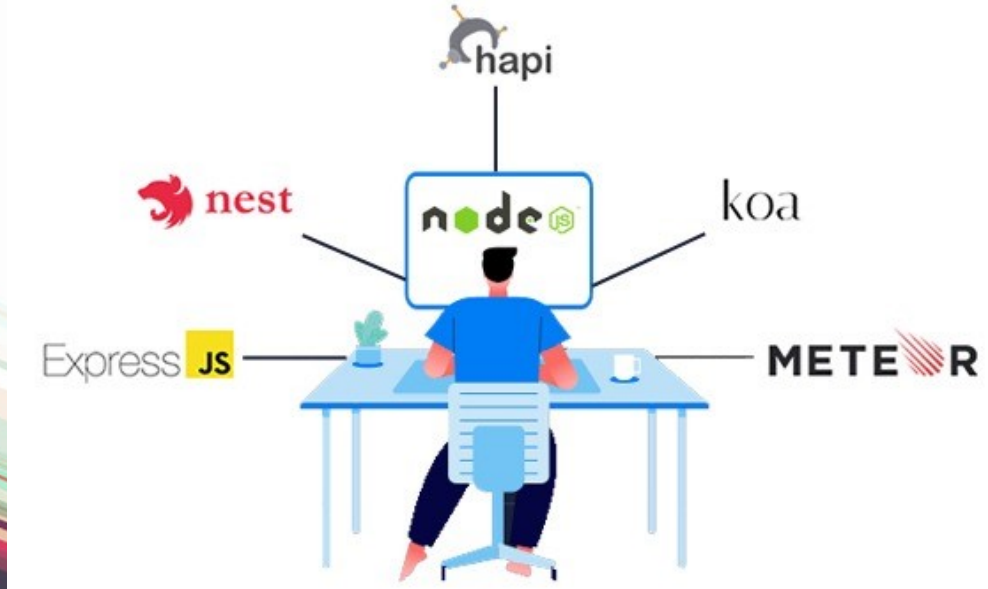
Unit – 5 Node JS



TECHNOSTACKS

NODE.JS FRAMEWORKS

What is a Node Framework ?

- A framework is a collection of various libraries and tools that are required in the development process of a software application.
- It acts as a base on which different software applications can be developed.
- Every web-app technology offers different kinds of frameworks with each one supporting a specific use case in the development lifecycle.
- Node.js frameworks are mainly of three types — MVC, Full-Stack MVC, and REST API frameworks.

MVC frameworks

- MVC frameworks splits application logic into three essential parts:

1. models,
2. views, and
3. controllers.

- MVC framework makes very simple to maintain and scale the app.
- Express.js is a classic example of an MVC framework.

Full-stack MVC frameworks

- When building a realtime app,
- Full-Stack MVC frameworks offer a great deal with scaffolding, libraries, template engines, and a range of other development capabilities.
- Additionally, they can take care of both frontend and backend development of applications.
- Derby.js is a full-stack web app development framework based on Node.js technology.

REST API (Representational state transfer)/ REST API frameworks

- There is also Node.js frameworks to build REST API which works very fast.
- HAPI is one such framework used to develop REST API.

Express.JS

- Express.js is also known as express.
- It is one of the most popular framework for node JS.
- It is open-source, free and under the MIT license Framework.
- It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.

Express js Features

- It is the fastest node js framework which provides a robust set of the features that can ensure flexible development of web applications.
- can be use to develop API.
- Its fast, robust, and asynchronous architecture jives well with Node.
- it’s quite suitable for almost any kind of web and mobile application development from small to large scale.

More about express js

- ExpressJS is a web application framework that provides you with a simple API to build websites, web apps and back ends.
- you need not worry about low level protocols, processes, etc.
- It is flexible as there are numerous modules available on **npm**, which can be directly used with express.
- Express was developed by **TJ Holowaychuk** and is maintained by the Node.js foundation and numerous open source contributors.
- It is used along with mongoDB with the help of **Mongoose**
- MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++.
- Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js.

LET US SEE AN EXAMPLE OF EXPRESS JS CODE

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

app.get('/', function(request, response){
   response.send("this is home page");
});
app.get('/aboutus', function(request, response){
    response.send("this is about us page");
 });
 app.get('/contactus', function(request, response){
    response.send("this is contact us page");
 });

app.listen(5000);
console.log("we are ready...");
```

HOW IT WORKS?

- The first line of our code is using the require function to include the express module.
- Before we start using Express, we need to define an instance of it which handles the request and response from the server to the client. In our case, it is the variable app.
- app.get() is a function that tells the server what to do when a get request at the given route is called.
- It has a callback function (request, response) that listen to the incoming request object and respond accordingly using response object.
- The request object represents the HTTP request and has properties for the request query string, parameters, body, and HTTP headers.
- The response object represents the HTTP response that an Express app sends when it gets an HTTP request.
- Lastly, app.listen() is the function that starts a port and host, in our case the localhost for the connections to listen to incoming requests from a client. We can define the port number such as 5000.

Routing in express JS

- Routing refers to how an server side application give reponse to a client request to a particular endpoint.
- This endpoint has of a URI (a path such as / or /aboutus) and an HTTP method such as GET, POST, PUT, DELETE, etc.
- Routes can be either web pages or REST API endpoints.
- In both cases the syntax is similar syntax for a route can be defined as:
- app.METHOD(PATH, HANDLER FUNCTION);
- Routers are helpful in separating app code such as different endpoints and keep relevant portions of the source code together.
- They help in building maintainable code.
- All routes are defined before the function call of app.listen().

Routing Methods

- HTTP protocol provides different methods for a client to make request.
- There are four main methods, GET, POST, PUT and DELETE.
- There is one function for each method.
- For example, a route of app.get() is used to handle GET requests and in return send simple message as a response.

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

// GET method route
app.get('/contact', function(request, response){
    response.send("this is contact page for get request");
 });

// POST method route
app.post('/contact', function(request, response){
    response.send("this is contact page post request");
 });

app.listen(5000);
console.log("we are ready...");
```

example of put and delete method

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

// put method route
app.put('/contact', function(request, response){
    response.send("this is contact page for put request");
 });

// delete method route
app.delete('/contact', function(request, response){
    response.send("this is contact page delete request");
 });

 var portno = 5000
 app.listen(portno, function(error)
 {
    if (error!=null)
        console.log(error);
    else
        console.log("Ready to accept request on portno", portno);
});
```

Dynamic URL

- In all our examples, we have build only static routes, but in real world work, we need to use dynamic url,
- Dynamic url means url which has some input attached to it, commonly known as query string.
- We can pass input as per requirement while sending request on specific route. Let us see an example

dynamic routing

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

// http://127.0.0.1:5000/product/1
app.get('/product/:id', function(req, res){
    res.send('product id = ' + req.params.id);
 });

 //http://127.0.0.1:5000/product/apple/1
app.get('/product/:name/:id', function(req, res){
    res.send('product name ' + req.params.name + ' product id = ' + req.params.id);
 });

app.listen(5000);
console.log('ready to accept request')
```

Middleware

- Express.js Middleware are different types of functions that are invoked by the Express.js routing layer before the final request handler execute.
- As the name suggest, Middleware appears in the middle between an initial request and final intended route.
- In stack, middleware functions are always invoked in the order in which they are added.
- Middleware is commonly used to perform tasks like body parsing for URL-encoded or JSON requests, cookie parsing for basic cookie handling, or even building JavaScript modules on the fly.

What middleware can do?

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

Types of middleware

1. Application-level middleware
2. Router-level middleware
3. Error-handling middleware
4. Built-in middleware
5. Third-party middleware

example of middleware

```
var express = require('express');
var app = express();
app.use(function(request,response, next) {
    console.log('I am first middleware');
    next();
});
app.use(function(request,response, next) {
    console.log('I am second middleware');
    console.log('request method = %s requested url = %s', request.method,request.url);
    next();
});
app.get('/contact', function(request, response){
    response.send("this is contact page for get request");
 });
 app.get('/aboutus', function(request, response){
    response.send("this is about page for get request");
 });
app.listen(5000);
console.log("we are ready...");
```