Introduction

- ✓ 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 JS

- ✓ Develops Node.js web applications quickly and easily.
- ✓ It's simple to set up and personalize.
- ✓ 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 pug(Jade), Vash, and EJS.
- ✓ Allows you to specify a middleware for handling errors.

Environment Setup

Node Package Manager(npm)

- ✓ **npm** is the package manager for node.
- ✓ The **npm** Registry is a public collection of packages of open-source code for Node.js, front-end web apps, mobile apps, robots, routers, and countless other needs of the JavaScript community.
- ✓ **npm** allows us to access all these packages and install them locally. You can browse through the list of packages available on npm at npmJS.

How to use npm?

There are two ways to install a package using npm: globally and locally.

- **Globally** – This method is generally used to install development tools and CLI based packages. To install a package globally, use the following code.

```
npm install -g <package-name>
```

- **Locally** – This method is generally used to install frameworks and libraries. A locally installed package can be used only within the directory it is installed. To install a package locally, use the same command as above without the **-g** flag.

```
npm install <package-name>
```

Whenever we create a project using npm, we need to provide a **package.json** file, which has all the details about our project. npm makes it easy for us to set up this file. Let us set up our development project.

Step 1 – Start your terminal/cmd, create a new folder named hello-world and cd (create directory) into it

```
mkdir express
cd express
```

Step 2 – Now to create the package.json file using npm, use the following code.

```
npm init
```

It will ask you for the following information.

```
{
 "name": "new-folder",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "",
 "license": "ISC"
}
Is this OK? (yes)
```

Just keep pressing enter, and enter your name at the “author name” field.

Set proxy if required: npm config set proxy http://192.168.10.252:808

Step 3 – Now we have our package.json file set up, we will further install Express. To install Express and add it to our package.json file, use the following command –

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

The --**save** flag can be replaced by the **-S** flag. This flag ensures that Express is added as a dependency to our **package.json** file. This has an advantage, the next time we need to install all the dependencies of our project we can just run the command *npm install* and it will find the dependencies in this file and install them for us.

We can also install express by below command.

```
npm install express
```

OR

```
npm i express
```

To check version of express run below command or you can check in package.json file.

```
npm list express
```

This is all we need to start development using the Express framework.

Create a new file called index.js and type the following in it. (You can create file with any name e1.js,test.js etc)

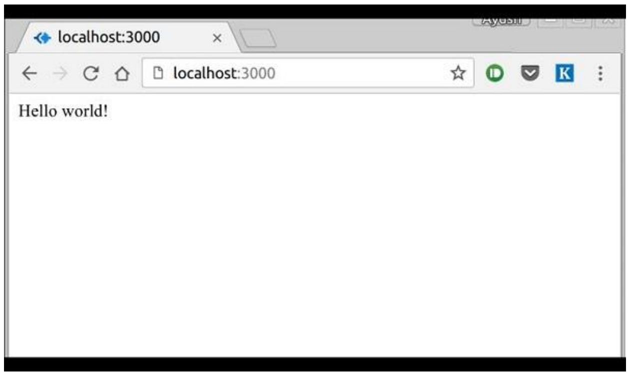
Index.js

```
var express = require('express');
var app = express();
app.get('/', function(req, res){
 res.set ("content-type","text/plain");
 res.send("Hello world!");

});
app.listen(3000);
```

Save the file, go to your terminal and type the following.

```
node index.js
```

This will start the server. To test this app, open your browser and go to **http://localhost:3000** and a message will be displayed as in the following screenshot.



Hello world!

How the App Works?

- ✓ The first line imports Express in our file, we have access to it through the variable Express.
- ✓ We use it to create an application and assign it to var app.

res.set()

The res.set() function is used to set the response HTTP header field to value. To set multiple fields at once, pass an object as the parameter.

```
res.set(field [, value])
```

Parameters: The field parameter is the name of the field and the value parameter is the value assigned to the field parameter.

Return Value: It returns an Object.

app.get(route, callback)

- ✓ This function tells what to do when a **get** request at the given route is called. The callback function has 2 parameters, ***request(req)*** and ***response(res)***.
- ✓ The request **object(req)** represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, etc. Similarly, the response object represents the HTTP response that the Express app sends when it receives an HTTP request.

res.send()

- ✓ This function takes an object as input and it sends this to the requesting client. Here we are sending the string *"Hello World!"*.

app.listen(port, [host], [backlog], [callback]])

- ✓ This function binds and listens for connections on the specified host and port. Port is the only required parameter here.

Sr. No.	Argument & Description
1	**port** A port number on which the server should accept incoming requests.
2	**host** Name of the domain. You need to set it when you deploy your apps to the cloud.
3	**backlog** The maximum number of queued pending connections. The default is 511.
4	**callback** An asynchronous function that is called when the server starts listening for requests.

Response methods

The methods on the response object (res) in the following table can send a response to the client, and terminate the request-response cycle.

Method	Description
res.end()	End the response process.
res.json()	Send a JSON response.
res.redirect()	Redirect a request.
res.render()	Render a view template.

Method	Description
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.

- ✓ **res.end:** comes from NodeJS core. In Express JS if you need to end request in a quick way and do not need to send any data then you can use this function.
- ✓ **res.send:** Sends data and end the request.
- ✓ **res.json:** Sends data in JSON format and ends the request.
- ✓ **res.json()** allows for extra formatting of the JSON data - if this is not required **res.send()** can also be used to return a response object using Express. Both of these methods also end the response correctly, and there's no further action required.

Example

1.js

```
const express = require("express");
const app = express();
app.get ("/", (req,res)=>
{
   res.set ("content-type","text/plain");
   res.send ("<h1>Hello</h1>");
});
app.get ("/about", (req,res)=>
{
   res.set ("content-type","text/html");
   res.write ("Hello");
   res.send ("<h1> Hello from home</h1>");
   //res.write ("hello");
});
app.listen (5504,()=>
{
   console.log ("server started");
})
```

Output

In terminal > node 1.js

Console >>

```
server started
```

In browser >>

- **localhost:5504**
  <h1>Hello</h1>
- **localhost:5504/about**
  Hello

Note: Like node JS we cannot set headers after they are sent to the client. Means if we send response by writing in "res.write()" then "res.send()" must be blank.

*Here in example on about page only hello will be displayed. And in console it will give an error "Cannot set headers after they are sent to the client". Means we cannot write anything in **res.send()**.*

***res.write()** after the res.send() will generate an error "write after end".(Commented line)*

Just for the information

In Node JS we learnt http module.

```
var http=require("http");
var server=http.createServer(
    function(req,res)
    {
    if(req.url=="/")
    {
        res.writeHead(200,{"content-type":"application/json"});
        res.write("<h1>Thank you..!</h1>");
        res.end();
    }
});
server.listen(6001);
```

Routing

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

Each route can have one or more handler functions, which are executed when the route is matched.

The main difference between a URL and a URI is that a URL specifies the location of a resource on the internet, while a URI can be used to identify any type of resource, not just those on the internet.

Route definition takes the following structure:

```
app.METHOD(PATH, HANDLER)
```

- **app** is an instance of express.
- **METHOD** is an HTTP request method, in lowercase.
- **PATH** is a path on the server.
- **HANDLER** is the function executed when the route is matched.

The following examples illustrate defining simple routes.

Respond with Hello World! on the homepage:

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

Respond to POST request on the root route (/), the application’s home page:

```
app.post('/', (req, res) => {
  res.send('Got a POST request')
})
```

✓ The **app.all()** function is used to route all types of HTTP requests.
✓ Like if we have POST, GET, PUT, DELETE, etc, requests made to any specific route, let’s say */user*, so instead of defining different APIs like app.post(‘/user’), app.get(‘/user’), etc, we can define single API **app.all(‘/user’)** which will accept all type of HTTP request.

Route parameters

- ✓ Route parameters are named URL segments that are used to capture the values specified at their position in the URL.
- ✓ The captured values are populated in the **req.params** object, with the name of the route parameter specified in the path as their respective keys.

Route path: /calendar/:day/event/:ename
Request URL: http://localhost:6001/calendar/monday/event/birthday
req.params: {"day":"monday","ename":"birthday"}

To define routes with route parameters, simply specify the route parameters in the path of the route as shown below.

```
var express = require("express");
var app = express();
app.get('/calendar/:day/event/:ename', (req, res) => {
   res.send(req.params);
 });
app.listen (6001,()=>
{
   console.log ("server started");
})
```

To check: http://localhost:6001/**calendar**/monday/**event**/birthday

Output:

```
{"day":"monday","ename":"birthday"}
```

JSON Processing

We can define a JSON object and directly pass it inside res.send(). For this, JSON.stringify() method is not required. To send JSON object in res.write(), convert JSON object to string using JSON.stringify().

Example : Write Express JS script to request server to display json object on browser.

Send JSON object using res.write	**Send JSON object using res.send**

const express=require("express")
const app=express();
student={name:"LJU",age:28}
  app.get("/",(req,res)=>{
    res.write(JSON.stringify(student))
    res.send()
  })
app.listen(6007)
	const express=require("express")
	const app=express();
	student={name:"LJU",age:28}
	  app.get("/",(req,res)=>{
	    res.send(student)
	  })
	app.listen(6007)

Output in browser: {"name":"LJU","age":28}	**Output in browser:** {"name":"LJU","age":28}

Write express script to send json object to server and display it.

Send JSON object using res.json

```
const express=require("express")
const app=express();
student={name:"LJU",age:28}
  app.get("/",(req,res)=>{
    res.json(student)
  })
app.listen(6007)
```

Output in browser: {"name":"LJU","age":28}

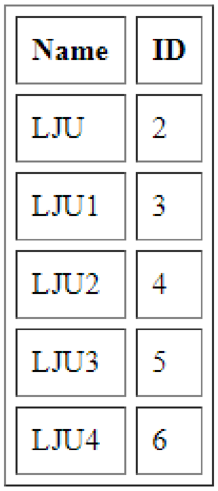
Example : Write Express JS script to request server to display only Age on browser.

```
const express=require("express")
const app=express();
student={name:"LJU",age:28}
  app.get("/",(req,res)=>{
    res.write(student.age+"")  //needs to convert to string so appended string
    res.send()
  })
  app.get("/j",(req,res)=>{
    res.send(student.age+"")  //needs to convert to string so appended string
  })
  app.get("/j1",(req,res)=>{
     res.json(student.age)
  })
app.listen(6001)
```

Example : Write Express JS script to request server to display json object (Array of Objects) in table form on browser.

```
const express=require("express")
const app=express();
student={
   u1:[{name:"LJU",id:2},
   {name:"LJU1",id:3},
   {name:"LJU2",id:4},
   {name:"LJU3",id:5},
   {name:"LJU4",id:6}]}
      app.get("/student",(req,res)=>{
         res.set("content-type","text/html")
            res.write("<center><table  cellspacing='5px'  cellpadding='8px'
border='1px solid'><tr><th>Name</th><th>ID</th></tr>")
          for(i of student.u1){
             res.write("<tr><td>" + i.name + "</td>")
             res.write("<td>" + JSON.stringify(i.id) + "</td></tr>")
          }
         res.write("</table></center>")
         res.send()
      })
app.listen(6007)
```

Output:



Name	ID
LJU	2
LJU1	3
LJU2	4
LJU3	5
LJU4	6

Write an express js script to define one JSON array of 3 objects having properties name and age. Sort this object according to age. If user requests sorted names in url then all names along with age should be printed according to descending order of age. Also, display these sorted values on "Sort page" and display JSON object on "Home page"

```
const express = require("express")
const app = express();
student = [{name:"abc",age:28},
        {name:"def",age:40},
        {name:"xyz",age:50}]
 //home page
app.get ("/",(req,res)=>{
   res.set ("content-type","text/html")
   res.send (student)
})
//sort page
app.get ("/sort",(req,res)=>{
   res.set ("content-type","text/html")
```

```
    var des = student.sort((a,b)=>b.age-a.age)
    console.log(des)
    for (k=0;k<des.length;k++){
       res.write ("<center><h2>"+des[k].name+ " = " +des[k].age +"</h2></center>")
    }
    res.send ()
})
app.listen (5200)
```

OR

```
const express = require("express")
const app = express();
student = [{name:"abc",age:28},
        {name:"def",age:30},
        {name:"xyz",age:50}]
 //home page
app.get ("/",(req,res)=>{
   res.set ("content-type","text/html")
   res.send (student)
})
//sort page
app.get ("/sort",(req,res)=>{
   res.set ("content-type","text/html")
   for (i=0;i<student.length;i++){
      for (j=0;j<student.length;j++){
         if (student[i].age>student[j].age){
            temp = student[i];
            student[i] = student[j];
            student[j] = temp
         }
      }
   }
   for (k=0;k<student.length;k++){
      res.write ("<center><h2>"+student[k].name+ " = " +student[k].age +"</h2></center>")
   }
   res.send ()
})
app.listen (5200)
```

Output:

On home page

```
[{"name":"abc","age":28},{"name":"def","age":40},{"name":"xyz","age":50}]
```

On sort page

xyz = 50

def = 40

abc = 28

Link HTML, CSS and JS files

app.use()

function is used to mount the specified middleware function(s) at the path which is being specified. It is mostly used to set up middleware for your application.

Syntax:

```
app.use(path, callback)
```

Parameters:

- **path:** It is the path for which the middleware function is being called. It can be a string representing a path or path pattern or a regular expression pattern to match the paths.
- **callback:** It is a middleware function or a series/array of middleware functions.

Serving static files in Express

✓ To serve static files such as images, CSS files, and JavaScript files, use the express.static built-in middleware function in Express.

The function signature is:

```
express.static(root, [options])
```

The root argument specifies the root directory from which to serve static assets. For more information on the options argument, see express.static.

path.join()

✓ The path.join() method joins the specified path segments into one path.
✓ You can specify as many path segments as you like.
✓ The specified path segments must be strings, separated by comma.

__dirname

✓ __dirname is an environment variable that tells you the absolute path of the directory containing the currently executing file.

Folder structure

Express **|-----1.js** **|-----index.html** **|-----1.css** **|-----1.png** **Code:** **app.use (express.static (__dirname));**	**Express** **|---1.js** **|-----Frontend** **|------|---index.html** **|------|---1.css** **|------|---1.png** **Code:** **app.use (express.static ("frontend"));**	**Express** **|------Backend** **|--------|--1.js** **|---index.html** **|---1.css** **|---1.png** Here we need to join the path as we need to go to one step up to access the frontend files. **Code:** **const sp = path.join (__dirname ,"../");** **app.use (express.static (sp));**	**Express** **|-------Backend** **|---------|---1.js** **|-------Frontend** **|---------|---index.html** **|---------|---1.css** **|---------|---1.png** Here we need to join the path as we need to go to one step up to access the frontend files. **Code:** **const sp = path.join (__dirname ,"../frontend");** **app.use (express.static(sp));**

Load HTML file named index.html (All files in same folder)

```
Express
|-----1.js
|-----index.html
|-----1.css
```

Example

/express/index.html	**/express/1.css**	**/express/1.js**

`<html>`
`<head><link rel="stylesheet" href="1.css"></head>`
`<body><p class="lj_p"> File name must be index.html </p>`
`</body>`
`</html>`
	`.lj_p {`
	`font-size:100;`
	`color:blueviolet;`
	`}`
		`const express = require ("express")`
		`const app = express();`
		`// name of file must be index.html`
		`app.use(express.static(__dirname))`
		`app.listen (5200)`

To run in terminal : go to "express" folder by command cd express and then node 1.js

Output in browser:

```
File name must be index.html
```

Load HTML file named index.html (create a frontend folder inside express folder and create index.html and 1.css inside it)

```
Express
|---1.js
|-----frontend
|------|---index.html
|------|---1.css
```

Example

/express/frontend/index.html

```
<html>
   <head>
         <link rel="stylesheet"
               href="1.css">
   </head>
   <body>
         <p class="lj_p"> File name must be index.html </p>
   </body>
</html>
```

/express/frontend/1.css

```
.lj_p {
   font-size:100;
   color:blueviolet;
}
```

/express/1.js

```
const express = require ("express")
const app = express();
// name of file must be index.html
app.use(express.static("frontend"))
app.listen (5200)
```

To run in terminal : **go to "express" folder by command cd express and then node 1.js**

Output in browser:

```
File name must be index.html
```

Load HTML file named index.html (create a frontend folder inside express folder and create index.html and 1.css inside it. Also, create backend folder and create 1.js file inside it.)

```
Express
|-------backend
|---------|---1.js
|-------frontend
|---------|---index.html
|---------|---1.css
```

Example

/express/frontend/index.html

```
<html>
   <head>
         <link rel="stylesheet"
               href="1.css">
   </head>
   <body>
         <p class="lj_p"> File name
must be index.html </p>
   </body>
</html>
```

/express/frontend/1.css

```
.lj_p {
   font-size:100;
   color:blueviolet;
}
```

/express/backend/1.js

```
const express = require ("express")
const app = express();
const path = require ("path");
const staticpath = path.join (__dirname,
"../frontend");
// name of file must be index.html
app.use(express.static(staticpath))
app.listen (5200)
```

To run in terminal : **go to "backend" folder by command cd backend and then node 1.js**

Output in browser:

```
File name must be index.html
```

Note that if you don't want to use path module to join the path then you can use below code.

```
const express = require ("express")
const app = express();
app.use(express.static((__dirname, "../frontend")))
app.listen (5200)
```

Load HTML file named index.html (create index.html and 1.css files inside express folder. Create backend folder inside express folder and create 1.js file inside backend folder.)

```
Express
|------backend
|--------|--1.js
|---index.html
|---1.css
```

Example

/express/index.html	/express/1.css	/express/backend/1.js

`<html>`
`<head><link rel="stylesheet" href="1.css"></head>`
`<body><p class="lj_p"> File name must be index.html </p>`
`</body>`
`</html>`
	`.lj_p {`
	`font-size:100;`
	`color:blueviolet;`
	`}`
		`const express = require ("express")`
		`const app = express();`
		`const path = require ("path");`
		`const staticpath = path.join (__dirname, "../");`
		`// name of file must be index.html`
		`app.use(express.static(staticpath))`
		`app.listen (5200)`

To run in terminal : go to "backend" folder by command cd backend and then node 1.js

Output in browser:

```
File name must be index.html
```

load HTML file using res.sendFile()

express/frontend/1.html

```
<html>
  <head>
       <link rel="stylesheet"
            href="1.css">
  </head>
  <body>
       <p class="lj_p"> File name must be index.html </p>
  </body>
</html>
```

express/frontend/1.css

```
.lj_p {
  font-size:100;
  color:blueviolet;
}
```

express /backend/1.js

```
const express = require ("express")
const app = express();
const path = require ("path");
const staticpath = path.join
(__dirname, "../frontend");

//to load css file which is linked in html file
app.use(express.static(staticpath))
//Another way to load HTML file
app.get('/',(req,res)=>{
res.sendFile(staticpath +
"/1.html”);
 })
app.listen (5200)
```

Output in browser:

```
File name must be index.html
```

Load HTML file named other than index.html file.

express/frontend/2.html

```
<html>
  <head>
       <link rel="stylesheet"
href="2.css"></head>
  <body>
         <p class="lj_p">File name is other than index.html.</p>
  </body>
</html>
```

express/frontend/2.css

```
.lj_p {
  font-size:100;
  color:blueviolet;
}
```

express/backend/2.js

```
const express = require ("express")
const path = require ("path");
const app = express();
const staticpath = path.join
(__dirname,"../frontend");
app.use
(express.static(staticpath,{index:"2.html"}));
app.listen(5200)
```

Output in browser:

```
File name can be different.
```

Example to understand concept of all files in different folders.

```
Express
|-------js
|---------|---1.js
|-------html
|---------|---1.html
|-------css
|---------|---1.css
|-------image
|---------|--- folderstructure.png
```

express/js/1.js

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

var css_path = path.join(__dirname,"../css")
var img_path= path.join(__dirname,"../image")
var html_path = path.join(__dirname,"../html")

app.use(express.static(css_path))
app.use(express.static(img_path))
app.use(express.static(html_path,{index:"1.html"}))

app.listen(5001,()=>console.log("Started"))
```

express/html/1.html

```
<html>
   <head>
<link rel="stylesheet" href="1.css">
   </head>
   <body>
      <p class="lj_p"> Hello </p>
      <img src="folderstructure.png">
   </body>
</html>
```

express/css/1.css

```
.lj_p{
   font-size: 60px;
   color:purple;
   text-align: center;
}
img{ width:400px; height:auto;}
```

GET and POST

- ✓ GET and POST both are two common HTTP requests used for building REST API's.
- ✓ GET requests are used to send only limited amount of data because data is sent into header while POST requests are used to send large amount of data because data is sent in the body.
- ✓ Express.js facilitates you to handle GET and POST requests using the instance of express.

Get method

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

Example 1

Write express script to get form data using get method and display data in JSON format in next page named "process_get"

index.html

```
<html>
<body>
<form action="/process_get" method="get">
First Name: <input type="text" name="first_name"> <br>
Last Name: <input type="text" name="last_name">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

example1.js

```
var express = require('express');
var app = express();
app.use(express.static(__dirname))
app.get('/process_get', function (req, res) {
response = {
      fname:req.query.first_name,
      lname:req.query.last_name
  };
  console.log(req.query);  //req.query is used to get the submitted data in json object format. To access particular key value we have to write "req.query.first_name" or
  res.send(response);
})
app.listen(8000)
```

Example 2

Write express js script to print message in next line splitting by "." And use get method to submit the data. HTML file contains form of text area for the message and submit button.

frontend/2.html

```
<html>
  <head><title>Get Method</title></head>
  <body>
    <form method="get" action="/login">
    <textarea rows="10" cols="10" name="message">Hi.Hello.how are you
    </textarea>
    <button name="Submit" value="Submit">Submit</button>
  </form>
  </body>
</html>
```

backend/2.js

```
var express = require("express");
var app = express();
var p = require("path");
const staticp = p.join(__dirname,"../frontend");

app.use(express.static(staticp,{index: '2.html'}));

app.get("/login",(req,res)=>{
  res.set("content-type","text/html");

  t1 = (req.query.message).split(".");
  for(i in t1){
    res.write(t1[i]+ "</br>");
  }
  res.send();

});
app.listen(5122);
```

Output:

```
Hi
Hello
how are you
```

Example 3

Write express js script to perform tasks as asked below.

1. **Create one HTML file which contains two number type input fields, one dropdown which contains options like (select, addition, subtraction, multiplication, division) and one submit button.**
2. **The input fields must contain the value greater than 0 else it will give a message “Please enter the valid number”. Also, user must select any of the formula from the dropdown else give a message “You have not selected any formula”. (Message will be displayed on “/calc” page.)**
3. **If one formula is selected and numbers are entered then respective calculations will be performed on the page “/calc”.**
4. **Use get method to request data.**

/backend/calc.js

```
var express = require("express");
var app = express();
var p = require("path");

//css file in folder css
var css = p.join(__dirname,"../css");
app.use(express.static(css));

// Html file in folder frontend
const staticp = p.join(__dirname,"../frontend");
app.use(express.static(staticp,{index:'calc.html'}));

app.get("/calc",(req,res)=>{
   res.set("content-type","text/html");
   var n1 = parseInt(req.query.n1);
   var n2 = parseInt(req.query.n2);
   if ((n1>0) && (n2>0)) {
      if(req.query.formula == "addition"){
         a = n1+n2;
         res.write("<h1>Addition is : " + a + "</h1>");
      }
      else if(req.query.formula == "subtraction"){
         s = n1-n2;
         res.write("<h1>Subtraction is : " + s + "</h1>");
      }
      else if(req.query.formula == "multi"){
         m = n1*n2;
         res.write("<h1>Multiplication is : " + m + "</h1>");
      }
      else if(req.query.formula == "div"){
         d = n1/n2;
```

```
            res.write("<h1>Division is : " + d + "</h1>");
        }
        else{
            res.write("<h1>You have not selected any formula.</h1>");
        }
    }else{
        res.write("<h1>Please enter the valid number/s.</h1>");
    }
    res.send()
})
app.listen(6012);
```

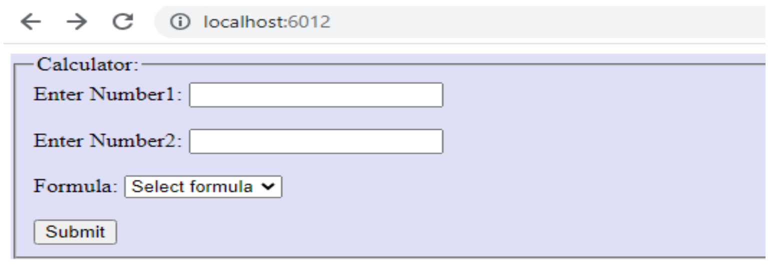
/css/t1.css

```
.lj_form{
    background-color: rgb(158, 158, 219);
}
```

/frontend/calc.html

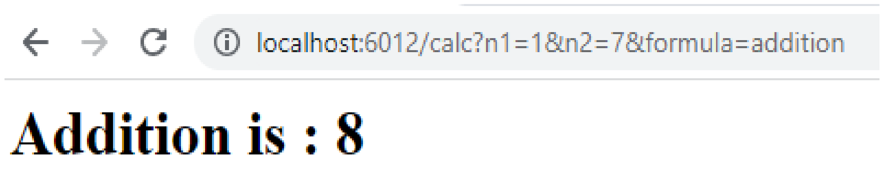
```
<html>
<head>
    <link rel="stylesheet" href="/t1.css">
</head>
<body>
    <form action="/calc" method="get" class="lj_form">
    <fieldset>
        <legend>Calculator:</legend>
            <label for="n1">Enter Number1:</label>
            <input type="number" name="n1"><br><br>

            <label for="n2">Enter Number2:</label>
            <input type="number" name="n2"><br><br>

            <label for="formula">Formula: </label>
            <select name="formula" id="formula">
                <option value="">Select formula</option>
                <option value="addition">Addition</option>
                <option value="subtraction">Subtraction</option>
                <option value="multi">Multiplication</option>
                <option value="div">Division</option>
            </select><br><br>

            <input type="submit" value="Submit">
    </fieldset>
    </form>
</body>
</html>
```

Output:



localhost:6012

Calculator:
Enter Number1:
Enter Number2:
Formula: Select formula
Submit



localhost:6012/calc?n1=1&n2=7&formula=addition

Addition is : 8

Post method

It facilitates you to send large amount of data because data is sent in the body. Post method is secure because data is not visible in URL.

Please note that new version of express contains built-in body-parser module. We don't need to install it and import it to our program.
We can access it's functionality by just adding below line to our program.

```
var express = require("express");
app.use(express.urlencoded());
```

If above won't work properly then follow the following steps of installation and importing module to the program.

Installation

```
npm install body-parser
```

API

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

- Body-parser parses is an HTTP request body that usually helps when you need to know more than just the URL being hit.
- It provides four express middleware for parsing JSON, Text, URL-encoded, and raw data sets over an HTTP request body.
- Using **body-parser** allows you to access **req.body** from within routes and use that data.

- The **urlencoded()** function is a built-in middleware function in Express.
- It parses incoming requests with URL-encoded payloads and is based on a body parser.
- Express.urlencoded() expects request data to be sent encoded in the URL, usually in strings or arrays

    - **extended**
        - The extended option allows to choose between parsing the URL-encoded data with the **querystring** library (**when false**) or the **qs** library (**when true**).

Default value is true, but using the default has been deprecated.

Example 1

Write express script to retrieve form data using post method and display data in JSON format in next page named “process”

express/frontend/p.html

```
<html>
<body>
<form action="/process" method="post">
First Name: <input type="text" name="first_name">  <br>
Last Name: <input type="text" name="last_name">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

express/backend/p.js

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

app.use(express.urlencoded({ extended: false }));
var staticp = path.join(__dirname, "../frontend");

app.use(express.static(staticp,{index: 'p.html'}));

app.post('/process', function (req, res) {
  console.log(req.body.first_name+” “ +req.body.last_name)
  res.send(req.body);
})

app.listen(7002)
```

Example 2

Write express js script to perform the tasks as asked below.

1)Create one HTML file and add one form which contains username, password and submit button. Data should be submitted by HTTP post method.
2)Submit button is of black color with white text. (External CSS)
3)On home page form should be displayed and while submitting the form, on next page named “/login” if username is admin then it will display “Welcome admin” else display “Please login with admin name”.

express/backend/po2.js

```
var express = require("express");
var app = express();
var p = require("path");
app.use(express.urlencoded());

const staticp = p.join(__dirname,"../frontend");
app.use(express.static(staticp,{index: 'po2.html'}));

app.post("/login",(req,res)=>{
   if( req.body.uname == 'admin' ){
      res.write("<h1> Welcome " + req.body.uname + "</h1>")
   }
   else{
      res.write(`<h1 style="color:red">Please login with admin name</h1>`);
   }
   res.send();
})
app.listen(5222);
```

express/frontend/po2.html

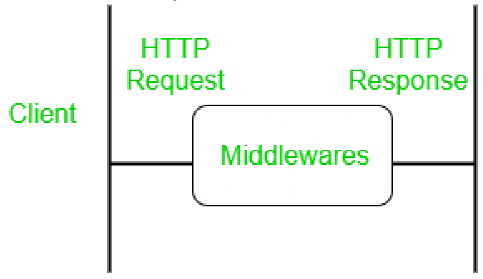
```
<html>
   <head><title>Get Method</title>
      <link rel="stylesheet" href="po2.css">
   </head>
   <body>
      <form method="post" action="/login">
      <div class="lj-div">
         <label>User name:</label>
         <input type = "text" name="uname" ></input>
      </div>
      <div class="lj-div">
         <label>Password:</label>
         <input type = "text" name="password" ></input>
      </div>
      <button name="Submit" value="Submit">Submit</button>
   </form>
   </body>
</html>
```

express/frontend/po2.css

```
button{
    padding: 10px;
    background-color: black;
    color: #fff;
}
```

Express Middleware

Express.js is a routing and Middleware framework for handling the different routing of the webpage and it works between the request and response cycle. Middleware gets executed after the server receives the request and before the controller actions send the response. Middleware has the access to the request object, responses object, and next, it can process the request before the server send a response.

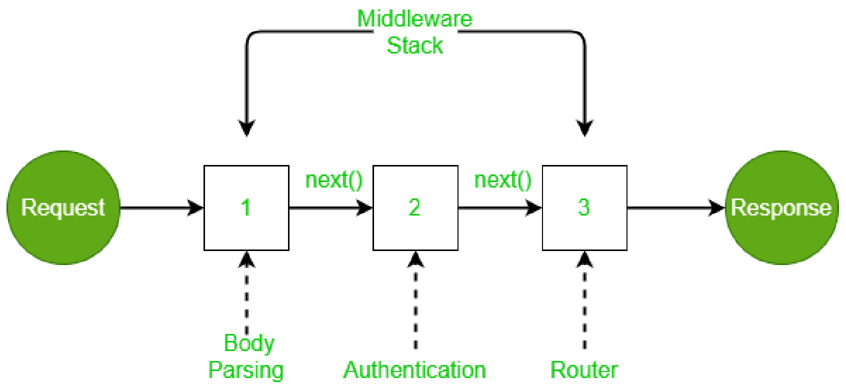


An Express-based application is a series of middleware function calls.

Advantages of using middleware:

- 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.

Middleware Chaining: Middleware can be chained from one to another, Hence creating a chain of functions that are executed in order. The last function sends the response back to the browser. So, before sending the response back to the browser the different middleware process the request.
The next() function in the express is responsible for calling the next middleware function if there is one.
Modified requests will be available to each middleware via the next function –



Middleware chaining example

In the above case, the incoming request is modified and various operations are performed using several middlewares, and middleware is chained using the next function. The router sends the response back to the browser.

Below is example to understand concept of middleware.

```
var express = require("express");
var app = express();
//Method 1
const cb=(req,res,next)=>
{
   console.log("Initialized");
   res.set("content-type","text/html")
   res.write("<strong>First</strong>");
   next();
}
const cb1=(req,res,next)=>
{
   res.write("<p>Addition = " + (5+5) + "</p>");
   next();
}
app.use("/ee",cb,cb1);
app.get("/ee",(req,res)=>
{
   res.write("<h1>Hello Welcome to LJU</h1>");
   res.send();
});

//Method 2
app.use(
   "/xyz",
   (req, res, next) => {
     console.log("request received on"+new Date());
     next();
   },
   (req, res, next) => {
      res.set("content-type","text/html")
      res.write("Hello");
      next();
     },
   (req, res) => {
     res.write(`<div>
     <h2>Welcome to LJU</h2>
     <h5>Tutorial on Middleware</h5>
   </div>`);
   res.send();
   }
  );
app.listen(6001)
```

Output:

localhost:6001/ee

<u>On browser</u>

```
First
Addition = 10
Hello Welcome to LJU
```

<u>In console</u>

```
Initialized
```

localhost:6001/xyz

<u>On browser</u>

```
Hello
Welcome to LJU
Tutorial on Middleware
```

<u>In console</u>

```
request received onMon Jun 10 2024 15:22:44 GMT-0700 (Pacific Daylight Time)
```

Example 1

Write express js script to perform following tasks.

1. **Create one html file which contains one text field for name, email field and checkbox for subscription. Html file will be loaded on home page. Email and name fields are required fields.**
2. **On login page welcome user and email id data should be printed.**
   a. **If user checked the subscription then "Thank you for the subscription" message will be printed and "logout" link will be displayed under the message. If user clicks logout link then he/she will be redirected to the home page.**
   b. **If user has not opted for the subscription then "You can subscribe to get daily updates" message will be printed and "subscribe" link will be displayed under the message.**
   c. **If user clicks subscribe link then he/she will be redirected to the subscription page. In this page "Thank you for the subscription" message will be printed and "logout" link will be displayed under the message. If user clicks logout link then he/she will be redirected to the home page.**

Use concept of the middleware and you can use any of http methods(get/post).

backend/t1.js

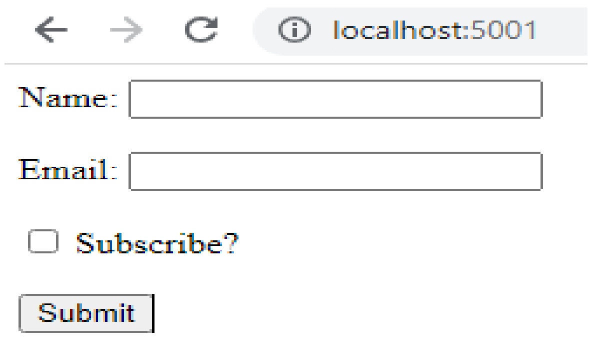
```
var express = require("express");
var app = express();
var p = require("path");

const staticp = p.join(__dirname,"../frontend");
app.use(express.static(staticp,{index:'t1.html'}));

app.use(express.urlencoded());

app.post("/login",(req,res,next)=>{
   console.log(req.query);
   res.set("content-type","text/html");
   res.write("<center><h1>Welcome " + req.body.name + "</h1>");
   res.write("<center><h2>Your email id is " + req.body.email + "</h2>");
   next();
});

app.post("/login",(req,res)=>{
   if(req.body.newsletter == "on"){
      res.write("<h3>Thank you for your subscription</h3><a href='/'>Logout</a>");
   }else{
      res.write("<h3>You can subscribe to get daily updates</h3><a
href='/subscribe'>Subscribe</a></center>");
   }
  res.send();
});
```

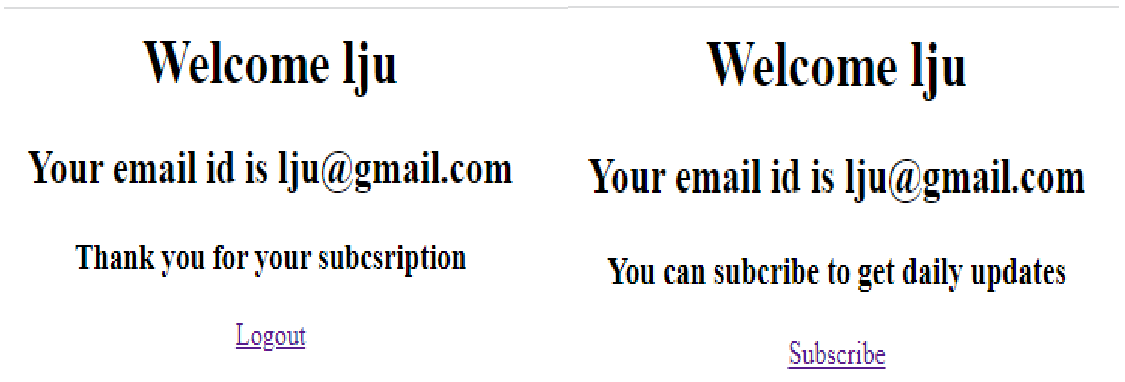
```
app.get("/subscribe",(req,res)=>{
   res.set("content-type","text/html");
   res.write("<h3>Thank you for your subscription</h3></center><a href='/'>Logout</a>");
   res.send();
});

app.listen(5001);
```

/frontend/t1.html

```
<form action="/login" method="post">
   <label for="name">Name:</label>
   <input type="text" name="name" required><br><br>

   <label for="email">Email:</label>
   <input type="email" name="email" required><br><br>

   <input type="checkbox" name="newsletter" id="newsletter">
    <label for="newsletter">Subscribe?</label><br><br>

    <input type="submit" value="Submit">
</form>
```

Output:





```
Welcome lju

Your email id is lju@gmail.com

Thank you for your subcsription

Logout
```

```
Welcome lju

Your email id is lju@gmail.com

You can subcribe to get daily updates

Subscribe
```

Example 2

Write an express.js script to load an HTML file having username and password and submit button. On clicking submit button. It should jump on "check" page using "POST" method. If username is "admin" , then jump on next middleware to print "welcome… admin" , if usernameis not "admin" , then stay on same middleware to print "warning msg" in red color.

Index.html

```
<html>
<body>
<form action ="/check" method="post">
Username: <input type="text" name="uname"/>
password: <input type="text" name="pwd"/>
<input type="submit"/>
</form>
</body>
</html>
```

App.js

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

app.use(express.urlencoded({extended:false}));
app.use(express.static(__dirname));

app.post("/check",(req,res,next)=>{
if(req.body.uname=="admin")
{
   next();
}
else {
   res.send("<h1 style='color:red'>wrong credentials</h1>") }
});

app.post("/check",(req,res)=>
{
 res.write("<h1>welcome...”+ req.body.uname+ “</h1>”)
 res.send();
}
).listen(3001);
```

Cookie

How cookies work

- When a user visits a cookie-enabled website for the first time, the browser will prompt the user that the web page uses cookies and request the user to accept cookies to be saved on their computer. Typically, when a makes a user request, the server responds by sending back a cookie (among many other things).
- This cookie is going to be stored in the user's browser. When a user visits the website or sends another request, that request will be sent back together with the cookies. The cookie will have certain information about the user that the server can use to make decisions on any other subsequent requests.

For example, **Facebook** from a browser.
When you want to access your Facebook account, you have to log in with the correct credentials.
When you first make a login request and the server verifies your credentials, the server will send your Facebook account content. It will also send cookies to your browser. The cookies are then stored on your computer and submitted to the server with every request you make to that website. A cookie will be saved with an identifier that is unique to that user.
When you revisit Facebook, the request you make, the saved cookie, and the server will keep track of your login session and remember who you are and thus keep you logged in.

The main difference between a session and a cookie

The major difference between sessions and cookies is that sessions live on the server-side (the webserver), and cookies live on the client-side (the user browser). Sessions have sensitive information such as usernames and passwords. This is why they are stored on the server. Sessions can be used to identify and validate which user is making a request.

As we have explained, cookies are stored in the browser, and no sensitive information can be stored in them. They are typically used to save a user's preferences.

- cookie-parser - cookie-parser looks at the headers in between the client and the server transactions, reads these headers, parses out the cookies being sent, and saves them in a browser. In other words, cookie-parser will help us create and manage cookies depending on the request a user makes to the server.

Run the following command to install these NPM packages:

```
npm install cookie-parser
```

We will create a simple example to demonstrate how cookies work.

Step 1 - Import the installed packages
To set up a server and save cookies, import the cookie parser and express modules to your project. This will make the necessary functions and objects accessible.

```
const express = require('express')
const cookieParser = require('cookie-parser')
```

Step - 2 Get your application to use the packages

You need to use the above modules as middleware inside your application, as shown below.

```
//setup express app
const app = express()
// let’s you use the cookieParser in your application
app.use(cookieParser());
```

This will make your application use the cookie parser and Express modules.

Example:

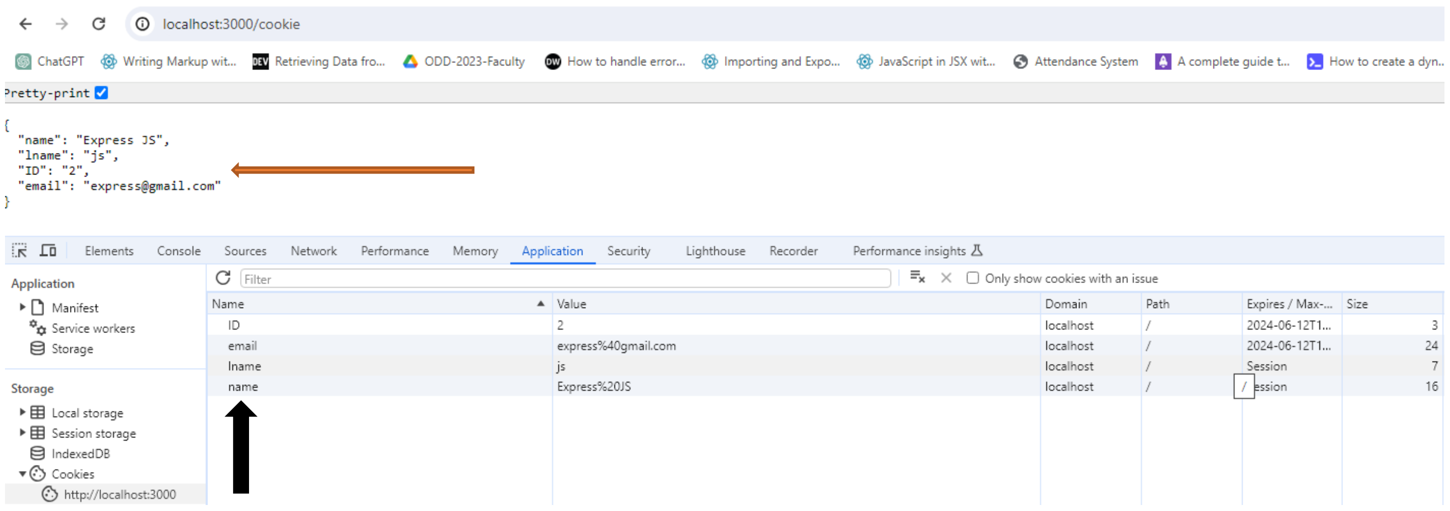
app.js

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

app.get('/cookie', function(req, res){
res.cookie('name', 'Express JS');
res.cookie('fname', 'express'); //Sets fname = express
res.cookie('lname', 'js'); //Sets lname = js
res.cookie('ID','2',{ expires:new Date(Date.now()+10000)}); //expires after 10 secs
res.cookie('email', 'express@gmail.com',{maxAge:2000}); //expires after 2 secs

res.clearCookie('fname');

//show the saved cookies
const cookies = req.cookies;
res.send(cookies);  //Use req.cookies method to check the saved cookies
});
app.listen(3000);
```

When the above route is executed from a browser, the client sends a get request to the server. But in this case, the server will respond with a cookie and save it in the browser.

Open http://localhost:3000/cookie your browser and access the route.

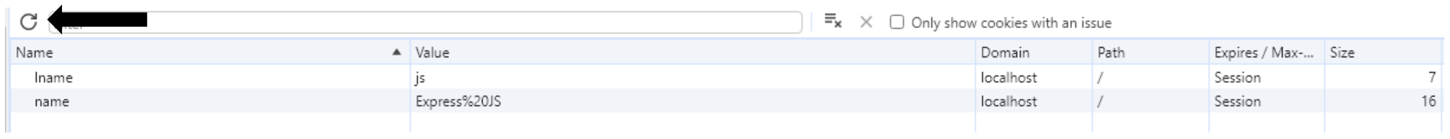
To confirm that the cookie was saved, go to your browser’s Inspect element > select the application tab > cookies > select your domain URL.



```
{
  "name": "Express JS",
  "lname": "js",
  "ID": "2",
  "email": "express@gmail.com"
}
```

Name	Value	Domain	Path	Expires / Max-...	Size
ID	2	localhost	/	2024-06-12T1...	3
email	express%40gmail.com	localhost	/	2024-06-12T1...	24
lname	js	localhost	/	Session	7
name	Express%20JS	localhost	/	Session	16

After 10 seconds check cookies



Name	Value	Domain	Path	Expires / Max-...	Size
lname	js	localhost	/	Session	7
name	Express%20JS	localhost	/	Session	16

To check if your cookie is set or not, just go to your browser, fire up the console, and enter –

console.log(document.cookie);

Example

Write an express js script to set cookies of submitted values of form. Perform following tasks.

- **Create a HTML file which contains a form with fields first name, last name, password and a submit button.**
- **Once form submitted, store all these entered values to the respective cookies on '/next' page.**
- **Then redirect user to "/admin" page and clear the cookie set for the last name. Display remaining set cookie values on this page. (Using post method)**

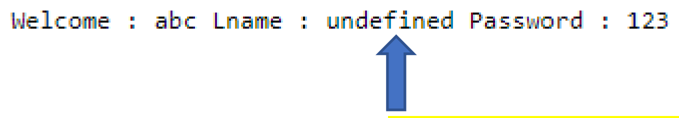
c1.html

```
<html>
  <body>
    <form method="post" action="/next">
    First Name : <input type = "text" name="fname" ></input>
    Last Name : <input type = "text" name="lname" ></input>
    Password : <input type = "password" name="password" ></input>
```

```
      <button name="Submit" value="Submit">Submit</button>
   </form>
   </body>
</html>
```

c1.js

```
var express = require("express");
var app = express();
var cp = require("cookie-parser");
app.use(cp());
app.use(express.static(__dirname,{index: 'c1.html'}));
app.use(express.urlencoded())
app.post("/next",(req,res,next)=>{
   res.cookie("fname", req.body.fname);
   res.cookie("password", req.body.password)
   res.cookie("lname", req.body.lname);
   res.redirect("/admin");
})
app.get("/admin",(req,res)=>{
   res.clearCookie('lname');
   res.write(" Welcome : " + req.cookies.fname);
   res.write(" Lname : " + req.cookies.lname);
   res.write(" Password : " + req.cookies.password);

   res.send()
});
app.listen(3000);
```



```
Welcome : abc Lname : undefined Password : 123
```

We have cleared lname cookie. So, req.cookies.lname is undefined.

Example

You have been assigned to develop a user feedback form for a website using Express.js and cookies. Implement the following requirements:

- **Create a form with the following fields:**
    - **Name (input field)**
    - **Email (input field)**
    - **Message (textarea field)**
    - **Rating (radio buttons: Bad, Average, Good, Very Good, Excellent)**
- **When the user submits the form, store their feedback information (name, email, message, and rating) in a cookie named "feedback" that expires in 10 seconds.**

- **Display a confirmation message to the user after successfully submitting the form & Create a link to display the feedback details stored in the "feedback" cookie.**
- **When the user click to the link, retrieve the feedback information from the cookie and display it on the page also include a link on the feedback details page to Logout.**
- **When the user clicks the link, user redirected to home page.**

feedback.js

```
const express = require('express');
const cp = require('cookie-parser');
const app = express();
app.use(cp());
app.use(express.urlencoded({ extended: true }));
app.use(express.static(__dirname,{index:'feedback.html'}))

app.post('/submit-feedback', (req, res) => {
  const { name, email, message, rating } = req.body;
  // Create the feedback object
  const feedback = {name,email,message,rating};
  // Set the feedback cookie with a 10-second expiration
  res.cookie('feedback', feedback, { maxAge: 10000 });
  res.send('Thank you for your feedback! <br> <a href="/feedback-details"> Show Feedback </a>');
});

app.get('/feedback-details', (req, res) => {
  const feedback = req.cookies.feedback;

  if (feedback) {
    res.send(`
      <h1>Feedback Details</h1>
      <p><strong>Name:</strong> ${feedback.name}</p>
      <p><strong>Email:</strong> ${feedback.email}</p>
      <p><strong>Message:</strong> ${feedback.message}</p>
      <p><strong>Rating:</strong> ${feedback.rating}</p>
      <a href="/" > logout </a>`);
  } else {
    res.send('No feedback available.');
  }
});

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

feedback.html

```
<!DOCTYPE html>
<html>
<head>
  <title>User Feedback Form</title>
</head>
<body>
  <h1>User Feedback Form</h1>
  <form action="/submit-feedback" method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br><br>

    <label for="message">Message:</label>
    <textarea id="message" name="message" required></textarea><br><br>

    <label for="rating">Rating:</label>
    <input type="radio" id="rating1" name="rating" value="Bad" required>
    <label for="rating1">Bad</label>
    <input type="radio" id="rating2" name="rating" value="Average" required>
    <label for="rating2">Average</label>
    <input type="radio" id="rating3" name="rating" value="Good" required>
    <label for="rating3">Good</label>
    <input type="radio" id="rating4" name="rating" value="Very Good" required>
    <label for="rating4">Very Good</label>
    <input type="radio" id="rating5" name="rating" value="Excellent" required>
    <label for="rating5">Excellent</label><br><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Session

HTTP is stateless; in order to associate a request to any other request, you need a way to store user data between HTTP requests. 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. Install the using the command:

```
npm install express-session
```

To set up the session, you need to set a couple of Express-session options, as shown below.

```
const oneDay = 1000 * 60 * 60 * 24; // creating 24 hours from milliseconds
app.use(sessions({
   secret: "thisismysecretkey",
   saveUninitialized:true,
   cookie: { maxAge: oneDay },
   resave: false
}));
```

secret - a random unique string key used to authenticate a session. It is stored in an environment variable and can't be exposed to the public. The key is usually long and randomly generated in a production environment.

resave - Forces the session to be saved back to the session store, even if the session was never modified during the request. Depending on your store this may be necessary, but it can also create race conditions where a client makes two parallel requests to your server and changes made to the session in one request may get overwritten when the other request ends, even if it made no changes (this behavior also depends on what store you're using).

The default value is true, but using the default has been deprecated, as the default will change in the future. Please research into this setting and choose what is appropriate to your use-case. Typically, you'll want false.
resave = true
It means when the modification is performed on the session it will re write the req.session.cookie object.

resave = false
It will not rewrite the req.session.cookie object. the initial req.session.cookie remains as it is.
saveUninitialized - Forces a session that is "uninitialized" to be saved to the store. A session is uninitialized when it is new but not modified. Choosing false is useful for implementing login sessions, reducing server storage usage, or complying with laws that require permission before setting a cookie. Choosing false will also help with race conditions where a client makes multiple parallel requests without a session.

The default value is true, but using the default has been deprecated, as the default will change in the future. Please research into this setting and choose what is appropriate to your use-case.
Uninitialised = false
It means that Your session is only Stored into your storage, when any of the Property is modified in req.session

Uninitialised = true
It means that Your session will be stored into your storage Everytime for request. It will not depend on the modification of req.session.

cookie: { maxAge: oneDay } - this sets the cookie expiry time. The browser will delete the cookie after the set duration elapses. The cookie will not be attached to any of the requests in the future. In this case, we've set the maxAge to a single day as computed by the following arithmetic.

Examples:

Write express js script using session to display how many times a user visited a website. If user is visiting a website for the first time then display "Welcome! Thank you for visiting our website!" else display the count of user (How many times) for that particular session.

```
const express=require("express");
const app=express();
const sess=require("express-session");
app.use(sess(
  {
    resave:true,
    saveUninitialized:true,
    secret:"LJU123"
  }
));
app.get("/",(req,res)=>
{
  if(req.session.page_views)
  {
    req.session.page_views++;
    res.send(`<h1 style="color:blue;"> You have visited page ${req.session.page_views} times <h1>`);
  }
  else{
    req.session.page_views=1;
    res.send(`<h1 style="color:green;"> Welcome! Thank you for visiting our website!<h1>`);
  }
});
app.listen(8003,()=>{console.log("server running at 8003");});
```

Example:
write a script to meet following requirements:

- **Create index.html file page which contains form(username,password,login button). and open it on localhost.**
- **After clicking submit button, it should jump on "savesession" page. Store username and password in session.**
- **After saving session, redirect to "fetchsession" page and read value. Put a LOGOUT link button here.**
- **Jump on delete session page after clicking LOGOUT button.**
- **Destroy the session on this page and redirect to index.html page.**

session.js

```
var express = require("express");
var app = express();
var es = require("express-session");
app.use(es({
   resave:false,
   saveUninitialized:false,
   secret:"lju1"
}));
app.use(express.static(__dirname));

app.get("/savesession",(req,res)=>{
   req.session.uname = req.query.uname;
   req.session.password = req.query.password;
   res.redirect("fetchsession")
})
app.get("/fetchsession",(req,res)=>{
   res.write("<h1>Welcome " + req.session.uname +"</h1>")
   res.write("<a href='/deletesession'>Logout</a>")
   res.send();
});
app.get("/deletesession",(req,res)=>{
  req.session.destroy()
  res.redirect('/')
});
app.listen(6177);
```

index.html

```
<html>
   <body>
      <center>
      <form method="get" action="/savesession">
         <div><input type = "text" name="uname" placeholder="Username" ></input></div>
         <div><input type = "password" name="pass" placeholder="Password"></input></div>
```

```
        <input class="lj_in" type="submit" name="submit" value="Login">
      </form>
   </center>
   </body>
</html>
```

Example:
write a script to meet following requirements:

- **Create session.html file page which contains form(username,password,login button). and open it on localhost.**
- **After clicking submit button, it should jump on "save" page. Store username and password in session.**
- **After saving session, redirect to "fetchdata" page and read value. On this page check authentication of user. User name and password must be "admin" and "admin@123" respectively.**
    - **If this condition is true then display welcome admin and display logout link on this page(fetchdata).**
        - **By clicking on logout link user should jump to "destroy" page and destroy the session there and display the message "Session destroyed". And give the link of "login" under that message. By clicking that link user will be redirected to the home page.**
    - **Else display "Please enter valid username and password" and login link on this page(fetchdata).**

session.html

```
<html>
   <body>
      <center>
      <form method="get" action="/save">
         <div><input type = "text" name="uname" placeholder="Username" ></input></div>
         <div><input type = "password" name="pass" placeholder="Password"></input></div>
         <input type="submit" name="submit" value="Login">
      </form>
   </center>
   </body>
</html>
```

Session.js

```
var express = require("express");
var app = express();
var es = require("express-session");
app.use(es({
   resave:false,
   saveUninitialized:false,
   secret:"lju1"
}));
```

```
app.use(express.static(__dirname,{index:'session.html'}));

app.get("/save",(req,res)=>{
   req.session.uname = req.query.uname;
   req.session.password = req.query.pass;
   res.redirect("fetchdata")

})
app.get("/fetchdata",(req,res)=>{
   if(req.session.uname == "admin" && req.session.password=="admin@123"){
      res.write("<h1 style='color:green;'>Welcome Admin</h1>")
      res.write("<a href='/destroy'>Logout</a>")
   }else{
      res.write("<h1 style='color:red;'>Please enter valid username or password</h1><a href='/'>Login</a>")
   }
   res.send();
});
app.get("/destroy",(req,res)=>{
   req.session.destroy()
    res.write("<h1>Session Destroyed</h1><a href='/'>Login</a>")
    res.send();
});
app.listen(6177);
```

Miscellaneous Task

If you want to set 404 status code on page not found error.

```
const express=require("express");
const app=express();
app.get("/",(req,res)=>
        {
        res.write("Home Page"); res.send();
        })
app.get("/student",(req,res)=>
        {
        res.write("Student Page"); res.send();
        })
app.get("*",(req,res)=>
        {
        res.status(404).end("page not found");
        })
app.listen(8081,()=>{console.log("server started");})
```