

Unit-4 Express.js

Express.js is a web framework for Node.js. It is a fast, robust and asynchronous in nature.

What is Express.js

- Express is a fast, assertive, essential and moderate web framework of Node.js.
- Assume express as a layer built on the top of the Node.js that helps manage a server and routes
- It provides a robust set of features to develop web and mobile applications.

Features of Express

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

Why use Express

- o Ultra fast I/O
- o Asynchronous and single threaded
- o MVC like structure
- o Robust API makes routing easy

ExpressJS - Environment

Assuming that you have installed node.js on your system, the following steps should be followed to install express on your Windows:

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

```
$ mkdir express_practice  
$ cd express_practice
```

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

```
$ npm init
```

STEP-3: Installing Express

Now in your express_practice(*name of your folder*) folder type the following command line:

```
$ npm install express
```

Create a new file called index.js and type the following in it.

Index.js

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

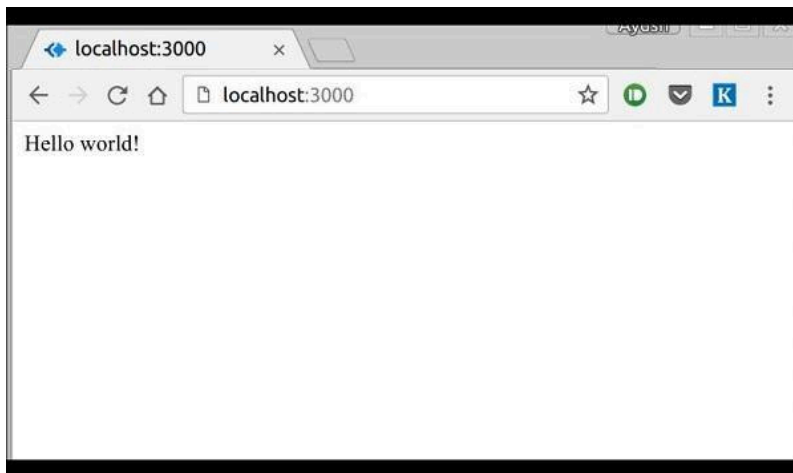
app.get('/', function(req, res){
  res.send("Hello world!");
});

app.listen(6000);
```

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.



Request & Response

Express application uses a callback function whose parameters are request and response objects.

```
app.get('/', function (req, res) {
  // --
})
```

Request Object – The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.

Response Object – The response object represents the HTTP response that an Express app sends when it gets an HTTP request.

Response methods

The methods on the response object (res) in the following table can send a response to the client.

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

Example

index.js

```

const expr = require("express");
const app = expr();
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.write("<h1> Hello from home</h1>");
    res.send();
});
app.listen (5000,()=>>

```

```
{ console.log ("server started");  
}}
```

Example: if you want to display page not found error .

```
const expr=require("express");  
const app=expr();  
app.get("/",(req,res)=>  
{  
    res.write("hello");  
    res.send();  
})  
app.get("/about",(req,res)=>  
{  
    res.write("world");  
    res.send();  
})  
app.get("*(req,res)=>  
{  
    res.status(404).send("page not found");  
})  
app.listen(5001,()=>  
{  
    console.log("server started");  
});
```

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.

Route definition takes the following structure:

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

Where:

- 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.

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.

```
app.all('/', function(req, res){  
  res.send("HTTP method doesn't have any effect on this route!");  
});
```

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.

To use the dynamic routes, we should provide different types of routes. Using dynamic routes allows us to pass parameters and process based on them.

Here is an example of a dynamic route:

```
var express = require('express');  
var app = express();  
app.get('/:id', function(req, res){  
  res.send('The id you specified is ' + req.params.id);  
});  
app.listen(3000);
```

To test this go to `http://localhost:3000/123`. The following response will be displayed.

You can replace '123' in the URL with anything else and the change will reflect in the response.

A more complex example of the above is:

```
var express = require('express');  
var app = express();  
app.get('/things/:name/:id', function(req, res){  
  res.send('id: ' + req.params.id + ' and name: ' + req.params.name);  
});  
app.listen(3000);
```

To test the above code, go to `http://localhost:3000/things/practice/12345`.

You can use the `req.params` object to access all the parameters you pass in the url. Note

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

```
var expr = require("express");
var app = expr();
app.get('/user/:userId/test/:test', (req, res) => {
  req.params;
  res.send(req.params);
});
app.listen(5000)
```

Output: { "userId": "342", "test": "889" }

Route path: /user/:userId/test/:test

Request URL: http://localhost:5000/user/342/test/889

req.params: { "userId": "342", "test": "889" }

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.

Sending a response

How to send a response back to the client using Express

Response.send() method is used to send a simple string as a response, and to close the connection:

```
(req, res) => res.send('Hello World!')
```

If you pass in a string, it sets the Content-Type header to text/html .

if you pass in an object or an array, it sets the application/json Content-Type header, and parses that parameter into JSON.

send() automatically sets the Content-Length HTTP response header.

send() also automatically closes the connection.

Use end() to send an empty response .An alternative way to send the response, without any body, it's by using the Response.end()

method:

```
res.end()
```

Sending a JSON response

When you listen for connections on a route in Express, the callback function will be invoked on every network call with a Request object instance and a Response object instance.

Example:

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

Here we used the `Response.send()` method, which accepts any string.

You can send JSON to the client by using `Response.json()`, a useful method.

It accepts an object or array, and converts it to JSON before sending it:

```
res.json({ username: 'Flavio' })
```

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()`.

Send JSON object using res.write

```
const expr=require("express")
const app=expr();
student={name:"xyz",age:31}

app.get("/",(req,res)=>{
  res.set("content-type", "text/html")
  res.write(JSON.stringify(student))
  res.send()
})
app.listen(5000);
```

Send JSON object using res.send

```
const expr=require("express")
const app=expr();
student={name:"xyz",age:31}

app.get("/",(req,res)=>{
```

```

    res.set("content-type","text/html")

    res.send(student)    //res.send automatically converts data in string format
  })
  app.listen(5000);

```

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

```

const expr=require("express")
const app=expr();
  student={name:"xyz",age:31}    //JSON OBJECT
  app.get("/",(req,res)=>{
    res.write( student.name +"\n" + JSON.stringify(student.age))    //student age must be converted to
string type before sending to server

    res.send()
  })
  app.listen(5000)

```

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

```

const expr=require("express")
const app=expr();
student={
  A1:[{name:"xyz",id:1},
    {name:"pqr",id:2},
    {name:"mnc",id:3},
    {name:"abc",id:4}, ]
  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.A1){
      res.write("<tr><td>" + i.name + "</td>")
      res.write("<td>" + JSON.stringify(i.id) + "</td></tr>")
    }
  })
  app.listen(5000)

```



```
}  
  
res.write("</table></center>")  
  
res.send()  
  
}).listen(5000);
```

Example:

Write an express js script to define one JSON array of 3 objects having properties name and age. Sort these objects according to age. If user request [sortednames](#) 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 expr = require("express")  
const app = expr();  
student = [{name:"abc",age:28},  
            {name:"def",age:40},  
            {name:"xyz",age:10}]  
app.get ("/", (req, res) => {  
    res.set ("content-type", "text/html")  
    res.send (student)  
})  
app.get ("/sortednames", (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)
```

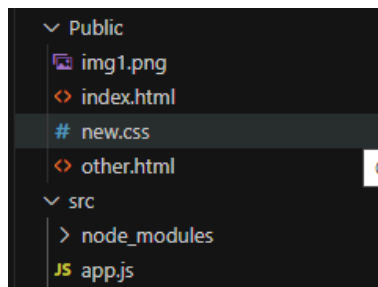
Serving Static Files(HTML,CSS,Javascript,image files)

Express provides a built-in middleware **express.static** to serve static files, such as HTML, images, CSS, JavaScript, etc.

You simply need to pass the name of the directory where you keep your static assets, to the **express.static** middleware to start serving the files directly. For example, if you keep your images, CSS, and JavaScript files in a directory named public, you can do this –

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

We will keep a few images and HTML file in **public** sub-directory as follows –



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

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

app.get('/', function (req, res) {
  res.send();
})

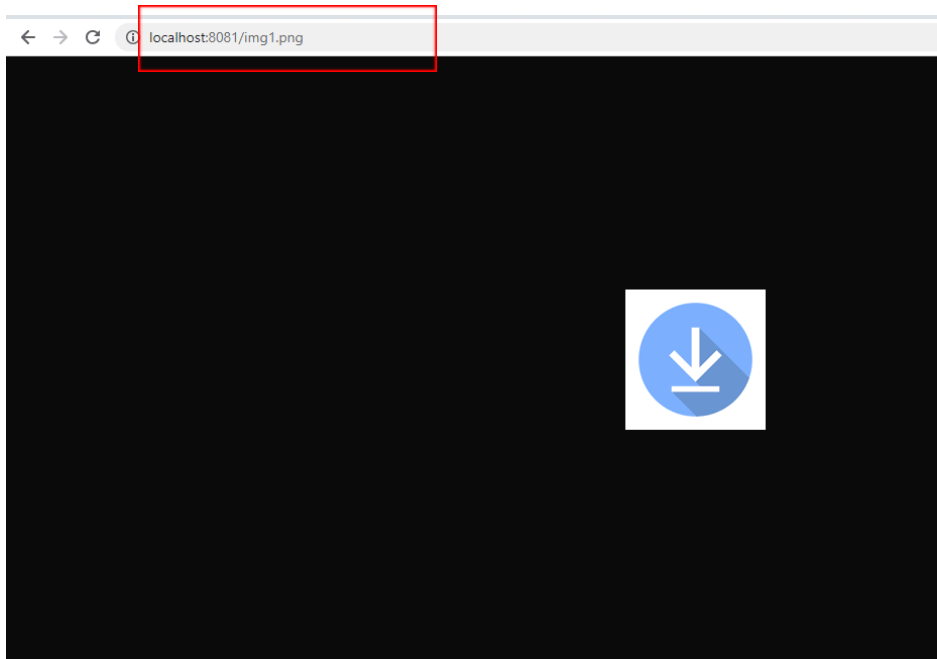
app.listen(8081, function () {

  console.log("server start")
})
```

Save the above code in a file named **app.js** and run it with the following command.

```
$ node app.js
```

Now open localhost:8081/img1.png in any browser and see observe following result.



Create one folder for public and one folder src

Suppose, here we have created folder named **public** to create HTML, CSS file inside it and **src** folder to create JS files inside it. So, we can manage the files easily.

Inside Folder public >> Create one HTML file named **index.html** and CSS file named **express.css**

```
/public/index.html
<html>
  <head>
    <title> Website</title>
    <link href="express.css" rel="stylesheet"/>
  </head>
  <body>

    <form action="get">
      <fieldset>
        <legend>Sign-up</legend>
        Username: <input type="text" name="username"/>
      </br></br>
      password: <input type="text" name="password"/>
      <br></br>
      <textarea rows="5" cols="10" name="ta">

      </textarea>
      <br></br>
      <input type="submit"/>
    </fieldset>
  </form>
</body></html>
```

/public/express.css

```
h1{
  font-style:italic;
  text-align:center;
  text-decoration:line-through;
  color:white;
  background-color: red;
}
p{
  border: 1px solid black;
  text-align: justify;
  text-transform: capitalize;
  color: blue;
}
```

/src/app.js

```
const expr=require("express");
const path=require("path");
const app=expr();
const staticPath=path.join(__dirname,"../Public");
console.log(staticPath);
app.use(expr.static(staticPath));
app.listen(5004,()=>
{
  console.log("server running");
});
```

Create **one.html** file and **app.js** file. If Both this files are in same folder then **res.sendFile()** function is used to display one.html on browser.

One.html

```
<html>
  <head>
    <title> Website</title>
  </head>
  <body>

    <form method="get">
      <fieldset>
        <legend>Sign-up</legend>
        Username: <input type="text" name="uname"/>
      </br></br>
        password: <input type="text" name="pwd"/>
      <br></br>

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

App.js

```
const expr=require("express");
const path=require("path");
const app=expr();
const staticPath=path.join(__dirname);
console.log(staticPath);
app.use(expr.static(staticPath));
app.get("/",(req,res)=>
{
    res.sendFile("/one.html");
})
app.listen(5004,()=>
{
    console.log("server running");
});
```

HTTP methods

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

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

Solution: create two file in same folder (index.html and app.js)

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>
```

app.js

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.sendFile(__dirname + "/index.html" );
})
app.get('/process_get', function (req, res) {
  response = {
    first_name:req.query.first_name,
    last_name:req.query.last_name
  };
  res.send(response);
})
app.listen(8000)
```

OR

Solution: create two file in different folder (**public/index.html** and **src/app.js**)

public/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>
```

Src/app.js

```
const expr=require("express");
const path=require("path");
const app=expr();
const staticpath=path.join(__dirname,"../public");
console.log(staticpath);
app.use(expr.static(staticpath));
app.get("/process_get",(req,res)=>
{
  const response=
  {
    fname:req.query.firstname,
    lname:req.query.lastname
  };
  console.log(response);
  res.end(JSON.stringify(response));
})
```

```
});  
  
app.listen(5000,()=>  
{  
    console.log("server starte");  
});
```

To run : > node app.js

On browser: localhost:5000/

Above code will automatically open index.html file on browser

If you want to open html file with other name like second.html then write below code.

Here **second.html** file is in **public** folder and **app.js** is in **src** folder

app.js

```
const expr=require("express");  
const path=require("path");  
const app=expr();  
const staticpath=path.join(__dirname,"../public");  
console.log(staticpath);  
app.use(expr.static(staticpath,{index:"second.html"}));  
app.get("/process_get",(req,res)=>  
{  
    const response=  
    {  
        fname:req.query.firstname,// req.query will give querystring from URL  
        lname:req.query.lastname  
    };  
    console.log(response);  
    res.end(JSON.stringify(response));  
});  
  
app.listen(5000,()=>  
{  
    console.log("server starte");  
});
```

Program:

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.

src/app.js

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

```

const staticpath=path.join(__dirname);
console.log(staticpath);
app.use(express.static(staticpath,{index:"third.html"}));
app.get("/process_get",(req,res)=>
{
    res.set("content-type","text/html");
    data=req.query.textarea1
    console.log(data);
    a=data.split(',');
    for(x in a)
        {res.write(a[x]+"<br>");
        }
    res.end();
});
app.listen(2501,()=>
{
    console.log("server starte");
});

```

src/third.html

```

<html>
  <head>

  </head>
  <body>
    <form action="/process_get" method="get">
      <textarea row="5" cols="15" name="textarea1"> this is an textarea tag,
we are doing express js,this is demo</textarea>
      <input type="submit"/>
    </form>

  </body>
</html>

```

Post Method

The most common way to send diverse and large amounts of data via HTTP is to use the POST method.

Before we can easily access this data on the server side in Express, we need to use some middleware, like the `body-parser` package

`body-parser` package, to parse the data in to a format that we can easily access.

Express has a module `body-parser` that includes a parser for URL-encoded request bodies, like the ones submitted by HTML forms.

Example:


```

const expr=require("express");
const app=expr();
const bp=require("body-parser");
const url=bp.urlencoded({extended:true});
const staticpath=__dirname;
app.use(expr.static(staticpath));
app.post("/process_post",url,(req,res)=>
{
    res.write(`<h1 style="color:red">username:${req.body.firstname}</h1>`)
    res.write(`<h1 style="color:yellow">username:${req.body.lastname}</h1>
    <h1>gender=${req.body.a1}</h1>`);

    res.end();

}
);
app.listen(5000,()=>
{
    console.log("server start");
});

```

index.html

```

<html>
  <head>
    <link href="new.css" rel="stylesheet"/>
  </head>
  <body>
    <h1>this is a demo</h1>
    <form action="/process_post" method="post">
      <input type="text" name="firstname"/>
      <input type="text" name="lastname"/>

      <input type="radio" name="a1" value="male"/>MAle
      <input type="radio" name="a1" value="female"/>FEMAlE
      <input type="submit"/>
    </form>
  </body>
</html>

```

New.css

```

h1
{
    text-align:center;
    color:red;
    text-transform:uppercase;
}

```

```
background-color:black;
}
```

Program:

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 message". 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 post method to request data.

App.js

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

const bp=require("body-parser");
const url=bp.urlencoded({extended:true});
const staticpath = p.join(__dirname,"../public");
app.use(expr.static(staticpath,{index:'calculate.html'}));

app.post("/calc",url,(req,res)=>{
  res.set("content-type","text/html");
  var n1 = parseInt(req.body.n1);
  var n2 = parseInt(req.body.n2);
  if ((n1>0) && (n2>0)) {
    if(req.body.formula == "addition"){
      a = n1+n2;
      res.write("<h1>Addition is : " + a + "</h1>");
    }
    else if(req.body.formula == "subtraction"){
      s = n1-n2;
      res.write("<h1>Subtraction is : " + s + "</h1>");
    }
    else if(req.body.formula == "multi"){
      m = n1*n2;
      res.write("<h1>Multiplication is : " + m + "</h1>");
    }
    else if(req.body.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(5000);

```

calculate.html

```

<html>
<body>
    <form action="/calc" method="post">
        <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>

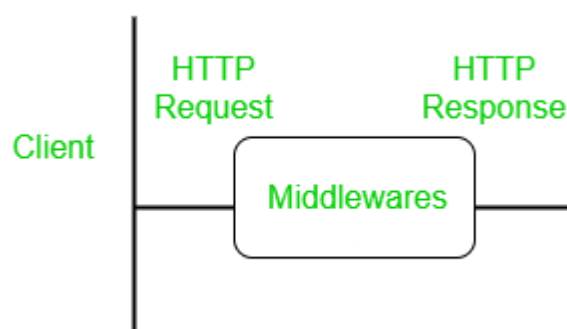
            <input type="submit" value="Submit">
        </fieldset>
    </form>
</body>
</html>

```

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.



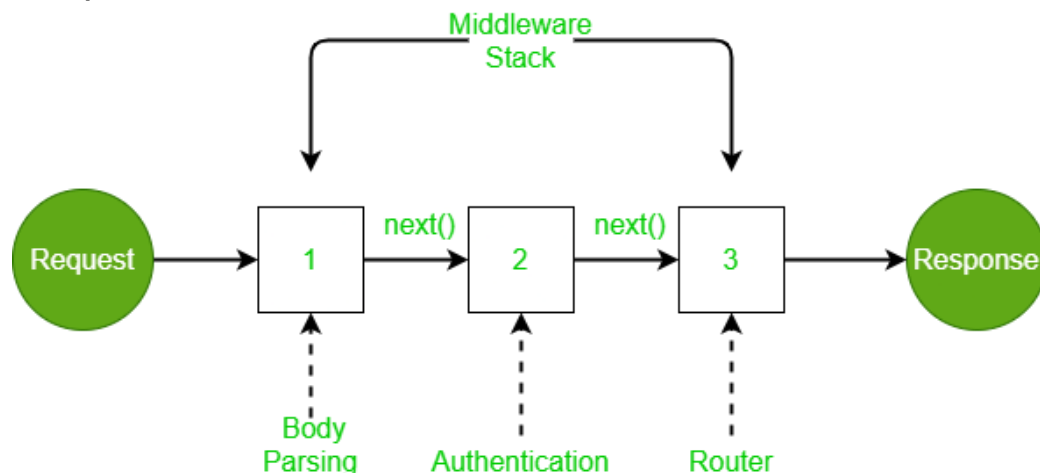
Advantages of using middleware:

- 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

Modified requests will be available to each middleware via the next function -



Middleware Syntax:

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

Middleware functions take 3 arguments: the **request object**, the **response object**, and the **next function** in the application's request-response cycle, i.e., two objects and one function.

Example:

```
const expr=require("express");
const app=expr();
const path=__dirname;
app.use((req,res,next)=>
```

```

{
  console.log("initialize");
  res.write("hello1");
  next();
});
app.get("/",(req,res)=>
{
  res.write("hello2");
  res.send();
});
app.listen(5001,()=>
{
  console.log("server start");
});

```

Example:

```

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

app.get("/hello",(req,res,next)=>
{
  res.write("request received on"+ new Date());
  next();
});
app.get("/hello",(req,res,next)=>
{
  console.log("hello1");
  next();
});
app.get("/hello",(req,res)=>
{
  res.send();
}).listen(5000);

```

Program:

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. 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).

src/app.js

```
var expr = require("express");
var app = expr();
var p = require("path");
const staticp = p.join(__dirname, "../public");
app.use(expr.static(staticp, {index: 'home.html'}));

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

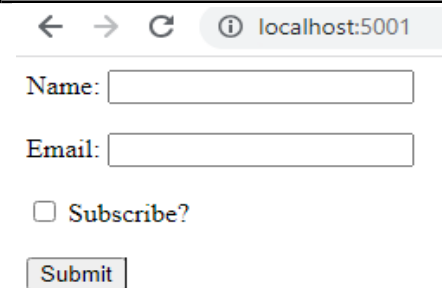
/public/home.html

```
<form action="/login" method="get">
  <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>
```



← → ↻ ⓘ localhost:5001

Name:

Email:

☐ Subscribe?

Welcome lju

Your email id is lju@gmail.com

Thank you for your subscription

[Logout](#)

Welcome lju

Your email id is lju@gmail.com

You can subscribe to get daily updates

[Subscribe](#)

Program(QB-120)

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 username is 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"/>
    </br></br>
    password: <input type="text" name="pwd"/>
    <br></br>
    <input type="submit"/>
</form>
</body>
</html>
```

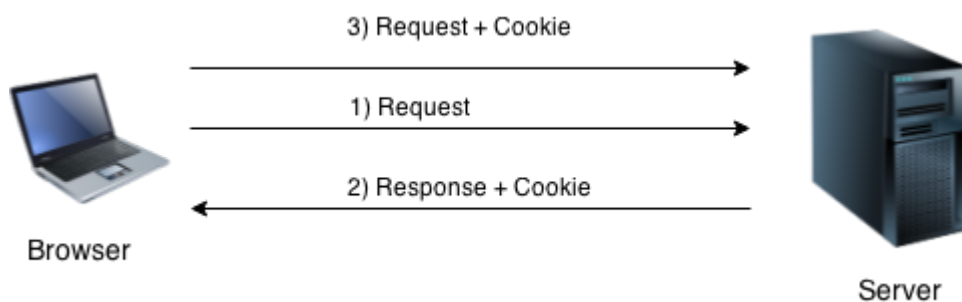
App.js

```
const expr=require("express");
const app=expr();
const bp=require("body-parser");
const url=bp.urlencoded(
{
    extended:false
});
console.log(url);
const staticPath=__dirname;
app.use(expr.static(staticPath));
app.post("/check",url,(req,res,next)=>
{
    if(req.body.uname=="admin")
    {
        next();
    }

else{
    res.send("<h1 style='color:red'>wrong credentials</h1>")
}
});
app.post("/check",url,(req,res,next)=>
{
    res.write(`<h1>welcome...${ req.body.uname}</h1>`)
res.send();
}
).listen(3001);
```

Cookies

Cookies are simple files that are stored on user's computer. It stores the data in a text file. This helps us to keep track of the user action. Cookies can be accessed either by the web server or client computer. This allows the server to deliver a page to a particular user, or the page itself can be stored some script which is aware of the data in the cookie and so is able to carry information from one visit to the website or related site to the next.



What's in a cookie?

Each cookie is finally small lookup table involves pairs of key-values - for example (first name, Steve) (last name, jobs). Once the cookie has been read by the code on server or client computer, the data can be restored and used to customize the web page accordingly.

Why are Cookies Used?

Cookies are an adequate way to carry data between sessions on a website, without loading a server machine with huge amounts of data storage. If we store data on a server without cookies it would be insecure and inconvenient because then it will get difficult to restore users data without requiring a login on each visit to the website.

Installation:-

To use cookies with express, firstly we have to install cookie-parser middleware. To install it, use the following command-

```
npm install cookie-parser
```

Now to use cookies with express, we will have to require the cookie-parser module. Cookie-parser is a middleware which parses cookies to attach with the client request object.

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

Adding cookies

To use a new cookie, first define a new route in your express.js app like:-

```
var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');
app.use(cookieParser());
app.get('/', function(req, res) {
  res.cookie('mycookies', 'express')
  res.send('cookie set');
});
app.listen(3000, function(err, message) {
  console.log("server start.....")
});
```

Browser always sends back cookies every time it queries the server.

Display cookie

To view cookies from your server `req.cookies` function is used. add the following console to that route.

```
console.log('cookies:' , req.cookies)
```

How Long Does a Cookie Store?

The time of expiry of a cookie can be set when the cookie is created. If time is not set by the user then by default, the cookie is destroyed when the current browser window is closed.

Adding Cookies with Expiration Time:-

You can add cookies that expire. Just pass an object with property `'expire'` and set to the time when you want it to expire from your browser. Following is an example of this method.

```
//Expires after 360000 ms from the time it is set.  
res.cookie(name, 'value', {expires:new Date(Date.now()+1000)});
```

The other way to set the expiration time is using `'maxAge'` property. Following is an example of this method.

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

Delete existing cookies:-

To delete a cookie, use the `clearCookie` function.

```
res.clearCookie('mycookies');
```

Example:

```
const expr=require("express");  
const app=expr();  
const cp=require("cookie-parser");  
app.use(cp());  
app.get("/",(req,res,next)=>  
{  
  res.cookie("fname","xyz");  
  res.cookie("lname","pqr");  
  next();  
})
```

```

});
app.get("/",(req,res,next)=>
{
    console.log(req.cookies);
    next();
});
app.get("/",(req,res)=>
{
    res.clearCookie("fname");
    res.send("cookie deleted");
});
app.listen(5001,()=>>
{
    console.log("listen on 5001");
}
);

```

Example:

```

const expr=require("express");
const app=expr();
const cp=require("cookie-parser");
app.use(cp());

app.get("/",(req,res,next)=>
{
    res.cookie("uname","xyz",{
        expires:new Date(Date.now()+20000)
    });
    res.cookie("lname","xyz1",{
        expires:new Date(Date.now()+10000)
    });
    res.cookie("abc","pqr");
    next();
});
app.get("/",(req,res,next)=>
{
    console.log(req.cookies.uname);
    next();
    //res.send()
});
app.get("/",(req,res)=>
{
    res.clearCookie("uname");
    res.send("Cookie deleted");
});
app.listen(8001,()=>>
{
    console.log("server running at 8001");
});

```

Program:

write a script to create a login form on **index.html** file. After clicking submit button it should jump to **/next** page and value of username should be stored inside cookie. perform the task using **get method**. observe and check inside browser that the cookie is stored perfectly or not.

index.html

```
<html>
  <body>

    <form action="/next" method="get">
      <fieldset>
        <legend>Sign-up</legend>
        Username: <input type="text" name="uname"/>
      </br>
      password: <input type="text" name="pwd"/>
      <br>

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

App.js

```
var expr=require("express")
var cp=require("cookie-parser")
var app=expr()
app.use(cp())
app.get("/",(req,res)=>{
  res.sendFile(__dirname+"/index.html")}))
app.get("/next",(req,res,next)=>{
  const response={
    u:req.query.uname,
    p:req.query.pwd
  }
  res.cookie("uname",response.u)
  next()
})
app.get("/next",(req,res)=>{
  console.log(req.cookies)
  res.send(req.cookies)
})
app.listen(3000)
```

Program:

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.

Make simple.html file and app.js file use get method in express js.

Simple.html

```
<html>
<head>
  <title>User Feedback Form</title>
</head>
<body>
  <h1>User Feedback Form</h1>
  <form action="/submit-feedback" method="get">
    <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>

App.js

```
const express = require('express');
const cp = require('cookie-parser');
const app = express();
app.use(cp());
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/simple.html');
});
app.get('/submit-feedback', (req, res) => {
  const { name, email, message, rating } = req.query;

  // 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.');
```

← → ↻ ⓘ localhost:5000

User Feedback Form

Name:

Email:

Message:

Rating: ☐ Bad ☐ Average ☐ Good ☐ Very Good ☐ Excellent

← → ↻ ⓘ localhost:5000/submit-feedback?nam

Thank you for your feedback!
[Show Feedback](#)

← → ↻ ⓘ localhost:5000/feedback-details

Feedback Details

Name: L J university
Email: xyz@lju.com
Message: very nice
Rating: Very Good
[logout](#)

Session

A website is based on the HTTP protocol. HTTP is a stateless protocol which means at the end of every request and response cycle, the client and the server forget about each other.

This is where the session comes in. A session will contain some unique data about that client to allow the server to keep track of the user's state.

In session-based authentication, the user's state is stored in the server's memory or a database.

How sessions works

When the client makes a login request to the server, the server will create a session and store it on the server-side.

When the server responds to the client, it sends a cookie. This cookie will contain the session's unique id stored on the server, which will now be stored on the client.

This cookie will be sent on every request to the server.

Why Use Sessions?

Sessions are used to store information such as User ID over the server more securely, where it cannot be altered. This prevents the information from being tampered with.

In addition to this, sessions can transfer the information from one web page to another in the form of value.

The difference between session and cookie

Basis of Comparison	Cookie	Session
Definition	Cookies are client-side files that are stored on a local computer and contain user information.	Sessions are server-side files that store user information.
Expiry	Cookies expire after the user specified lifetime.	The session ends when the user closes the browser or logs out of the program.
Data storage	It can only store a limited amount of data.	It is able to store an unlimited amount of information.
Capacity	Cookies can only store up to a maximum of 4 KB of data in a browser.	There is a maximum memory restriction of 128 megabytes that a script may consume at one time. However, we are free to maintain as much data as we like within a session.
Data Format	Cookies are used to store information in a text file.	The data is saved in an encrypted format during sessions.

Installation

Installation is done using the `npm install` [command](#):

```
npm install express-session
```

require express-session module

```
var session = require('express-session')
```

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

```
app.use(session({
  {
    resave:false,// Do not save session if not modify
    saveUninitialized:false,// Do not create session if something is not
stored
    secret:"Hello"
  }
})
```



```
));
```

resave: It basically means that for every request to the server, it reset the session cookie. Even if the request was from the same user or browser and the session was never modified during the request.

saveUninitialized: When an empty session object is created and no properties are set, it is the uninitialized state. So, setting saveUninitialized to false will not save the session if it is not modified.

Secret: is a random unique string key used to authenticate a session. secret parameter allows express-session to use it to encrypt the session-Id

Example:

```
const expr=require("express");
const app=expr();
const sess=require("express-session");
app.use(sess(
  {
    resave:true,
    saveUninitialized:true,
    secret:"Hello"
  }
));
app.get("/",(req,res)=>
{
  if(!req.session.fname)
  {   req.session.fname="hiral"
      res.redirect("/fetchsession")
  }
  else
  {
    console.log("Session is already set...");
  }
})
app.get("/fetchsession",(req,res)=>
{
  res.write(`<h1> Welcome... ${req.session.fname} <h1>`);
  res.write(`<button><a href="deletesession">Delete session</a></button>`);
  res.send();
})
app.get("/deletesession",(req,res)=>
{
  req.session.destroy();
  res.send("Session destroy")
})
app.listen(8004,()=>=)
```

```
{  
  console.log("server running at 8004");  
});
```

Program(QB-121): write a script to meet following requirements.

- 1) create index.html page and open it on localhost
- 2) after clicking submit button, it should jump to **savesession** page. store username in session.
- 3) After saving session, redirect to **fetchsession** page and read session value. put a logout link button here.
- 4) destroy the session on this page and redirect to index.html

index.html

```
<html>  
  <head>  
    <title> Website</title>  
  </head>  
  <body>  
  
    <form action="/savesession" method="get">  
      <fieldset>  
        <legend>Sign-up</legend>  
        Username: <input type="text" name="uname"/>  
      </br>  
        password: <input type="text" name="pwd"/>  
      <br>  
  
      <input type="submit"/>  
    </fieldset>  
  </form>  
</body>  
</html>
```

App.js

```
const expr=require("express");  
const app=expr();  
const sess=require("express-session");  
app.use(expr.static(__dirname,{index:"sessiontask.html"}))  
app.use(sess(  
  {  
    resave:true,  
    saveUninitialized:true,  
    secret:"Heloo"  
  }  
));
```

```

app.get("/save-session", (req, res) =>
{
    if(!req.session.u)
    {
        req.session.u = req.query.username;
        res.redirect("/fetch-session");
    }
    else
    {
        console.log("Session is already set...");
    }
})
app.get("/fetch-session", (req, res) =>
{
    res.write(`<h1 style="color:blue;border:5px solid black;font-style:italic">
Welcome... ${req.session.u} <h1>`);
    res.write(`<button><a href="/delete-session">Logout</a></button>`);
    res.send();
})
app.get("/delete-session", (req, res) =>
{
    req.session.destroy();
    res.redirect("/")
})
app.listen(8004, () =>
{
    console.log("server running at 8001");
});

```

Example :

write express script to print how much time user visit the page. For ex., if user visit first time , "you have visited page First time" message will print. if user visit second time , "you have visited page second time" message will print. and so on.

```

const expr = require("express");
const app = expr();
const sess = require("express-session");
app.use(sess(
{
    resave: true,
    saveUninitialized: true,
    secret: "Hello1"
}
));

app.get("/", (req, res) =>
{
    if(req.session.page_views)
    {

```

```
        req.session.page_views++;
        res.send(`<h1 style="color:blue;border:5px solid black;font-style:italic">
You have visited page ${req.session.page_views} times <h1>`);
    }
    else{
        req.session.page_views=1,
        res.send(`<h1 style="color:green;border:5px solid
black;font-style:italic"> Welcome.. You have visited page
${req.session.page_views} times<h1>`);
    }
});

app.listen(8001,()=>
{
    console.log("server running at 8001");
});
```

Programs:

Write express script to get form data using post method and display First name in Red color, Password in Green Color and Gender in Black color.

trial.html (file in public folder)

```
<html>

<body>

<form method="post" action="/process-post">

<fieldset>

<legend style="color:blue;text-align:center" >Log In </legend>

</br>

<lable> User Name</lable>

<input type="text" name="firstname"/>

</br>

<lable> Password</lable>

<input type="password" name="password"/>

</br>

<lable> Gender</lable>

<input type="radio" name="r1" value="Male">Male</input>

<input type="radio" name="r1" value="Female">Female</input>

</br>

<lable> Comment</lable>

<textarea rows="5" cols="15" name="comment"> </textarea>

</br>

<input type="Submit">

<input type="Reset">

</fieldset>

</form>

</body>

</html>
```

trial.js (file in src folder)

```

const expr=require("express")
const path=require("path");
const app=expr();
const body=require("body-parser")
const url=body.urlencoded({extended:false})
const staticpath=path.join(__dirname,"../public");
console.log(staticpath)
app.use(expr.static(staticpath,{index:"trial.html"}))
app.post("/process-post",url,(req,res)=>
{
res.write(`<h1 style="color:red">Firstname=${req.body.firstname}</h1>
<h1 style="color:green">Password=${req.body.password}</h1>
<h1>Gender=${req.body.r1}</h1>`)
res.end()
})
app.listen(5000)
console.log("Server is running")

```

Output:

After filling form on localhost:5000 with name:"abc", password: "abcd", gender: Male

Output on: http://localhost:5000/process-post

Firstname=abc

Password=abcd

Gender=Male

2. Write express JS script to load in html file having username, password and submit button. On clicking of submit button, it should jump on check page using post method. If username is "admin" then jump on next middleware to print "welcome admin". For any other username it should stay on same middleware to print warning message in red color.

two.html (file in src folder)

```

<html>
<head>
<title> Website</title>
<link href="new.css" rel="stylesheet"/>
</head>
<body>
<form action="/check" method="post">
<fieldset>
<legend>Sign-up</legend>
Username: <input type="text" name="uname"/>
</br></br>
password: <input type="text" name="pwd"/>
<br></br>
<input type="submit"/>
</fieldset>
</form>
</body>
</html>

```

task.js (file in src folder)

```

const expr=require("express");
const app=expr();
const bp=require("body-parser");
const url=bp.urlencoded(
{
extended:false
});
const staticPath=__dirname;
app.use(expr.static(staticPath,{index:"two.html"}));
app.post("/check",url,(req,res,next)=>
{

```

```

const u1=req.body.username
if(u1=="admin")
{
next();
}
else
{
res.write(`<h1 style="color:red"> You are not admin </h1>`);
res.send();
} });
app.post("/check",url,(req,res)=>
{
res.write("<h1>welcome admin</h1>");
res.send() });
app.listen(7002)

```

3. Write express js script to perform tasks as asked below. (Get Method)

- 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 message”. 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 expr = 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, next) => {
  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>
```

4. Write express js script to perform following tasks. (Middleware)

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.

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. 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 expr = require("express");
var app = expr();
```

```

var p = require("path");
const staticp = p.join(__dirname, "../frontend");
app.use(express.static(staticp,{index:'t1.html'}));
app.get("/login",(req,res,next)=>{
  console.log(req.query);
  res.set("content-type","text/html");
  res.write("<center><h1>Welcome " + req.query.name + "</h1>");
  res.write("<center><h2>Your email id is " + req.query.email + "</h2>");
  next();
})
app.get("/login",(req,res,next)=>{
  if(req.query.newsletter && req.query.newsletter == "on"){
    res.write("<h3>Thank you for your subcsription</h3><a href='/'>Logout</a>");
  }else{
    res.write("<h3>You can subscribe to get daily updates</h3><a href='/subscribe'>Subscribe</a></center>");
  }
  next();
});
app.get("/subscribe",(req,res)=>{
  res.set("content-type","text/html");
  res.write("<h3>Thank you for your subcsription</h3></center><a href='/'>Logout</a>");
  res.send();
});
app.listen(5001);

```

/frontend/t1.html

```

<form action="/login" method="get">
<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>
<label for="message">Message:</label><br>
<textarea name="message" id="message" cols="30" rows="4"></textarea><br><br>
<input type="checkbox" name="newsletter" id="newsletter">
<label for="newsletter">Subscribe?</label><br><br>
<input type="submit" value="Submit">
</form>

```

Task: Write express js script to perform the tasks as asked below. (Post method)

- 1) Create one HTML file named ljform.html 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”.**

In 3.js file in src folder

```

var expr = require("express");
var app = expr();
var bp = require("body-parser");
var p = require("path");
app.use(bp.urlencoded());
const staticp = p.join(__dirname, "../public");
app.use(expr.static(staticp, {index: 'ljform.html'}));
app.post("/login", (req, res) => {
  if (req.body.uname == 'admin') {
    res.write(`<h1 style="color:green">Hey ${req.body.uname}, Welcome!</h1><br>`);
  }
  else {
    res.write(`<h1 style="color:red">Please login with Admin name</h1>`);
  }
}

```

```
res.send();  
})  
app.listen(5222);
```

In 1.css file in public folder

```
.click{  
background-color: black;  
color: white  
}
```

In ljform.html file in public folder

```
<html>  
<head>  
<link rel="stylesheet" href="1.css">  
</head>  
<body>  
<form method="post" action="/login">  
<label>User name:</label>  
<input type = "text" name="uname" ></input>  
<label>Password:</label>  
<input type = "text" name="password" ></input>  
<button name="Submit" value="Submit" class="click">Submit</button>  
</form>  
</body>  
</html>
```

Task : write a script to meet following requirements (Authentication)

1)create index.html file page and open it on localhost.

2)after clicking submit button, it should jump on “savesession” page. Store username in session.

3)After saving session, redirect to “fetchsession” page and read value. Put a LOGOUT link

button here.

4)Jump on delete session page after clicking LOGOUT button.

5)Destroy the session on this page and redirect to index.html page.

session.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="Submit">

</form>

</center>

</body>

</html>
```

Session.js

```
var expr = require("express");

var app = expr();

var es = require("express-session");

app.use(es({

  resave:false,

  saveUninitialized:false,

  secret:"lju1"

}));


app.get('/',(req,res)=>{

  res.sendFile(__dirname + "/redirect.html");

});


app.get("/savesession",(req,res)=>{

  var uname=req.query.uname;
```

```

var pass=req.query.pass;
req.session.uname = uname;
req.session.password = pass;
res.redirect("fetchsession")
})
app.get("/fetchsession",(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='/deletesession'>Delete Session</a>")
}else{
res.write("<h1 style='color:red;'>Please enter valid username or password</h1><a href='/'>Login</a>")
}
res.send();
});
app.get("/deletesession",(req,res)=>{
req.session.destroy();
res.write("<h1>Session Destroyed</h1><a href='/'>Login</a>")
res.send();
});
app.listen(6177);

```

Task: 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. Make simple.html file and app.js file use get method in express js.

In cookietask2.html file:

```
<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>
```

In task_cookie2.js file:

```
const express = require('express');
const cp = require('cookie-parser');
const bp=require('body-parser')
const app = express();
app.use(cp());
app.use(bp.urlencoded({ extended: true }));
app.get('/', (req, res) => {
res.sendFile(__dirname + '/cookietask2.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.');
```

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

```

const expr=require("express");
const app=expr();
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).send("page not found");  
})  
app.listen(8081,()=>  
{  
  console.log("server started");  
})
```

Unit-5 Express.js

Templates -View Engine(pug)

A **template engine** enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. This approach makes it easier to design an HTML page.

Some popular template engines that work with Express are [Pug](#), [Mustache](#), and [EJS](#)

Pug is a very powerful templating engine which has a variety of features including **filters, includes, inheritance, interpolation**.

To use Pug with Express, we need to install it,

```
npm install pug
```

Now that Pug is installed, set it as the templating engine for your app. You **don't** need to 'require' it. Add the following code to your **.js** file.

```
app.set('view engine', 'pug');  
app.set('views', '_dirname');
```

Now create a file called **first_view.pug**, and enter the following data in it.

```
doctype html  
html  
  head  
    title = "Hello Pug"  
  body  
    p people Hello World!
```

To run this page, add the following route to your app –

```
app.get('/', function(req, res){  
  res.render('first_view'); // this will display your pug file on browser  
});
```

Attributes

To define attributes, we use a comma separated list of attributes, in parenthesis. Class and ID attributes have special representations. The following line of code covers defining attributes, classes and id for a given html tag.

```
div.container.column.main#division(width = "100", height = "100")
```

This line of code, gets converted to the following. –

```
<div class = "container column main" id = "division" width = "100" height =  
"100"></div>
```

Unit-5 Express.js

Example:

Simple example to understand the concept of pug

```
html
head
title PUG example
body
  h1 LJU
  p lets learn pug
  i Thank you

  // apply style or attribute
  h5(style="text-transform:uppercase;color:pink",align="center") hello world

  //to write multiple line in same tag. but it will print in one line
  p hello world
  |The file will not
  |render properly if the
  |programmer does not make
  |sure of proper indentation

  // create list
  ul
    li Mango
    li Watermelon
    li Pineapple
```

Unit-5 Express.js

```
// to create table

table(border='1px red solid')

  tr

    th name

    th id

  tr

    td abc

    td 1
```

Passing Values to Templates

When we render a Pug template, we can actually pass it a value from our route handler, which we can then use in our template. Create a new route handler with the following.

```
var express = require('express');
var app = express();
app.set("view engine", "pug")
app.set("views", __dirname)
app.get('/', function(req, res){
  res.render('dynamic', { name: "learning", url: "http://www.gmail.com" });
});

app.listen(3000);
```

And create a new file in same directory, called **dynamic.pug**, with the following code –

```
html
  head
    title=name
  body
    h1 #{name} //fetch value of name from js file
    a(href = url) URL // fetch value of url from js file
```

Include

Pug provides a very intuitive way to create components for a web page. For example, if you see a news website, the header with logo and categories is always fixed. Instead of copying that to every view we create, we can use the **include** feature. Following example shows how we can use this feature Create 3 views with the following code –

Unit-5 Express.js

HEADER.PUG

```
div.header.  
  I'm the header for this website.
```

CONTENT.PUG

```
html  
  head  
    title Simple template  
  body  
    include ./header.pug  
    h3 I'm the main content  
    include ./footer.pug
```

FOOTER.PUG

```
div.footer.  
  I'm the footer for this website.
```

Create a route for this as follows –

App.js

```
var express = require('express');  
var app = express();  
app.set("view engine","pug")  
app.set("views",__dirname)  
app.get('/', function(req, res){  
  res.render('content'); // here content is pug file name  
});  
  
app.listen(3000);
```

External Stylesheet

To apply external stylesheet , first you have to create style.css file ,style.pug and style.js file Then use include command to link external stylesheet in your pug file.

Style.css

```
#c1  
  
{ color:red;  
  
}
```


Unit-5 Express.js

Style.pug

```
Html
  Head
    Style
      Include ./style.css
  Body
    H1#c1 Hello world
```

Style.js

```
var express = require('express');
var app = express();
app.set("view engine","pug")
app.set("views",__dirname)
app.get('/', function(req, res){
  res.render('style'); // it will render style.pug file
});

app.listen(3000);
```

Example

Main.pug

```
html
head
title hello world
body
h1(align="center") pug example
p(style="color:red;font-size:50px") lets lern example
i thanks
```

main.js

```
const expr=require("express");
const app=expr();
app.set("view engine","pug");
app.get("/",(req,res)=>
{
  res.render(__dirname+"/main");
});

app.listen(3000,()=>>
```

Unit-5 Express.js

```
{  
  console.log("server start");  
});
```

Program 1: Create one pug file which contains a text field and a password field. By submitting the form, on next page called “/login” submitted data will be displayed

Form.pug

```
html  
  head  
    title Login Form  
  body  
    h1 Login  
    form(action="/login" method="post")  
      label Username:  
      input(type="text" id="username" name="username" required)  
      br  
      label Password:  
      input(type="password" id="password" name="password" required)  
      br  
      input(type="submit" value="Login")
```

display.pug

```
html  
  head  
    title Login Data  
  body  
    h1 Welcome, #{username}!  
    p Your username is: #{username}  
    p Your password is: #{password}
```

pug.js

```
const express = require('express');  
const app = express();  
var bp=require("body-parser");  
var url=bp.urlencoded();
```

Unit-5 Express.js

```
app.set('view engine', 'pug');
//app.set('views', __dirname);
app.get('/', (req, res) => {
  res.render(__dirname + '/form');});
app.post('/login',url, (req, res) => {
  const { username, password } = req.body;
  res.render(__dirname + '/display', { username: username, password:
password });
});
app.listen(3000)
```

Program-2 Write express JS script to pass data like message, name and id from express application to pug template in h1, h2 and h3 tags respectively and display data in browser.

Pug.js

```
const express = require('express');
const path = require('path');
const app = express();
app.set('view engine', 'pug');
app.set('views', __dirname);
app.get('/', (req, res) => {
  res.render('example', {message: 'Hello from Express!',name: 'lju',id: 2 });
});
app.listen(6002)
```

example.pug

```
html
head
title My Express App
body
h1 #{message}
h2 #{name}
h3(style="color:red") #{id}
```

Example:3

Write express js script to load student form using pug file which contains following fields Name(text),Email(email),Course(radio : CE, IT, CSE).Once form submitted then data must be displayed on '/student' page using pug file. Means data should be submitted from express application to PUG file.

Unit-5 Express.js

Public/student.pug

```
html
head
title Pug Form
body
  h2 Student Form
  form(action="/data",method="get")
    div
      label Enter Your Name
      input(type="text", name="name")
    div
      label Enter Your Email
      input(type="email", name="email")
    div
      label Course
      input(type="radio", name="course", value="IT",id="IT")
      | IT
      input(type="radio", name="course", value="CE",id="CE")
      | CE
      input(type="radio", name="course", value="CSE",id="CSE")
      | CSE
    br
    input(type="submit",value="Submit")
```

src/student_form.js

```
var expr = require("express");
var app = expr();
app.set("view engine","pug");
var p = require("path");
const staticpath = p.join(__dirname,"../public");
app.get("/",(req,res)=>{
  res.render(staticpath+"/student");
});
app.get("/student",(req,res)=>{
  res.render(staticpath+"/form_output",{name:req.query.name, email:req.query.email,
  course:req.query.course});
});
app.listen(5000);
```

public/form_output.pug

Unit-5 Express.js

```
html
head
title FORM Data
body
h1 Welcome!
table(border='1px solid',style='border-collapse:collapse;color:blue', cellpadding=10)
tr
th Name
th Email
th Course
tr
td #{name}
td #{email}
td #{course}
```

File Upload

File upload is a common operation for any applications. In Node.js, with the Express web framework and the **Multer** library, adding file upload feature to your app is very easy.

What is Multer?

Multer is a node.js middleware for handling **multipart/form-data**, which is primarily used for uploading files.

What is Multipart Data?

In general, when a “form” is submitted, browsers use “*application-xx-www-form-urlencoded*” content-type. This type contains only a list of keys and values and therefore are not capable of uploading files. Whereas, when you configure your form to use “***multipart/form-data***” content-type, browsers will create a “multipart” message where **each part will contain a field of the form**. A multipart message will consist of text input and file input. This way using multipart/form-data you can upload files.

Multer adds a body object and a file or files object to the request object. The body object contains the values of the text fields of the form, the file or files object contains the files uploaded via the form.

Multer will not process any form which is not multipart (*multipart/form-data*).

Unit-5 Express.js

Adding Multer

Before using multer, we have to install it using npm.

```
npm install multer
```

We will now load **multer** in the **app.js** file using the **require()** method.

```
const multer = require('multer');
```

DiskStorage

The disk storage engine gives you full control over storing files to disk. We will create a storage object using the **diskStorage** () method.

The following code will go in **app.js** :

```
var storage = multer.diskStorage({
  destination: 'uploads',
  filename: function (req, file, cb) {
    cb(null , file.originalname);
  }
});
```

destination – destination is used to give folder name where file will upload. If no destination is given, the operating system's default directory for temporary files is used.

filename - It is used to determine what the file should be named inside the folder. If you don't provide any filename, each file will be given a random name without any file extension. (here, cb is callback function). The 2 arguments to cb are:

- **null** - as we don't want to show any error.
- **file.originalname** - here, we have used the same name of the file as they were uploaded. You can use any name of your choice.

Other Options in Upload

1. **limits** - You can also put a limit on the size of the file that is being uploaded with the help of using **limits**.

The following code will go inside the **multer()** .

Unit-5 Express.js

```
// inside multer({}), file upto only 1MB can be uploaded
const upload = multer({
  storage: storage,
  limits : {fileSize : 1000000}
});
```

Here, fileSize is in bytes. (1000000 bytes = 1MB)

2. fileFilter - Set this to a function to control which files should be uploaded and which should be skipped

```
function fileFilter (req, file, cb) {
  // Allowed ext
  const filetypes = /jpeg|jpg|png|gif/;

  // Check ext
  const extname =
filetypes.test(path.extname(file.originalname).toLowerCase())
;
  // Check mime
  const mimetype = filetypes.test(file.mimetype);

  if(mimetype && extname){
    return cb(null, true);
  } else {
    cb('Error: Images Only!');
  }
}
```

Uploading Multiple Files

We can upload multiple files as well. In this case, multer gives us another function called **. arrays(fieldname[, max_count])** that accepts an array of files, all with the name **fieldname**. It generates an error if more than **max_count** files are uploaded. The array of files will be stored in req.files.

```
// uploading multiple images together
app.post("/images", upload.array("demo_images", 4), (req, res) =>{
  try {
    res.send(req.files);
  } catch (error) {
    console.log(error);
    res.send(400);
  }
})
```

Unit-5 Express.js

```
});
```

Example: Write an express js script that accepts **single file** to be uploaded using the multer middleware and saves the file to the specific directory called **“upload”**.

FileUpload.js

```
const expr=require("express");
const app=expr();
const path=require("path");
const multer=require("multer");
var storage=multer.diskStorage(
  {
    destination:"upload",
    filename:function(req,file,cb)
    {
      cb(null,file.fieldname+"-"+Date.now()+".jpg");
    }
  });
var upload1=multer(
  {
    storage:storage,

  }).single("mypic");
app.get("/",(req,res)=>
{
  res.sendFile(__dirname+"/index.html");
});
app.post("/uploadfile",upload1,(req,res)=>
{
  //Const f=req.file;
  //res.send(f);
  res.send("file uploaded");
});
});
app.listen(8000,()=>
{
  console.log("server running at 8000");
});
```

Index.html

```
<form action="/uploadfile" enctype="multipart/form-data" method="post">
  <input type="file" name="mypic"/>
  <input type="submit" value="Upload"/>
</form>
```


Unit-5 Express.js

Example: Write an express js script that accepts **multiple files max number 5** to be uploaded using the multer middleware and saves the files to the specific directory called **"multiple"**. limit the size of uploaded file to 1MB.

```
const expr=require("express");
const app=expr();
const path=require("path");
const multer=require("multer");
var storage=multer.diskStorage(
  {
    destination:"multiple",
    filename:function(req,file,cb)
    {
      cb(null,file.originalfilename);
    }
  });
const maxSize=1*1024*1024;
var upload1=multer(
  {
    storage:storage,
    limits:
    {
      fileSize:maxSize
    },

  }).array("mypic",5);
app.get("/",(req,res)=>
{
  res.sendFile(__dirname+"/index.html");
});
app.post("/uploadfile",upload1,(req,res)=>
{
  Const f=req.file;
  res.send(f);
});
});
app.listen(8000,()=>
{
  console.log("server running at 8000");
});
```

Index.html

```
<form action="/uploadfile" enctype="multipart/form-data" method="post">
  <input type="file" name="mypic"/>
  <input type="submit" value="Upload"/>
</form>
```

Unit-5 Express.js

Example

Write an express js script that allows **only pdf** type file to be uploaded using the multer middleware and saves the file to the specific directory called **“specific”**. If file other than pdf has been uploaded then it will give an error message that **“Only PDF format allowed”**. Also, final output to be displayed in browser through pug file.

Specific.js

```
const express = require('express')
const multer = require('multer');
const app = express();
app.set("view engine", "pug");
app.get('/', (req, res) => {
  res.sendFile(__dirname + " /index.html");
})
var store = multer.diskStorage({
  destination: "specific",
  filename: function (req, file, cb) {
    cb(null, file.originalname)
  }
})
// Allow only pdf to be uploaded
const filter = (req, file, cb) => {
  if (file.mimetype == "application/pdf" ) {
    cb(null, true);
  }
  else {
    return cb('Only pdf format allowed!');
  }
}
var upload = multer({
  storage: store,
  fileFilter: filter
}).single("mypic")
app.post('/uploadfile', upload, (req, res) => {
  const file = req.file
  if (file) {
    res.render(__dirname + "/output", {file: file.originalname})
  }
})
app.listen(6789);
```

output.pug

```
html
  head
    title FORM Data
  body
```

Unit-5 Express.js

```
h1 File #{file} have been uploaded
```

index.html

```
<html>
  <form action="/uploadfile" method="post" enctype="multipart/form-data">
    <input type="file" name="mypic" accept=".pdf"/>
    <input type="submit" value="Upload"/>
  </form>
</html>
```

RESTful API

What is REST?

The REST stands for **REpresentational State Transfer**.

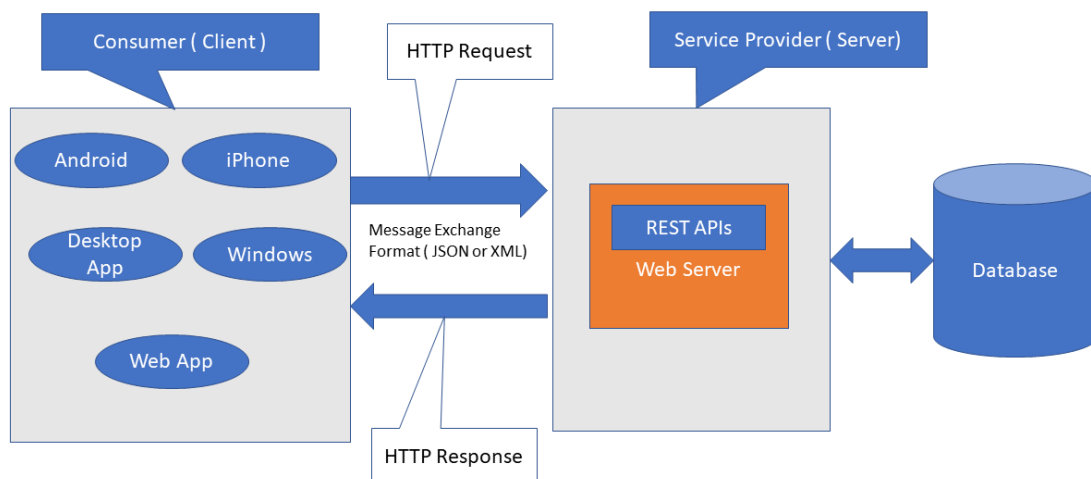
Let's understand the meaning of each word in the REST acronym.

- **State** means data
- **Representational** means formats (such as XML, JSON, YAML, HTML, etc)
- **Transfer** means carrying data between consumer and provider using the HTTP protocol

A REST API is an intermediary Application Programming Interface that enables two applications to communicate with each other over HTTP, much like how servers communicate to browsers.

The need for REST APIs increased a lot with the drastic increase of mobile devices. It became logical to build REST APIs and let the web and mobile clients consume the API instead of developing separate applications.

REST Architecture



Unit-5 Express.js

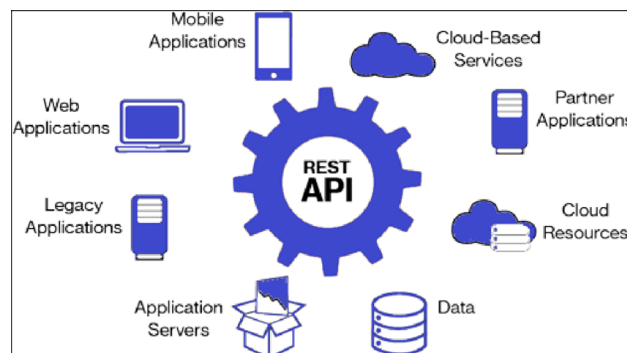
Request and Response: Request is the input to a web service, and the response is the output from a web service.

Message Exchange Format: It is the format of the request and response. There are two popular message exchange formats: XML and JSON.

Service Provider or Server: The service provider is one that hosts the web service.

Service Consumer or Client: A service consumer is one who is using a web service.

In simple words A RESTful API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data.



What Is `express.Router()` For

Express Routers are a way to organize your Express application such that your primary `app.js` file does not become bloated and difficult to reason about

To create an instance of an Express Router, we call the **`.Router()`** method on the top-level Express import. Then to use that router, we mount it at a certain path using **`app.use()`** and pass in the router as the second argument. This router will now be used for all paths that begin with that path segment. To create a router to handle all requests beginning with **`/hello`**, the code would look like this

```
const helloRouter = express.Router();

app.use('/hello', helloRouter);
```

Unit-5 Express.js

EXAMPLE: IN below example **movies.js** is serve as **provider** and **index.js** is serve as **consumer**

Index1.js

```
const expr=require("express");
const app=expr();
const movies=require("./movies"); // require movies.js file
app.use("/",movies);
app.listen(3000,()=>{
  console.log("Running at 3000");
});
```

movies.js

```
const expr=require("express");
const router1=expr.Router();
const movi=[
  {
    id:102,
    name:"abc1",
    year:1999,
    rating:8.1
  },
  {
    id:202,
    name:"xyz1",
    year:2000,
    rating:9.1
  }
];
module.exports=router1;
router1.get("/",(req,res)=>{
  res.json(movi);
});
```

To Retrieve specified element from JSON Array(API)

Index1.js

```
const expr=require("express");
const app=expr();
const movies=require("./movies"); // require movies.js file
app.use("/",movies);
app.listen(3000,()=>{
  console.log("Running at 3000");
});
```

Unit-5 Express.js

movies.js

```
const expr=require("express");
const router1=expr.Router();
const movi=[
  {
    id:102,
    name:"abc1",
    year:1999,
    rating:8.1
  },
  {
    id:202,
    name:"xyz1",
    year:2000,
    rating:9.1
  }
];
module.exports=router1;
router1.get("/:id1([0-9]{3,})",(req,res)=>

{
  var currMovi=movi.filter((m)=>
  {
    console.log(m)
    if(m.id==req.params.id1)
    {
      return true;
    }
  });
  console.log(currMovi)
  if(currMovi.length==1)
  {
    res.json(currMovi[0]);
  }
  else
  {
    res.json("Not Found");
  }
});
```

NodeMailer

The Nodemailer module makes it easy to send emails from your computer.

The Nodemailer module can be downloaded and installed using npm:

```
>npm install nodemailer
```

After you have downloaded the Nodemailer module, you can include the module in any application:

Unit-5 Express.js

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

Example

```
var nm=require("nodemailer");
var trans=nm.createTransport(
{
  service:"gmail",
  host:"smtp.gmail.com",      //optional
  port:465,      //optional
  secure:"true",
  auth:
  {
    user:"xyz.mnc123@gmail.com",
    pass:"vcguapkksfcungnd"
  }
});
var mailOption=
{
  from:"xyz.mnc123@gmail.com",
  to:"pqr.45abc@gmail.com",
  subject:"test mail",
  text:"its easy"
}
trans.sendMail(mailOption,(err,info)=>
{
  if(err)
  {
    console.log(err);
  }
  else
  {
    console.log("email send"+info.response);
  }
});
```

Unit-5 Express.js

What Is `express.Router()` For

Express Routers are a way to organize your Express application such that your primary `app.js` file does not become bloated and difficult to reason about

Express.Router

To create an instance of an Express Router, we call the **.Router()** method on the top-level Express import. Then to use that router, we mount it at a certain path using **app.use()** and pass in the router as the second argument. This router will now be used for all paths that begin with that path segment. To create a router to handle all requests beginning with **/hello**, the code would look like this:

```
const helloRouter = express.Router();  
  
app.use('/hello', helloRouter);
```


QB-288 Write a code to set up nodemailer in Express.js. Sender email id: "lju@gmail.com". Receiver email ids: "student@gmail.com and faculty@gmail.com". Mail subject should be "LJ University" Mail body contains "Welcome Student" in h3 tag and in table display data Date 28/06/23, Exam name FSD-2.

```
const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'lju@gmail.com',
    pass: 'yourpassword'
  }
});

const mailOptions = {
  from: 'lju@gmail.com',
  to: 'student@gmail.com, faculty@gmail.com',
  subject: 'LJ University',
  html: '<h3>welcome student</h3><table border="1px"><tr><th>date</th><th>Exam  
name</th></tr><tr><td> 28/06/23</td><td>FSD-2</td></tr>'
};

transporter.sendMail(mailOptions, (error, info) => {
  if (error) {
    console.log(err)
  }
  console.log("mail send")
});
```

QB-289 Write an express js script to configure the multer middleware. Perform following tasks. 1) Create a pug file named "file.pug". This file contains heading(h3) "Upload your CV" in red color. And, a form with input type file(to browse and select file) and submit(to upload the file). 2) Create a js file named "file.js" and link this js and pug file to browse pug file on "/home" page. 3) After uploading a file display message on "/upload" page "(file original name) has been uploaded". 4) Save uploaded files to specific directory named "upload". And in this folder file must be stored in format of "lju-file.pdf" where "lju" is the field name.

xyz.pug

```
html
body
  form(action="/upload" method="post" enctype="multipart/form-data")
    input(type="file" name="lju")
    input(type="submit" value="upload")
```

xyz.js

```
const express = require('express');
const multer = require('multer');
const app = express();
app.set('view engine', 'pug')
app.set('views', __dirname)
const storage = multer.diskStorage({
  destination: 'file',
  filename: function (req, file, cb) {
    cb(null, file.fieldname+'-file.pdf');
  }
});
const upload = multer({ storage: storage });
app.get("/", (req, res) => {
  res.render('xyz')
})

app.post('/upload', upload.single('lju'), (req, res) => {
  res.send(`<h3 style="text-align: center;">${req.file.originalname} has been uploaded</h3>`);
}).listen(3000);
```

QB-283 Write express js script to load student form using pug file which contains following fields

Name(text)

Email(email)

Course(radio : CE, IT, CSE)

Once form submitted then data must be displayed on '/data' page using pug file. Means data should be submitted from express application to PUG file.

form.pug(public folder)

html

body

h2 Student Form

form(action="/data",method="get")

div

label Enter Your Name

input(type="text", name="name")

div

label Enter Your Email

input(type="email", name="email")

div

label Course

input(type="radio", name="course", value="IT",id="IT")

| IT

input(type="radio", name="course", value="CE",id="CE")

| CE

input(type="radio", name="course", value="CSE",id="CSE")

| CSE

div

input(type="submit",value="Submit")

form_output.pug (public folder)

html

body

h1 Welcome!

table(border='1px solid',style='border-collapse:collapse;color:blue', cellpadding=10)

tr

th Name

th Email

th Course

tr

td #{name}

td #{email}

td #{course}

pug_form.js

```
var expr = require("express");
```

```
var app = expr();
```

```
app.set("view engine", "pug");
```

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

```
const staticp = p.join(__dirname, "../frontend");
```

```
app.get("/",(req,res)=>{
```

```
res.render(staticp+"/form");
```

```
});
```

```
app.get("/data",(req,res)=>{
```

```
res.render(staticp+"/form_output",{name:req.query.name, email:req.query.email,
```

```
course:req.query.course});
```

```
});
```

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

API TASK

App.js

```

const expr = require("express");
const router = expr.Router();
const data=[
  { id:101,name:"ABC",branch:"CSE",contact:9876543210,city:"Ahmedabad" },
  { id:102,name:"BCD",branch:"CE",contact:9876543210,city:"Ahmedabad" },
  { id:103,name:"XYZ",branch:"CSE",contact:9876543210,city:"Rajkot" },
  { id:104,name:"PQR",branch:"IT",contact:9876543210,city:"Ahmedabad" },
  { id:105,name:"ABC",branch:"CSE",contact:9876543210,city:"Surat" },
  { id:106,name:"ABC",branch:"IT",contact:9876543210,city:"Rajkot" }
]
module.exports = router;
router.get("/",(req,res)=>{
  res.set("content-type","text/html")
  for(i of data){
    res.write("<h3>ID: " + JSON.stringify(i.id) + ", Name: " + i.name + ", Branch: " + i.branch + ",
    Contact: " +
    i.contact + ", City: " + i.city + "</h3>");
  }
  res.send();
})
router.get("/:id([0-9]{3,})",(req,res)=>{
  var current_data = data.filter((i1)=>{
    if(i1.id == req.params.id){
      return true;
    }
  })
  if(current_data.length == 1){
    res.send(current_data);
  }else{
    res.send("Not Found")
  }
}

```

```

}
})
router.get("/:branch",(req,res)=>{
var current_data = data.filter((b)=>{
b1 = b.branch.toLowerCase();
if(b1 == req.params.branch){
return true;
}
})
if(current_data.length >= 1){
res.send(current_data);
}else{
res.send("Not Found")
}
})

```

main.js

```

const expr = require("express");
const app = expr();
const api = require("./api")
app.use("/api",api);
app.listen(7899);

```

For output in browser

http://localhost:7899/api

http://localhost:7899/api/101

<http://localhost:7899/api/it>

QB-271 Write an expressJS code in which RESTapi is created for json object named Places I love which contains name,country,state,city and rating out of 10(no decimal points) is given.upon passing

ratings on the browser it should display the places having that rating.i.e. on localhost:30001/a/10 should display all the places having 10 ratings.

api.js

```
const expr = require("express");
const router = expr.Router();
const PlacesIlove=[
{name:'manali',country:'india',state:'jammu',city:'manali',rating:8},
{name:'kullu',country:'india',state:'jammu',city:'manali',rating:9},
{name:'kashmir',country:'india',state:'jammu',city:'manali',rating:10}]
module.exports = router;
router.get("/:rating([0-9]){1,2})",(req,res)=>{
var current_data = data.filter((i1)=>{
if(i1.rating == req.params.rating){
return true;
}
})
if(current_data.length >0){
res.send(current_data);
}else{
res.send("Not Found")
}
})
```

main.js

```
const expr = require("express");
const app = expr();
const api = require("./api")
```

```
app.use("/a",api);  
app.listen(30001);
```

Extra Program:

Perform the following tasks as asked.

Create a HTML file for feedback form and this file should be loaded on home('/') page.

Fields are :

name, email, dropdown for the feedback ,text area for the additional comments, and submit button.

Once feedback submitted, message "Thank you for your feedback." Will be displayed on page '/feedback' and also send mail to the entered email id with the submitted feedback data. (Data can be submitted using get/post method)

mail_form.js

```
var expr = require("express");  
const app = expr();  
var nm = require("nodemailer");  
app.get('/',(req,res)=>{  
  res.sendFile(__dirname + '/mail.html')  
})  
app.get('/feedback',(req,res)=>{  
  var trans = nm.createTransport({  
    host : "smtp.gmail.com",  
    port: 465,  
    auth:{  
      user : "sender@gmail.com",  
      pass : "2stepverificationkey"  
    }  
  });  
  var mailoption = {
```



```

from:"noreply@gmail.com",
to: req.query.mail,
bcc:"bcc@gmail.com",
subject : "Feedback response",

html: "<h1>Hello "+req.query.uname+"!</h1><h2> Thank you for submitting the
feedback.<br>

Your feedback: <span style=color:red>"+req.query.feedback+"</span></h2><h3> Additional
comments: "+req.query.comments+"</h3>"

};

trans.sendMail(mailoption,(err,info)=>{
  if(err){
    console.error(err);
  }
  console.log(info);
});

res.send("Thank you for your feedback.")
})

app.listen(5001)

```

mail.html

```

<html>
<head><title>Feedback Form</title></head>
<body>
<form action="/feedback" method="get">
Name: <input type="text" name="uname"/>
Email: <input type="text" name="mail"/>
Feedback: <select name="feedback">
<option value="bad">Bad</option>
<option value="good">Good</option>

```

```

<option value="verygood">Very Good</option>
<option value="excellent">>Excellent</option>
</select>
Comments: <textarea name="comments" rows="10" cols="15"
placeholder="Comments"></textarea>
<input type="submit" value="Submit"/>
</form>
</body>
</html>

```

Program:

Write express JS script to pass data like message, name and id from express application to pug template in h1, h2 and h3 tags respectively and display data in browser. (Both file in same folder)

Pug.js

```

const express = require('express');
const path = require('path');
const app = express();
app.set('view engine', 'pug');
app.get('/', (req, res) => {
  res.render(__dirname+'/two', {message: 'Hello from Express!',name: 'lju',id: 2 });
});
app.listen(6002)

```

two.pug (same folder)

```

html
head
title My Express App
body
h1 #{message}

```

h2 #{name}

h3(style="color:red") #{id}