Express.js

"Express is a fast, minimalist web framework for Node.js"

Vraj Suratwala

Department of ICT, VNSGU

Topics Covered : Express

- Node Framework and What is Framework ?
- Why Express ? - Featues and Advantages.
- How to install an express ?
- nodemon and pug
- request and response object.
- Routing and Routers along with HTTP Methods.
- Dynamic and Static Routing.

Node Frameworks

- as we know that Node is runtime envirnment rather then a framework so we need one structural framework in that we can easily do our work and create the application.

- it open the door for many in-built librariesn,tools, and pre-written code snippets for seamless development.

- so, let;s understand the concept Frmaework!

What is Framework ?

- Node.js frameworks are tools that help developers make web applications with Node.js.
- They provide a structured way to build apps and include pre-made code for common tasks like handling web requests and managing databases.
- Using these frameworks makes development faster and helps keep code organized.
- Popular Node.js frameworks include Express.js, NestJS, Koa.js, Hapi.js, Meteor.js, and Next.js.

Some Node Frameworks

- Express.js
- Koa.js
- Meteor.js
- Nest.js
- Hapi.js
- Sails.js
- Adonis.js
- LoopBack.js
- Total.js
- Restify.js

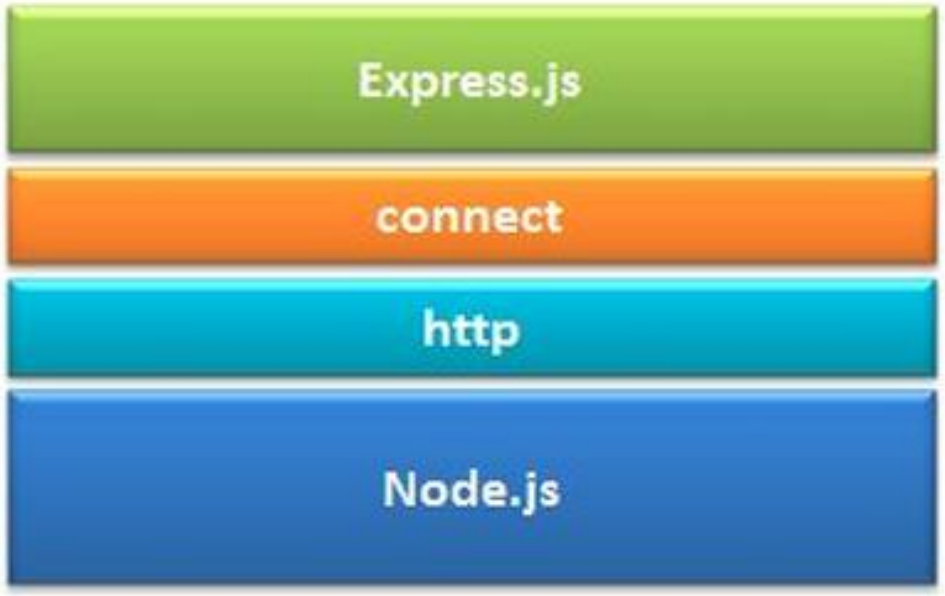
- Reference : https://www.geeksforgeeks.org/node-js/best-nodejs-frameworks-for-app-development/ - For Pros and cons of each one refer this link.

Why Express.js ?

- Express.js is a popular Node.js framework for building web applications in Node.js.
- It provides a robust set of features that help you build web apps quickly and easily.
- Routing
- Middleware
- Plugins

- So Let’s get Started with Express.js...

Express - Where it’s Resides ? along with features.



Features :

- robust for App and Web Development.
- can be used to create SPA’s, Multi Page Application and Hybrid Web Applications as well!
- allows Middleware.
- uses routing table for HTTP Requests.
- dynamic Rendering
- Ultra fast I/O
- Async and sigle threaded.
- MVC Architecture.
- Reliable API Makeing Features.
- Minimalistic and lightweight
- Extensive Plugin Making Eco System.
- Flexible file serving and templating.

Advantages of Express.js

- Makes Node.js web application development and I/O fast and easy.
- Easy to configure and customize.
- Allow Application based URL.
- Easy to Integrate with different templates.
- Allows to Define Error Handling Middleware.
- Allows REST API.
- Easy to Connect with Multiple Database.

How to Install an Express.js ?

- Go to https://nodejs.org/ - (official Node Website!)
- Download the LTS (Long-Term Support) version.

- go to Command Propmt : (to check the version of node.)
  - node -v
  - npm -v

- navigate to respactive path where you created the folder : (fire this command) : npm init -y or npm init

Install an Express.js

- Local : npm install express or npm i express
- Global : npm install express - g or npm i express -g

- it will create the node_modules directory and mendatory packages.
- you should also install this packages as per your need :
  - body-parser : middleware for handling JSON, Raw, Text and URL encoded form data.
  - cookie-parser : Parse Cookie header and populate req.cookies with an object keyed by the cookie names.

Install an Express.js

- multer - this is the middlerware which is responsible for multipart/form-data.

- Commands :
  - $ npm install body-parser --save
  - $ npm install cookie-parser --save
  - $ npm install multer --save

Install an Express.js

- COMMAND : Install an Express.js
  - 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.

Question : if you really worked with node and express then you may find one disadvantage that is to restart the server when you make any small or big change in your app ? so how to overcome that ? - Let’s Understand!

nodemon - IMP

- This tool restarts our server as soon as we make a change in any of our files, otherwise we need to restart the server manually after each file modification.
- How to install it ? - npm install -g nodemon. (where -g refers to Global flag which will install this package at Globally!)

- Once it installed then you should use nodemon instaed of node.

Pug - Old but Gold - Good to Know this!

- What is Pug ?
  - basically, pug is a template engine for Node.js (often paired with Express.js) that allows you to write cleaner, shorter and more dynamic HTML with JS.
  - Previously it was knows as Jade but because of copywrite issues, it was renamed as Pug.

- Features of Pug.
  - Produces HTML
    - Example :
    - HTML Syntax : <h1 class="title">Hello, World!</h1>
    - Pug Syntax : h1.title Hello, World!

Pug : Other Features

- Supports Dynamic Code
- Reusability (DRY - Don’t Repeat Yourself)
- Variables and Interpolation

- installing a pug : npm install pug

Simple Express Example.

```
import express from 'express';

const app = express();
app.listen(8003, "Localhost", console.log("This is Server 8003"));


 app.use("/",(req,res)=>{
          console.log(req.method + req.url + Date());
          next();
     });
```

request and response object.

- this both objects are the parameters of callback function in an express applications.
- Example :

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

request object

1. req.app This is used to hold a reference to the instance of the express application that is using the middleware.
2. req.baseurl It specifies the URL path on which a router instance was mounted.
3. req.body It contains key-value pairs of data submitted in the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as body-parser.
4. req.cookies When we use cookie-parser middleware, this property is an object that contains cookies sent by the request.
5. req.fresh It specifies that the request is "fresh." it is the opposite of req.stale.
6. req.hostname It contains the hostname from the "host" http header.

request object

7. req.ip It specifies the remote IP address of the request.
8. req.ips When the trust proxy setting is true, this property contains an array of IP addresses specified in the ?x-forwarded-for? request header.
9. req.originalurl This property is much like req.url; however, it retains the original request URL, allowing you to rewrite req.url freely for internal routing purposes.
10. req.params An object containing properties mapped to the named route ?parameters?. For example, if you have the route /user/:name, then the "name" property is available as req.params.name. This object defaults to {}.
11. req.path It contains the path part of the request URL.

request object

12. req.protocol The request protocol string, "http" or "https" when requested with TLS.

13. req.query An object containing a property for each query string parameter in the route.

14. req.route The currently-matched route, a string.

15. req.secure A Boolean that is true if a TLS connection is established.

17. req.stale It indicates whether the request is "stale," and is the opposite of req.fresh.

18. req.subdomains It represents an array of subdomains in the domain name of the request.

19. req.xhr A Boolean value that is true if the request's "x-requested-with" header field is "xmlhttprequest", indicating that the request was issued by a client library such as jQuery

request object methods

1. req.accepts (types)

This method is used to check whether the specified content types are acceptable, based on the request's Accept HTTP header field.

```
req.accepts('html');
//=>?html?
req.accepts('text/html');
// => ?text/html?
```

2. req.get(field)

This method returns the specified HTTP request header field.

```
req.get('Content-Type');
// => "text/plain"
req.get('content-type');
// => "text/plain"
```

request object methods

3. req.is(type)

This method returns true if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the type parameter.

```
// With Content-Type: text/html; charset=utf-8
req.is('html');
req.is('text/html');
req.is('text/*');
// => true
```

4. req.param(name [, defaultValue])

This method is used to fetch the value of param name when present.

```
// ?name=sasha
req.param('name')
// => "sasha"
// POST name=sasha
req.param('name')
// => "sasha"
// /user/sasha for /user/:name
req.param('name')    // => "sasha"
```

response object

The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an HTTP request.

It sends response back to the client browser.

It facilitates you to put new cookies value and that will write to the client browser.

Once you res.send() or res.redirect() or res.render(), you cannot do it again, otherwise, there will be uncaught error.

Properties:

1. res.app It holds a reference to the instance of the express application that is using the middleware.
2. res.headersSent It is a Boolean property that indicates if the app sent HTTP headers for the response.
3. res.locals It specifies an object that contains response local variables scoped to the request

response object

- res.append(Field,[,value]) : This method appends the specified value to the HTTP response header field.
  - Example : res.append('Warning', '199 Miscellaneous warning');

- res.attachment([filename]) : This method facilitates you to send a file as an attachment in the HTTP response.
  - Example : res.attachment('path/to/js_pic.png');

- res.cookie(name, value [, options]) : This method is used to set a cookie name to value. The value can be a string or object converted to JSON.
  - Example : res.cookie('name', 'Aryan', { domain: '.xyz.com', path: '/admin', secure: true });

response object

- res.clearCookie(name [, options]) : As the name specifies, the clearCookie method is used to clear the cookie specified by name.
  - Example : res.clearCookie('name', { path: '/admin' });

- res.download(path [, filename] [, fn]) : This method transfers the file at path as an "attachment" and enforces the browser to prompt user for download.
  - Example : res.download('/report-12345.pdf');

- res.end([data] [, encoding]) : This method is used to end the response process.
  - Example : res.end();
  - res.status(404).end();

9. res.format(object)

This method performs content negotiation on the Accept HTTP header on the request object, when present.

```
res.format({
  'text/plain': function(){
    res.send('hey');
  },
 'text/html': function(){
    res.send('
hey');
  },
  'application/json': function(){
    res.send({ message: 'hey' });
  },
 'default': function() {
    // log the request and respond with 406
    res.status(406).send('Not Acceptable');
  }
});
```

10. res.get(field)

This method provides HTTP response header specified by field.

```
res.get('Content-Type');
```

11. res.json([body])

This method returns the response in JSON format.

```
res.json(null)
```

12. res.json({ name: 'ajeet' })

Response JSONP method

13. res.jsonp([body])

This method returns response in JSON format with JSONP support.

```
res.jsonp(null)
res.jsonp({ name: 'ajeet' })
```

14. res.links(links)

This method populates the response?s Link HTTP header field by joining the links provided as properties of the parameter.

```
res.links({
  next: 'http://api.rnd.com/users?page=5',
  last: 'http://api.rnd.com/users?page=10'
});
```

15. res.location(path)

This method is used to set the response location HTTP header field based on the specified path parameter.

```
res.location('http://xyz.com');
```

16. res.redirect([status,] path)

This method redirects to the URL derived from the specified path, with specified HTTP status

```
res.redirect('http://example.com');
```

17. res.render(view [, locals] [, callback])

This method renders a view and sends the rendered HTML string to the client.

```
// send the rendered view to the client
res.render('index');
// pass a local variable to the view
res.render('user', { name: 'aryan' }, function(err, html) {
  // ...
});
```

18. res.send([body])

This method is used to send HTTP response.

```
res.send(new Buffer('whoop'));
res.send({ some: 'json' });
res.send('
.....some html
');
```

19. res.sendFile(path [, options] [, fn])

This method is used to transfer the file at the given path. It sets the Content-Type response HTTP header field based on the filename's extension.

```
res.sendFile(fileName, options, function (err) {
  // ...
});
```

Other Methods

- res.set(field,[,value])
- res.status(code)
- res.type(type)

Routing

- Specify the behavoiur of an Express Application.
- it is based on the HTTP Methods : GET, POST, PUT, PETCH, DELETE etc..
- Web framework will be provide specific thing at different routes.
  - Example : HTML Page, Scripts, Image or files.

- Methods : app.method(path,handler) : where the method can be any HTTP method.
- Methods like : app.all() , app.get(), app.post(), app.delete(), app.put().

an Important Methods.

- app.listen(3000) , where 3000 is port number on which the server will run.
- app.use() : which is specifying the middlewares.
- app.use() : method may seem similar to app.all() but there are a lot of differences between them which makes app.use() perfect for declaring middlewares.

app.use() vs app.all()

DIFFERENCE

Path - app.use() only see whether url starts with specified path where app.all() will match complete path.

```
app.use( "/product" , mymiddleware);
// will match /product
// will match /product/cool
// will match /product/hello
app.all( "/product" , handler);
// will match /product
// won't match /product/cool   <-- important
// won't match /product/hello    <-- important
app.all( "/product/*" , handler);
// won't match /product        <-- Important
// will match /product/cool
// will match /product/hello
```

Router

- when you are working with real or big application that time you need to separate the routes from main app.
- so Express.Router is used to got that.
- Example :

```
var express = require('express');
var router = express.Router();
router.get('/', function(req, res){
   res.send('GET route on things.');
});
```

HTTP Methods

- The HTTP method is supplied in the request and specifies the operation that the client has requested.

Method	Purpose	Use Case Example
GET	Fetch data	Load a user's profile, fetch products
POST	Send new data	Submit a form, create a new account
PUT	Update data	Edit a user's details, update a product
DELETE	Remove data	Delete a post, remove an item from cart

Dynamic Routes

- Example :

```
app.get('/products/:name/:id', function(req, res) {
  res.send('id:'+ req.params.id + ' and name: ' + req.params.name);
});
app.listen(8000);
```

Static Routing or URL for Static Routes/Files

Create a new directory, **public**.

Express, by default does not allow you to serve static files (html, images). You need to enable it using the following built-in middleware.

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

- Express looks up the static files relative to the static directory, so the name of the static directory is not part of the URL.

- Multiple Static Directories

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

Express - Middleware

an Important Component

What is Middleware ?

- Middleware functions are functions that have access ro request object and response object and to the next middleware which is in the req-res cycle.
- these functions are used to modify the request and response object.
- like body parser,express.static etc...

Task of Middleware

- It can execute any code.
- it can make changes to req-res objects.
- it can also end the res-req cycle.
- it can go to next middleware.

- others are : Body Parsing, Error handling, Routing Control

Types of Middleware

- 1. Application level middleware
- 2. Router level middleware
- 3. Error Handling middleware
- 4. Built-in middleware
- 5. Thied-Party middleware

Example of Middleware

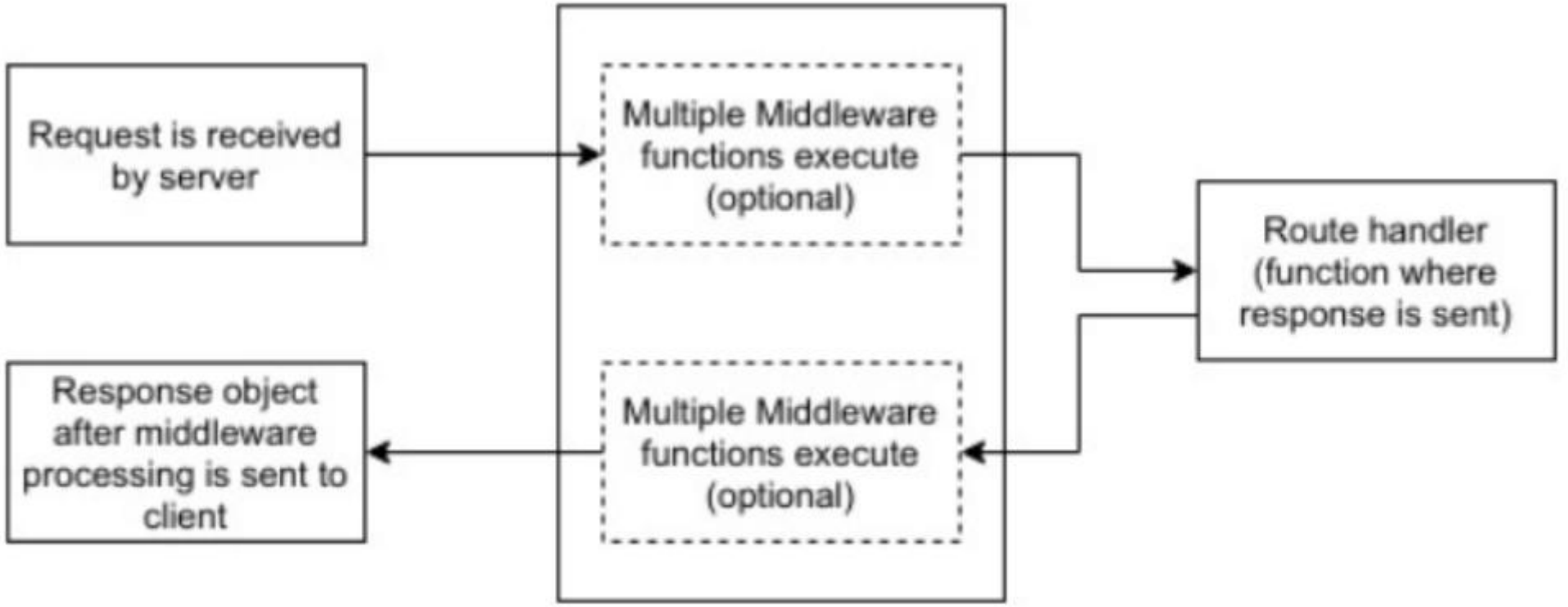
- // Custom Middleware
- const loggerMiddleware = (req, res, next) => {
- console.log(`${req.method} ${req.url} at ${new Date().toISOString()}`);
- next(); // Pass control to the next middleware or route
- };

- // Use the middleware in the app
- app.use(loggerMiddleware);

Chain of Middleware

```
//First middleware before response is sent
app.use(function(req, res, next){
   console.log("Start");
   next();
});
//Route handler
app.get('/', function(req, res, next){
   res.send("Middle");
   next();
});
app.use('/', function(req, res){
   console.log('End');
});
app.listen(3000);
```

Flow of Middleware



Example : 1. Application Level Middleware

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

app.use((req, res, next) => {
  console.log(‘Application Middleware triggered');
  next();
});

app.get('/', (req, res) => {
  res.send('Welcome to Home');
});
```

Runs on every request made to the application (unless filtered by route or method).

Example 2 : Route-Level Middleware router.js

```
import express, { Router } from 'express';
const app  = express();
const router = Router();

router.use((req, res, next) => {
  console.log('Router Middleware');
  next();
});

router.get('/user', (req, res) => {
  res.send('User Page');
});

2 references
export default router;
```

index.js

```
import express from 'express';
const app  = express();
import router from './routes/route-level.js';

app.use((req,res,next)=>{
    console.log("Hello Vraj Suratwala");
    next();
});

app.use('/api',router);
```

3. Error-Level Middleware

```
import express from 'express';
const app = express();
app.get('/crash_app',(req,res,next) =>{
    const error = new Error("The Server is Crashed!Pls Try After Some times!");
    error.status  = 500;
    next(error);
});

app.use((err,req,res,next)=>{
    if(err)
    {
        console.log(err); // printing an error
    }else{
        console.log("You are good to go ahead!");
    }
})

app.listen(3000, 'localhost', (err)=>{
    if(err)
    {
        console.log(err)
    }else{
        console.log("Server has been started!");
    }
})
```

Output :


```
Server has been started!
Error: The Server is Crashed!Pls Try After Some times!
```

So this is the output when we run this localhost and any error occurs.

4. Built-in middleware

- app.use(express.json());
- some popular middleware
  - express.json()
  - express.urlencoded()
  - express.static() etc...

5. Third Party Middlewares

Middleware	Purpose	NPM Package Name
`morgan`	HTTP request logging	`morgan`
`cors`	Enable Cross-Origin requests	`cors`
`body-parser`	Body parsing (pre-Express 4.16)	`body-parser`
`cookie-parser`	Parse cookies from request	`cookie-parser`
`express-session`	Manage sessions and cookies	`express-session`
`multer`	Handle file uploads	`multer`

Some Third Party Middlewares

body-parser

```
var bodyParser = require('body-parser');

//To parse URL encoded data
app.use(bodyParser.urlencoded({ extended: false }))

//To parse json data
app.use(bodyParser.json())
```

cookie-parser

It parses Cookie header and populate req.cookies with an object keyed by cookie names.

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

Express - fileupload

- multer : Use the body-parser for parsing json and x-www-form-urlencoded header requests.
- Use multer for parsing multipart/form-data.
- Example as below :

Cookies

Install package

npm install cookie-parser

Import cookie-parser into your app.

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

Define a route:

Cookie-parser parses Cookie header and populate req.cookies with an object keyed by the cookie names. Browser sends back that cookie to the server, every time when it requests that website.

Define a new route in your express app like set a new cookie:

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

Set Cookie

res.cookie(name, value [, options])

This method is used to set a cookie name to value. The value can be a string or object converted to JSON.

res.cookie('name', 'Aryan', { domain: '.xyz.com', path: '/admin', secure: true });

res.cookie('Section', { Names: [Aryan,Sushil,Priyanka] });

res.cookie('Cart', { items: [1,2,3] }, { maxAge: 900000 });

Get Cookie

req.cookies array

Clear Cookie

res.clearCookie(name [, options])

As the name specifies, the clearCookie method is used to clear the cookie specified by name.

Encrypted cookie

req.signedCookies array

Other Middleware

- 6. Custom Middleware
- 7. Conditional Middleware
- 8. Async. Middleware
- 9. Middleware Stack
- 10. Staic Middleware

Thank You

Vraj Suratwala

Department of ICT, VNSGU, Surat