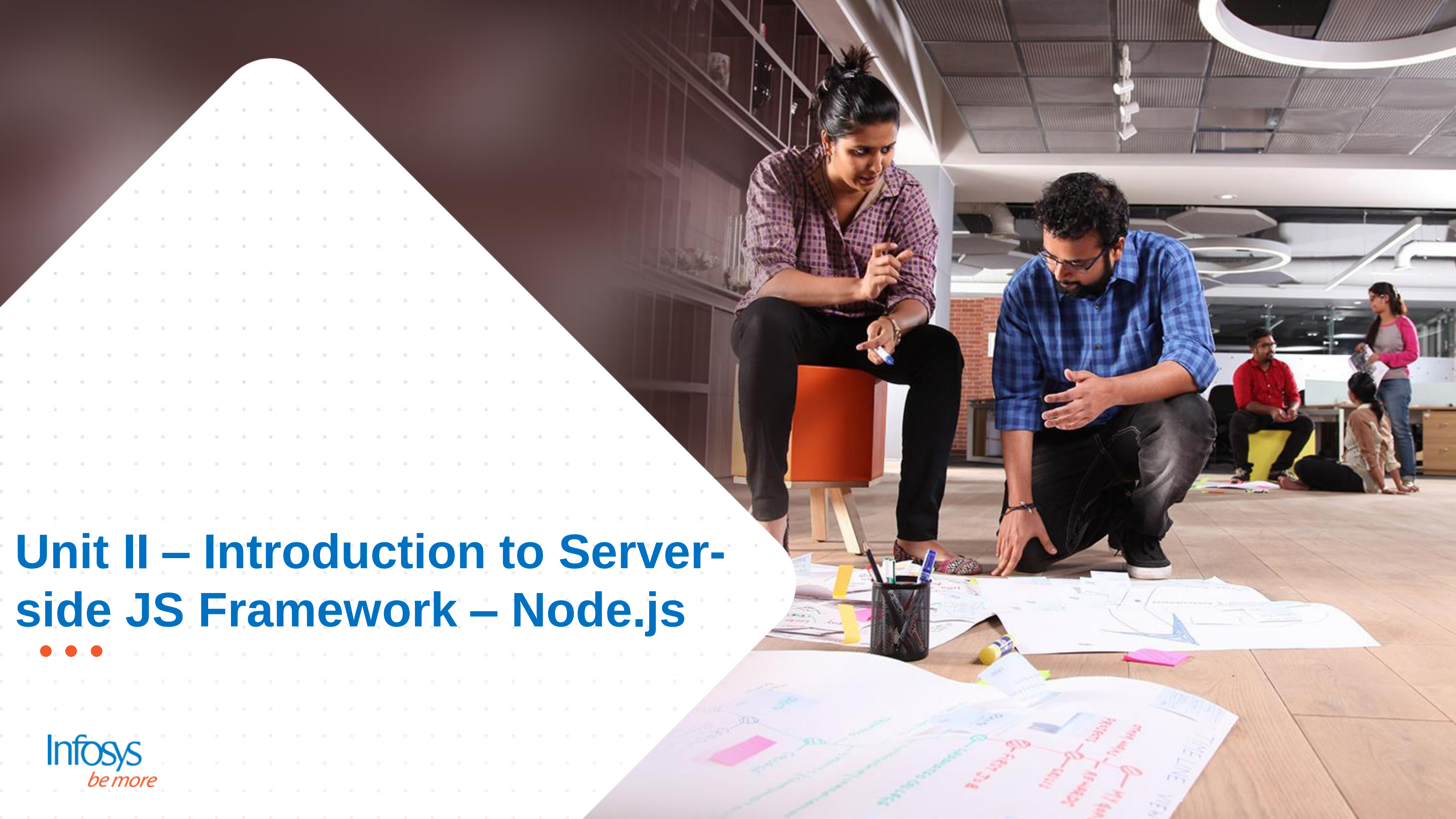


Unit II – Introduction to Server-side JS Framework – Node.js

Agenda

- Introduction
  - What is Node JS
  - Architecture
  - Feature of Node JS
  - Installation and setup
- Creating web servers with HTTP (Request & Response)
- Event Handling - GET & POST implementation
- Connect to NoSQL Database using Node JS
- Implementation of CRUD operations

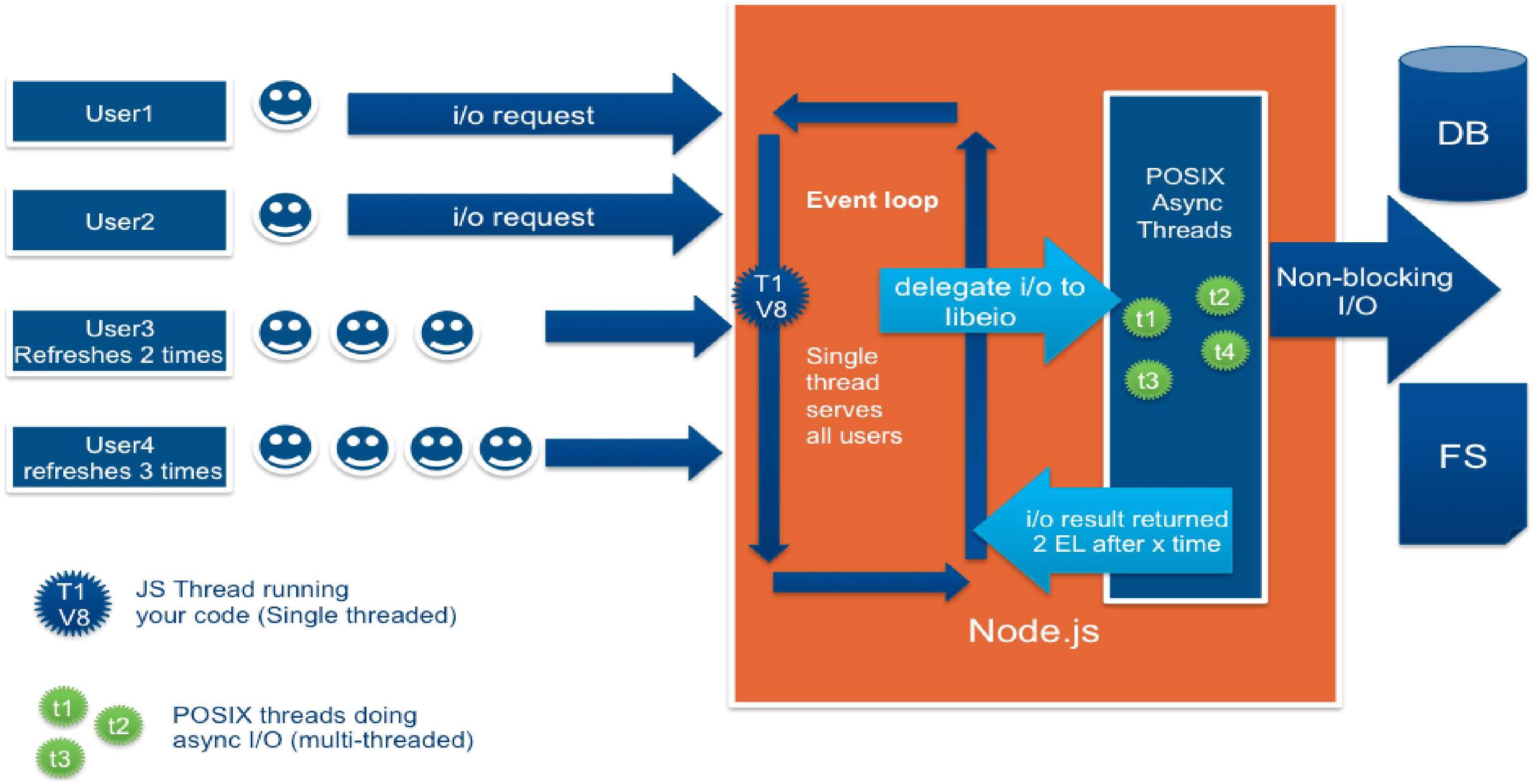
Introduction to Node.js

Infosys® | Education, Training and Assessment

Needs of Node.js

- Enables JavaScript to run outside the browser
- Makes use of Google's V8 VM for interpreting JavaScript
- Additional modules which simplifies JavaScript development
- It's a runtime and also a library!
- JavaScript is an event-driven language, and Node takes this as an advantage to produce highly scalable servers, using an architecture called an *event loop.*
- For Creating High Performance Servers.
- Non-Blocking I/O.
- Event and Callback based.
- Only one Thread and one call stack

Node.js Architecture



IDEs for Node

Sublime Text Editor

Cloud9 IDE
Your code anywhere, anytime

WebMatrix

Microsoft® WebMatrix

JetBRAINS

IntelliJ IDEA

WebStorm

Nide

Nodeclipse

node

node IDE ™

Installing Node

- Follow the instructions at https://github.com/joyent/node/wiki/Installation to get Node setup in the system or go to the official site and download the latest version of Node – http://nodejs.org/

- Once installed it will be available under **C → Program Files → nodejs**

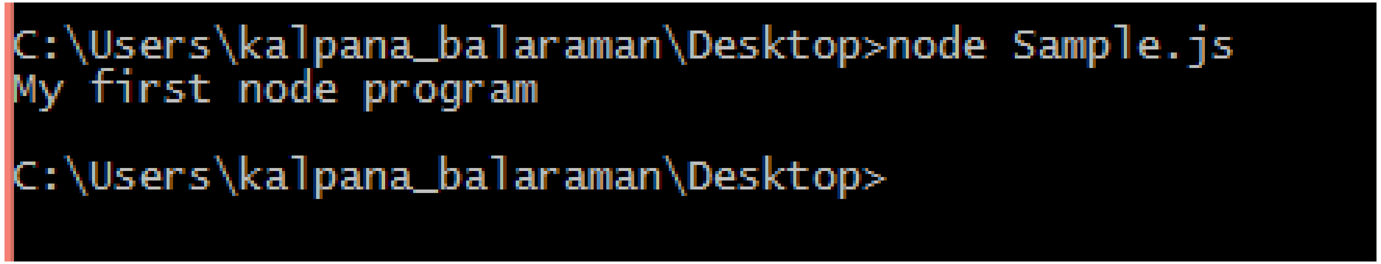
- Go to command prompt and type **node –v** to ensure the installation

First Demo

- Create a Simple JavaScript file as “Sample.js”

```
console.log("My first node program");
```

- Open Node command prompt and run the JavaScript file as:



```
C:\Users\kalpana_balaraman\Desktop>node Sample.js
My first node program

C:\Users\kalpana_balaraman\Desktop>
```

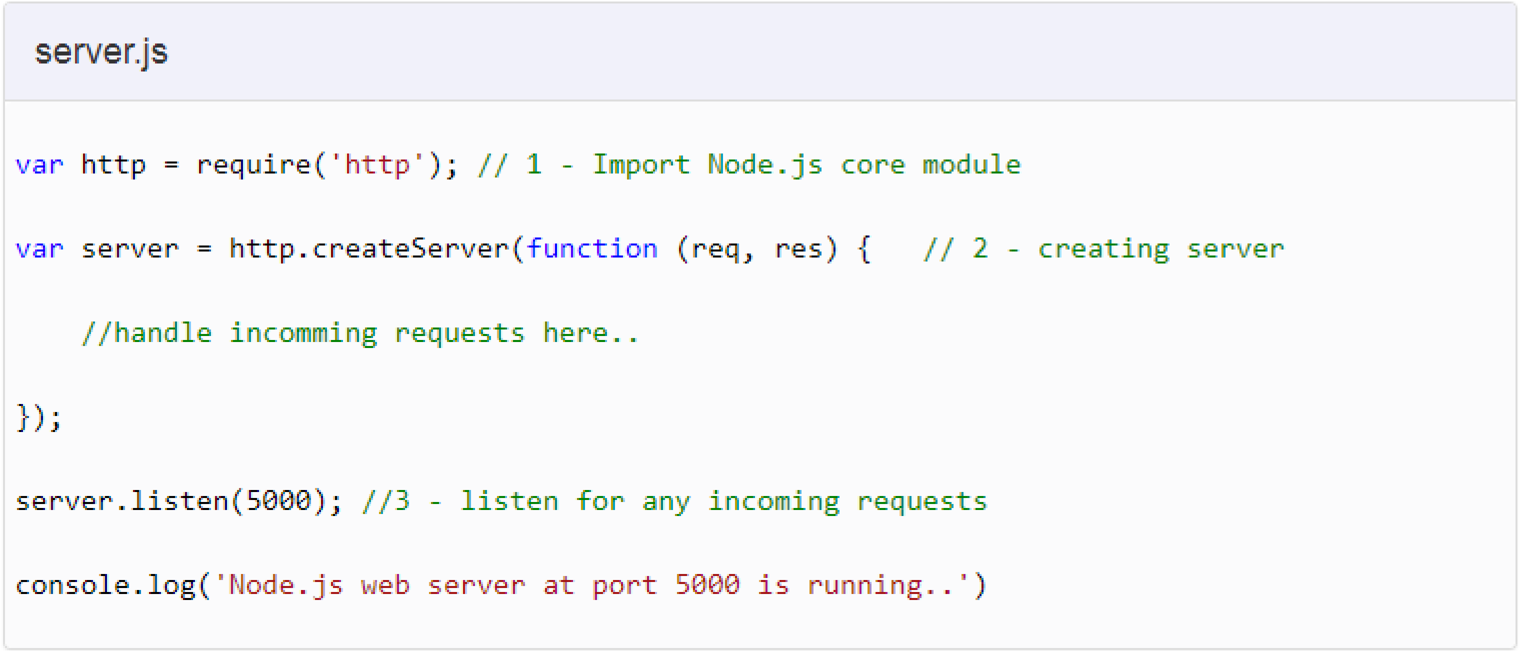
Creating web servers with HTTP (Request & Response)

Web Server using Node.js

- Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously.
- There should be a server listening in a port to service every request
- Handling of request data and processing it
- View generation logic execution and rendering back to the user

Create a Web Server(1/2)

- Create a JavaScript file with below code to create a web server and listen to our own port number:
- Eg., Server.js

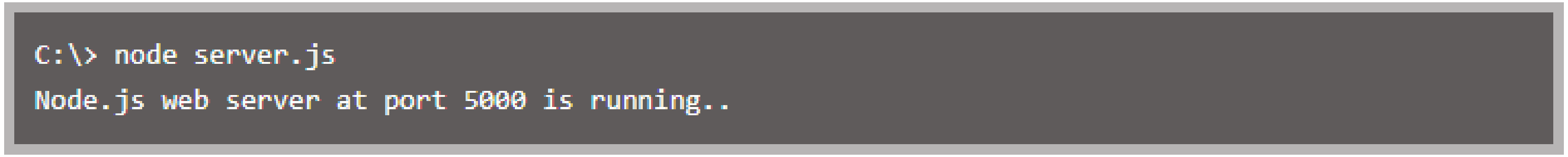


server.js

```
var http = require('http'); // 1 - Import Node.js core module

var server = http.createServer(function (req, res) {   // 2 - creating server

    //handle incomming requests here..

});

server.listen(5000); //3 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

Create a Web Server(2/2)

• Execute the JavaScript file using Node command prompt to start the server



```
C:\> node server.js
Node.js web server at port 5000 is running..
```

• Once server started, then open the browser with the below URL:

http://localhost:5000

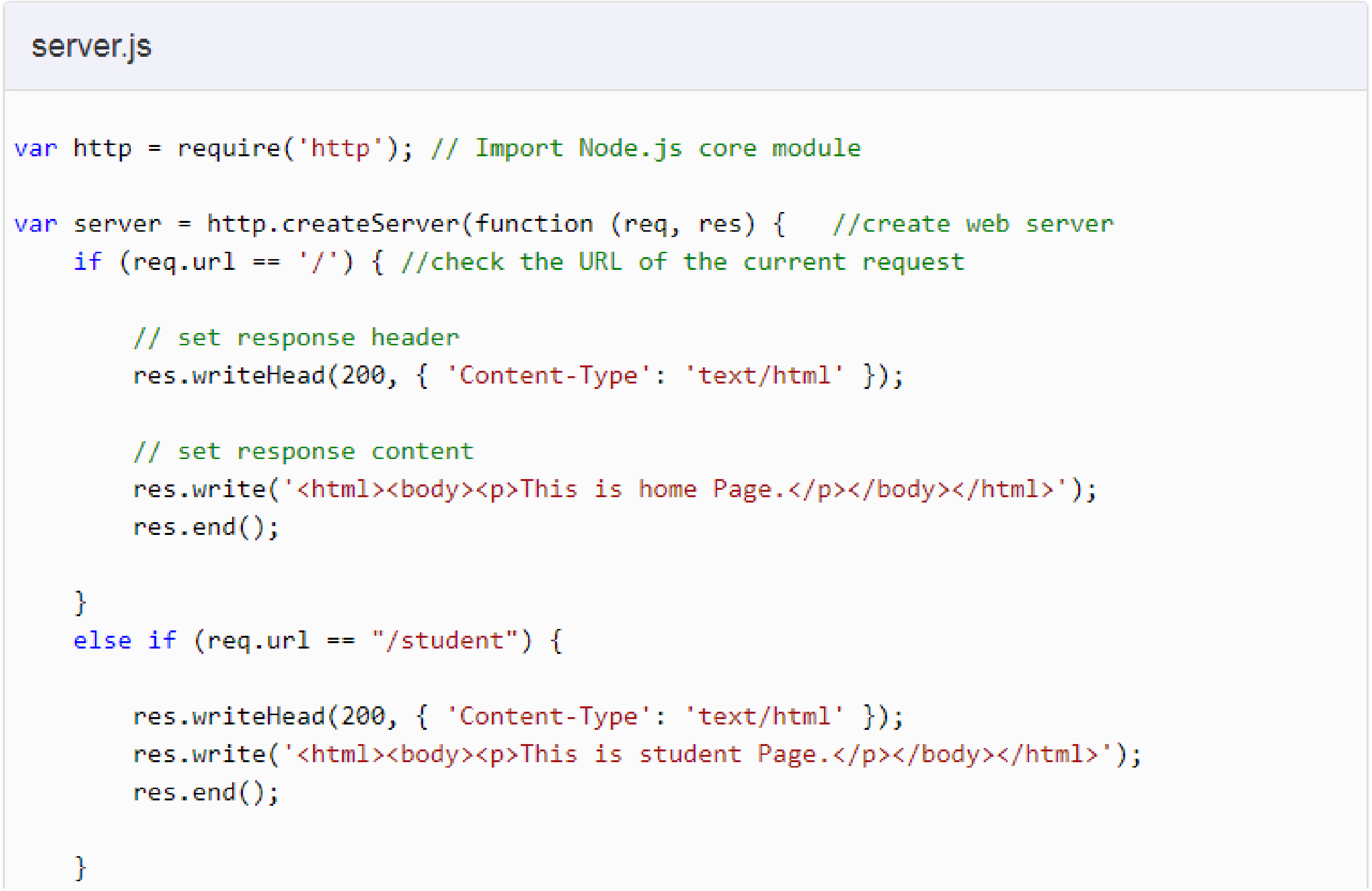
Handling HTTP Request & Response

- http.createServer() method includes request and response parameters which is supplied by Node.js.

- The request object can be used to get information about the current HTTP request e.g., url, request header, and data

- The response object can be used to send a response for a current HTTP request.

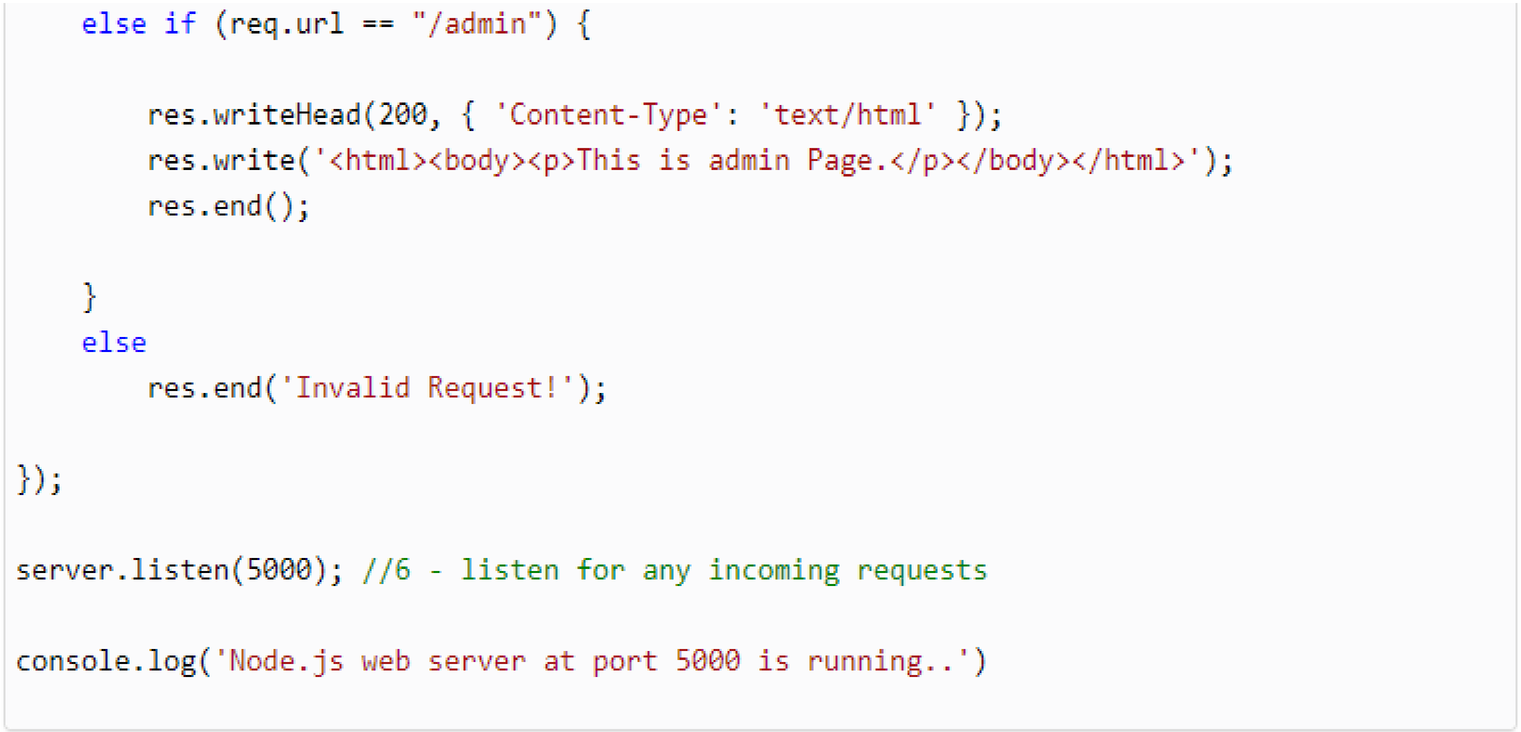
Example for HTTP Request & Response (1/4)



server.js

```
var http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) {   //create web server
    if (req.url == '/') { //check the URL of the current request

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

        // set response content
        res.write('<html><body><p>This is home Page.</p></body></html>');
        res.end();

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

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student Page.</p></body></html>');
        res.end();

    }
```

Example for HTTP Request & Response (2/4)



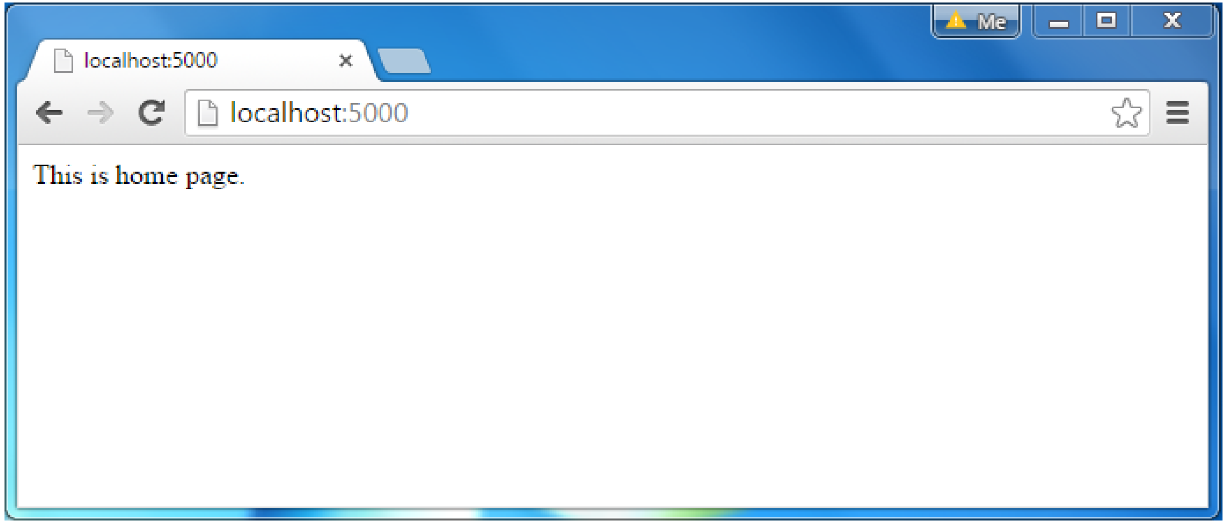
```
    else if (req.url == "/admin") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is admin Page.</p></body></html>');
        res.end();

    }
    else
        res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

Example for HTTP Request & Response (3/4)

- Execute using node command prompt to start the server

```
C:\> node server.js
Node.js web server at port 5000 is running..
```

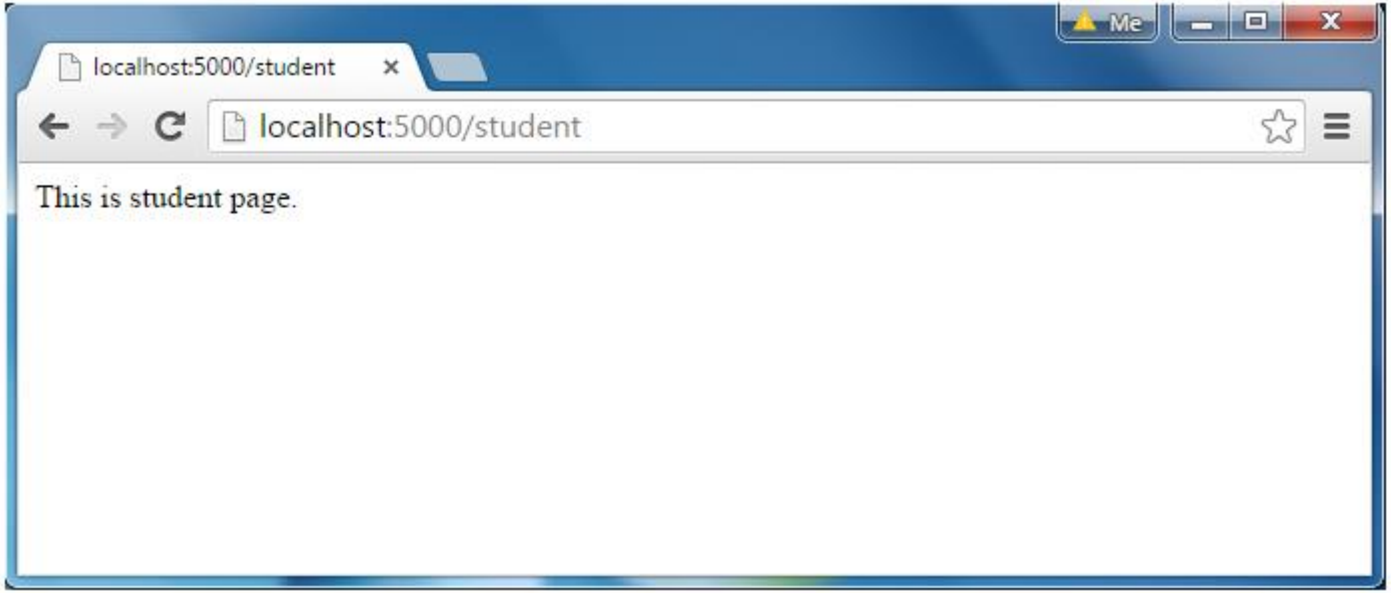
- Open in the browser with http://localhost:5000/ and if req.url path is “/”, then the below output will be displayed



This is home page.

Example for HTTP Request & Response (4/4)

- Open in the browser with http://localhost:5000/student and if req.url path is “/student”, then the below output will be displayed



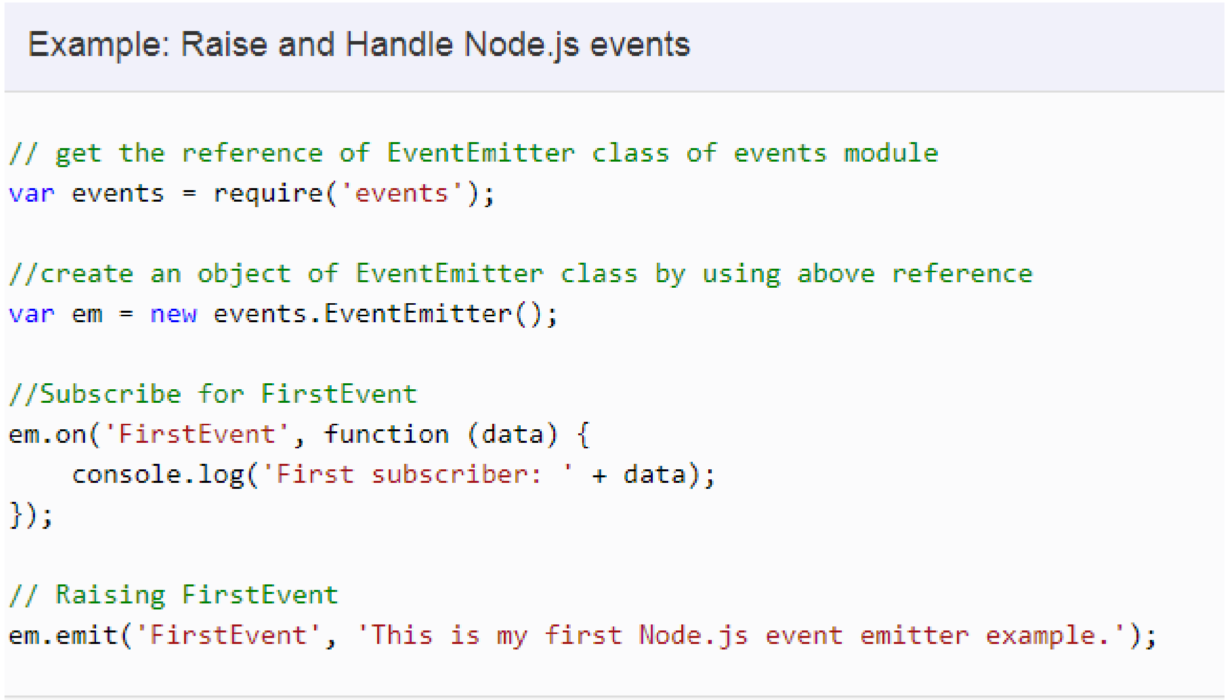
This is student page.

Event Handling - GET & POST implementation

Infosys® | Education, Training and Assessment

Node.js EventEmitter Class

- Node.js allows us to create and handle custom events easily by using events module.
- Event module includes EventEmitter class which can be used to raise and handle custom events.



Example: Raise and Handle Node.js events

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

on() and emit()

- on() – used to create the event with 2 arguments
  - Event name as 1st argument
  - Function to be executed as 2nd argument

```
em.on('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});
```

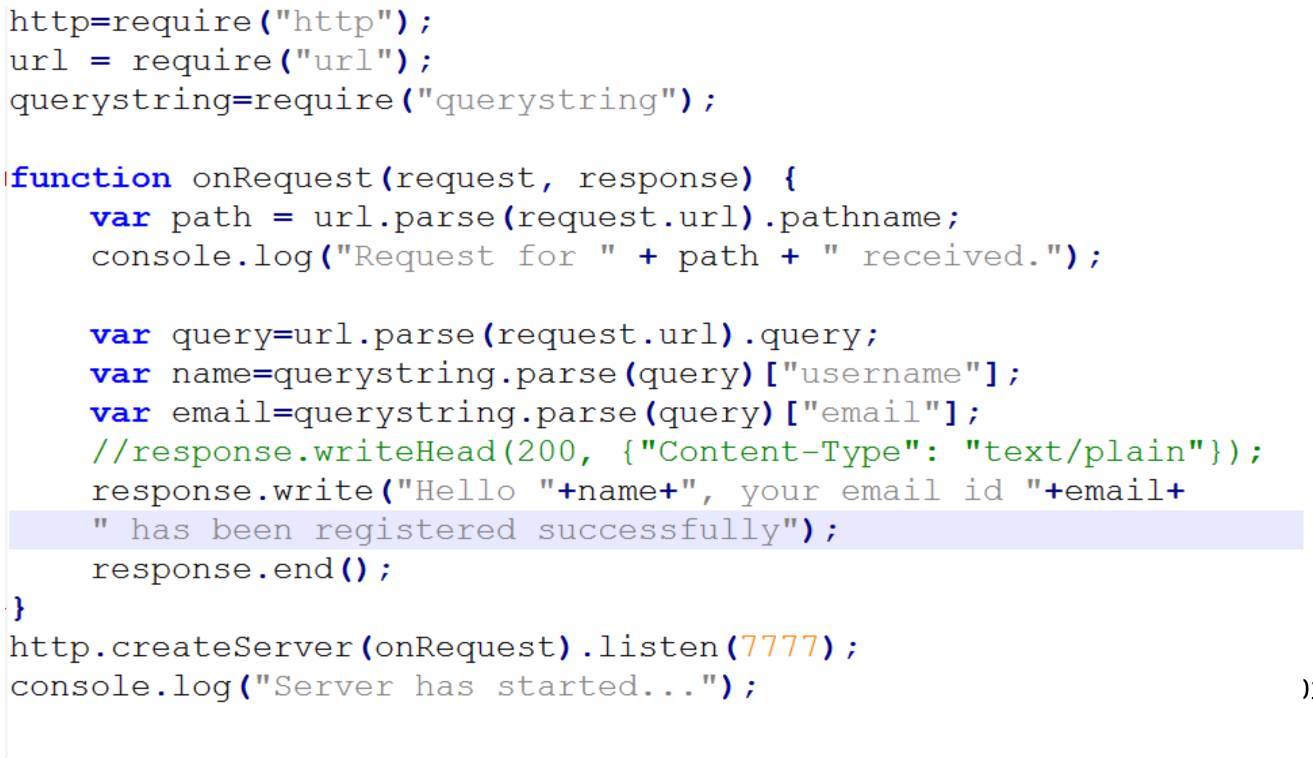
- emit() - used to invoke the event when required

```
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

Event Handling GET method (1/3)

- get() will show the request parameter in the URL and it can be easily retrieved using URL and QueryString Modules
- Eg:

Server.js

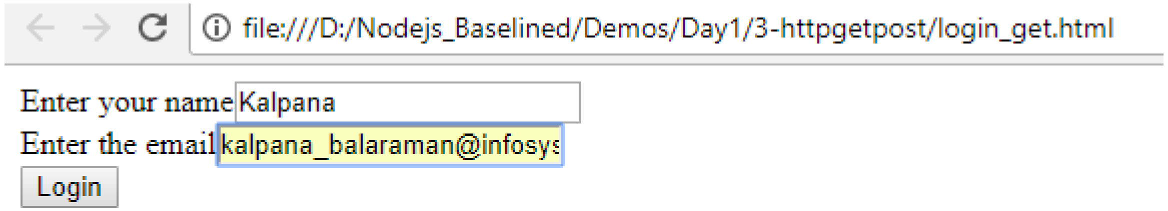


```
http=require("http");
url = require("url");
querystring=require("querystring");

function onRequest(request, response) {
    var path = url.parse(request.url).pathname;
    console.log("Request for " + path + " received.");

    var query=url.parse(request.url).query;
    var name=querystring.parse(query)["username"];
    var email=querystring.parse(query)["email"];
    //response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello "+name+", your email id "+email+
    " has been registered successfully");
    response.end();
}
http.createServer(onRequest).listen(7777);
console.log("Server has started...");
```

Event Handling GET method (2/3)

login_get.html

```
<html>
    <head>
        <title></title>
    </head>
    <body>
        <form action="http://localhost:7777/login" method="get">
            Enter your name<input type="text" name="username" value=""/><br/>
            Enter the email<input type="text" name="email" value=""/><br/>
            <input type="submit" name="login" value="Login"/>
        </form>
    </body>
</html>
```



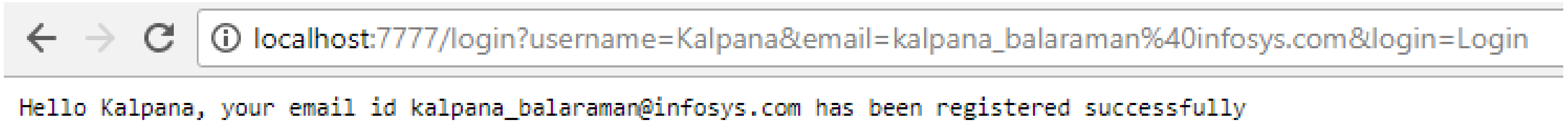
file:///D:/Nodejs_Baselined/Demos/Day1/3-httpgetpost/login_get.html

Enter your name Kalpana

Enter the email kalpana_balaraman@infosys

Event Handling GET method (3/3)

- Once the Login button is clicked, it will submit the action to http://localhost:7777/login to process http request and response using get()



```
localhost:7777/login?username=Kalpana&email=kalpana_balaraman%40infosys.com&login=Login
```

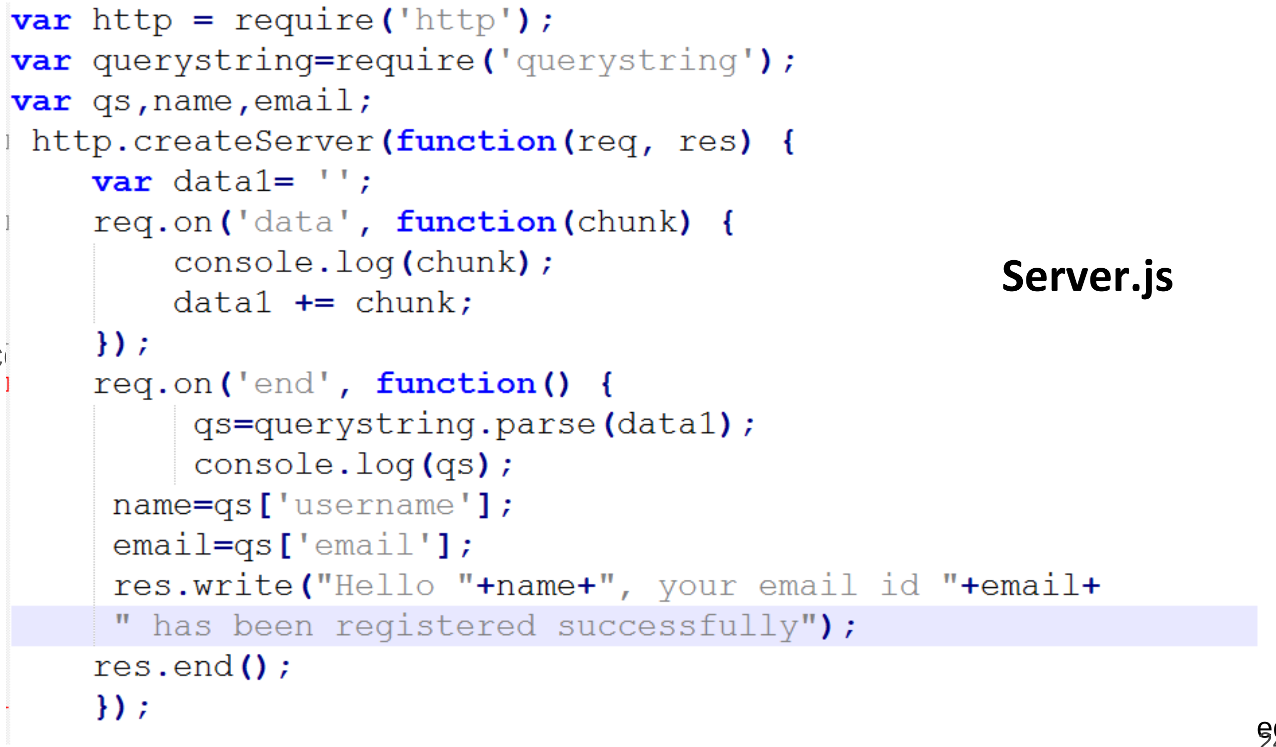
```
Hello Kalpana, your email id kalpana_balaraman@infosys.com has been registered successfully
```

Event Handling POST method (1/3)

- **post()** will process the request parameters using on() method of EventEmitter class and build-in events like ‘data’ and ‘end’.

- **‘data’ event** is used to read the request parameters

- **‘end’ event** is invoked onc ‘data’ event is completed to process the data retrieved using ‘data’ event



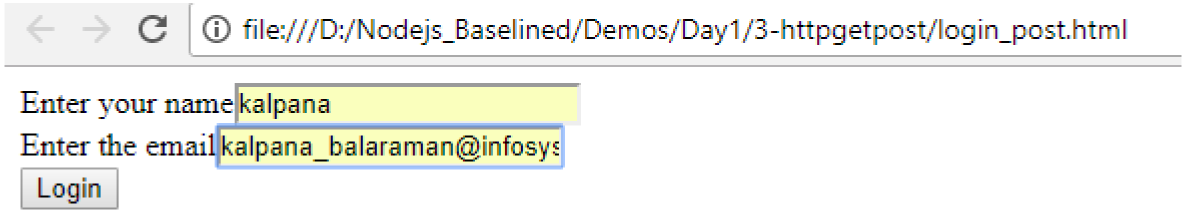
```
var http = require('http');
var querystring=require('querystring');
var qs,name,email;
 http.createServer(function(req, res) {
    var data1= '';
    req.on('data', function(chunk) {
        console.log(chunk);
        data1 += chunk;
    });
    req.on('end', function() {
         qs=querystring.parse(data1);
         console.log(qs);
     name=qs['username'];
     email=qs['email'];
     res.write("Hello "+name+", your email id "+email+
     " has been registered successfully");
    res.end();
    });
```

Server.js

Event Handling POST method (2/3)

login_post.html

```
<html>
    <head>
        <title></title>
    </head>
    <body>
        <form action="http://localhost:7777/login" method="post">
            Enter your name<input type="text" name="username" value=""/><br/>
            Enter the email<input type="text" name="email" value=""/><br/>
            <input type="submit" name="login" value="Login"/>
        </form>
    </body>
</html>
```

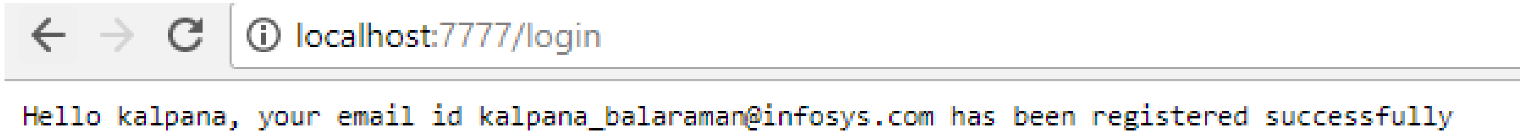

file:///D:/Nodejs_Baselined/Demos/Day1/3-httpgetpost/login_post.html

Enter your name kalpana

Enter the email kalpana_balaraman@infosys

Event Handling POST method (3/3)

- Once the Login button is clicked, it will submit the action to http://localhost:7777/login to process http request and response using post()



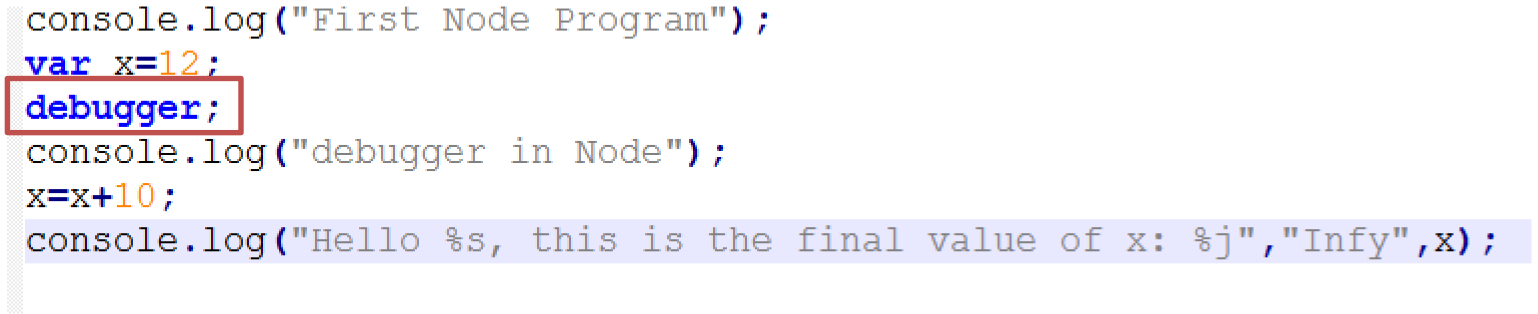
```
Hello kalpana, your email id kalpana_balaraman@infosys.com has been registered successfully
```

Debugging(1/2)

Debugging of applications can be done using the following tools

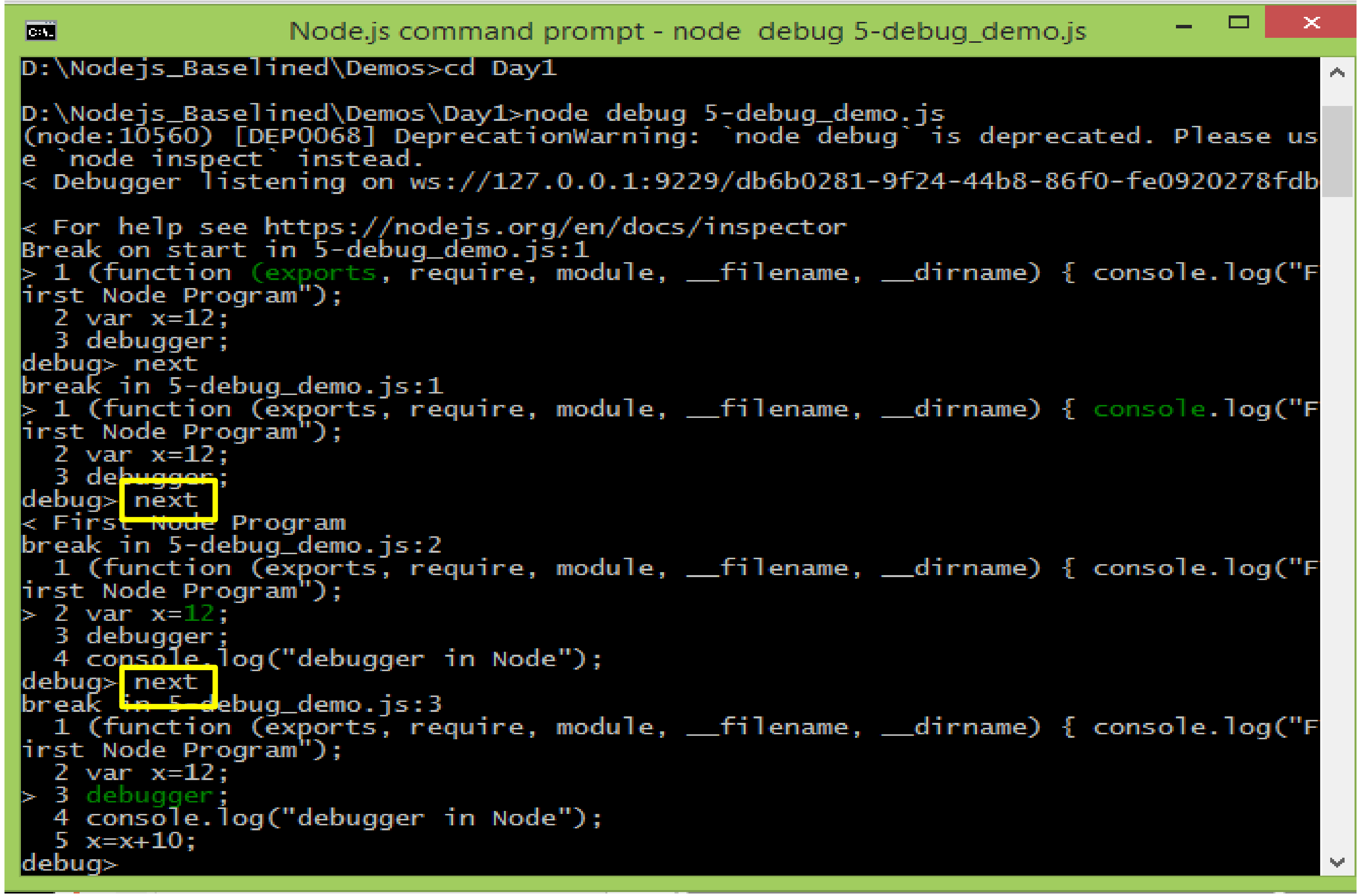
1. **Console.log()** - use simple console.log statements wherever required and Press F12 in browser to view the console.log() information's

2. Node's built-in debugger - set the break point using **"debugger"** command in the program and execute in debug mode using node command prompt as **"node debug filename.js"**



```
console.log("First Node Program");
var x=12;
debugger;
console.log("debugger in Node");
x=x+10;
console.log("Hello %s, this is the final value of x: %j","Infy",x);
```

Debugging(2/2)

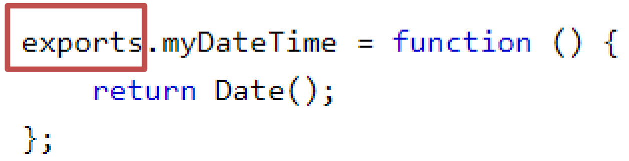

Node.js command prompt - node debug 5-debug_demo.js

```
D:\Nodejs_Baselined\Demos>cd Day1

D:\Nodejs_Baselined\Demos\Day1>node debug 5-debug_demo.js
(node:10560) [DEP0068] DeprecationWarning: `node debug` is deprecated. Please us
e `node inspect` instead.
< Debugger listening on ws://127.0.0.1:9229/db6b0281-9f24-44b8-86f0-fe0920278fdb

< For help see https://nodejs.org/en/docs/inspector
Break on start in 5-debug_demo.js:1
> 1 (function (exports, require, module, __filename, __dirname) { console.log("F
irst Node Program");
  2 var x=12;
  3 debugger;
debug> next
break in 5-debug_demo.js:1
> 1 (function (exports, require, module, __filename, __dirname) { console.log("F
irst Node Program");
  2 var x=12;
  3 debugger;
debug> next
< First Node Program
break in 5-debug_demo.js:2
  1 (function (exports, require, module, __filename, __dirname) { console.log("F
irst Node Program");
> 2 var x=12;
  3 debugger;
  4 console.log("debugger in Node");
debug> next
break in 5-debug_demo.js:3
  1 (function (exports, require, module, __filename, __dirname) { console.log("F
irst Node Program");
  2 var x=12;
> 3 debugger;
  4 console.log("debugger in Node");
  5 x=x+10;
debug>
```

Modules in Node.js

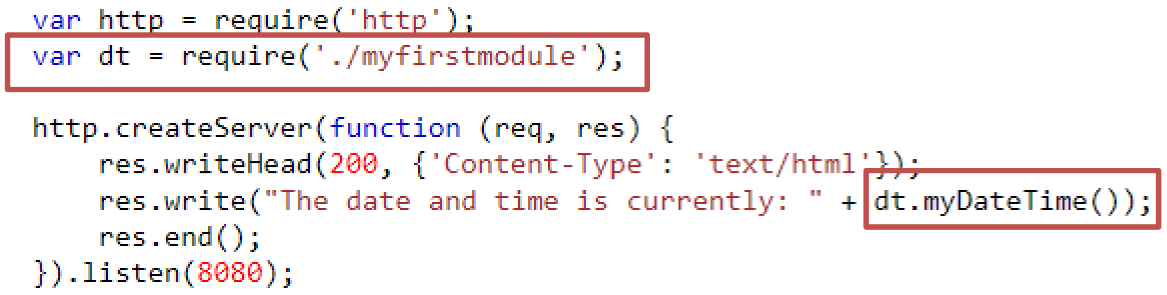
- Each JavaScript file in Node.js is a Module.
- We can reuse the code by importing one module into another module
- “**export**” keyword is used to provide permission to functions to access outside the module
- Module System in node is based on CommonJS module specification

myfirstmodule.js



```
exports.myDateTime = function () {
    return Date();
};
```

server.js



```
var http = require('http');
var dt = require('./myfirstmodule');

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write("The date and time is currently: " + dt.myDateTime());
    res.end();
}).listen(8080);
```

Connect to NoSQL Database using Node JS

Infosys® | Education, Training and Assessment

Connect Node.js with NoSQL MongoDB Database

- MongoDB is one of the most popular databases used along with Node.js.
- Node.js has the ability to work with both MySQL and MongoDB as databases.
- You need to download and use the required modules using the Node package manager.

Databases	Module and Command to Install
MySQL	npm install mysql
MongoDB	npm install mongojs

Connection to MongoDB

- We can now start building our JavaScript application and connect to our MongoDB server:

```
// app.js
var databaseUrl = "mydb"; // "username:password@example.com/mydb"
var collections = ["users", "reports"]
var db = require("mongojs").connect(databaseUrl, collections);
```

- The databaseUrl can contain the database server host and port along with the database name to connect to. By default the host is “localhost” and the port is “27017”

Implementation of CRUD operations

Infosys® | Education, Training and Assessment

Retrieve Data from Collections

- Collections is a set (array) of documents.
- Find() – used to retrieve the data from collections

Syntax:

db.collectionname.find([condition], callback function());

```
// app.js
db.users.find({gender: "female"}, function(err, users) {
  if( err || !users)
        console.log("No female users found");
  else
        users.forEach( function(femaleUser) {
                console.log(femaleUser);
         } );
});
```

Insert a record to Collection

- Save() method is used to insert a new record to Collection.

Syntax:

db.collectionname.save([args], callback function());

- Let us see how do I save a new user in my collection:

```
// app.js
db.users.save({email: "srirangan@gmail.com", password: "iLoveMongo",
gender: "male"}, function(err, saved) {
  if( err || !saved )
        console.log("User not saved");
  else
        console.log("User saved");
});
```

Modify a record in Collection

- Update() – used to modify the existing data in a collection.

Syntax:

db.collectionname.update([args], callback function());

- Let us see how do I modify the existing user:

```
// app.js
db.users.update({email: "srirangan@gmail.com"}, {$set: {password: "iReallyLoveMongo"}},
function(err, updated) {
  if( err || !updated )
        console.log("User not updated");
  else
        console.log("User updated");
});'
```

Delete a record in Collections

- remove() – used to delete records from collection.

Syntax:

db.collectionname.remove([condition], callback function());

- Let us see how do I delete a record:

```
// app.js
db.users.remove({gender: "female"}, function(err, users) {
  if( err || !users)
        console.log("No female users found");
  else
        console.log(“Record Deleted Successfully”);
 });
```

Summary

- You are now knowledgeable on :
  - Features and Need of Node.js
  - Architecture, Installation and setup
  - Creating web servers with HTTP (Request & Response)
  - Event Handling - GET & POST implementation
  - Connect to NoSQL Database using Node JS
  - Implementation of CRUD operations

Reference

- Links:
  - https://www.w3schools.com/nodejs/default.asp
  - https://www.tutorialspoint.com/nodejs/index.htm
  - https://www.guru99.com/node-js-tutorial.html
  - https://www.javatpoint.com/nodejs-tutorial
  - www.tutorialsteacher.com/nodejs/nodejs-tutorials
  - https://www.codeschool.com/courses/real-time-web-with-node-js

- Videos:
  - Node.js Tutorial For Absolute Beginners - YouTube

Thank You

● ● ●

Infosys
be more