Express

✳Express.js

- *Express.js is a small framework that works on top of Node.js web server functionality to simplify its APIs and add helpful new features.*
- *It makes it easier to organize your application's functionality with middleware and routing.*
- *It adds helpful utilities to Node.js HTTP objects and facilitates the rendering of dynamic HTTP objects.*

Features of Express framework:

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

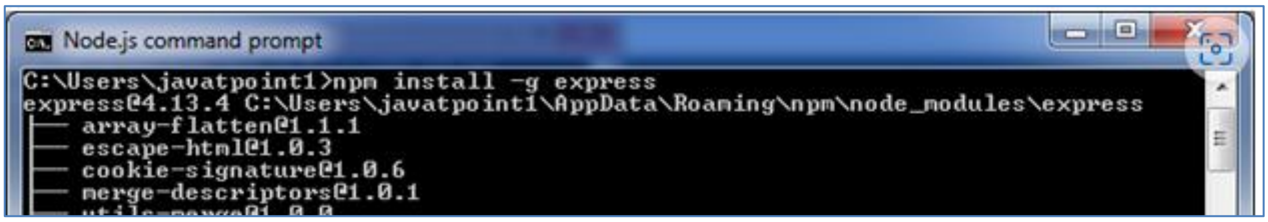
✳Why Express?

- *Develops Node.js web applications quickly and easily.*
- *It's simple to set up and personalise.*
- *Allows you to define application routes using HTTP methods and URLs.*
- *Includes a number of middleware modules that can be used to execute additional requests and responses activities.*
- *Simple to interface with a variety of template engines, including Jade, Vash, and EJS.*
- *Allows you to specify a middleware for handling errors.*

✳Install Express.js:

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

npm install -g express

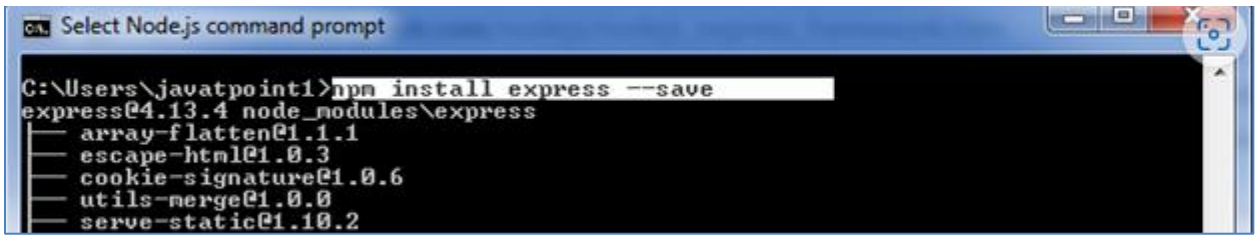


```
C:\Users\javatpoint1>npm install -g express
express@4.13.4 C:\Users\javatpoint1\AppData\Roaming\npm\node_modules\express
├── array-flatten@1.1.1
├── escape-html@1.0.3
├── cookie-signature@1.0.6
├── merge-descriptors@1.0.1
```

✳Installing Express

Use the following command to install express:

npm install express -save



```
C:\Users\javatpoint1>npm install express --save
express@4.13.4 node_modules\express
├── array-flatten@1.1.1
├── escape-html@1.0.3
├── cookie-signature@1.0.6
├── utils-merge@1.0.0
├── serve-static@1.10.2
```

The above command install express in node_module directory and create a directory named express inside the node_module. You should install some other important modules along with express. Following is the list:

- *body-parser: This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.*
- *cookie-parser: It is used to parse Cookie header and populate req.cookies with an object keyed by the cookie names.*
- *multer: This is a node.js middleware for handling multipart/form-data.*

Ex hello

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
res.send("Welocme to hello!");
});
app.listen(5000);
```

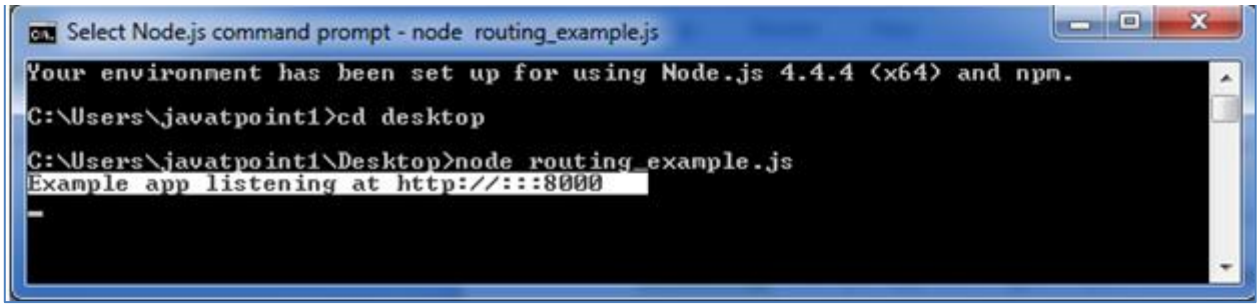
Express.js Routing:

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

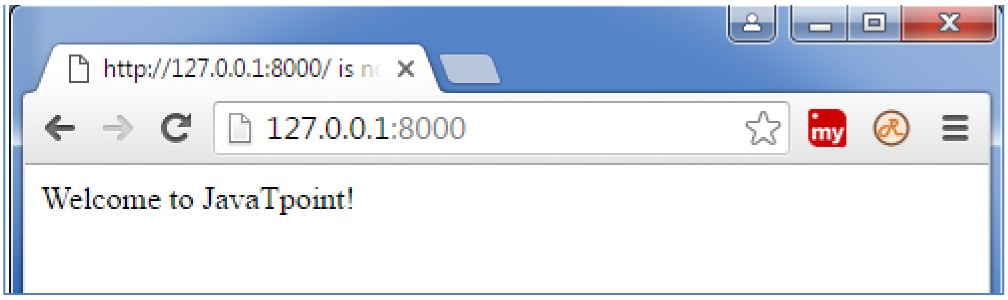
Let's take an example to see basic routing.

File: routing_example.js

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
   console.log("Got a GET request for the homepage");
   res.send('Welcome to JavaTpoint!');
})
app.post('/', function (req, res) {
   console.log("Got a POST request for the homepage");
   res.send('I am Impossible! ');
})
app.delete('/del_student', function (req, res) {
   console.log("Got a DELETE request for /del_student");
   res.send('I am Deleted!');
})
app.get('/enrolled_student', function (req, res) {
   console.log("Got a GET request for /enrolled_student");
   res.send('I am an enrolled student.');
})
// This responds a GET request for abcd, abxcd, ab123cd, and so on
app.get('/ab*cd', function(req, res) {
   console.log("Got a GET request for /ab*cd");
   res.send('Pattern Matched.');
})
var server = app.listen(8000, function () {
var host = server.address().address
  var port = server.address().port
console.log("Example app listening at http://%s:%s", host, port)
})
```



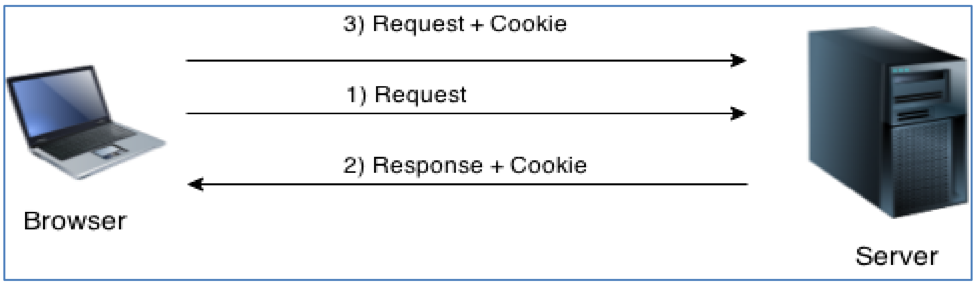
```
Your environment has been set up for using Node.js 4.4.4 (x64) and npm.

C:\Users\javatpoint1>cd desktop

C:\Users\javatpoint1\Desktop>node routing_example.js
Example app listening at http://:::8000
_
```



Welcome to JavaTpoint!

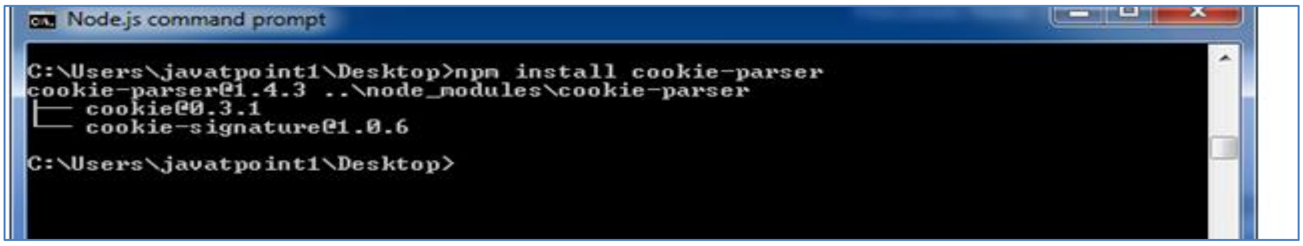
✳Express.js Cookies

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



Install cookie

You have to acquire cookie abilities in Express.js. So, install cookie-parser middleware through npm by using the following command:



```
C:\Users\javatpoint1\Desktop>npm install cookie-parser
cookie-parser@1.4.3 ..\node_modules\cookie-parser
├── cookie@0.3.1
└── cookie-signature@1.0.6

C:\Users\javatpoint1\Desktop>
```

Import cookie-parser into your app.

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

Define a route:

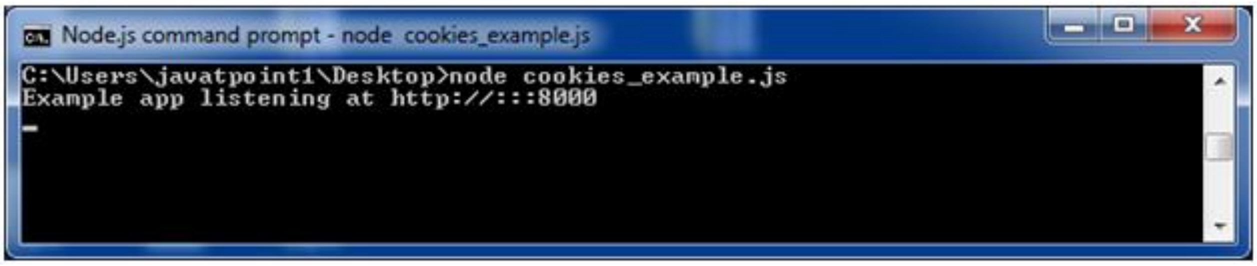
Cookie-parser parses Cookie header and populate req.cookies with an object keyed by the cookie names.

```
app.get('/cookie',function(req, res){
    res.cookie('cookie_name' , 'cookie_value').send('Cookie is set');
});
app.get('/', function(req, res) {
  console.log("Cookies :  ", req.cookies);
});
```

Express.js Cookies Example

File: cookies_example.js

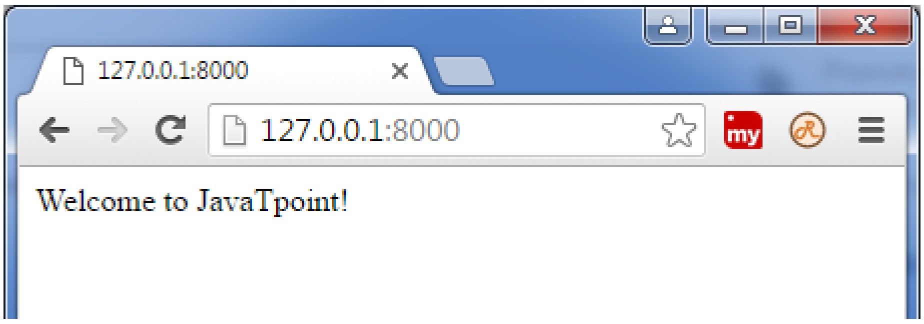
```
var express = require('express');
var cookieParser = require('cookie-parser');
var app = express();
app.use(cookieParser());
app.get('/cookieset',function(req, res){
res.cookie('cookie_name', 'cookie_value');
res.cookie('company', 'javatpoint');
res.cookie('name', 'sonoo');

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 JavaTpoint!');
});
var server = app.listen(8000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('Example app listening at http://%s:%s', host, port);
});
```



```
C:\Users\javatpoint1\Desktop>node cookies_example.js
Example app listening at http://:::8000
```

Output:

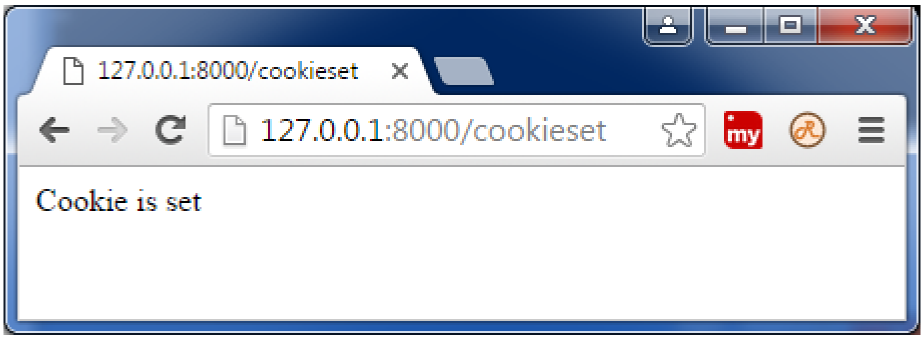
Open the page **http://127.0.0.1:8000/** on your browser:



Welcome to JavaTpoint!

Set cookie:

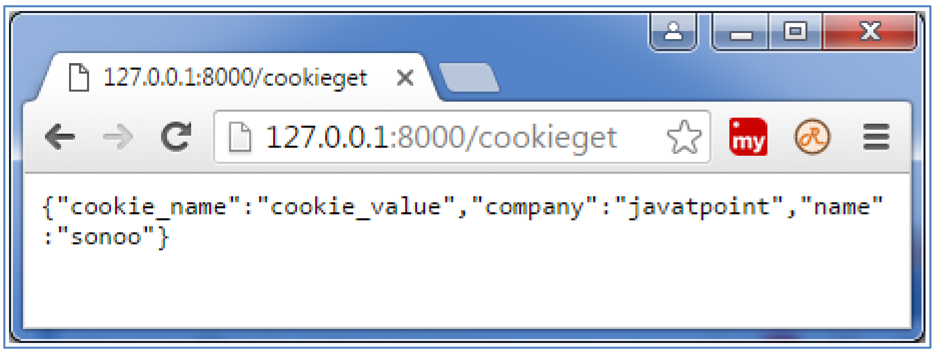
Now open **http://127.0.0.1:8000/cookieset** to set the cookie:



Cookie is set

Get cookie:

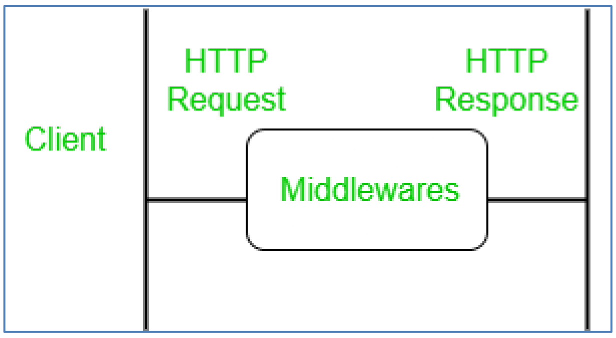
Now open http://127.0.0.1:8000/cookieget to get the cookie:



```
{"cookie_name":"cookie_value","company":"javatpoint","name"
:"sonoo"}
```

☸**Express.js Middleware:**

- *Express.js Middleware are different types of functions that are invoked by the Express.js routing layer before the final request handler. As the name specified, Middleware appears in the middle between an initial request and final intended route.*
- *Middleware has the access to the request object, responses object, and next, it can process the request before the server send a response.*
- *An Express-based application is a series of middleware function calls.*
- *In stack, middleware functions are always invoked in the order in which they are added.*
- *Middleware is commonly used to perform tasks like body parsing for URL-encoded or JSON requests, cookie parsing for basic cookie handling, or even building JavaScript modules on the fly.*



Middleware Syntax: The basic syntax for the middleware functions are as follows –

app.get(path, (req, res, next) => {}, (req, res) => {})

What is a Middleware function

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

Task middleware

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

Express.js Middleware

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

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

Let us create our middleware and see that how it executes.

Step 1: Go to your project directory and enter the following command to create a NodeJs project. Make sure that NodeJs is installed in your machine.

```
npm init -y
```

It will create a package.json file.

Step 2: Install two dependencies using the following command.

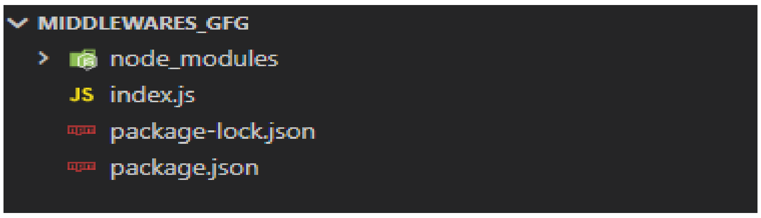
```
npm install express nodemon
```

Step 3: In the scripts section of the package.json file, add the following code line.

```
"start": "nodemon index.js",
```

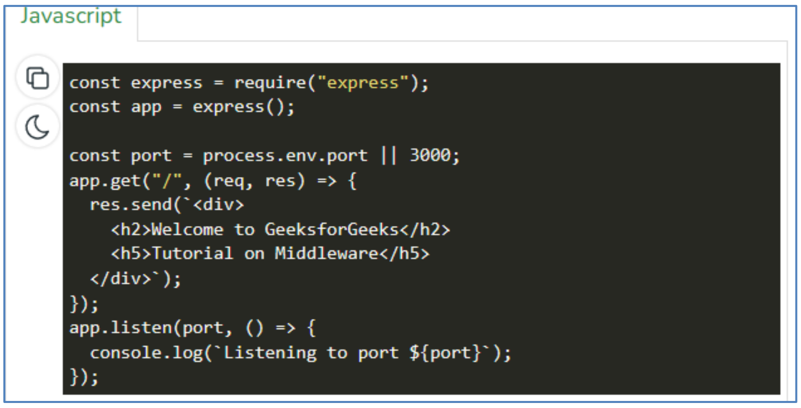
Step 4: Create an index.js file in the directory. Make sure that it is not inside any subdirectories of the directory you are working in.

Project Structure: It will look like the following.



```
MIDDLEWARES_GFG
  > node_modules
    index.js
    package-lock.json
    package.json
```

Now we will set up our express app and send a response to our server.

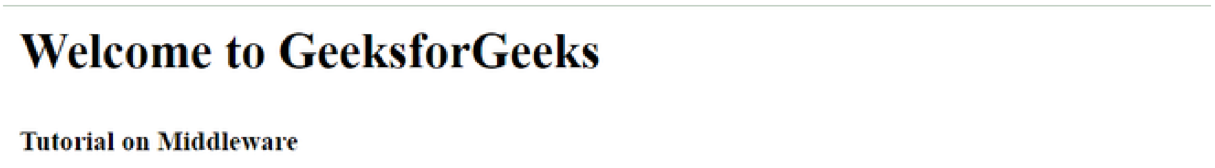


Javascript

```javascript
const express = require("express");
const app = express();

const port = process.env.port || 3000;
app.get("/", (req, res) => {
  res.send(`<div>
    <h2>Welcome to GeeksforGeeks</h2>
    <h5>Tutorial on Middleware</h5>
  </div>`);
});
app.listen(port, () => {
  console.log(`Listening to port ${port}`);
});
```

Step to run the application: Run the code by entering the following command on the terminal.

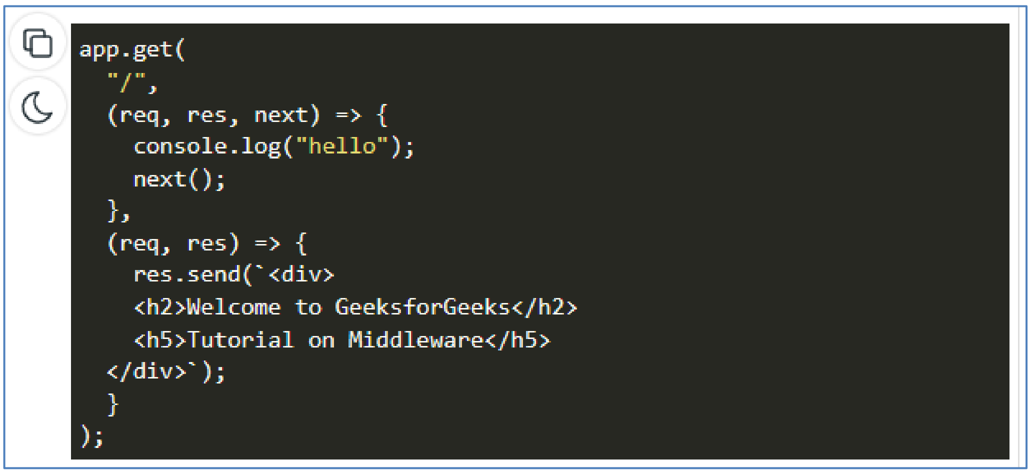
```
npm start
```



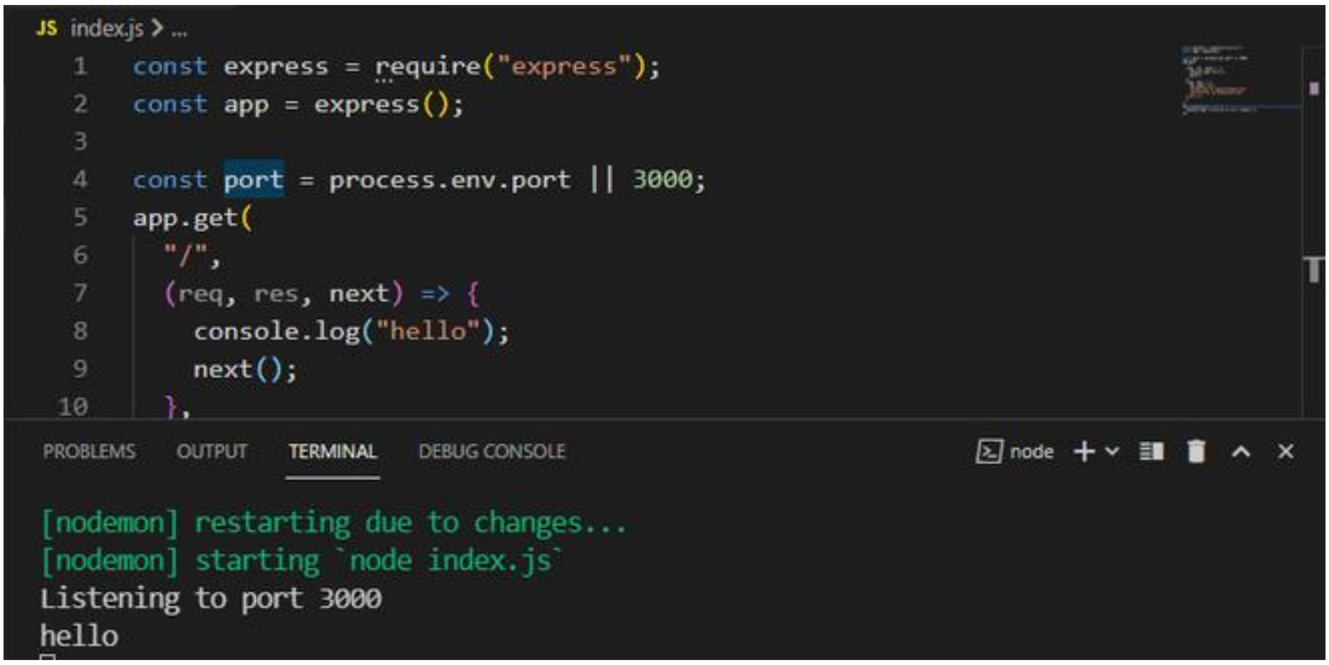
Welcome to GeeksforGeeks

Tutorial on Middleware

Create a Middleware: In the app.get() function, modify according to the following code.



```
app.get(
  "/",
  (req, res, next) => {
    console.log("hello");
    next();
  },
  (req, res) => {
    res.send(`<div>
    <h2>Welcome to GeeksforGeeks</h2>
    <h5>Tutorial on Middleware</h5>
  </div>`);
  }
);
```



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

const port = process.env.port || 3000;
app.get(
  "/",
  (req, res, next) => {
    console.log("hello");
    next();
  },
```

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Listening to port 3000
hello
```

<u>Advantages of using middleware:</u>

- *Middleware can process request objects multiple times before the server works for that request.*
- *Middleware can be used to add logging and authentication functionality.*
- *Middleware improves client-side rendering performance.*
- *Middleware is used for setting some specific HTTP headers.*
- *Middleware helps for Optimization and better performance*

✡<u>ExpressJS – Sessions:</u>

Cookies and URL parameters are both suitable ways to transport data between the client and the server. But they are both readable and on the client side. Sessions solve exactly this problem. You assign the client an ID and it makes all further requests using that ID. Information associated with the client is stored on the server linked to this ID.

We will need the *Express-session*, so install it using the following code.

```
npm install --save express-session
```

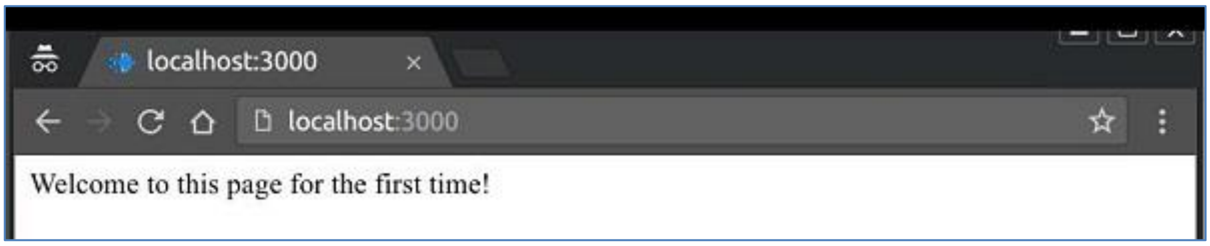
We will put the session and cookie-parser middleware in place. In this example, we will use the default store for storing sessions, i.e., MemoryStore. Never use this in production environments. The session middleware handles all things for us, i.e., creating the session, setting the session cookie and creating the session object in req object.

Whenever we make a request from the same client again, we will have their session information stored with us (given that the server was not restarted). We can add more properties to the session object. In the following example, we will create a view counter for a client.

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');

var app = express();

app.use(cookieParser());
app.use(session({secret: "Shh, its a secret!"}));

app.get('/', function(req, res){
  if(req.session.page_views){
     req.session.page_views++;
     res.send("You visited this page " + req.session.page_views + " times");
  } else {
     req.session.page_views = 1;
     res.send("Welcome to this page for the first time!");
  }
});
app.listen(3000);
```

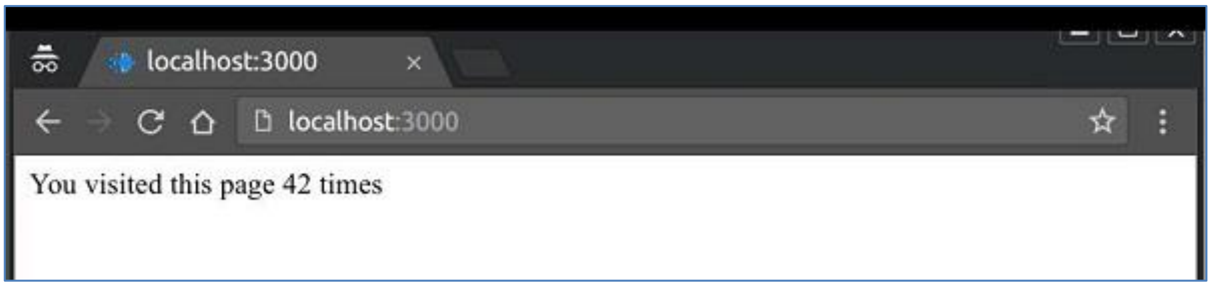
What the above code does is, when a user visits the site, it creates a new session for the user and assigns them a cookie. Next time the user comes, the cookie is checked and the page_view session variable is updated accordingly.

Now if you run the app and go to localhost:3000, the following output will be displayed.



Welcome to this page for the first time!

If you revisit the page, the page counter will increase. The page in the following screenshot was refreshed 42 times.



You visited this page 42 times