ADITYA
ENGINEERING COLLEGE (A)

What is Node.js?

- Node.js is an open source server environment.
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server.
- **Node.js uses asynchronous programming!**

ADITYA ENGINEERING COLLEGE (A)

What is Node.js?

- Node.js is an open-source JavaScript runtime environment that is built on Chrome's V8 JavaScript engine.
- It allows developers to execute JavaScript code on the server side, rather than just in a web browser.
- Node.js uses an event-driven, non-blocking I/O model that makes it efficient and lightweight.
- It has a rich ecosystem of modules and packages available through the Node Package Manager (NPM), which allows developers to easily add functionality to their applications.

What is Node.js?

- A common task for a web server can be to open a file on the server and return the content to the client.

- Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

- Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Why Node.js?

- Node.js eliminates the waiting and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

What Node.js can do?

- Node.js can generate dynamic page content.
- Node.js can create, open, read, write, delete, and close files on the server.
- Node.js can collect form data.
- Node.js can perform CRUD operations on your database.

What is a Node.js file?

- Node.js files contain tasks that will be executed on certain events.
- A typical event is someone trying to access a port on the server.
- Node.js files must be initiated on the server before having any effect.
- Node.js files have extension ".js“
- Download Node.js from https://nodejs.org

ADITYA ENGINEERING COLLEGE (A)

Steps to create and run a Node.js File

Step 1:

Create a Javascript file and save with .js in some directory

- NodeDemo.js

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

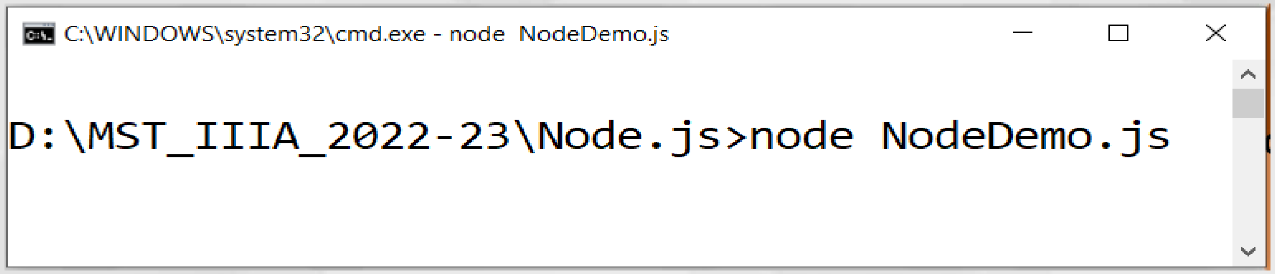
Step 2:

Go to Command Prompt and to the directory where the file is saved.

Steps to create and run a Node.js File

Step 3: Run the file using the following command

node NodeDemo.js

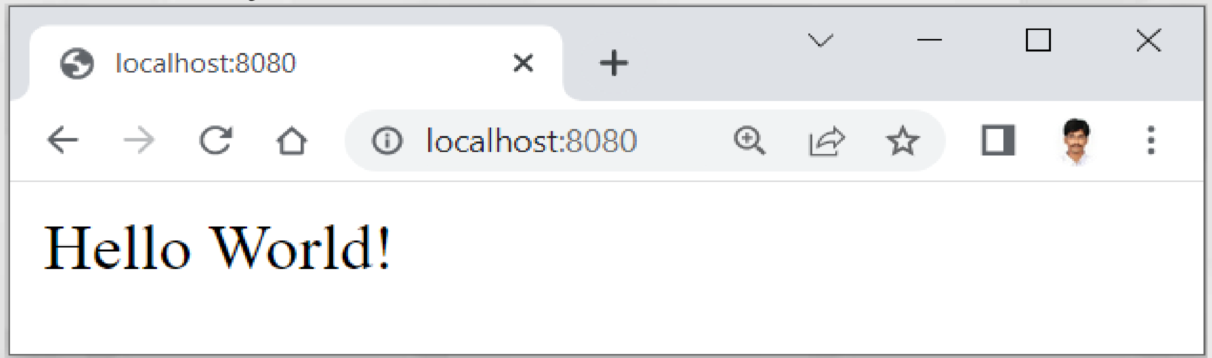


```
C:\WINDOWS\system32\cmd.exe - node  NodeDemo.js

D:\MST_IIIA_2022-23\Node.js>node NodeDemo.js
```

Step 4: Now, your computer works as a server and starts running.

Step 5: Now open the Browser and give URL

http://localhost:8080



Hello World!

Node.js Modules

- What is a Module in Node.js?

- Node.js modules are reusable pieces of code that can be used to add functionality to a Node.js application. They encapsulate related variables, functions, and classes, allowing them to be organized and imported into other parts of a program.

- Modules are of 2 types:
    1. Predefined-CORE MODULE,LOCAL MODULE
    2. User defined-THIRD PARTY MODULE

Node.js Modules

- 1. Core modules: These are built-in modules that come with the Node.js installation. They include modules such as `fs` (for file system operations), `http` (for creating web servers), `path` (for working with file paths), and many others.
- 2. Local modules: These are modules that are created within the project by the developer. They can be defined in separate JavaScript files and then imported into other files using the `require` keyword.

- 3. Third-party modules: These are modules developed by other developers and made available through the Node Package Manager (NPM). They can be installed in a Node.js project using the `npm install` command and then imported and used in the code.

Node.js HTTP Module

- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

- To include the HTTP module, use the require() method:

```
var http = require('http');
```

Node.js HTTP Module

- **Node.js as a Web Server**
- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

- Use the createServer() method to create an HTTP server:
-
```
var http = require('http');
//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!');//write response
to client
  res.end(); //end the response
}).listen(8080);//server object listens on
port 8080
```

ADITYA ENGINEERING COLLEGE (A)

Node.js HTTP Module

- **Add an HTTP Header:**
- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

- 1st argument is the status code, 200 --> all is OK, 2nd argument is an object containing the response headers.

Node.js HTTP Module

- **Add an HTTP Header:**
- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-
Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

- 1st argument is the status code, 200 --> all is OK, 2nd argument is an object containing the response headers.

Node Package Manager

- NPM (Node Package Manager) is the default package manager for Node.js and is written entirely in Javascript. Developed by Isaac Z. Schlueter, it was initially released in January 12, 2010.
- NPM manages all the packages and modules for Node.js and consists of command line client npm. It gets installed into the system with installation of Node.js. The required packages and modules in Node project are installed using NPM.

Installing NPM:

- To install NPM, it is required to install Node.js as NPM gets installed with Node.js automatically.

Checking the NPM Version:

Syntax: npm -v

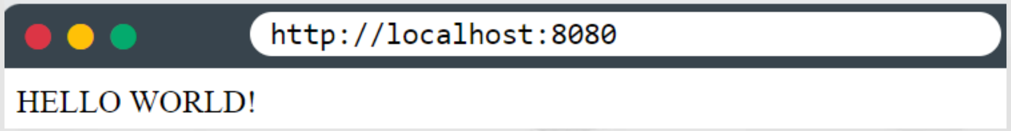
Node Package Manager

- **Download a Package:**
  Syntax: npm install upper-case

Program:

```
var http = require('http');
var uc = require('upper-case');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type':   'text/html'});
/*Use our upper-case module to upper case
a string:*/
res.write(uc.upperCase("Hello World!"));
res.end();
}).listen(8080);
```

Output:



```
http://localhost:8080
HELLO WORLD!
```

Modular Programming in Node JS

- In Node.js, Modules are the blocks of encapsulated code that communicates with an external application on the basis of their related functionality.
- Modules can be a single file or a collection of multiples files/folders.

Modules are of three types:
1.Core Modules
2.local Modules
3.Third-party Modules

Core Modules: Node.js has many built-in modules that are part of the platform and comes with Node.js installation. These modules can be loaded into the program by using the require function.

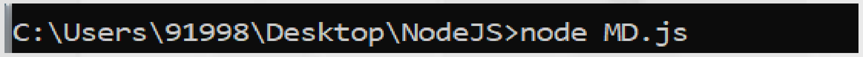
Syntax: var module = require('module_name');

Modular Programming in Node JS

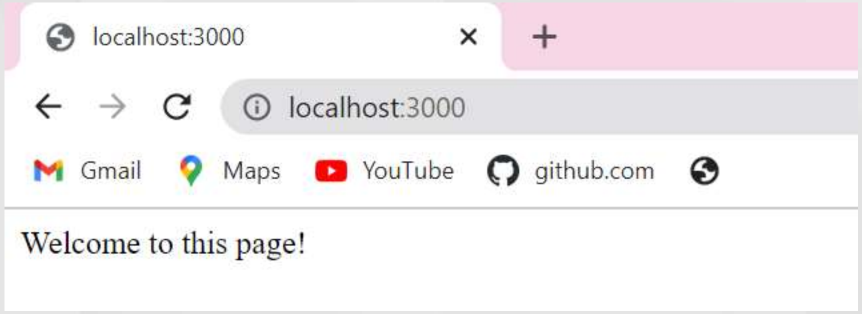
Program:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type':
'text/html'});
  res.write('Welcome to this page!');
  res.end();
}).listen(3000);
```

Output:



```
C:\Users\91998\Desktop\NodeJS>node MD.js
```

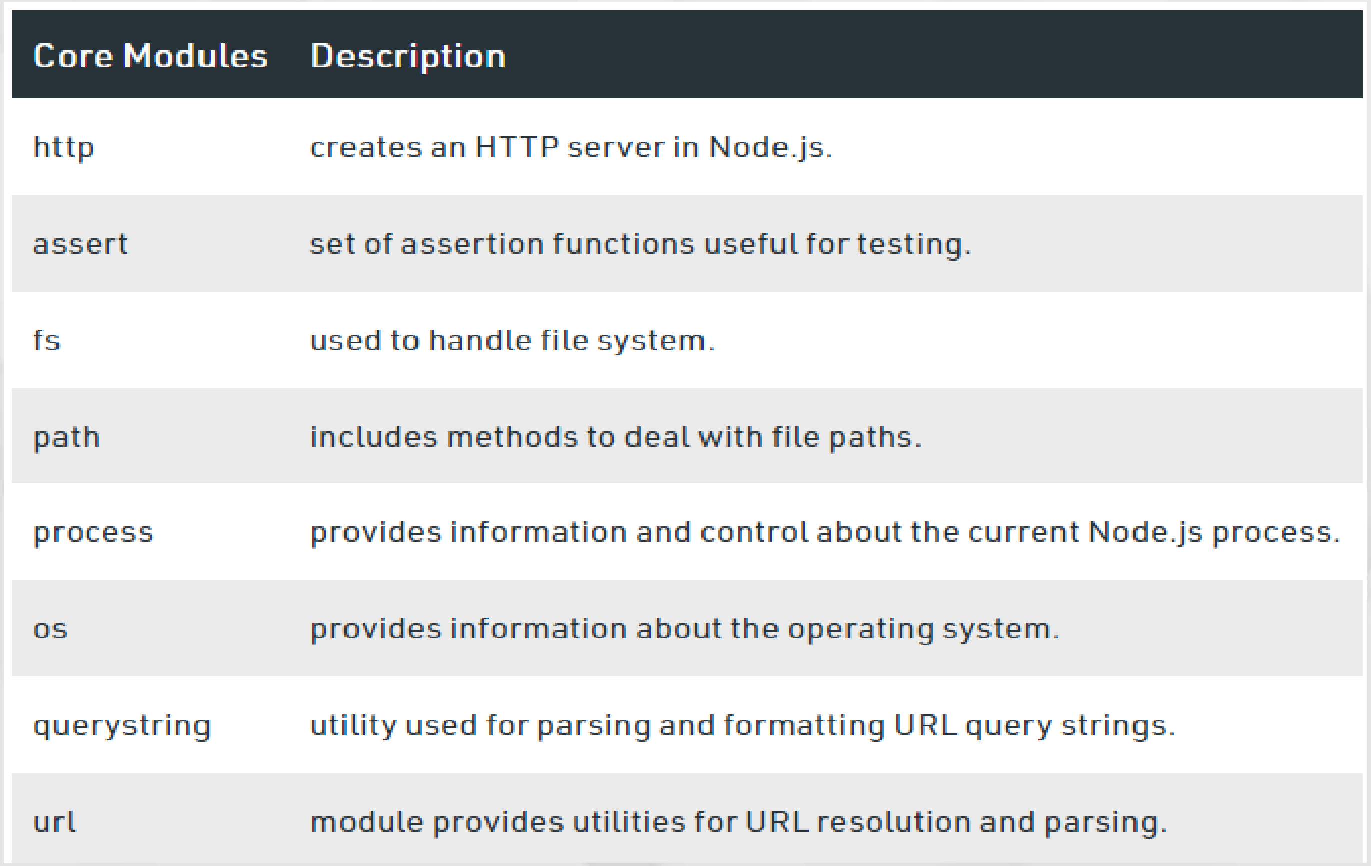


Welcome to this page!

ADITYA ENGINEERING COLLEGE (A)

Modular Programming in Node JS

The following list contains some of the important core modules in Node.js:



Core Modules	Description
http	creates an HTTP server in Node.js.
assert	set of assertion functions useful for testing.
fs	used to handle file system.
path	includes methods to deal with file paths.
process	provides information and control about the current Node.js process.
os	provides information about the operating system.
querystring	utility used for parsing and formatting URL query strings.
url	module provides utilities for URL resolution and parsing.

Modular Programming in Node JS

Local Modules: Unlike built-in and external modules, local modules are created locally in your Node.js application.

Filename: calc.js

```
exports.add = function (x, y) {
    return x + y;
};
exports.sub = function (x, y) {
    return x - y;
};
exports.mult = function (x, y) {
    return x * y;
};
exports.div = function (x, y) {
    return x / y;
};
```

Modular Programming in Node JS

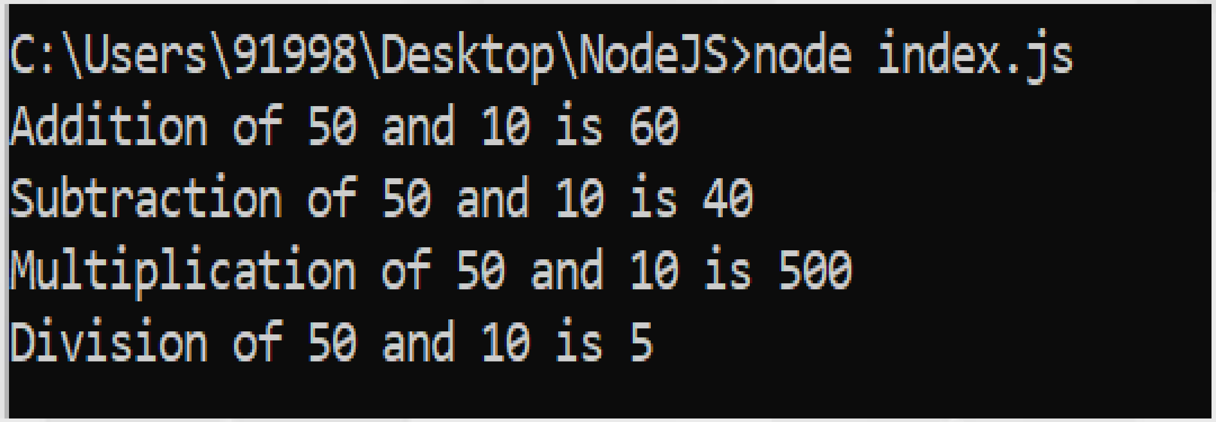
Since this file provides attributes to the outer world via exports, another file can use its exported functionality using the require() function.

Filename: index.js

```
var calculator = require('./calc');
var x = 50, y = 10;
console.log("Addition of 50 and 10 is "
                   + calculator.add(x, y));

console.log("Subtraction of 50 and 10 is "
                   + calculator.sub(x, y));

console.log("Multiplication of 50 and 10 is "
                   + calculator.mult(x, y));

console.log("Division of 50 and 10 is "
                   + calculator.div(x, y));
```

Modular Programming in Node JS

Output:



```
C:\Users\91998\Desktop\NodeJS>node index.js
Addition of 50 and 10 is 60
Subtraction of 50 and 10 is 40
Multiplication of 50 and 10 is 500
Division of 50 and 10 is 5
```

Third-party modules: Third-party modules are modules that are available online using the Node Package Manager(NPM). These modules can be installed in the project folder or globally. Some of the popular third-party modules are mongoose, express, angular, and react.

Example:

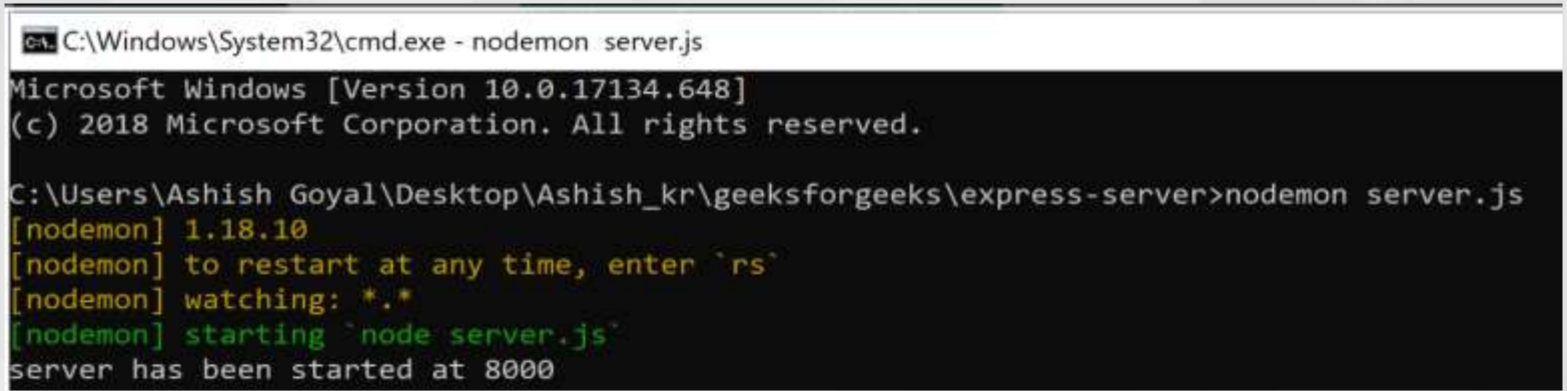
npm install express
npm install mongoose
npm install -g @angular/cli

Restarting Node Application

- Nodemon is a package for handling this restart process automatically when changes occur in the project file.
- Installing nodemon: nodemon should be installed globally in our system:
  Windows system: npm i nodemon -g
  Linux system: sudo npm i nodemon -g

To check the nodemon version:
Syntax: nodemon -v

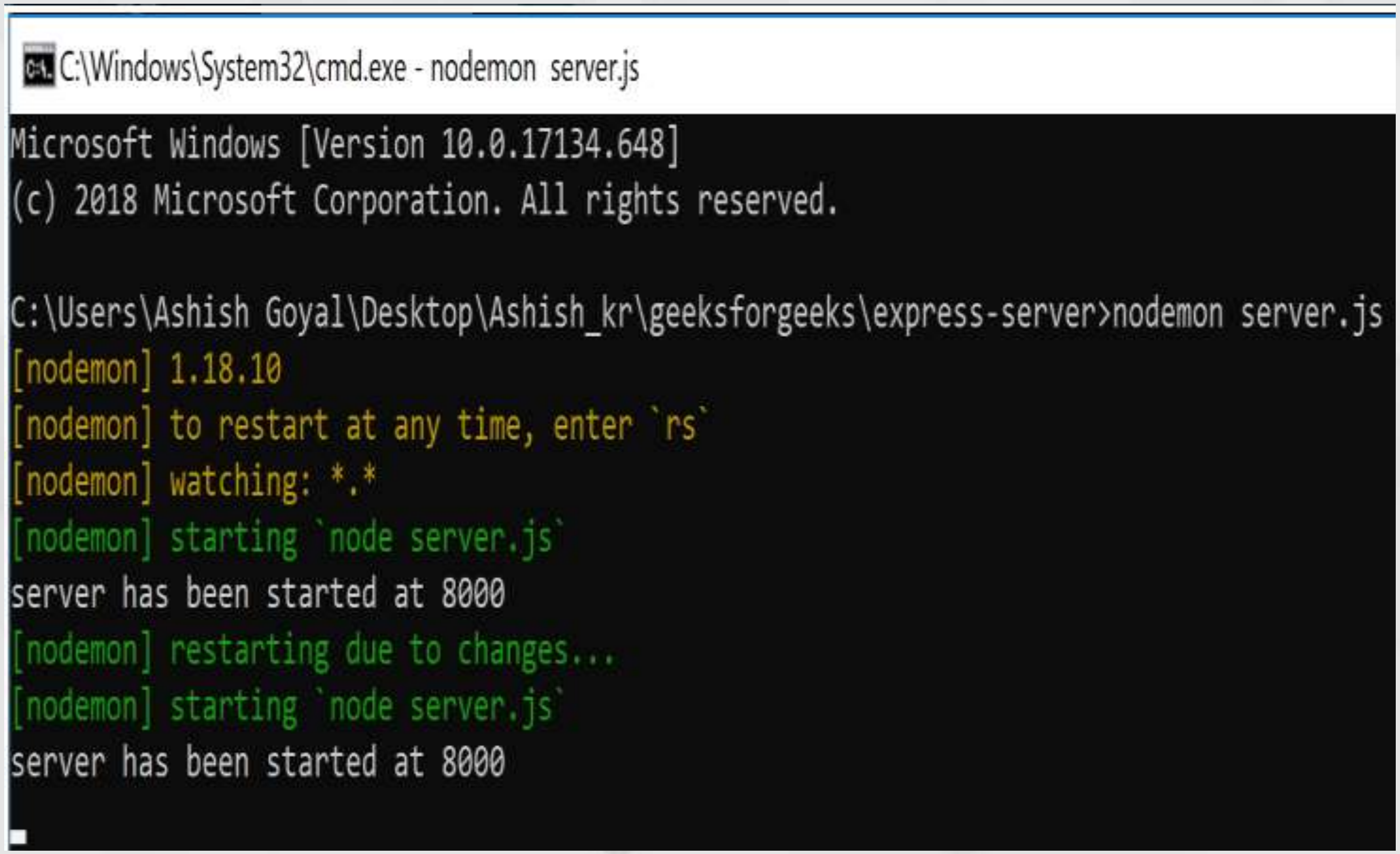
Starting node server with nodemon:


C:\Windows\System32\cmd.exe - nodemon server.js

```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ashish Goyal\Desktop\Ashish_kr\geeksforgeeks\express-server>nodemon server.js
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
server has been started at 8000
```

Restarting Node Application

Now, when we make changes to our nodejs application, the server automatically restarts by nodemon



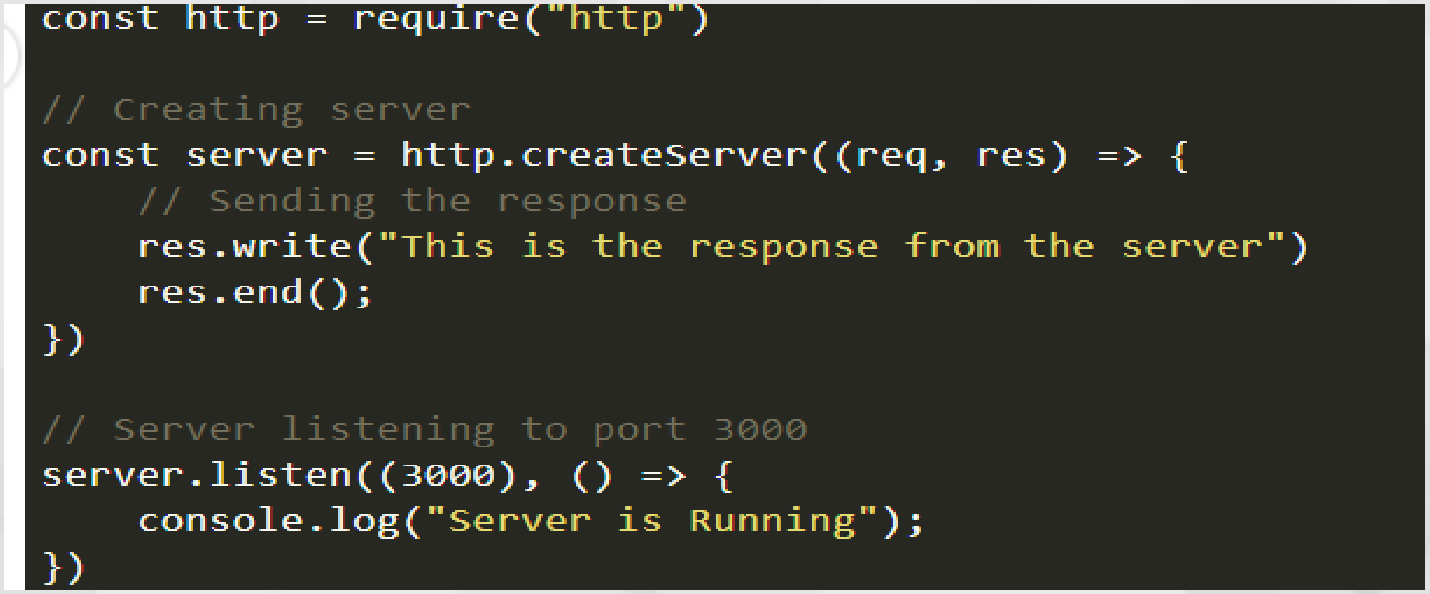
C:\Windows\System32\cmd.exe - nodemon server.js

```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ashish Goyal\Desktop\Ashish_kr\geeksforgeeks\express-server>nodemon server.js
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
server has been started at 8000
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
server has been started at 8000
```

Restarting Node Application

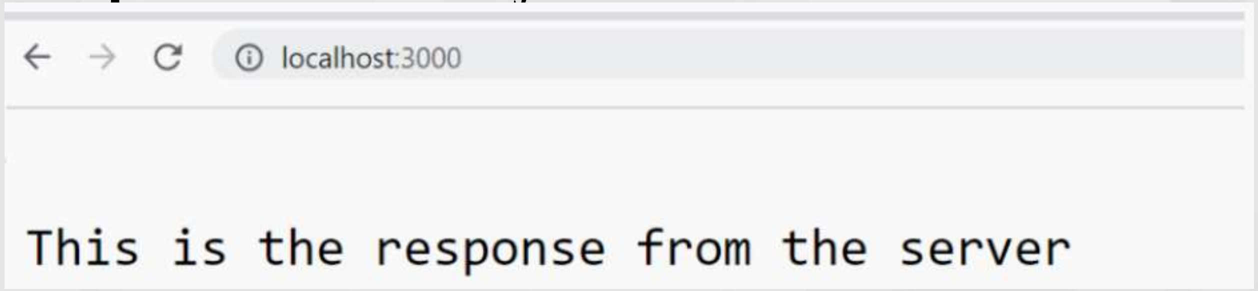
Program:

//File Name: index.js



```
const http = require("http")

// Creating server
const server = http.createServer((req, res) => {
    // Sending the response
    res.write("This is the response from the server")
    res.end();
})

// Server listening to port 3000
server.listen((3000), () => {
    console.log("Server is Running");
})
```

Output: node index.js



localhost:3000

```
This is the response from the server
```

File Operations in NodeJS

- Node.js as a File Server
- The Node.js file system module allows you to work with the file system on your computer.
- To include the File System module, use the require() method:
  var fs = require('fs');
- Common use for the File System module:
  1.Read files
  2.Create files
  3.Update files
  4.Delete files
  5.Rename files

- **Read Files:**
  The **fs.readFile()** method is used to read files on your computer.

File Operations in NodeJS

- //FileName: demofile1.html

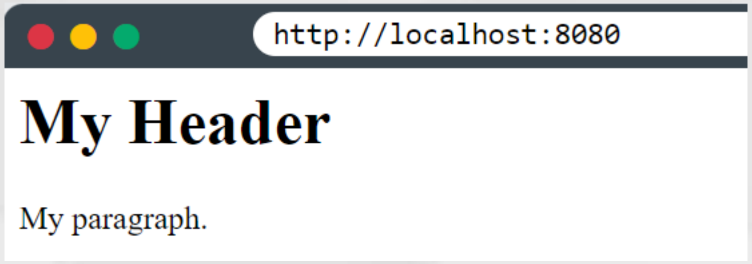
```
 <html>
<body>
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

- **Program:**

```
  var http = require('http');
  var fs = require('fs');
  http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(data);
  return res.end();
  });
}).listen(8080);
```

File Operations in NodeJS

Output:


```
http://localhost:8080
```
My Header

My paragraph.

- **Create Files:**

The File System module has methods for creating new files:
1. fs.appendFile()
2. fs.open()
3. fs.writeFile()

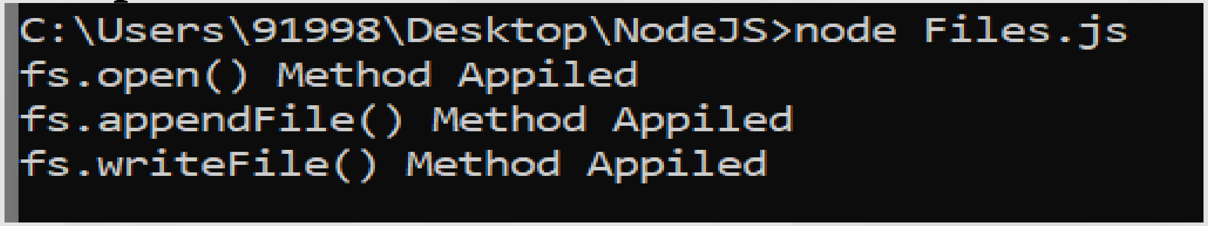
- The fs.appendFile() method appends specified content to a file.
- The fs.open() method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created
- The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created

File Operations in NodeJS

Program:

```
var fs = require('fs');
fs.appendFile('mynewfile1.txt', 'Hello content!', function
(err) {
  if (err) throw err;
  console.log('fs.appendFile() Method Appiled');
});
fs.open('mynewfile2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('fs.open() Method Appiled');
});
fs.writeFile('mynewfile3.txt', 'Hello content!', function
(err) {
  if (err) throw err;
  console.log('fs.writeFile() Method Appiled');
});
```

Output:



```
C:\Users\91998\Desktop\NodeJS>node Files.js
fs.open() Method Appiled
fs.appendFile() Method Appiled
fs.writeFile() Method Appiled
```

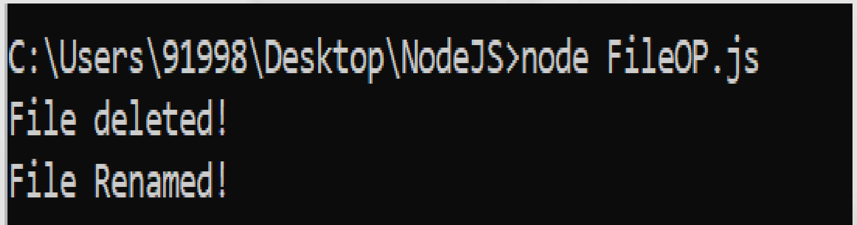
File Operations in NodeJS

- **Delete Files:**
  To delete a file with the File System module, use the fs.unlink() method.
  The **fs.unlink()** method deletes the specified file.
- **Rename Files:**
  To rename a file with the File System module, use the fs.rename() method.
  The **fs.rename()** method renames the specified file.

Program:

```
var fs = require('fs');
fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err)
{
  if (err) throw err;
  console.log('File Renamed!');
});
```

Output:



```
C:\Users\91998\Desktop\NodeJS>node FileOP.js
File deleted!
File Renamed!
```

Express Development Environment

We can install it with npm. Make sure that you have Node.js and npm installed.

Step 1: Creating a directory for our project and make that our working directory.

```
$ mkdir gfg
$ cd gfg
```

Step 2: Using npm init command to create a package.json file for our project.

```
$ npm init
```

Step 3: Now in your gfg(name of your folder) folder type the following command line:

Syntax : npm install express --save

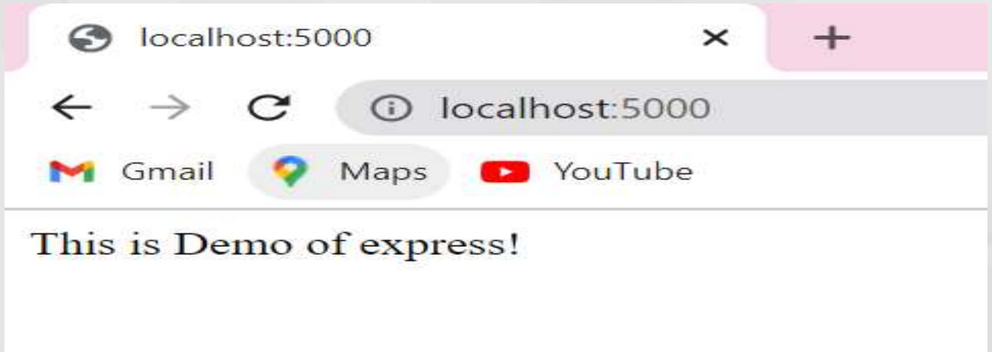
Express Development Environment

Program:

//File Name: app.js

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
res.send("This is Demo of express!");
});
app.listen(5000);
```

Output: node app.js



This is Demo of express!

Defining a route:

A route can be defined as shown below
router.method(path,handler)

router: express instance or router instance
method: one of the HTTP verbs
path: is the route where request runs
handler: is the callback function that gets triggered whenever a request comes to a particular path for a matching request type

Route Method:
The application object has different methods corresponding to each of the HTTP verbs (GET,POST, PUT, DELETE). These methods are used to receive HTTP requests.

Defining a route:

Below are the commonly used route methods and their description:

Method Description
get() Use this method for getting data from the server.
post() Use this method for posting data to a server.
put() Use this method for updating data in the server.
delete() Use this method to delete data from the server.
all() This method acts like any of the above methods. This can be used to handle all requests.

```
routing.get("/notes", notesController.getNotes);
routing.post("/notes", notesController.newNotes);
routing.all("*", notesController.invalid);
```

notesController is the custom js file created to pass the navigation to this controller file. Inside the controller, we can create methods like getNotes,newNotes, updateNotes.

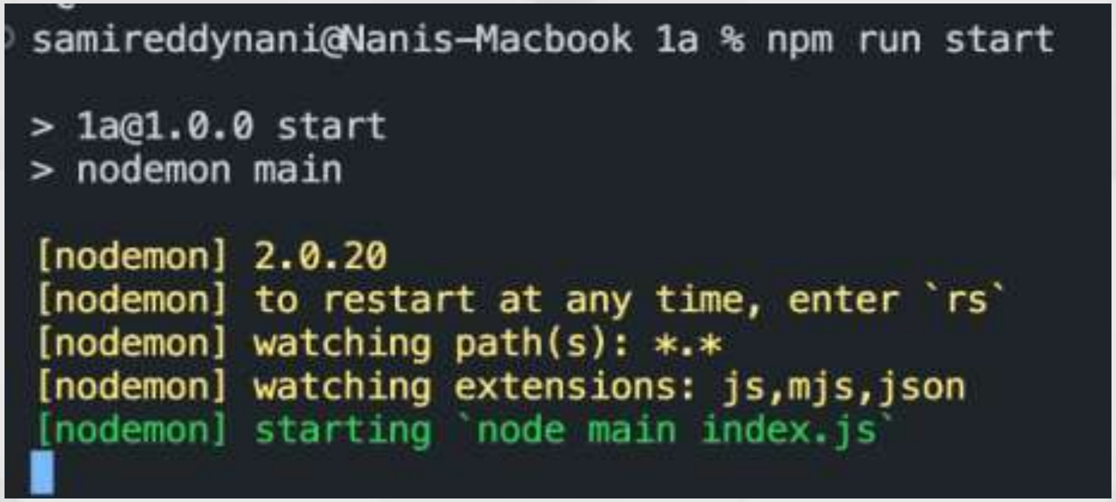
Handling Routes:

Program:

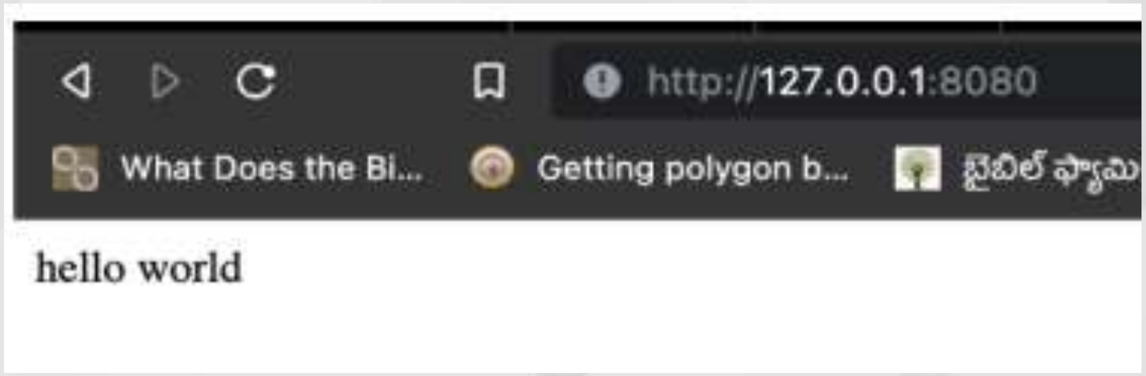
```
const express = require('express')
const app = express()
app.all((req, res, next) => {
console.log('Accessing the secret section ...')
next()
 })
app.get('/', (req, res) => {
res.send('hello world') })
app.get('/home', (req, res) => {
res.send('home page')
 })
app.get('/*', (req, res) => {
res.send('error 404 page not found')
 })
app.post('/', (req, res) => {
res.send('POST request to the homepage')
})
app.listen(8080, () => {
console.log('server started on port 8080')
})
```

Handling Routes:

Output:



```
samireddynani@Nanis-Macbook 1a % npm run start

> 1a@1.0.0 start
> nodemon main

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node main index.js`
```



hello world

Routing and Query Parameters

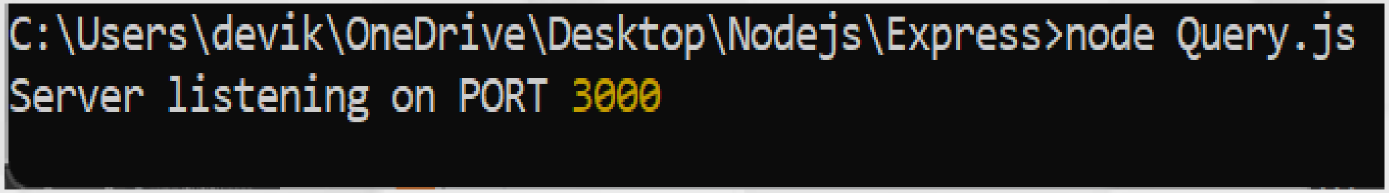
In Express.js, you can directly use the req.query() method to access the string variables. As per the documentation, the req.param method only gets the route parameters, whereas the req.query method checks the query string parameters.

Program:

```
var express = require('express');
var app = express();
var PORT = 3000;

app.get('/api', function(req, res){
   console.log('id: ' + req.query.id)
   res.send('id: ' + req.query.id);
});

app.listen(PORT, function(err){
   if (err) console.log(err);
   console.log("Server listening on PORT", PORT);
});
```

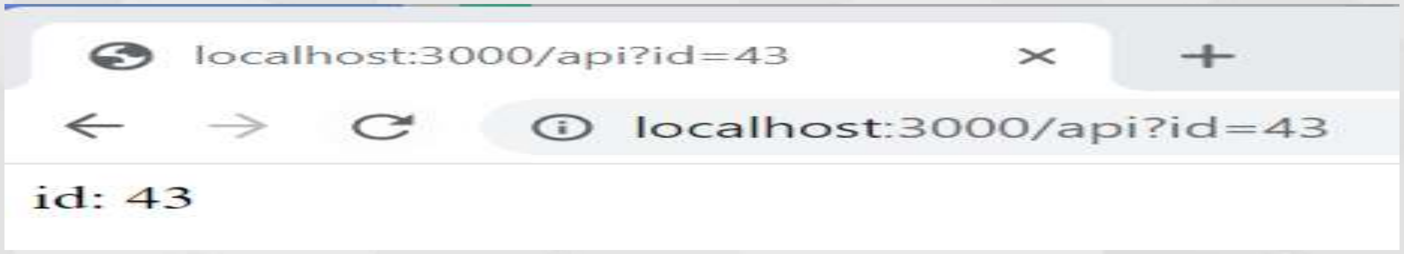
Routing and Query Parameters

Output:



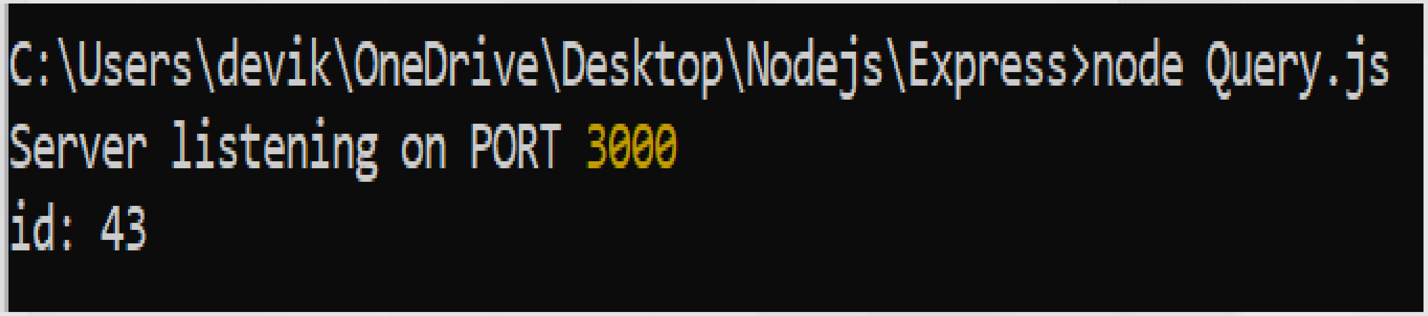
```
C:\Users\devik\OneDrive\Desktop\Nodejs\Express>node Query.js
Server listening on PORT 3000
```

Hit the endpoint "**localhost:3000/api?id=43**" with a GET Request –



```
localhost:3000/api?id=43
```

```
id: 43
```



```
C:\Users\devik\OneDrive\Desktop\Nodejs\Express>node Query.js
Server listening on PORT 3000
id: 43
```

Middlewares

- A middleware can be defined as a function for implementing different cross-cutting concerns such as authentication, logging, etc. The main arguments of a middleware function are the request object, response object, and the next middleware function defined in the application.
- A function defined as a middleware can execute any task mentioned below:

✓ Any code execution.

✓ Modification of objects - request and response.

✓ Call the next middleware function.

✓ End the cycle of request and response.

How Middleware works?

In order to understand how middleware works, let us take a scenario where we want to log

the request method and request URL before the handler executes. The route definition for which we want to add the middleware.

```
 app.get('/login', myController.myMethod);
exports.myMethod = async (req, res, next) => {
res.send('/login’);
};
```

Program:

Route1.js file

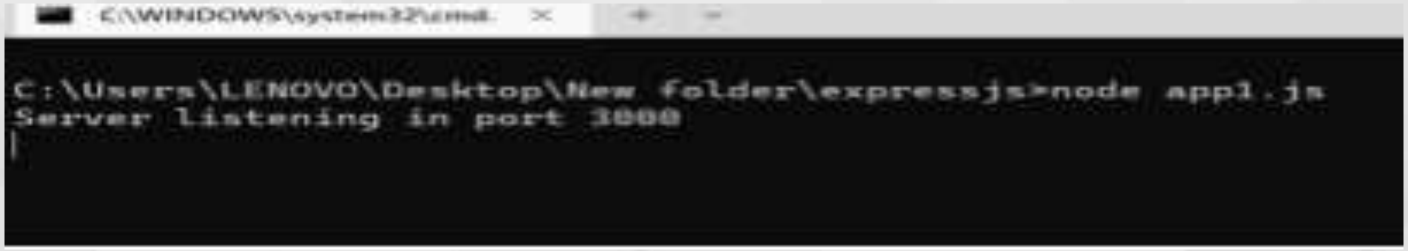
```
const express = require('express');
const router = express.Router();
const myController =
require('../Controller/myNotes1');
router.get('/', myController.myMethod);
router.get('/about', myController.aboutMethod);
module.exports = router;
```

myNotes.js File

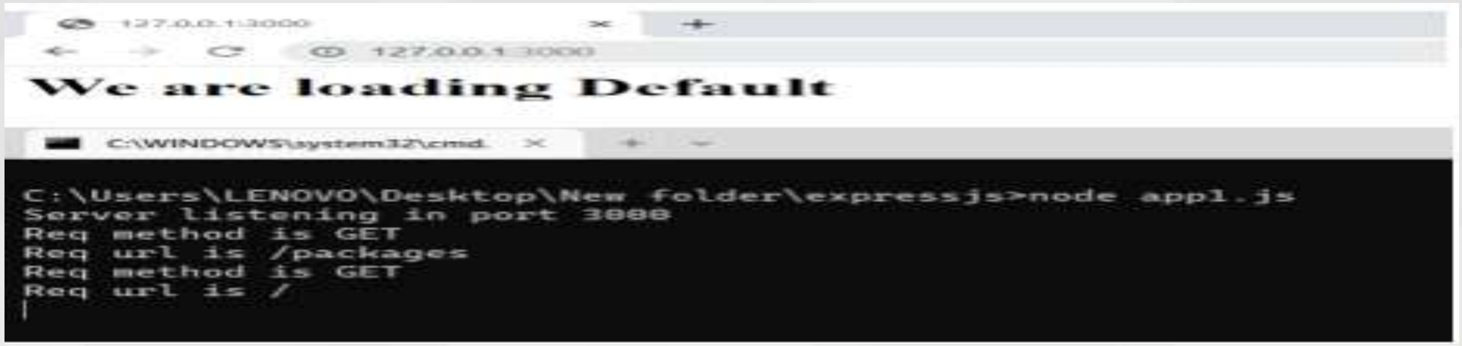
```
exports.myMethod = async (req, res, next) => {
res.send('<h1>Welcome</h1>');
};
exports.aboutMethod = async (req, res, next) => {
res.send('<h1>About Us Page</h1>');
};
```

```
app.use(mylogger);
app.use('/', router);
app.listen(3000);
console.log('Server listening in port 3000’);
```

Output:



```
C:\Users\LENOVO\Desktop\New folder\expressjs>node app1.js
Server listening in port 3000
```

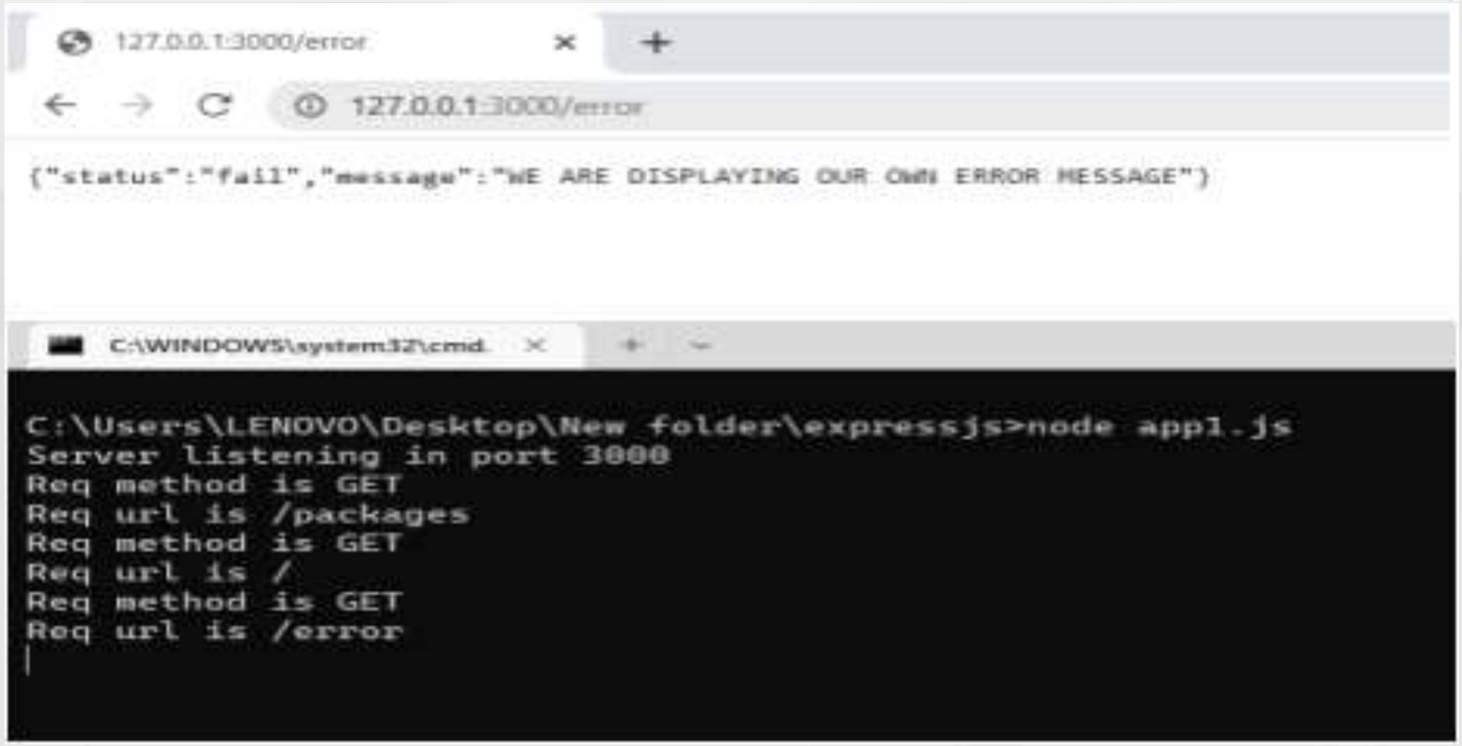


127.0.0.1:3000

We are loading Default

C:\WINDOWS\system32\cmd.

```
C:\Users\LENOVO\Desktop\New folder\expressjs>node appl.js
Server listening in port 3000
Req method is GET
Req url is /packages
Req method is GET
Req url is /
```



127.0.0.1:3000/error

```
{"status":"fail","message":"WE ARE DISPLAYING OUR OWN ERROR MESSAGE"}
```

C:\WINDOWS\system32\cmd.

```
C:\Users\LENOVO\Desktop\New folder\expressjs>node appl.js
Server listening in port 3000
Req method is GET
Req url is /packages
Req method is GET
Req url is /
Req method is GET
Req url is /error
```

Types of Middlewares

- **Application-level middlewares**

Application-level middlewares are functions that are associated with the application object. These middleware function will get executed each time an application receives a request.

- **Router-level middlewares**

Router-level middlewares are functions that are associated with a route. These functions are linked to express.Router() class instance.

- **Router-level middleware**

Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of express.Router().

- **Error-handling middleware**

Error-handling middleware always takes four arguments. You must provide four arguments to identify it as an error-handling middleware function. Even if you don’t need to use the next object, you must specify it to maintain the signature. Otherwise, the next object will be interpreted as regular middleware and will fail to handle errors.

- **Built-in Middleware**:

Express has the following built-in middleware functions:

- express.static serves static assets such as HTML files, images, and so on.
- express.json parses incoming requests with JSON payloads.
- express.urlencoded parses incoming requests with URL-encoded payloads.

• Third-party middleware

Use third-party middleware to add functionality to Express apps.

Install the Node.js module for the required functionality, then load it in your app at the application level or at the router level.

Program:

```
const express = require('express')
const app = express()
const router = express.Router()
// Application-level middleware
app.use((req, res, next) => {
console.log('Time:', Date.now())
next()
})
```

```
// Router-level middleware
router.use((req, res, next) => {
console.log('Time:', Date.now())
next()
})
app.use('/', router)
// error handler
app.use((err, req, res, next) => {
console.error(err.stack)
res.status(500).send('Something broke!')
})
// Third-party middleware
const cookieParser = require('cookie-parser')
// load the cookie-parsing middleware
app.use(cookieParser())
```

Chaining middlewares

- Middlewares can be chained.
- We can use more than one middleware on an Express app instance, which means that we can use more than one middleware inside app.use() or app.METHOD().
- We use a comma (,) to separate them.

Program:

```
var express = require('express');
var app = express();

const PORT = 3000;

app.get("/", function(req, res, next){
  //middleware 1
  req.middlewares = ["middleware1"];
  next()
},
function(req, res, next){
  //middleware 2
  req.middlewares.push("middleware2")
  next()
},
function(req, res, next){
  //middleware 3
  req.middlewares.push("middleware3")
  res.json(req.middlewares);
})

app.listen(PORT, ()=>{
  console.log("app running on port "+PORT)
})
```

Connecting to MongoDB with Mongoose

Connecting to MongoDB with Mongoose – Introduction

Organizations use various databases to store data and to perform various operations on the

data based on the requirements.

Some of the most popular databases are:

•Cassandra

•MySQL

•MongoDB

•Oracle

•Redis

•SQL Server

A user can interact with a database in the following ways.

•Using query language of the respective databases' - eg: SQL

•Using an ODM(Object Data Model) or ORM(Object-Relational Model)

Introduction to Mongoose Schema

A schema defines document properties through an object, where the key name corresponds to the property name in the collection. The schema allows you to define the fields stored in each document along with their validation requirements and default values.

```
    const mongoose = require('mongoose’);
    const schema = new mongoose.Schema
({ property_1: Number, property_2: String });
```

In a schema, we will define the data types of the properties in the document. The most used types in the schema are:

•**String** : To store string values. For e.g: employee name

•**Number**: To store numerical values. For e.g: employee Id

•**Date** : To store dates. The date will be stored in the ISO date format. For e.g. 2018-11-15T15:22:00.

•**Boolean** It will take the values true or false and is generally used for performing validations. We will discuss validations in the next resource.

•**ObjectId** ObjectId is assigned by MongoDB itself. Every document inserted in MongoDB will have a unique id which is created automatically by MongoDB, and this is of type ObjectId.

•**Array** To store an array of data, or even a sub-document array.

For e.g: ["Cricket","Football"]

ADITYA
ENGINEERING COLLEGE (A)

Validation through mongoose validator

- Mongoose provides a way to validate data before you save that data to a database.
- Data validation is important to make sure that "invalid" data does not get persisted in your application. This ensures data integrity.
- A benefit of using Mongoose, when inserting data into MongoDB is its built-in support for data types, and the automatic validation of data when it is persisted.
- Mongoose’s validators are easy to configure. When defining the schema, specific validations need to be configured.

Models

- To use our schema definition, we need to wrap the Schema into a Model object we can work with.
- The model provides an object which provides access to query documents in a named collection.
- Schemas are compiled into models using the model() method.

Syntax:

```
const Model = mongoose.model(name ,schema)
```

CRUD OPERATIONS

Mongoose library offers several functions to perform various CRUD (Create-Read-Update-Delete) operations.

- The Create operation is used to insert new documents in the MongoDB database.
- The Read operation is used to query a document in the database.
- The Update operation is used to modify existing documents in the database.
- The Delete operation is used to remove documents in the database

```
var mongoose = require("mongoose");
var dbHost = "mongodb://localhost:27017/test";
mongoose.connect(dbHost);
var bookSchema = mongoose.Schema({
name: String,
//Also creating index on field isbn
isbn: {type: String, index: true},
author: String,
pages: Number
});
var Book = mongoose.model("Book", bookSchema,
"mongoose_demo");
var db = mongoose.connection;
db.on("error", console.error.bind(console,
"connection error:"));
db.once("open", function(){
console.log("Connected to DB");
```

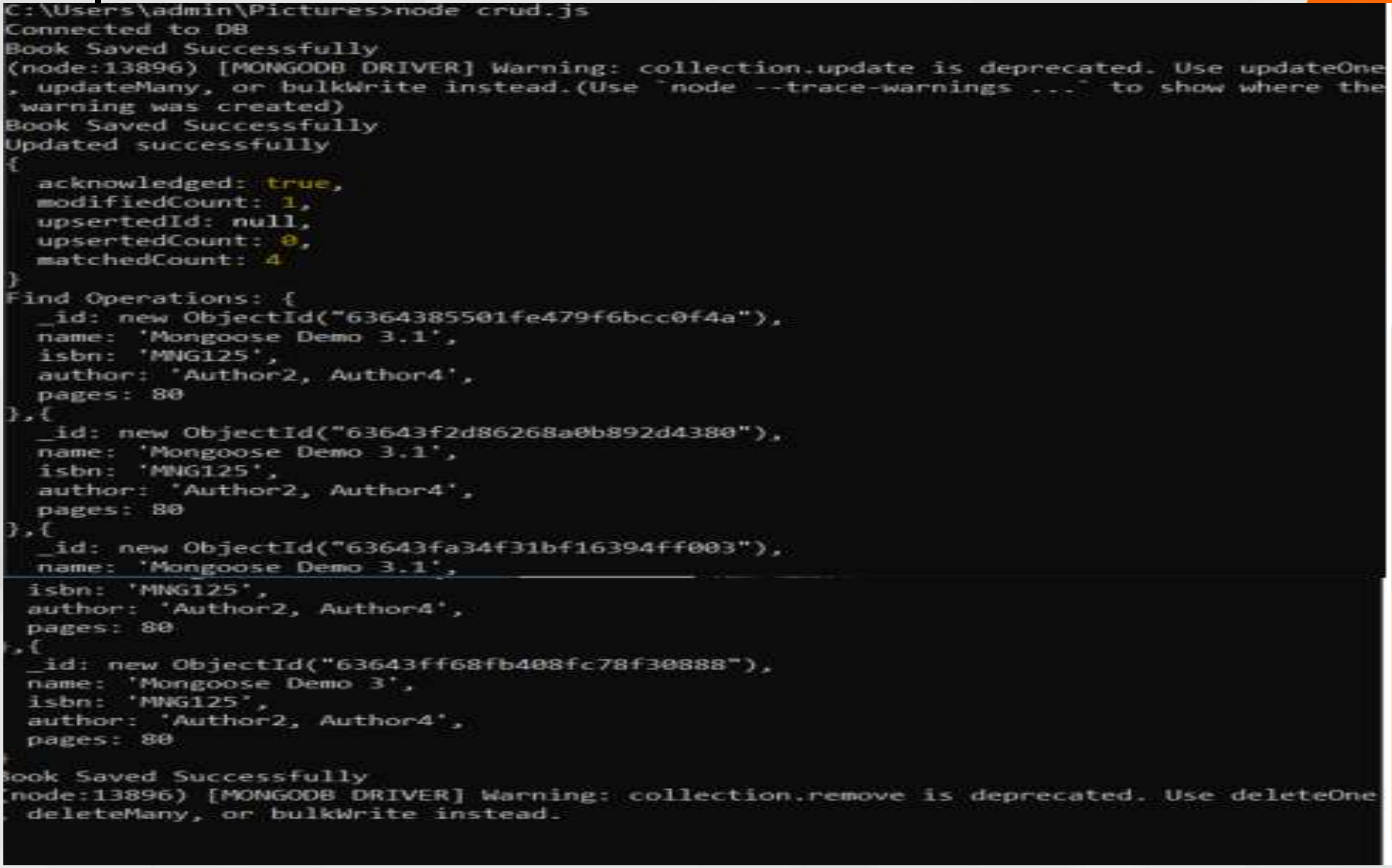
```
var book1 = new Book({
name:"Mongoose Demo 1",
isbn: "MNG123",
author: "Author1, Author2",
pages: 123
});
book1.save(function(err){
if ( err )
throw err;
console.log("Book Saved Successfully");
});
var book2 = new Book({
name:"Mongoose Demo 2",
isbn: "MNG124",
author: "Author2, Author3",
pages: 90
});
```

```
var book3 = new Book({
 name:"Mongoose Demo 3",
 isbn: "MNG125",
 author: "Author2, Author4",
 pages: 80
});
book3.save(function(err){
if ( err )
throw err;
console.log("Book Saved Successfully");
queryBooks();
updateBook();
});
});
```

```
var queryBooks = function(){
Book.find({pages : {$lt:100}}, "name isbn author pages",
function(err, result){
if ( err )
throw err;
console.log("Find Operations: " + result);
}); }
var updateBook = function(){
Book.update({isbn : {$eq: "MNG125"}}, {$set: {name:
"Mongoose Demo
3.1"}}, function(err, result){
console.log("Updated successfully");
console.log(result);
});}
var deleteBook = function(){
Book.remove({isbn:{$eq: "MNG124"}}).exec();
}
```

ADITYA ENGINEERING COLLEGE (A)

Output:



```
C:\Users\admin\Pictures>node crud.js
Connected to DB
Book Saved Successfully
(node:13896) [MONGODB DRIVER] Warning: collection.update is deprecated. Use updateOne
, updateMany, or bulkWrite instead.(Use `node --trace-warnings ...` to show where the
 warning was created)
Book Saved Successfully
Updated successfully
{
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 4
}
Find Operations: {
  _id: new ObjectId("6364385501fe479f6bcc0f4a"),
  name: 'Mongoose Demo 3.1',
  isbn: 'MNG125',
  author: 'Author2, Author4',
  pages: 80
},{
  _id: new ObjectId("63643f2d86268a0b892d4380"),
  name: 'Mongoose Demo 3.1',
  isbn: 'MNG125',
  author: 'Author2, Author4',
  pages: 80
},{
  _id: new ObjectId("63643fa34f31bf16394ff003"),
  name: 'Mongoose Demo 3.1',
  isbn: 'MNG125',
  author: 'Author2, Author4',
  pages: 80
},{
  _id: new ObjectId("63643ff68fb408fc78f30888"),
  name: 'Mongoose Demo 3',
  isbn: 'MNG125',
  author: 'Author2, Author4',
  pages: 80
}
Book Saved Successfully
(node:13896) [MONGODB DRIVER] Warning: collection.remove is deprecated. Use deleteOne
, deleteMany, or bulkWrite instead.
```

ADITYA ENGINEERING COLLEGE (A)

API Development - https

An application should be protected by HTTPS, even if it's not handling sensitive information for

communications.

•The integrity of the application will be protected by HTTPS.

•The privacy and security of the data will be protected by HTTPS.

Generate the SSH keys in the server where the application will be running.

ADITYA ENGINEERING COLLEGE (A)

Session Management,Cookies

Session Management is a technique used by the webserver to store a particular user's session information. The Express framework provides a consistent interface to work with the session-related data. The framework provides two ways of implementing sessions:

- By using cookies
- By using the session store

Cookies are a piece of information sent from a website server and stored in the user's web browser when the user browses that website. Every time the user loads that website back, the browser sends that stored data back to the website server, to recognize the user.

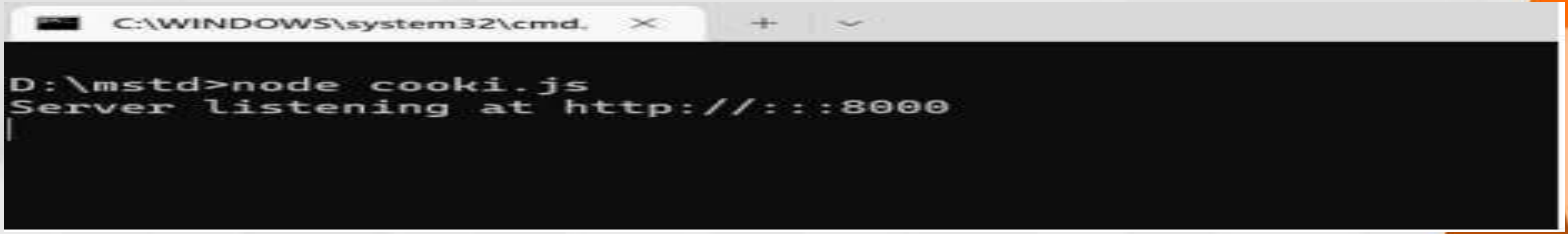
Program:

```
var express = require('express');
var cookieParser = require('cookie-parser'); var
app = express();
app.use(cookieParser());
app.get('/cookieset',function(req, res){
res.cookie('cookie_name', 'cookie_value');
res.cookie('College', 'Aditya');
res.cookie('Branch', 'Cse');
res.status(200).send('Cookie is set');
});
app.get('/cookieget', function(req, res) {
res.status(200).send(req.cookies);
});
```

```
app.get('/', function (req, res)
{
res.status(200).send('Welcome to Aditya');
});
var server = app.listen(8000, function ()
 {
 var host = server.address().address;
 var port = server.address().port;
console.log('Server listening at http://%s:%s',
host, port);
});
```

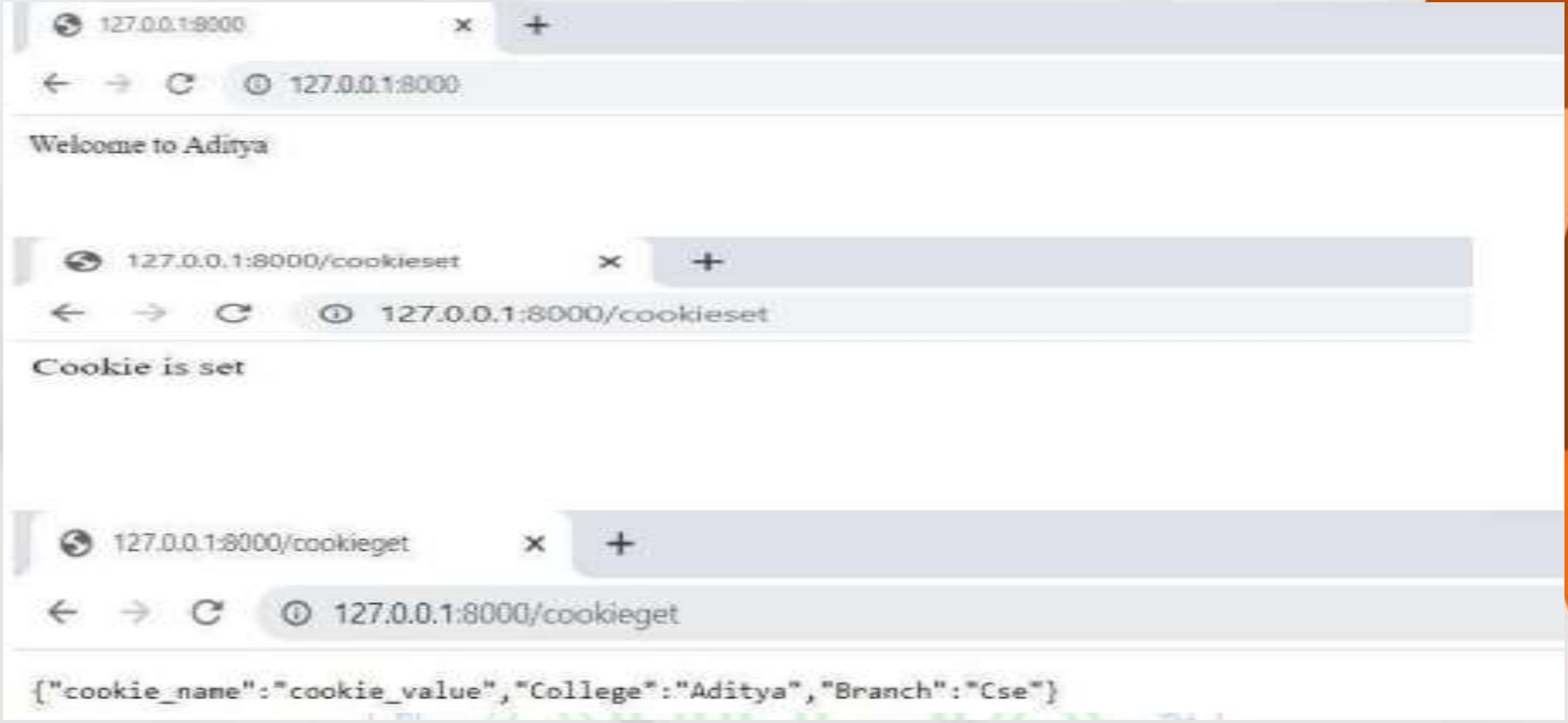
ADITYA ENGINEERING COLLEGE (A)

Output:



C:\WINDOWS\system32\cmd.

```
D:\mstd>node cooki.js
Server listening at http://:::8000
```



127.0.0.1:8000

Welcome to Aditya

127.0.0.1:8000/cookieset

Cookie is set

127.0.0.1:8000/cookieget

```
{"cookie_name":"cookie_value","College":"Aditya","Branch":"Cse"}
```

ADITYA ENGINEERING COLLEGE (A)

Why and What Security

- Attackers may attempt to exploit security vulnerabilities present in web applications in various ways.
- They may plan attacks like clickjacking, cross-site scripting, placing insecure requests and identifying web application frameworks etc.
- It is important to follow secure coding techniques for developing secure, robust and hack-resilient applications.

ADITYA ENGINEERING COLLEGE (A)

Helmet Middleware

- Without Helmet, default headers returned by Express expose sensitive information and make your Node.js app vulnerable to malicious actors.
- In contrast, using Helmet in Node.js protects your application from XSS attacks, Content Security Policy vulnerabilities, and other security issues.

Program:

App.js

```
const express = require('express’);
const helmet = require('helmet’);
const routing= require('./route');
const app = express(); app.use(helmet());
app.use('/', routing); app.listen(3000);
console.log('Server listening in port 3000');
```

route.js

```
const express = require('express’);
const router = express.Router();
router.get('/', function (req, res) {
res.send('<h1>Express</h1>');
});
router.get('/about', function (req, res) {
res.send('About Us Page');
});
module.exports = router;
```

test.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<style> p { color: red;
} iframe { width: 100%; height: 90%
}
</style>
</head>
<body>
<p>Clickjacked</p>
<iframe src="http://localhost:3000"></iframe>
</body>
</html>
```

Output:

```
Refused to frame '[illegible]' because an ancestor violates the following Content Security Policy directive: "frame-ancestors 'self'".
[illegible]
DevTools failed to load source map: Could not load content for [illegible]: System error: net::ERR_BLOCKED_BY_CLIENT
DevTools failed to load source map: Could not load content for [illegible]: System error: net::ERR_BLOCKED_BY_CLIENT
```

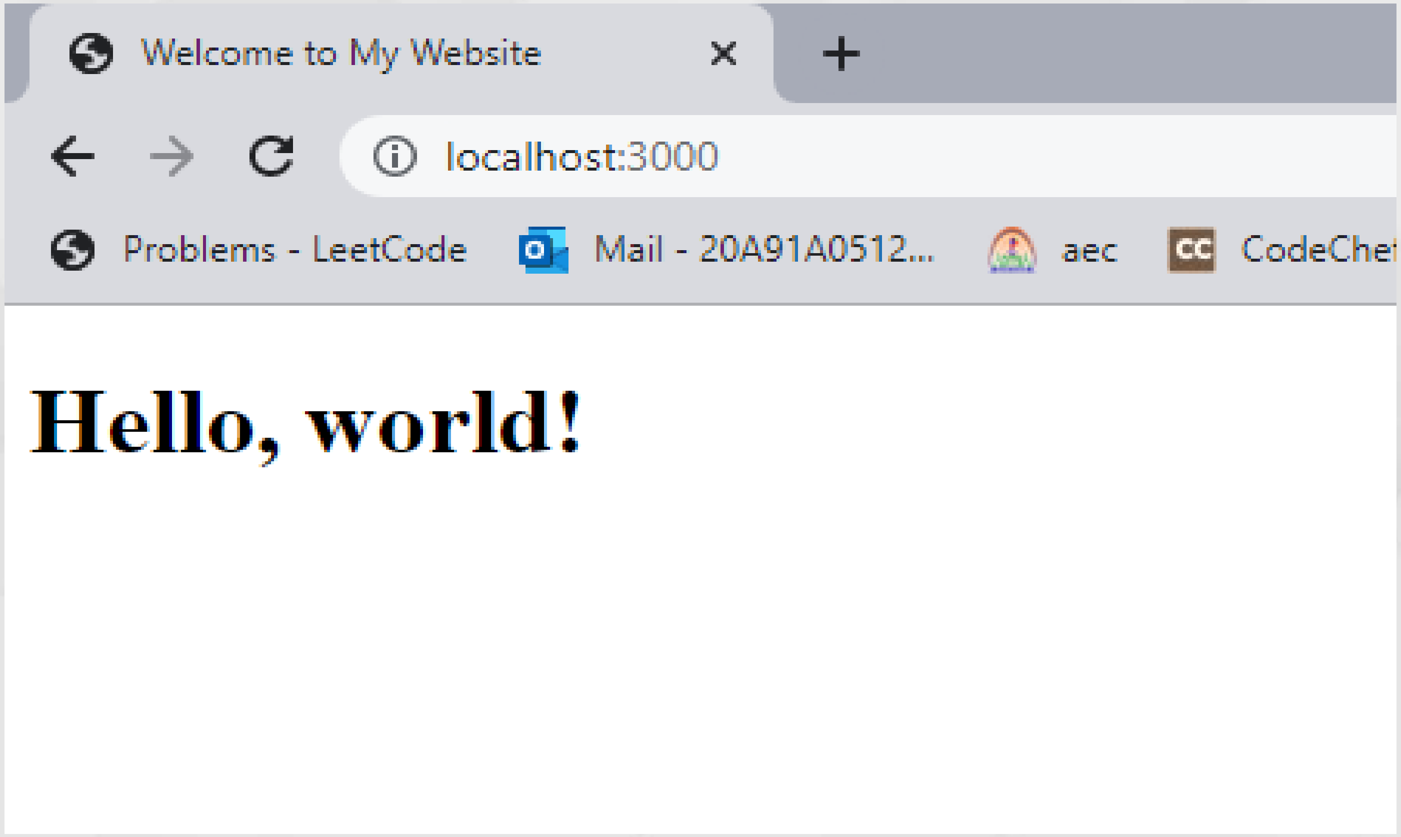
Template Engine Middleware

- A template engine enables you to use static template files in your application.
- At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.
- This approach makes it easier to design an HTML page, as it reduces duplicate code (DRY).
- Some popular template engines that work with Express are Pug, Mustache, and EJS.

Program:

```
const express = require('express');
const exphbs = require('express-handlebars');
const app = express();
const hbs = exphbs.create();
app.engine('handlebars', hbs.engine);
app.set('view engine', 'handlebars');
app.get('/', (req, res) => {
  res.render('home', {
    title: 'Welcome to My Website',
    message: 'Hello, world!' });});
app.listen(3000, () => {
  console.log('Server is running on
http://localhost:3000/');
});
```

Output:



Welcome to My Website

localhost:3000

Problems - LeetCode Mail - 20A91A0512... aec CodeChef

```
Hello, world!
```

Stylus CSS Preprocessor

- Stylus is a dynamic CSS preprocessor that allows you to write CSS in a more concise and expressive syntax.
- It was created by TJ Holowaychuk and is known for its minimalist approach and powerful features.
- Stylus compiles down to standard CSS and can be used in any web project.

To install stylus globally:

```
npm install -g stylus
```

Program: styles. styl

```
// Define variables
primary-color = #FF0000
border-radius = 5px
// Define mixin
rounded-box()
border-radius border-radius
// Define nested selector
.container
  background-color primary-color
  padding 10px
  // Extend mixin
  rounded-box()
// Compile to CSS
```

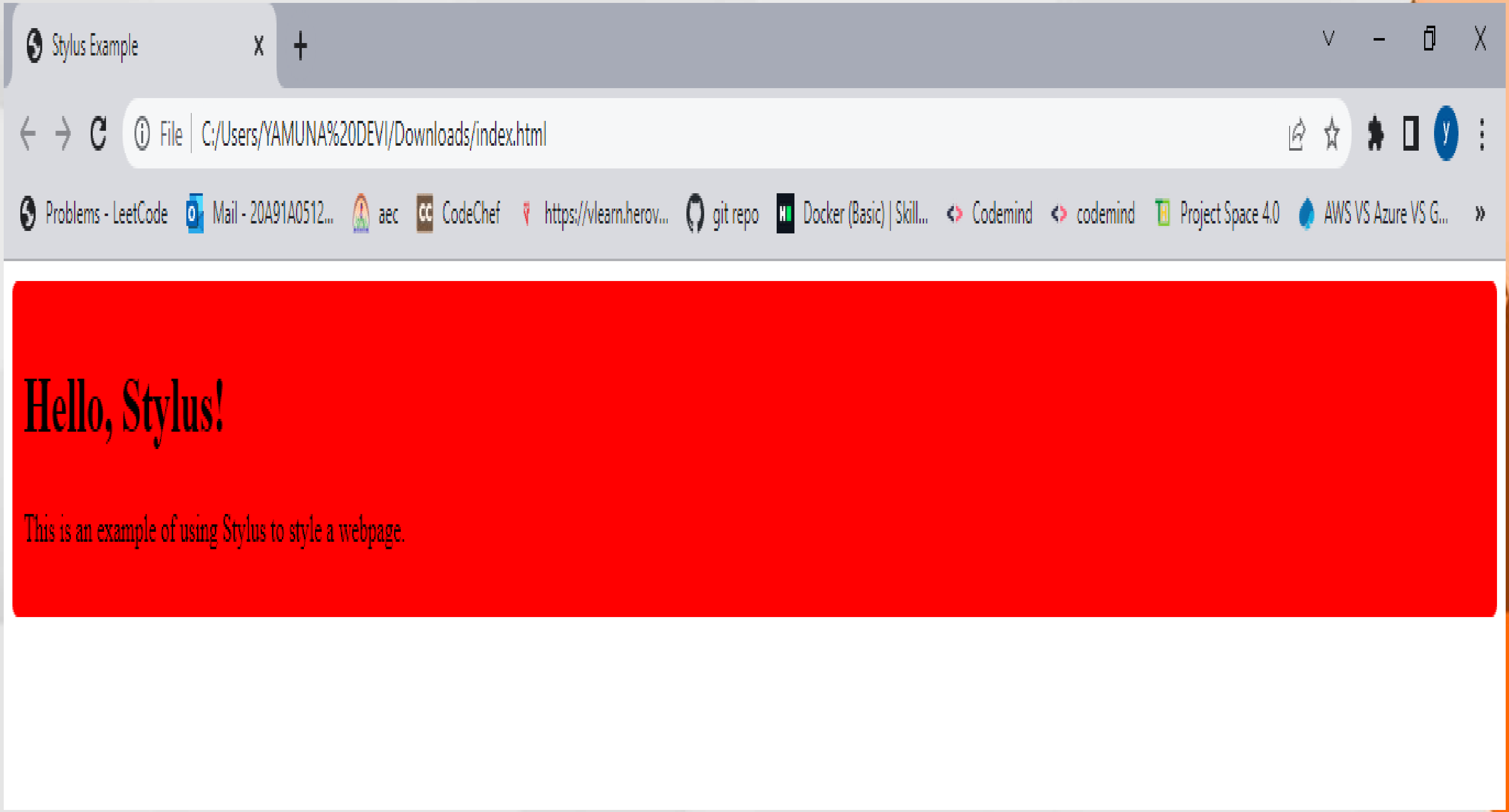
Execute by:stylus style.styl in terminal

Index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Stylus Example</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Hello, Stylus!</h1>
    <p>This is an example of using Stylus to style a
webpage.</p>
  </div>
</body>
</html>
```

ADITYA ENGINEERING COLLEGE (A)

Output :


Stylus Example

C:/Users/YAMUNA%20DEVI/Downloads/index.html

Problems - LeetCode   Mail - 20A91A0512...   aec   CodeChef   https://vlearn.herov...   git repo   Docker (Basic) | Skill...   Codemind   codemind   Project Space 4.0   AWS VS Azure VS G...

Hello, Stylus!

This is an example of using Stylus to style a webpage.