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

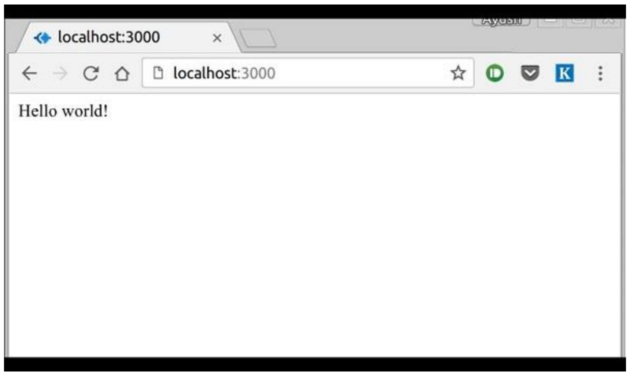
```
var express = require('express');
var app = express();
app.get('/', function(req, res){
 res.set ("content-type","text/plain");
 res.send("Hello world!");

});
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.

Method	Description
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.

- ✓ **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 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.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 expr = require("express");
var app = expr();
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

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

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

Send JSON object using res.send

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

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

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

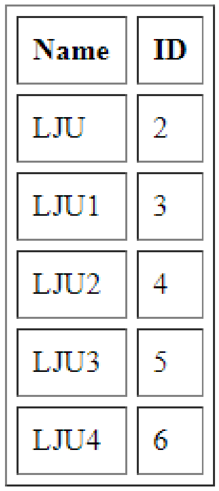
Example : Write Express JS script to request server to display only Age on browser.

```
const expr=require("express")
const app=expr();
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 expr=require("express")
const app=expr();
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 expr = require("express")
const app = expr();
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 expr = require("express")
const app = expr();
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 (expr.static (__dirname));**	**Express** **|---1.js** **|-----Frontend** **|------|---index.html** **|------|---1.css** **|------|---1.png** **Code:** **app.use (expr.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 (expr.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 (expr.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	/express/frontend/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, "../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 expr = require ("express")
const app = expr();
app.use(expr.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 and create 1.js file inside it.)

```
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;}
```

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");})
```