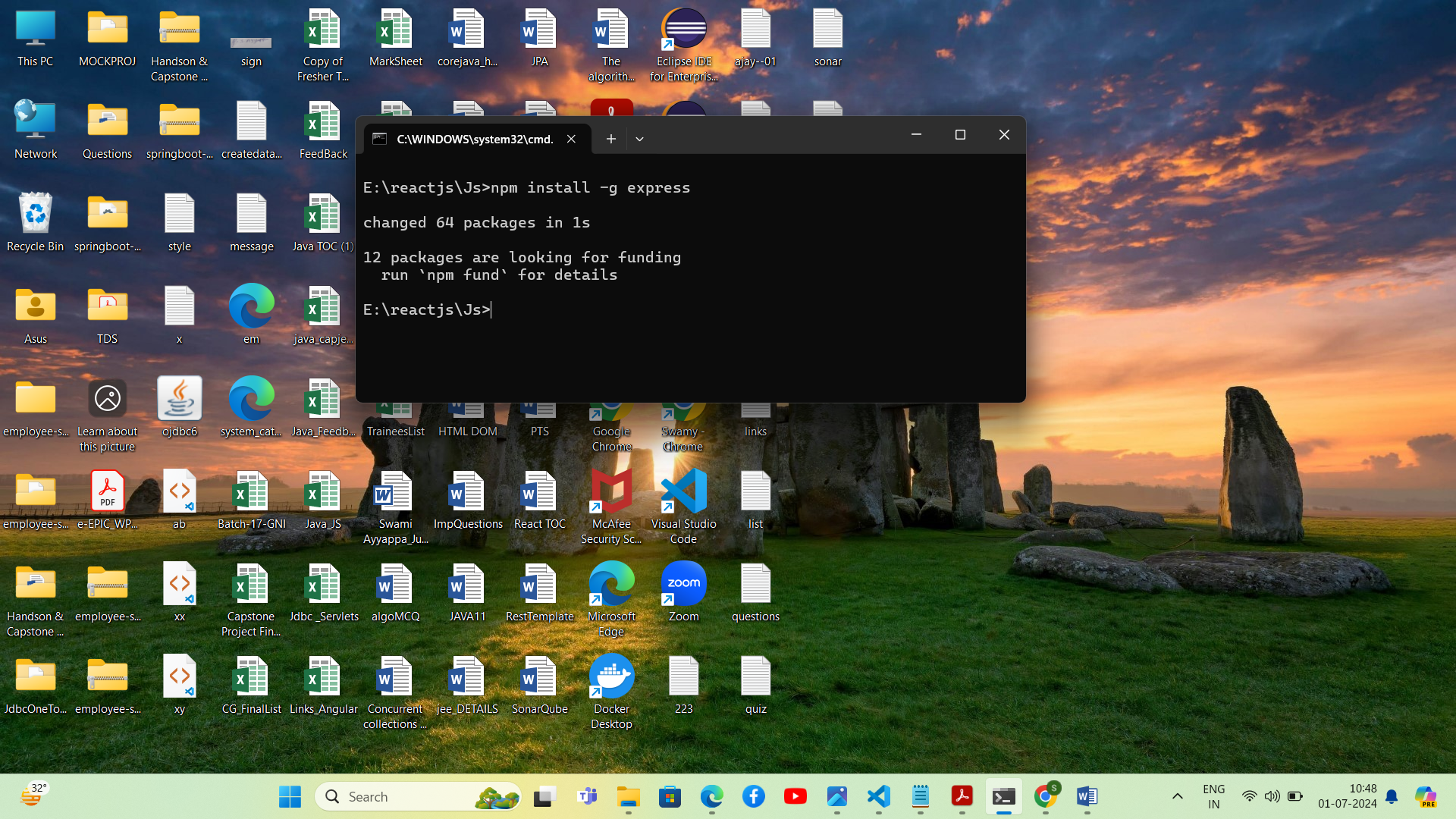
Why use Express

* Ultra fast I/O
* Asynchronous and single threaded
* MVC like structure
* Robust API makes routing easy

# **Install Express.js**

Firstly, you have to install the express framework globally to create web application using Node terminal. Use the following command to install express framework globally.

npm install -g express

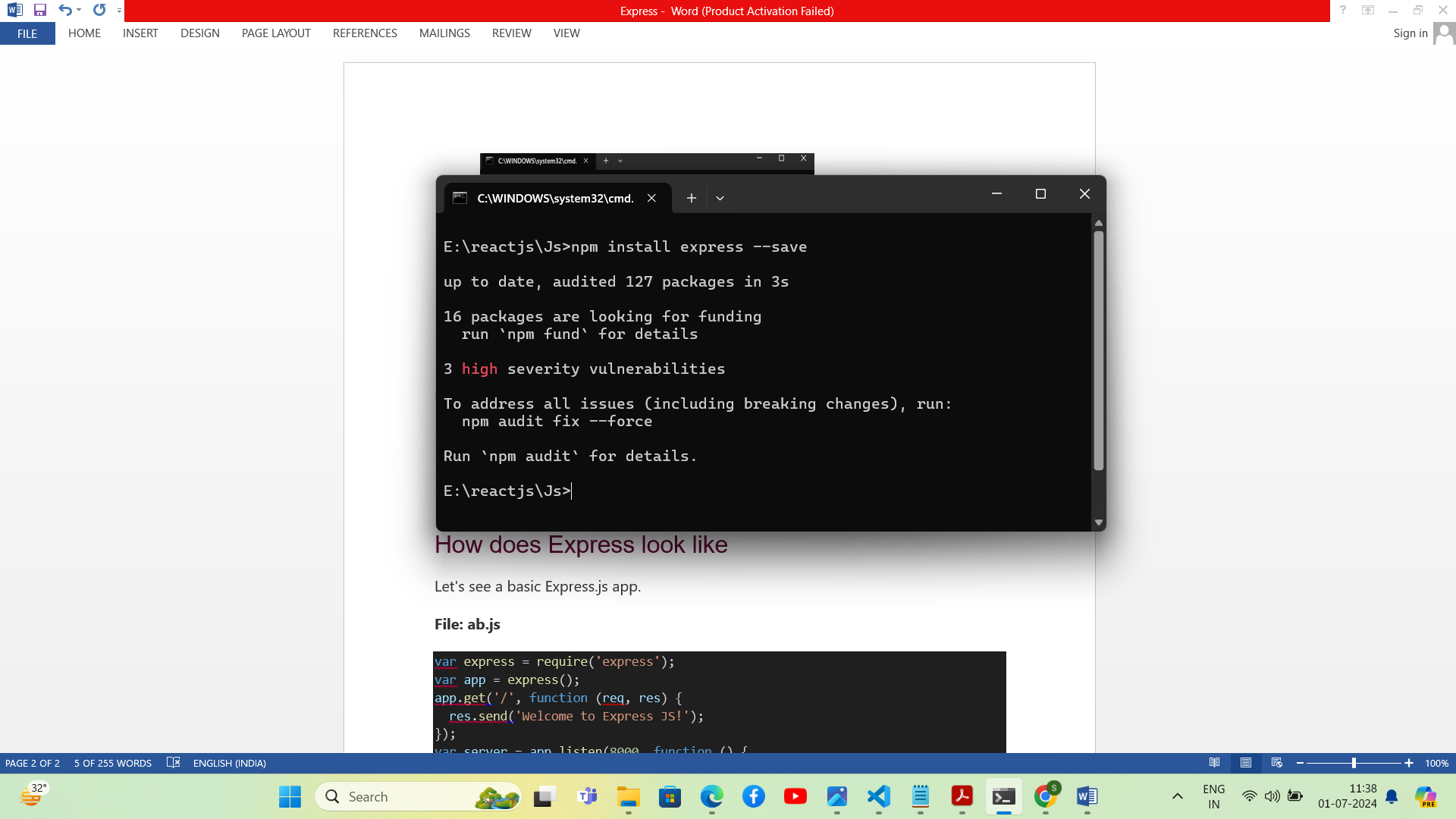


## Installing Express

Use the following command to install express:

--save will add the dependency to the package.json file.

npm install express --save



How does Express look like

Let's see a basic Express.js app.

**File: ab.js**

var express = require('express');

var app = express();

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

  res.send('Welcome to Express JS!');

});

var server = app.listen(8000, function () {

  var host = server.address().address;

  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);

});

# **Express.js GET Request**

GET and POST both are two common HTTP requests used for building REST API's. GET requests are used to send only limited amount of data because data is sent into header while POST requests are used to send large amount of data because data is sent in the body.

Express.js facilitates you to handle GET and POST requests using the instance of express.

## Express.js GET Method Example 1

**Fetch data in JSON format:**

Get method facilitates you to send only limited amount of data because data is sent in the header. It is not secure because data is visible in URL bar.

Index.html

<html>

<body>

<form action="http://127.0.0.1:8000/process\_get" method="GET">

First Name: <input type="text" name="first\_name">  <br>

Last Name: <input type="text" name="last\_name">

<input type="submit" value="Submit">

</form>

</body>

</html>

Ab.js

var express = require('express');

var app = express();

app.use(express.static('public'));

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

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

})

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

response = {

       first\_name:req.query.first\_name,

       last\_name:req.query.last\_name

   };

   console.log(response);

   res.end(JSON.stringify(response));

})

var server = app.listen(8000, function () {

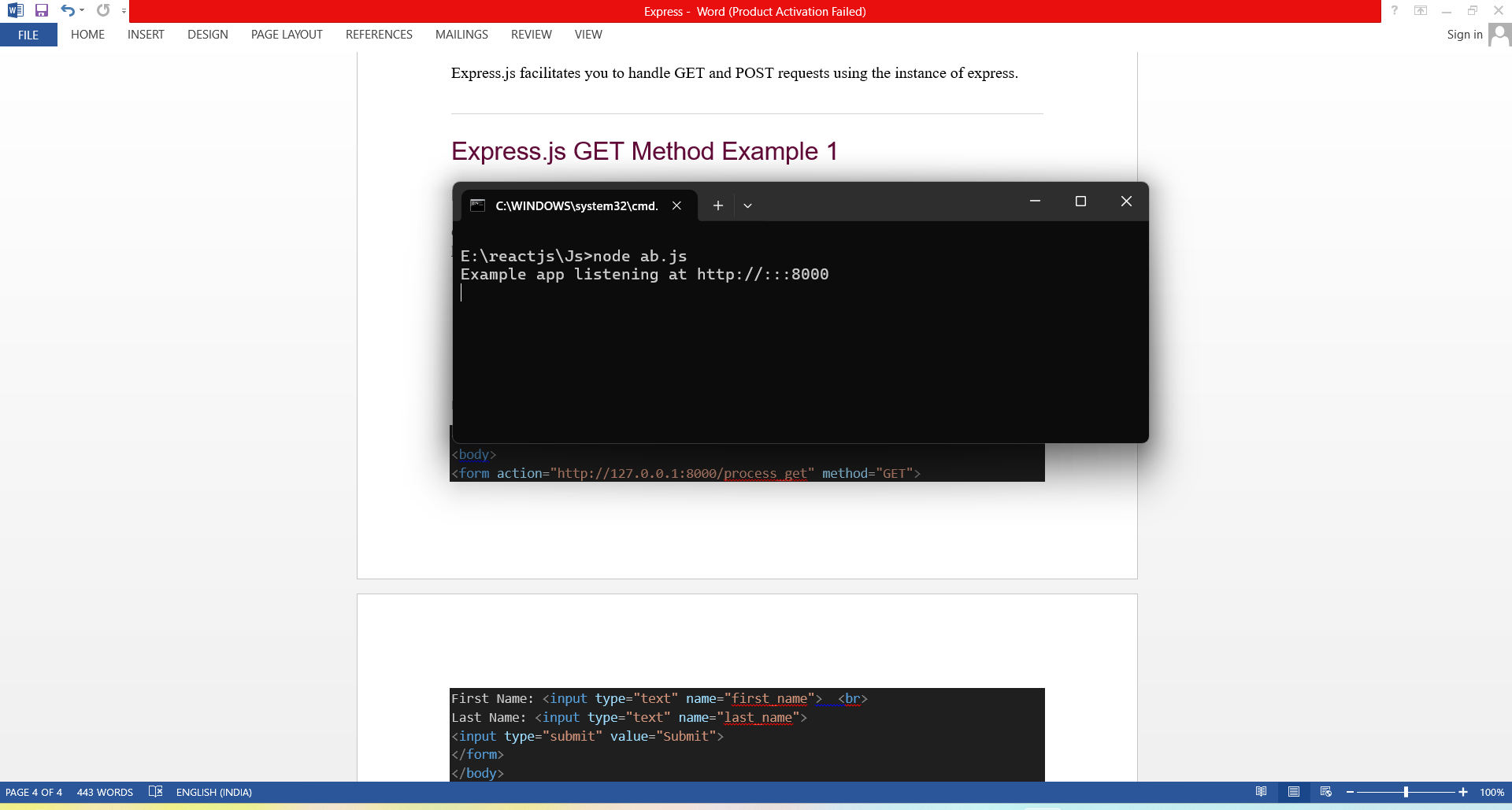
  var host = server.address().address

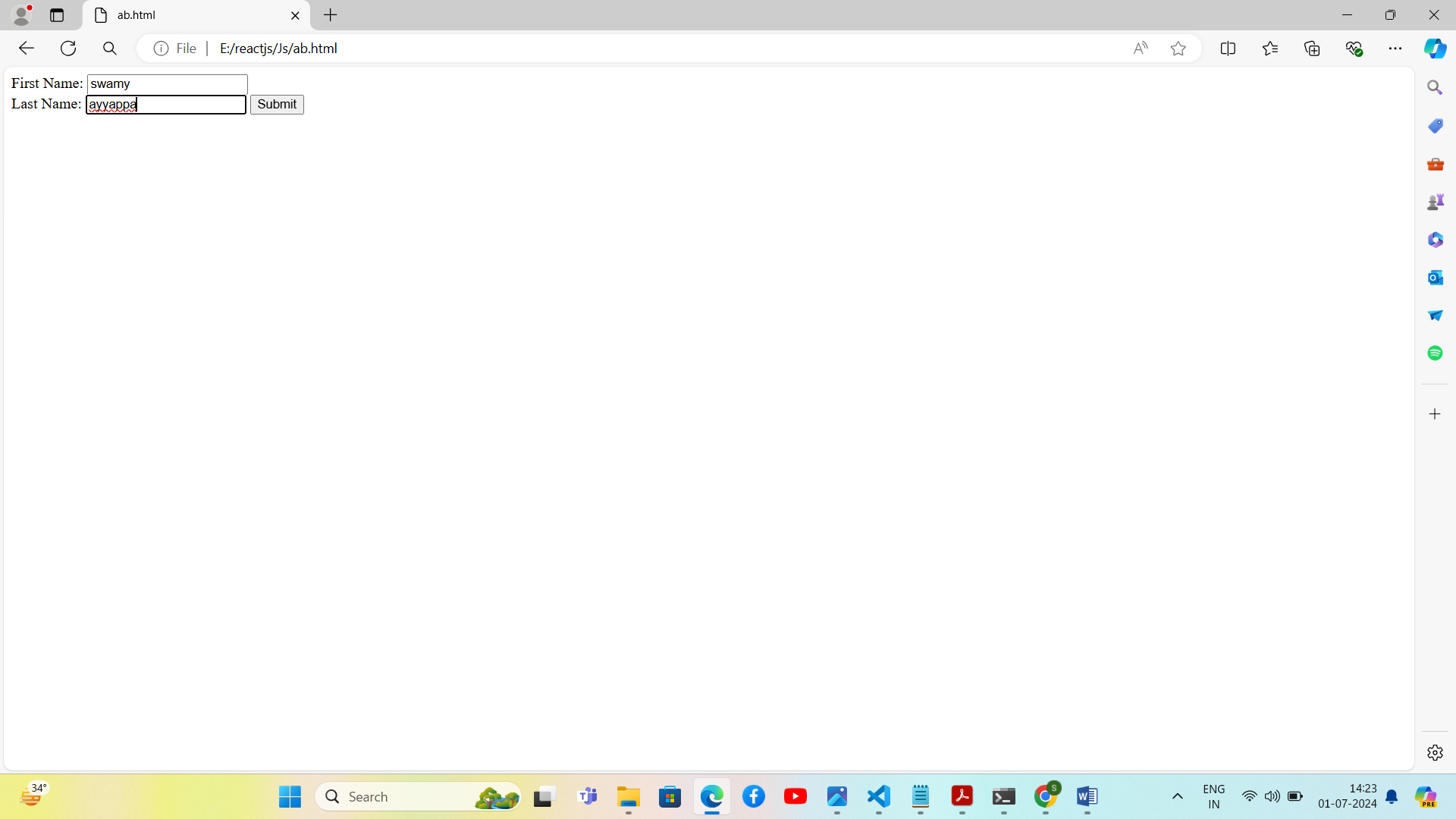
  var port = server.address().port

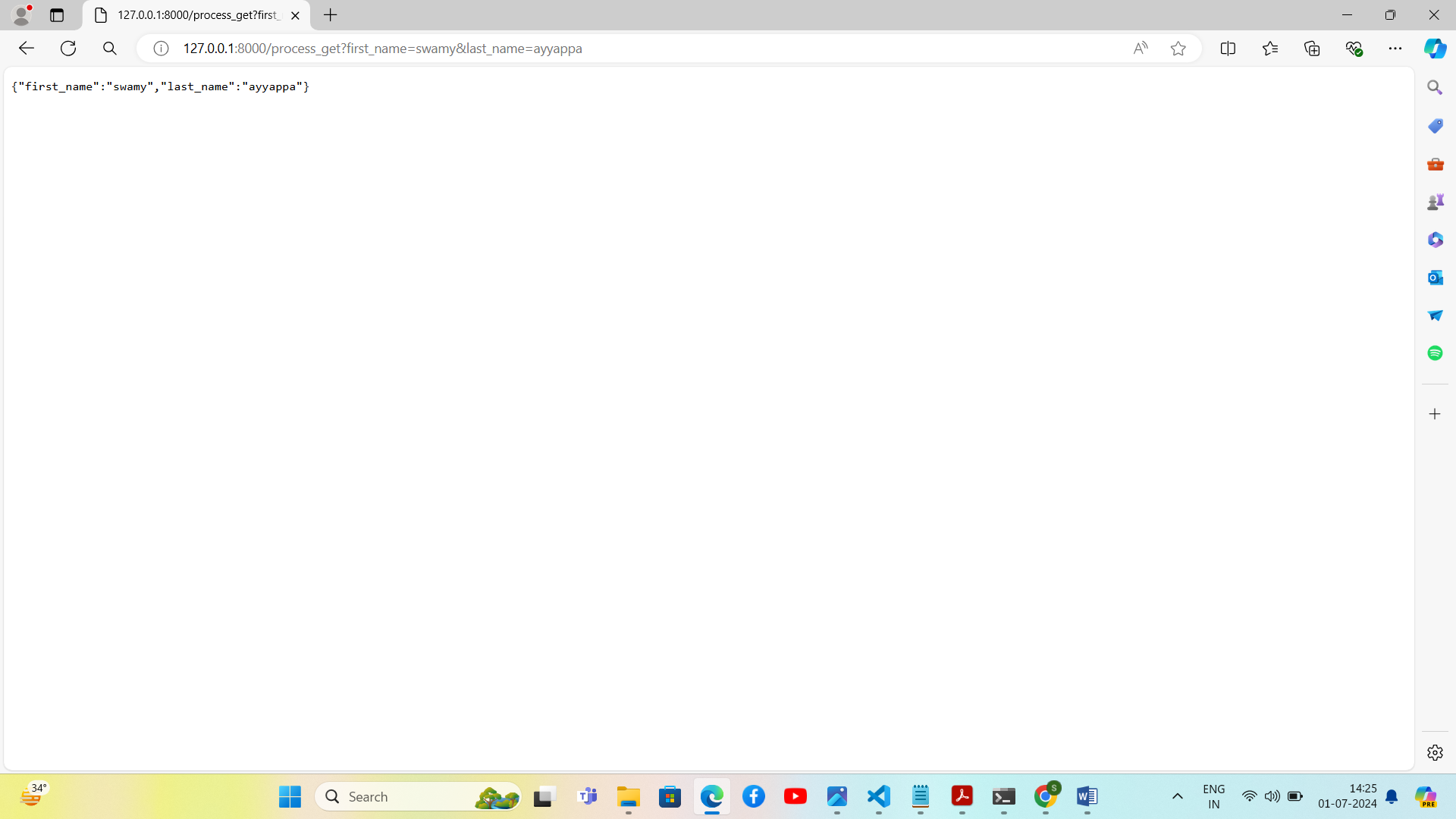
  console.log("Example app listening at http://%s:%s", host, port)

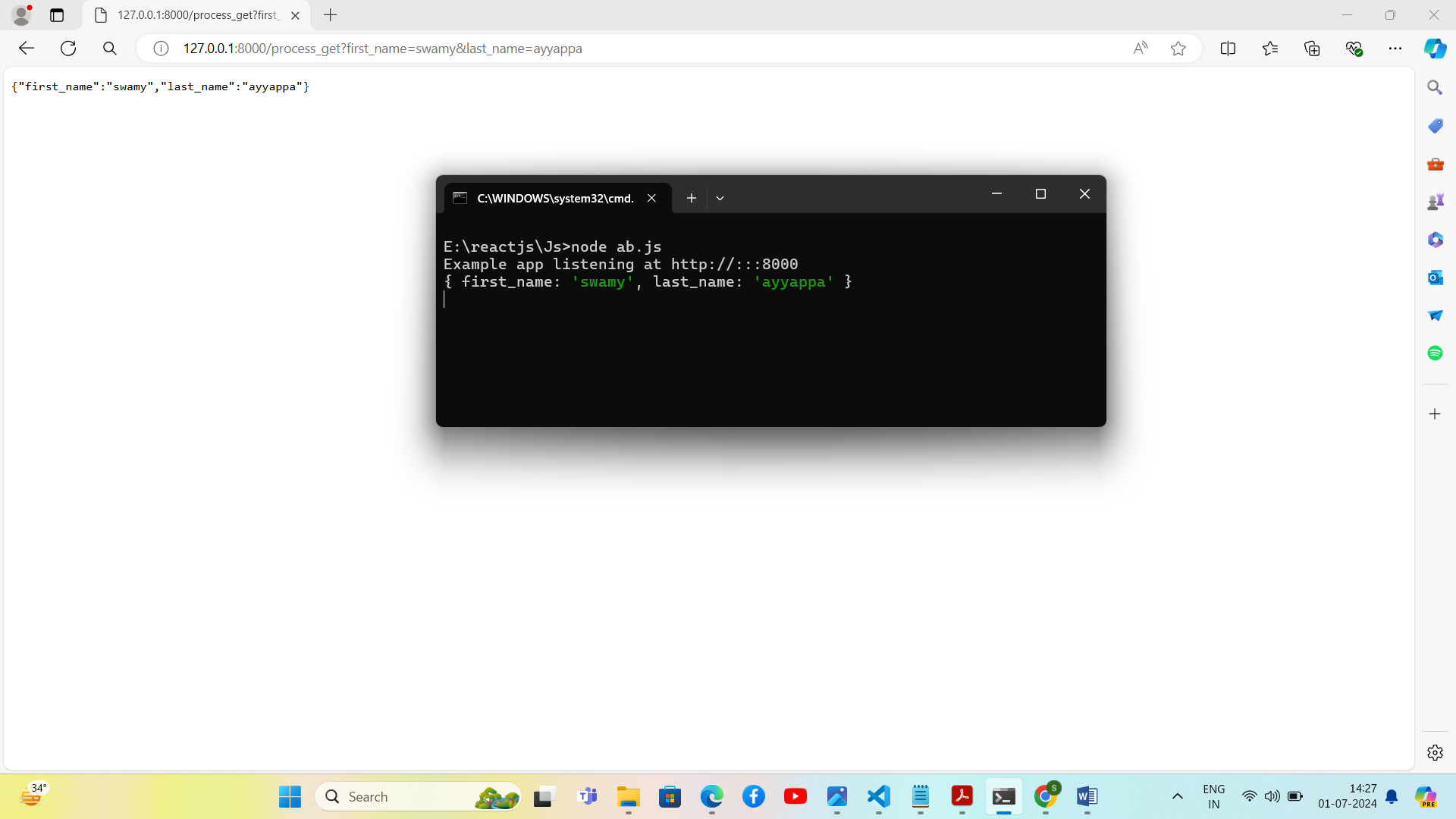
})

How To Run









## Express.js GET Method Example 2

<html>

<body>

<form action="http://127.0.0.1:8000/get\_example2">

<table>

<tr><td>Enter First Name:</td><td><input type="text" name="firstname"/><td></tr>

<tr><td>Enter Last Name:</td><td><input type="text" name="lastname"/><td></tr>

<tr><td>Enter Password:</td><td><input type="password" name="password"/></td></tr>

<tr><td>Sex:</td><td>

<input type="radio" name="sex" value="male"> Male

<input type="radio" name="sex" value="female">Female

</td></tr>

<tr><td>About You :</td><td>

<textarea rows="5" cols="40" name="aboutyou" placeholder="Write about yourself">

</textarea>

</td></tr>

<tr><td colspan="2"><input type="submit" value="register"/></td></tr>

</table>

</form>

</body>

</html>

Ab.js

var express = require('express');

var app=express();

app.get('/get\_example2', function (req, res) {

res.send('<p>Firstname: ' + req.query['firstname']+'</p>  <p>Lastname: '+req.query['lastname']+'</p><p>Password: '+req.query['password']+

'</p><p>AboutYou: '+req.query['aboutyou']+ '</p>');  })

var server = app.listen(8000, function () {

  var host = server.address().address

  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})

# **Express.js POST Request**

GET and POST both are two common HTTP requests used for building REST API's. POST requests are used to send large amount of data.

Express.js facilitates you to handle GET and POST requests using the instance of express.

## Express.js POST Method

Post method facilitates you to send large amount of data because data is send in the body. Post method is secure because data is not visible in URL bar but it is not used as popularly as GET method. On the other hand GET method is more efficient and used more than POST.

Let's take an example to demonstrate POST method.

Index.html

<html>

<body>

<form action="http://127.0.0.1:8000/process\_post" method="POST">

First Name: <input type="text" name="first\_name">  <br>

Last Name: <input type="text" name="last\_name">

<input type="submit" value="Submit">

</form>

</body>

</html>

Ab.js

var express = require('express');

var app = express();

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

// Create application/x-www-form-urlencoded parser

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

app.use(express.static('public'));

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

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

})

app.post('/process\_post', urlencodedParser, function (req, res) {

   // Prepare output in JSON format

   response = {

       first\_name:req.body.first\_name,

       last\_name:req.body.last\_name

   };

   console.log(response);

   res.end(JSON.stringify(response));

})

var server = app.listen(8000, function () {

  var host = server.address().address

  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})

# **Express.js Middleware**

Express.js Middleware are different types of functions that are invoked by the Express.js routing layer before the final request handler. As the name specified, Middleware appears in the middle between an initial request and final intended route. In stack, middleware functions are always invoked in the order in which they are added.

Middleware is commonly used to perform tasks like body parsing for URL-encoded or JSON requests, cookie parsing for basic cookie handling, or even building JavaScript modules on the fly.

## What is a Middleware function

Middleware functions are the functions that access to the request and response object (req, res) in request-response cycle.

A middleware function can perform the following tasks:

* It can execute any code.
* It can make changes to the request and the response objects.
* It can end the request-response cycle.
* It can call the next middleware function in the stack.

## Express.js Middleware

Following is a list of possibly used middleware in Express.js app:

* Application-level middleware
* Router-level middleware
* Error-handling middleware
* Built-in middleware
* Third-party middleware

Example:

Ab.js

var express = require('express');

var app = express();

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

  res.send('Welcome to Express Middleware');

});

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

  res.send('How can I help You?');

});

var server = app.listen(8000, function () {

  var host = server.address().address

  var port = server.address().port

console.log("Example app listening at http://%s:%s", host, port)

})

**Middleware example explanation**

* In the above middleware example a new function is used to invoke with every request via**app.use()**.
* Middleware is a function, just like route handlers and invoked also in the similar manner.
* You can add more middlewares above or below using the same API.

# **Express.js Routing**

Routing is made from the word route. It is used to determine the specific behavior of an application. It specifies how an application responds to a client request to a particular route, URI or path and a specific HTTP request method (GET, POST, etc.). It can handle different types of HTTP requests.

Ex:

Ab.js

var express = require('express');

var app = express();

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

   console.log("Got a GET request for the homepage");

   res.send('Welcome to Express Js');

})

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

   console.log("Got a POST request for the homepage");

   res.send('I am Impossible! ');

})

app.delete('/del\_student', function (req, res) {

   console.log("Got a DELETE request for /del\_student");

   res.send('I am Deleted!');

})

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

   console.log("Got a GET request for /enrolled\_student");

   res.send('I am an enrolled student.');

})

// This responds a GET request for abcd, abxcd, ab123cd, and so on

app.get('/ab\*cd', function(req, res) {

   console.log("Got a GET request for /ab\*cd");

   res.send('Pattern Matched.');

})

var server = app.listen(8000, function () {

var host = server.address().address

  var port = server.address().port

console.log("Example app listening at http://%s:%s", host, port)

})