SHREE H.N.SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University and G.T.U)

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Lt. Shree Chimanbhai Shukla

MScIT SEM-3 :: CS-14: Node.js

Shree H.N.Shukla College
Street No. 2, Vaishali Nagar,
Nr. Amrapali Under Bridge,
Raiya Road, Rajkot.
Ph. (0281)2440478, 2472590

Shree H.N.Shukla College
Street No. 3, Vaishali Nagar,
Nr. Amrapali Under Bridge,
Raiya Road, Rajkot.
Ph. (0281)2471645

Shree H.N.Shukla College of I.T & Management "Sky is the Limit"

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

CS – 14: NODE JS

MSc(IT) SEMESTER : 03 :: CS-14: Node.js		
No	**Topics**	**Details**
5.	**Express.js and Node.js**	• Introduction to Express Framework • Express Server Request-Response Routes • Route Parameters • Multiple Route Callback/Handler Functions • Methods of Response Object • Chaining Route Handlers • Send Static Files • Accept User Input • File Upload with Express • Manage Cookies • Send file as a response • Templates and Express

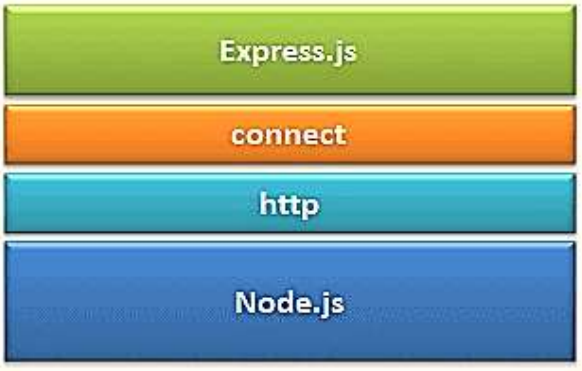
2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Introduction to Expressjs Framework

What is Expressjs and why is it used in Node.js applications?

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
- Express.js is a web application framework for Node.js.
- It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.
- It is used to build web applications and APIs with ease.
- Express.js is based on the Node.js middleware module called connect which in turn uses http module.
- So, any middleware which is based on connect will also work with Express.js.



Advantages of Express.js

- Makes Node.js web application development fast and easy.
- Easy to configure and customize.
- Allows you to define routes of your application based on HTTP methods and URLs.
- Includes various middleware modules which you can use to perform additional tasks on request and response.
- Easy to integrate with different template engines like Jade, Vash, EJS etc.
- Allows you to define an error handling middleware.
- Easy to serve static files and resources of your application.

- Allows you to create REST API server.
- Easy to connect with databases such as MongoDB, Redis, MySQL

Key features:

- **Simplified Routing:** Helps in managing different HTTP requests at different URLs.
- **Middleware Support:** Allows the inclusion of various middleware to handle requests.
- **Template Engines:** Facilitates dynamic content rendering.
- **Efficient Handling:** Manages both synchronous and asynchronous operations.

Example:

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

app.get('/', (req, res) => {
  res.send('Hello, World!');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Output:

```
Server is running on port 3000
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Install Express.js

You can install express.js using npm.
The following command will install latest version of express.js globally on your machine so that every Node.js application on your machine can use it.

npm install -g express

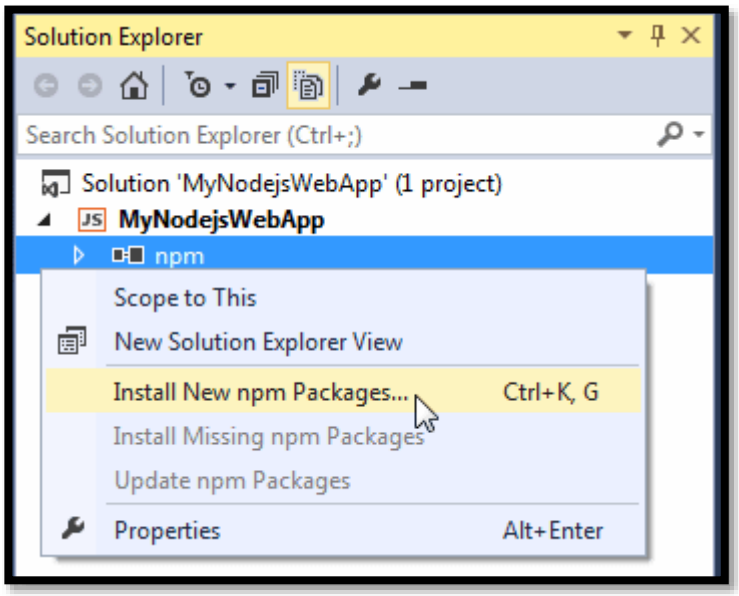
The following command will install latest version of express.js local to your project folder.

C:\MyNodeJSApp> npm install express –save

--save will update the package.json file by specifying express.js dependency.

Install Express.js in Visual Studio

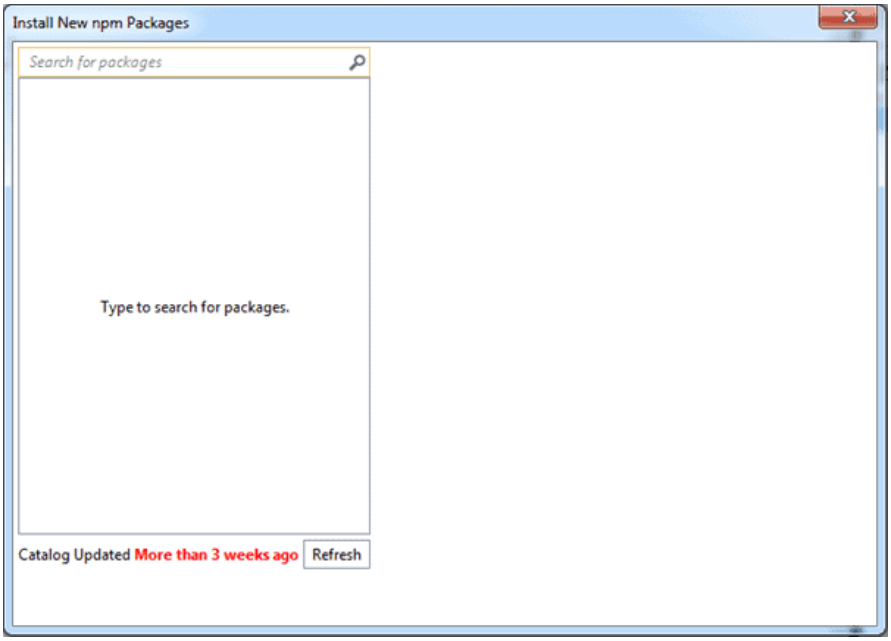
In the node.js web app, we created a simple node.js web application in Visual Studio. Now, to install Express.js, right click on the project MyNodejsWebApp -> select Install New npm Packages.



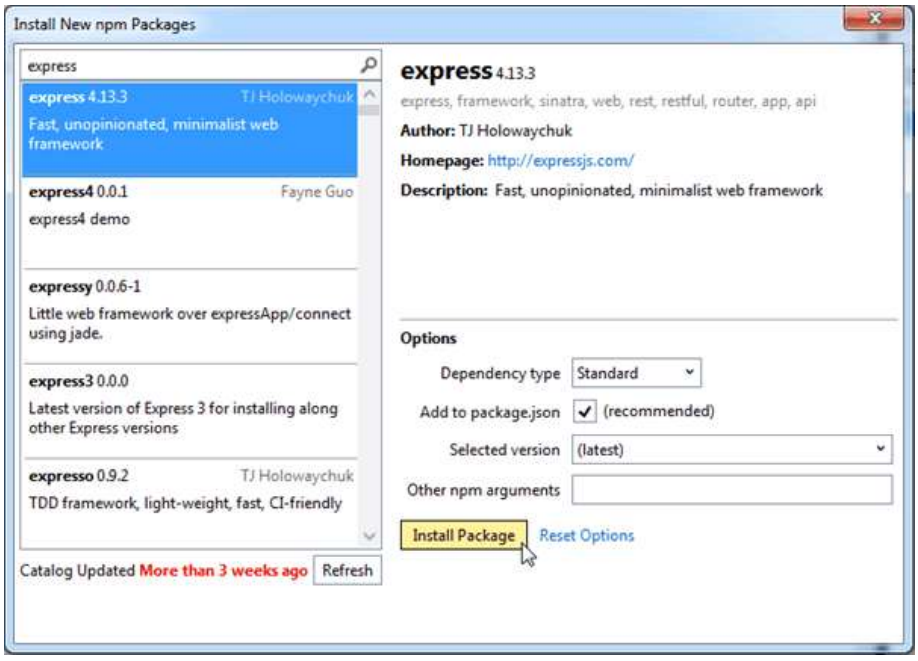
2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

It will open the following dialog box.



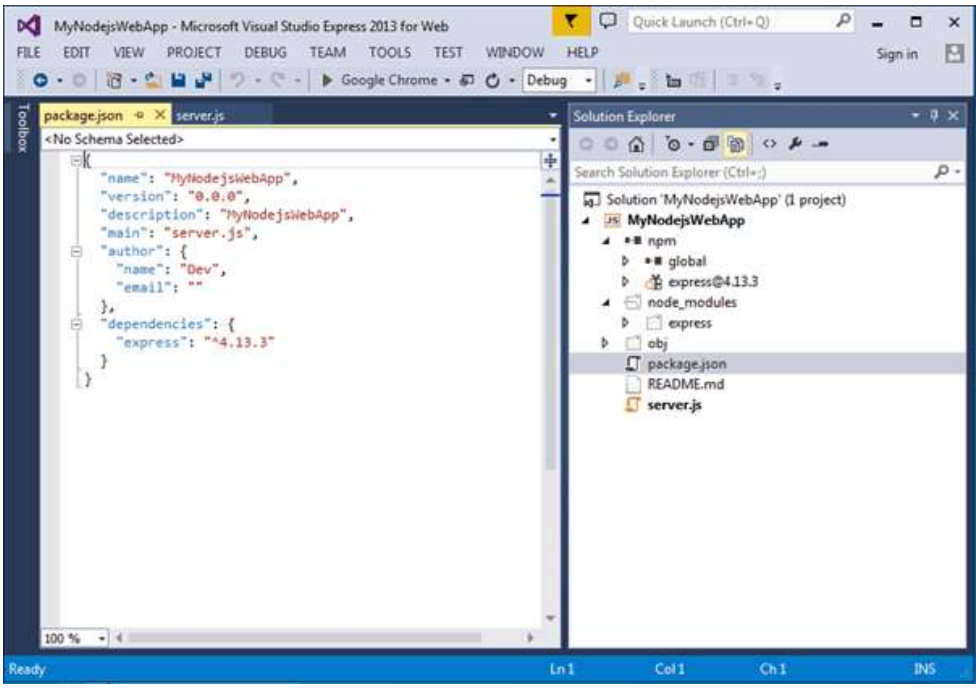
Now, type 'express' in the search box. It will display all the packages starting with express. Select latest version of express.js and select Add to package.json checkbox and then click Install Package button as shown below.



2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

So, this will install express.js package into your project under node_modules folder. It also adds a dependencies entry for express.js as shown below.



Now, you are ready to use Express.js in your application.

Express.js Web Application

Express.js provides an easy way to create web server and render HTML pages for different HTTP requests by configuring routes for your application.

Web Server

First of all, import the Express.js module and create the web server as shown below.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

File name: app.js

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

// define routes here..

var server = app.listen(4500, function () {
    console.log('Node server is running..');
});
```

In this example, Express.js module is imported using require() function.
The express module returns a function.
This function returns an object which can be used to configure Express application.

The app object includes methods for routing HTTP requests, configuring middleware, rendering HTML views and registering a template engine.

The **app.listen()** function creates the Node.js web server at the specified host and port. It is identical to Node's **http.Server.listen()** method.

Run the above example using node app.js command and point your browser to **http://localhost:4500**. It will display **Cannot GET /** because we have not configured any routes yet.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Create Node.js Web Server

Node.js makes it easy to create a simple web server that processes incoming requests asynchronously.

The following example is a simple Node.js web server contained in server.js file.

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..')
```

In this example, the http module is imported using require() function.

The http module is a core module of Node.js, so no need to install it using NPM.

The next step is to call createServer() method of http and specify callback function with request and response parameter.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Finally, call **listen()** method of server object which was returned from ***createServer()*** method with port number, to start listening to incoming requests on port 5000. You can specify any unused port here.

Run the above web server by writing **node server.js** command in command prompt or terminal window and it will display message as shown below.

```
C:\> node server.js

Node.js web server at port 5000 is running..
```

Handle HTTP Request

- The 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.
- The following example demonstrates handling HTTP request and response in Node.js.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

File Name : 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();
    }
    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..')
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

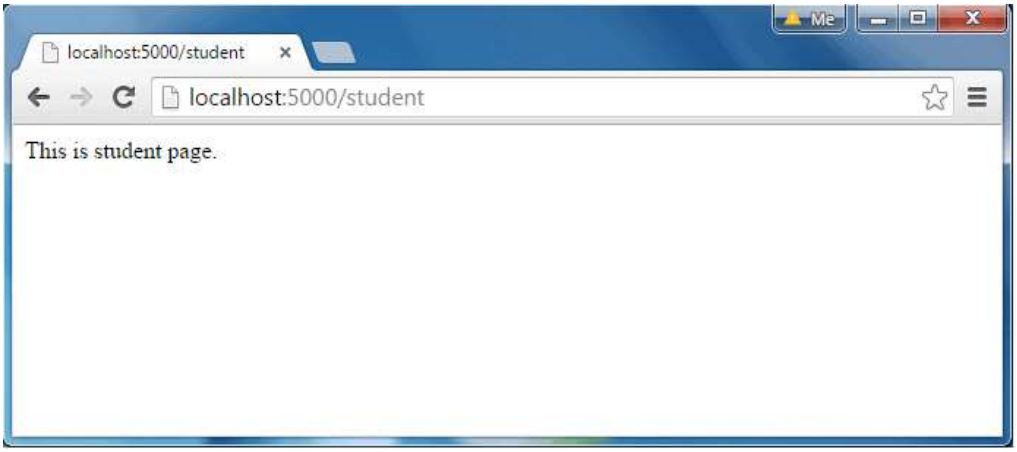
3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Run the above web server as shown below.

```
C:\> node server.js

Node.js web server at port 5000 is running..
```

For Windows users, point your browser to **http://localhost:5000** and see the following result.

The same way, point your browser to ***http://localhost:5000/student*** and see the following result.



2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Configure Routes

Use app object to define different routes of your application.

The app object includes get(), post(), put() and delete() methods to define routes for **HTTP GET**, **POST**, **PUT** and **DELETE** requests respectively.

The following example demonstrates configuring routes for HTTP requests.

Example: Configure Routes in Express.js

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

app.get('/', function (req, res) {
   res.send('<html><body><h1>Hello World</h1></body></html>');
});

app.post('/submit-data', function (req, res) {
   res.send('POST Request');
});

app.put('/update-data', function (req, res) {
   res.send('PUT Request');
});

app.delete('/delete-data', function (req, res) {
   res.send('DELETE Request');
});

var server = app.listen(5000, function () {
   console.log('Node server is running..');
});
```

In this example, app.get(), app.post(), app.put() and app.delete() methods define routes for **HTTP** ***GET***, ***POST***, ***PUT***, ***DELETE*** respectively.

The first parameter is a path of a route which will start after base URL.

The callback function includes request and response object which will be executed on each request.

Run the above example using node server.js command, and point your browser to ***http://localhost:5000*** and you will see the following result.

Handle POST Request

Learn how to handle HTTP POST request and get data from the submitted form.

First, create Index.html file in the root folder of your application and write the following HTML code in it.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Example: Configure Routes in Express.js

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <form action="/submit-student-data" method="post">
        First Name: <input name="firstName" type="text" />
<br />
        Last Name: <input name="lastName" type="text" /> <br
/>
        <input type="submit" />
    </form>
</body>
</html>
```

Body Parser

- To handle **HTTP POST** request in Express.js version 4 and above, you need to install middleware module called **body-parser**.
- The middleware was a part of Express.js earlier but now you have to install it separately.
- This body-parser module parses the JSON, buffer, string and url encoded data submitted using **HTTP POST** request. Install **body-parser** using **NPM** as shown below.

$ npm install body-parser --save

Now, import body-parser and get the POST request data as shown below.

Example: app.js: Handle POST Route in Express.js

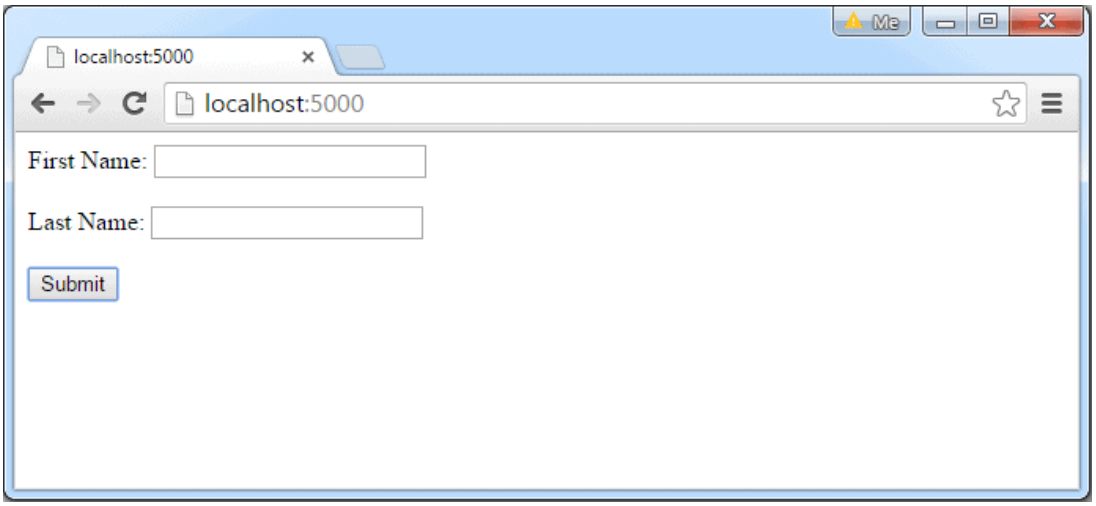
```
var express = require('express');
var app = express();

var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));

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

app.post('/submit-student-data', function (req, res) {
    var name = req.body.firstName + ' ' +
req.body.lastName;

    res.send(name + ' Submitted Successfully!');
});

var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

- In this example, **POST** data can be accessed using req.body.
- The req.body is an object that includes properties for each submitted form.
- **Index.html** contains **firstName** and **lastName** input types, so you can access it using req.body.firstName and req.body.lastName.

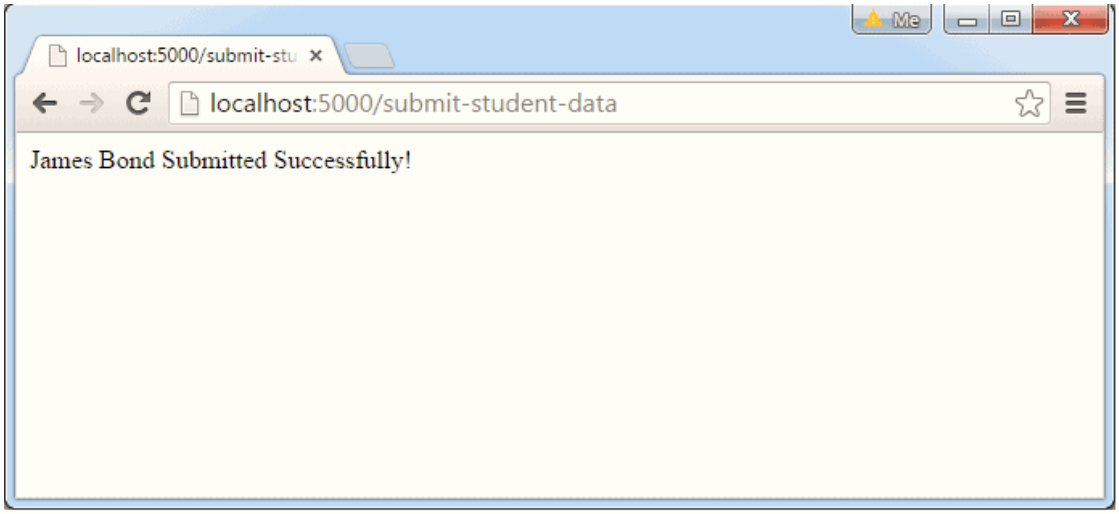
Now, run the above example using node server.js command, point your browser to **http://localhost:5000** and see the following result.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645



- Fill the First Name and Last Name in the above example and click on submit.
- For example, enter "James" in First Name textbox and "Bond" in Last Name textbox and click the submit button. The following result is displayed.



2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Express Server Request-Response Routes

How does Express handle request and response routes?

Express handles routes using the `app.METHOD(PATH, HANDLER)` syntax, where `METHOD` is an HTTP method (GET, POST, etc.), `PATH` is the endpoint, and `HANDLER` is a function executed when the route is matched.

Example:

```
app.get('/about', (req, res) => {
  res.send('About Page');
});

app.post('/submit', (req, res) => {
  res.send('Form Submitted');
});
```

Output:

- GET `/about` responds with `About Page`.
- POST `/submit` responds with `Form Submitted`.

Route Parameters

What are route parameters in Express and how are they used?

Route parameters are named URL segments used to capture values specified at their position in the URL. They are defined using :paramName in the route path.

```
app.get('/user/:id', (req, res) => {
  res.send(`User ID: ${req.params.id}`);
});
```

Output:

GET /user/123 responds with User ID: 123.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Multiple Route Callback/Handler Functions

How can you use multiple callback functions for a single route in Express?

The use of multiple route callback functions in Express.js allows you to chain multiple middleware functions or route handlers together for a specific route.

This provides several advantages:

1) **Modular and Reusable Code:**
   By breaking down route handling into multiple callback functions, you can create more modular and reusable code.
2) **Separation of Concerns:**
   Using multiple callbacks allows you to separate concerns in your application. For example, you can have one callback for authentication, another for data validation, and a final callback for processing the request and sending the response. This separation makes your code more organized and easier to understand.
3) **Middleware-like Behavior:**
   The multiple callbacks behave similarly to middleware in Express.js. They are executed in the order they are defined and can perform tasks such as parsing request bodies, adding response headers, or modifying the request/response objects.
4) **Bypassing Remaining Callbacks:**
   If a middleware function determines that the request should not proceed to the next callback, it can call next('route') to skip the remaining callbacks for that route and move on to the next matching route.
5) **Flexibility and Extensibility:**
   Adding or modifying functionality becomes easier when using multiple callbacks. You can insert new callbacks at specific points in the chain without affecting the existing ones, making your application more flexible and extensible.

Few examples of how multiple route callbacks can be used in Express.js:

1) Authentication and Authorization:

```
app.get('/protected', isAuthenticated, isAuthorized, (req, res) => {
  // Protected route logic
});
```

2) Data Validation and Sanitization:

```
app.post('/users', validateData, sanitizeData, (req, res) => {
  // Create user logic
});
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

3) Logging and Error Handling:

```
app.use((req, res, next) => {
 console.log(`${req.method} ${req.url}`);
        next();
}, (err, req, res, next) => {
        console.error(err);
        res.status(500).send('Internal Server Error');
});
```

4) Express allows chaining multiple callback functions for a single route, which are executed sequentially.

Example:

```
app.get('/example', (req, res, next) => {
  console.log('First callback');
  next();
}, (req, res) => {
  res.send('Second callback');
});
```

Output:

GET /example logs First callback and responds with Second callback.

EXAMPLE:

```
student-data-app/
├── app.js
├── package.json
└── routes/
    └── students.js
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
// app.js

const express = require('express');
const studentsRouter = require('./routes/students');

const app = express();
const port = 3000;

app.use(express.json());
app.use('/students', studentsRouter);

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

```
// package.json

{
  "name": "student-data-app",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

// **student.js**

```
const express = require('express');
const router = express.Router();

const students = [
  { id: 1, name: 'John Doe', grade: 85 },
  { id: 2, name: 'Jane Smith', grade: 92 },
  { id: 3, name: 'Michael Johnson', grade: 78 },
];

// Get all students
router.get('/', (req, res) => {
  res.json(students);
});

// Get a specific student by ID
router.get('/:id', (req, res) => {
  const student = students.find(s => s.id === parseInt(req.params.id));
  if (!student) {
    return res.status(404).json({ message: 'Student not found' });
  }
  res.json(student);
});

// Create a new student
router.post('/', (req, res) => {
  const { name, grade } = req.body;
  const newStudent = { id: students.length + 1, name, grade };
  students.push(newStudent);
  res.status(201).json(newStudent);
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
});

// Update a student
router.put('/:id', (req, res) => {
  const student = students.find(s => s.id === parseInt(req.params.id));
  if (!student) {
    return res.status(404).json({ message: 'Student not found' });
  }
  student.name = req.body.name || student.name;
  student.grade = req.body.grade || student.grade;
  res.json(student);
});

// Delete a student
router.delete('/:id', (req, res) => {
  const index = students.findIndex(s => s.id === parseInt(req.params.id));
  if (index === -1) {
    return res.status(404).json({ message: 'Student not found' });
  }
  const deletedStudent = students.splice(index, 1)[0];
  res.json(deletedStudent);
});

// Get students with a specific grade
router.get('/grade/:grade', (req, res) => {
  const grade = parseInt(req.params.grade);
  const filteredStudents = students.filter(s => s.grade === grade);
  res.json(filteredStudents);
});

// Get students sorted by name
router.get('/sort/name', (req, res) => {
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
  const sortedStudents = students.sort((a, b) =>
a.name.localeCompare(b.name));
  res.json(sortedStudents);
});

// Get students sorted by grade
router.get('/sort/grade', (req, res) => {
  const sortedStudents = students.sort((a, b) => a.grade - b.grade);
  res.json(sortedStudents);
});

module.exports = router;
```

The above student.js is converted to chaining route in the below example:

```
// student.js --- for chaining route example

const express = require('express');
const router = express.Router();

const students = [
  { id: 1, name: 'John Doe', grade: 85 },
  { id: 2, name: 'Jane Smith', grade: 92 },
  { id: 3, name: 'Michael Johnson', grade: 78 },
];

// Route for handling all student-related operations
router.route('/')
  .get((req, res) => {
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
    res.json(students);
  })
  .post((req, res) => {
    const { name, grade } = req.body;
    const newStudent = { id: students.length + 1, name, grade };
    students.push(newStudent);
    res.status(201).json(newStudent);
  });

// Route for handling specific student operations
router.route('/:id')
  .get((req, res) => {
    const student = students.find(s => s.id === parseInt(req.params.id));
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }
    res.json(student);
  })
  .put((req, res) => {
    const student = students.find(s => s.id === parseInt(req.params.id));
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }
    student.name = req.body.name || student.name;
    student.grade = req.body.grade || student.grade;
    res.json(student);
  })
  .delete((req, res) => {
    const index = students.findIndex(s => s.id === parseInt(req.params.id));
    if (index === -1) {
      return res.status(404).json({ message: 'Student not found' });
    }
    const deletedStudent = students.splice(index, 1)[0];
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
    res.json(deletedStudent);
  });

// Route for filtering students by grade
router.get('/grade/:grade', (req, res) => {
  const grade = parseInt(req.params.grade);
  const filteredStudents = students.filter(s => s.grade === grade);
  res.json(filteredStudents);
});

// Route for sorting students by name
router.get('/sort/name', (req, res) => {
  const sortedStudents = [...students].sort((a, b) =>
a.name.localeCompare(b.name));
  res.json(sortedStudents);
});

// Route for sorting students by grade
router.get('/sort/grade', (req, res) => {
  const sortedStudents = [...students].sort((a, b) => a.grade - b.grade);
  res.json(sortedStudents);
});

module.exports = router;
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Explanation of Changes

Chaining Routes for Student Operations:

The routes for GET, POST, PUT, and DELETE operations on students are now chained using router.route('/') and router.route('/:id').

This allows for a cleaner and more organized structure, making it clear which operations are related to the root path (/) and specific student IDs (/:id).

GET and POST for All Students:

The GET request on the root path (/) retrieves all students.

The POST request on the same path allows for creating a new student.

GET, PUT, and DELETE for Specific Students:

The GET, PUT, and DELETE requests for specific students are chained under the /:id route, allowing for easy management of individual student records.

Filtering and Sorting:

The routes for filtering students by grade and sorting students by name or grade remain unchanged but are organized clearly under their respective GET requests.

How to Test the API

You can test the API using tools like Postman or cURL. Here are some example requests:

Get all students: GET http://localhost:3000/students

Get a specific student: GET http://localhost:3000/students/1

Create a new student:

POST http://localhost:3000/students

Body: { "name": "Alice Brown", "grade": 90 }

Update a student:

PUT http://localhost:3000/students/1

Body: { "name": "John Doe", "grade": 88 }

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Delete a student: DELETE http://localhost:3000/students/1

Get students with a specific grade: GET http://localhost:3000/students/grade/85

Get students sorted by name: GET http://localhost:3000/students/sort/name

Get students sorted by grade: GET http://localhost:3000/students/sort/grade

This structure enhances the maintainability of your code and makes it easier to add or modify routes in the future.

Methods of Response Object

What are some commonly used methods of the response object in Express?

Common methods:

- `res.send()`: Sends a response of various types.
- `res.json()`: Sends a JSON response.
- `res.status()`: Sets the HTTP status for the response.
- `res.sendFile()`: Sends a file as a response.

```
app.get('/json', (req, res) => {
  res.status(200).json({ message: 'Hello, JSON' });
});
```

Output:

GET /json responds with {"message":"Hello, JSON"} and status 200.

Chaining Route Handlers

How do you chain route handlers in Express?

Chaining route handlers involves using the `.route()` method to handle multiple HTTP methods for a single route.

Example:

```
app.route('/book')
  .get((req, res) => {
    res.send('Get a book');
  })
  .post((req, res) => {
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Output:

- GET /book responds with Get a book.
- POST /book responds with Add a book.
- PUT /book responds with Update the book.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Send Static Files

How do you serve static files using Express?

Use the `express.static` middleware to serve static files like HTML, CSS, JavaScript, and images.

Example:

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

Output:

Files in the public directory are served at the root URL.

For example, public/style.css is accessible at /style.css.

Accept User Input

How does Express handle user input from forms or JSON payloads?

Express uses middleware like `body-parser` to handle user input.

Example

```
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.post('/user', (req, res) => {
  res.send(`User name: ${req.body.name}`);
});
```

Output:

POST /user with {"name": "John"} responds with User name: John.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

File Upload with Express

How can you handle file uploads in Express?

Use middleware like `multer` for handling file uploads.

Example

```
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });

app.post('/upload', upload.single('file'), (req, res) => {
  res.send(`File uploaded: ${req.file.originalname}`);
});
```

Output:

POST /upload with a file responds with File uploaded: filename.ext.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Manage Cookies

How do you manage cookies in Express?

Use the `cookie-parser` middleware to handle cookies.

Example:

```
const cookieParser = require('cookie-parser');
app.use(cookieParser());

app.get('/setcookie', (req, res) => {
  res.cookie('name', 'value');
  res.send('Cookie set');
});

app.get('/getcookie', (req, res) => {
  res.send(`Cookie value: ${req.cookies.name}`);
});
```

Output:

- GET `/setcookie` sets a cookie.
- GET `/getcookie` responds with `Cookie value: value`.

HTTP Cookies in Node.js

- Cookies are small data that are stored on a client side and sent to the client along with server requests.
- Cookies have various functionality, they can be used for maintaining sessions and adding user-specific features in your web app.
- For this, we will use cookie-parser module of npm which provides middleware for parsing of cookies.
- First set your directory of the command prompt to root folder of the project and run the following command:

  npm init

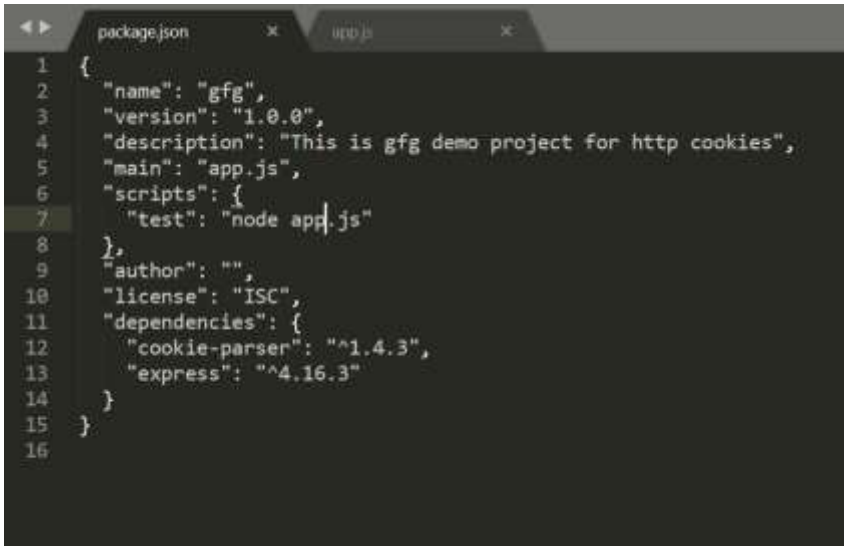
- This will ask you details about your app and finally will create a package.json file.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

- After that run the following command and it will install the required module and add them in your package.json file

npm install express cookie-parser --save



After that setup basic express app by writing following code in our **app.js** file in root directory.

```
let express = require('express');
//setup express app
let app = express()


//basic route for homepage
app.get('/', (req, res)=>{
        res.send('welcome to express app');
});

//server listens to port 3000
app.listen(3000, (err)=>{
        if(err)
                throw err;
                console.log('listening on port 3000');
});
```

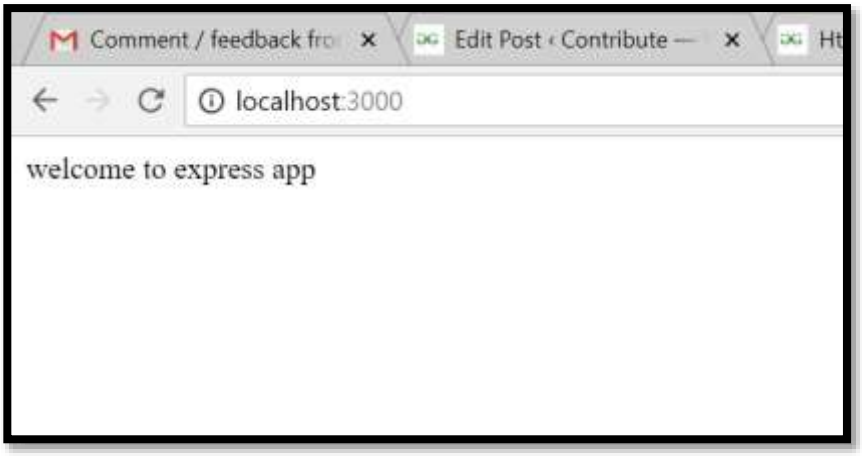
After that if we run the command

node app.js

It will start our server on port 3000 and if go to the url: localhost:3000, we will get a page showing the message :

welcome to express app

Here is screenshot of localhost:3000 page after starting the server :



So until now we have successfully set up our express app now let's start with cookies.

For cookies first, we need to import the module in our app.js file and use it like other middlewares.

```
var cookieParser = require('cookie-parser');
app.use(cookieParser());
```

we have to add that cookie to the response using the following code :

```
res.cookie(name_of_cookie, value_of_cookie);
```

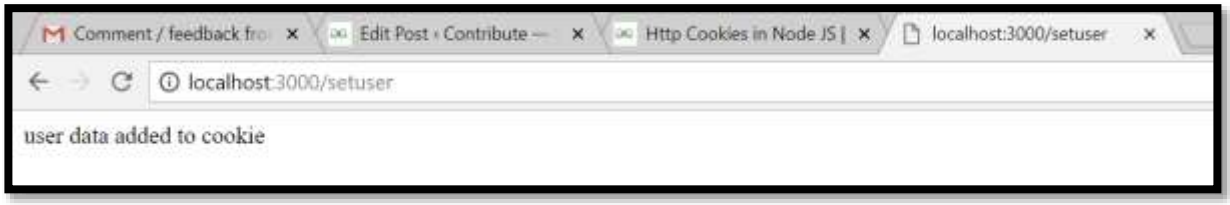
2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
let express = require('express');
let cookieParser = require('cookie-parser');
//setup express app
let app = express()

app.use(cookieParser());

//basic route for homepage
app.get('/', (req, res)=>{
        res.send('welcome to express app');
});

//JSON object to be added to cookie
let users = {
        name : "Ritik",
        Age : "18"
}

//Route for adding cookie
app.get('/setuser', (req, res)=>{
        res.cookie("userData", users);
        res.send('user data added to cookie');
});

//Iterate users data from cookie
app.get('/getuser', (req, res)=>{
        //shows all the cookies
        res.send(req.cookies);
});

//server listens to port 3000
app.listen(3000, (err)=>{
        if(err)
                throw err;
                console.log('listening on port 3000');
        });
```

If restart the server and make a get request to the route: **localhost:3000/getuser** before setting the cookies it is as follows :

After making a request to **localhost:3000/setuser** it will add user data to cookie and gives output as follows :



Now if we again make a request to **localhost:3000/getuser** as this route is iterating user data from cookies using req.cookies so output will be as follows :

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Adding Cookie with expiration Time

- We can add a cookie with some expiration time i.e. after that time cookies will be destroyed automatically.
- For this, we need to pass an extra property to the res.cookie object while setting the cookies.
  It can be done by using any of the two ways:

```
//Expires after 400000 ms from the time it is set.
res.cookie(cookie_name, 'value', {expire: 400000 + Date.now()});


//It also expires after 400000 ms from the time it is set.
res.cookie(cookie_name, 'value', {maxAge: 360000});
```

Destroy the cookies :

We can destroy cookies using following code :

```
res.clearCookie(cookieName);
```

Now let us make a logout route which will destroy user data from the cookie.
Now our app.js looks like:

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

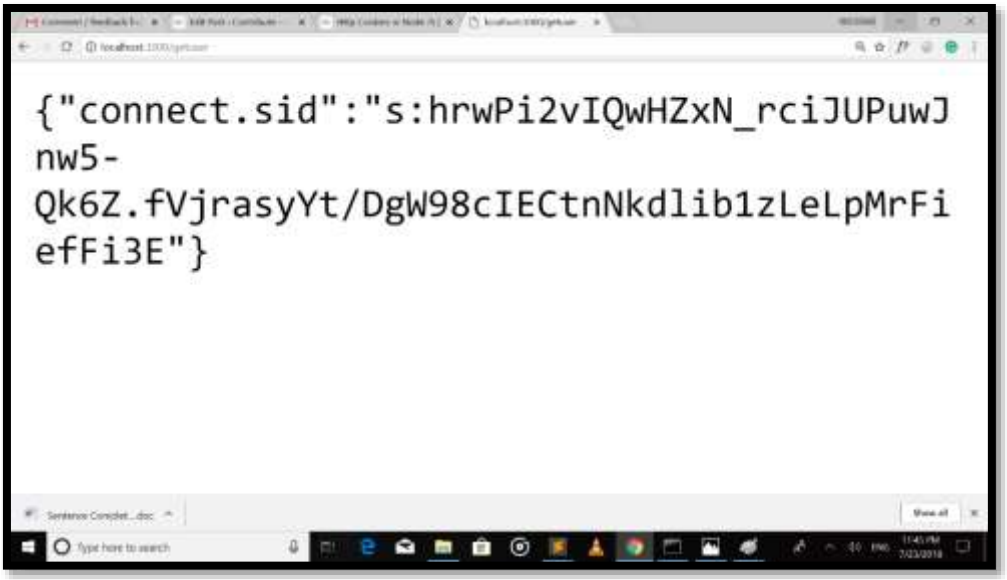
```
let express = require('express');
let cookieParser = require('cookie-
parser');
//setup express app
let app = express()
app.use(cookieParser());

//basic route for homepage
app.get('/', (req, res)=>{
        res.send('welcome to express
app');
});

//JSON object to be added to cookie
let users = {
        name : "Ritik",
        Age : "18"
}

//Route for adding cookie
app.get('/setuser', (req, res)=>{
res.cookie("userData", users);
res.send('user data added to cookie');
});

//Iterate users data from cookie
app.get('/getuser', (req, res)=>{
        //shows all the cookies
        res.send(req.cookies);
});
```

```
//Iterate users data from cookie
app.get('/getuser', (req, res)=>{
        //shows all the cookies
        res.send(req.cookies);
});

//Route for destroying cookie
app.get('/logout', (req, res)=>{
//it will clear the userData cookie
        res.clearCookie('userData');
        res.send('user logout
successfully');
});

//server listens to port 3000
app.listen(3000, (err)=>{
        if(err)
        throw err;
        console.log('listening on port
        3000');
});
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

For destroying the cookie make get request to following link: user logged out[/caption]
To check whether cookies are destroyed or not make a get request to **localhost:3000/getuserand** you will get an empty user cookie object.



2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Send File as a Response

How do you send a file as a response in Express?
Use the `res.sendFile` method to send a file.

Example:

```
app.get('/download', (req, res) => {
  res.sendFile(__dirname + '/files/sample.pdf');
});
```

Output:

GET `/download` prompts the user to download `sample.pdf`.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Template Engines for Node.js

Template engine helps us to create an HTML template with minimal code.

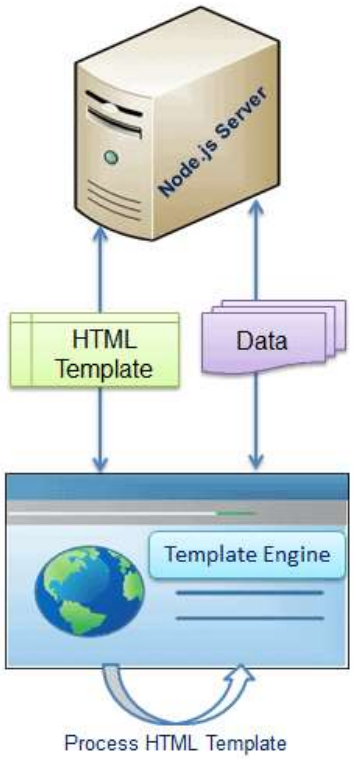
Also, it can inject data into HTML template at client side and produce the final HTML.

Template engines in Node.js allow developers to create dynamic HTML pages by embedding JavaScript logic within HTML templates.

This facilitates the rendering of views with data, making it easier to manage and produce HTML content.

EJS (Embedded JavaScript) is one of the popular template engines used in Node.js applications, particularly with the Express framework.

The following figure illustrates how template engine works in Node.js.



2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

As per the above figure, client-side browser loads HTML template, JSON/XML data and template engine library from the server.

Template engine produces the final HTML using template and data in client's browser.

However, some HTML templates process data and generate final HTML page at server side also.

Advantages of Template engine in Node.js

1. Improves developer's productivity.
2. Improves readability and maintainability.
3. Faster performance.
4. Maximizes client side processing.
5. Single template for multiple pages.
6. Templates can be accessed from CDN (Content Delivery Network).

There are many template engines available for Node.js. Each template engine uses a different language to define HTML template and inject data into it.

The following is a list of important (but not limited) template engines for Node.js

- Jade
- Vash
- EJS
- Mustache
- Dust.js
- Nunjucks
- Handlebars
- atpl
- haml

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Overview of EJS

EJS allows you to generate HTML markup with plain JavaScript.
It enables the embedding of JavaScript code directly into HTML, making it straightforward to inject dynamic content into your web pages.
The main advantages of using EJS include:

1. **Simplicity**: EJS syntax is easy to learn and use.
2. **Flexibility**: It allows for the inclusion of JavaScript logic, making it suitable for various applications.
3. **Performance**: EJS compiles templates into functions, which can be rendered quickly.

Setting Up EJS in a Node.js Application

To use EJS in a Node.js application, follow these steps:

1. Install EJS: First, you need to install EJS via npm:

   ```
   npm install ejs --save
   ```

2. **Set Up Express**: Create an Express application and set EJS as the templating engine:

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

// Set EJS as the templating engine
app.set('view engine', 'ejs');
app.set('views', './views'); // Directory for template files
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

3. **Create a Template:** In the views directory, create a file named ***student.ejs***. This file will serve as the template for displaying student data:

```
<!DOCTYPE html>
<html>
<head>
  <title>Student Details</title>
</head>
<body>
  <h1>Student Details</h1>
  <ul>
    <% students.forEach(function(student) { %>
      <li>Name: <%= student.name %>,
          Age: <%= student.age %>,
          Grade: <%= student.grade %>
      </li>
    <% }); %>
  </ul>
</body>
</html>
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

4. **Render the Template:** Set up a route to render the ***student.ejs*** template with some sample student data:

```
app.get('/students', (req, res) => {
   const students = [
      { name: 'Alice', age: 20, grade: 'A' },
      { name: 'Bob', age: 22, grade: 'B' },
      { name: 'Charlie', age: 21, grade: 'C' }
   ];
   res.render('student', { students: students });
});

const server = app.listen(4000, () => {
   console.log('Server is running on http://localhost:4000');
});
```

Explanation of the Example

In this example, the application sets up a route /students that renders a list of students. The students array is passed to the **student.ejs** template, where EJS syntax is used to loop through the array and display each student's details.

<% %> is used for control flow (like loops).

<%= %> is used to output values into the HTML.

When you navigate to **http://localhost:4000/students**, the server will respond with an HTML page displaying the list of students, dynamically generated using the EJS template.

Using EJS in Node.js applications simplifies the process of creating dynamic web pages, making it a valuable tool for developers.