Node JS

Learning Node JS

PRAVEEN NAIR

Top Backend Technologies

Node.js - is an open-source, cross-platform JavaScript framework that is used to build server-side and networking applications.

Django - is an open-source framework based on Python. It is a web framework from the server's side. Django follows the Model Template View (MTV) architecture.

Spring Boot - is an open-source web framework based on Java that allows developers to build production-grade and standalone applications.

ASP.NET - is an open source web framework, created by Microsoft, for building modern web apps and services that run on macOS, Linux, Windows, and Docker.

Laravel - is a PHP framework that provides a built-in user interface, flexibility, API support, creativity, and an extensive range of various libraries that help in the development process of secure web applications

Introduction to Node JS

Node.js was invented for server side scripting and uses JavaScript.

Node.js is free and open source server environment.

Node.js runs on various OS(Windows, Linux, etc.)

Node.js continues with the next request in case current request is delayed.

Node.js can collect data submitted by front end users.

Node.js can add, delete, modify data in your database

Visit https://nodejs.org/en to download and install node js

First Nodejs Program – index1.js

```
console.log("Hello, Node.js!");
```

Custom module

....index1.js....

```
import { add,multiply } from "./util/calc.js";
const result = add(3,4)
console.log(result)
```

.....calc.js.....

```
function add(a, b) {
  return a + b;
}
function multiply(a, b) {
  return a * b;
}
export {add,multiply}
```

Node.js built-in modules

fs (File System)

http (Create servers)

os (System info)

http module

```
import http from "http";

const server = http.createServer((req, res) => {
  res.end("Hello, World!");

});

server.listen(3000, () => {
  console.log("Server running at http://localhost:3000/");
});
```

Introduction to Express JS

Express.js is fast and minimalist web framework for Node.js. An Express application is essentially a series of middleware function calls.

Express.js has powerful models for URL routing (matching an incoming URL with a server function), and handling HTTP requests and responses.

By making XML HTTP Requests (XHRs) or GETs or POSTs from your React.js front end, you can connect to Express.js functions that power your application.

Create and setup server

Open terminal and type the following command
npm init –y //this will create package.json file

Add following line in package.json, below name attribute
"type":"module",

Create .gitignore file (Add /node_module and .gitignore in the file)

Installing required packages

```
npm install express
npm install mongodb
npm install mongoose
npm install bcrypt
npm install cors
npm install jsonwebtoken
npm install nodemon  (nodemon index.js)
```

index.js

```
import express from "express";
var app = express();
app.use(express.json());  // parses req.body
app.listen(8080, () => {
  console.log("Server Started on Port 8080");
});
app.get("/", (req, res) => {
  //   res.send("Hello"); //status is 200 by default
});
app.get("/users", (req, res) => {
  res.send("Got a GET request for /users");
});
app.get("/ab*cd", (req, res) => {
  res.send("Got a GET request for /ab*cd");
});
```

res.json

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

res.json ({"name":"John"})
  // res.status(200).json({"name":"John"})
});
```

Status code

Informational responses (100 – 199)

Successful responses (200 – 299)

Redirection messages (300 – 399)

Client error responses (400 – 499)

Server error responses (500 – 599)

req.params

```
app.get("/:name", (req, res) => {
  res.send(req.params.name);
});
app.get("/:name/:age", (req, res) => {
  res.send(req.params.name + req.params.age);
});
app.get("/name/:name/age/:age", (req, res) => {
  res.send(req.params.name + req.params.age);
});
```

req.headers

```
app.get("/", (req, res) => {
  res.send(req.headers.authorization);
});
```

req.query

```
//http://localhost:8080/?name=amy&age=23

app.get("/", (req, res) => {
  res.send(req.query.name+req.query.age);
});
```

app.use

```
const logger = (req, res, next) => {
  req.msg = "Hello World";
  next();
};

app.use(logger);
app.get("/", (req, res) => {
  res.send(req.msg);
});

app.get("/", logger, (req, res) => {
  res.send("Hello " + req.msg);
});

app.get("/users", (req, res) => {
  res.send("Hello" + req.msg);
});
```

express.json – parses req.body

```
app.use(express.json()) //parses req.body to json

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

  res.json(req.body);

});
```

express.static

```
app.use(express.static("images"))

app.use("/images",express.static("images"))

app.use("/public",express.static("public"))
```

API Doc – public folder-index.html

```
<h1>My API Docs</h1>
<p>Base URL: http://localhost:8080</p>
<h2>Login</h2>
<p>POST /login</p>
<pre>
  {
    "email": "john@gmail.com",
    "pass": "secret123“
  }
</pre>
<h2>USERS</h2>
<p>GET / (Authorization: Bearer &lt;token&gt;)</p>
```

bcrypt

```
import bcrypt from 'bcrypt'

const pwd = "pass1234"

const hashedpwd = await bcrypt.hash(pwd,10) //cost factor 12 more secure but slow

console.log(hashedpwd)

const validate = await bcrypt.compare(pwd,hashedpwd)

console.log(validate)
```

jsonwebtoken – jwt.sign

```
import jwt from "jsonwebtoken";

const SECRET = "sometext";

const user = {
  name: "john",
  email: "john@email.com",
  role: "customer",
};

const token = jwt.sign(user, SECRET, { expiresIn: "1h" });

console.log(token)
```

jsonwebtoken – jwt.verify

```
import jwt from "jsonwebtoken";

const SECRET = "sometext";

const token =

  "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiam9obiIsImVtYWlsIjoiam9obkBlbWFpbC5jb
20iLCJyb2xlIjoiY3VzdG9tZXIiLCJpYXQiOjE3NTE1NjE3NjQsImV4cCI6MTc1MTU2NTM2NH0.F6ZddfX
QtDzMmVneZ_2CjSA59V6i5z7mVgaGB5S1wAM";

const user = jwt.verify(token, SECRET);

console.log(user)
```

authenticate / authorize(“admin”)

```
const authenticate = (req, res, next) => {
  if (req.headers.authorization) {
    req.role = "user";
    next();
  } else {
    return res.json({ message: "Invalid Token" });
  }
};

const authorize = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.role)) {
      return res.send("Access Denied");
    } else {
      next();
    }
  };
};
app.get("/", authenticate, authorize("admin"), (req, res) => {
  res.json(users);
});
```

process.argv[2] – cli argument

```
import express from 'express'

const app = express()

const PORT = process.argv[2] || "8080"

app.listen(PORT,()=>{
    console.log(`Server started on ${PORT}`)
})

app.get("/",(req,res)=>{
res.send(`This app is running on PORT ${PORT}`)
})

//node index.js 8081

// node index.js 8081 3 4
//console.log(process.argv[3]+process.argv[4])
```

dotenv – process.env.VARIABLE

```
//.env file //
DBUSER=user1
DBPASS=pass1

//index.js//
import dotenv from 'dotenv’
dotenv.config()
const DBUSER = encodeURIComponent(process.env.DBUSER)
const DBPASS = encodeURIComponent(process.env.DBPASS)
console.log(DBUSER,DBPASS)
```

Express router

```
const userRouter = express.Router();
const postRouter = express.Router();

postRouter.get("/add", (req, res) => {
  res.send("add post");
});

userRouter.get("/add", (req, res) => {
  res.send("add user");
});

app.use("/users", userRouter);
app.use("/posts", postRouter);
```

Express router.use

```
router.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

using mongoose in index.js

```
import mongoose from "mongoose";

mongoose.connect("mongodb://127.0.0.1:27017/temp");

const productSchema = mongoose.Schema(
  {
    name: { type: String, required: true },
    price: { type: Number, required: true },
  },

  { timestamps: true }

);

const productModel = mongoose.model("products", productSchema);
```

mongoose -list products

```
app.get("/api/products/", async (req, res) => {
  const items = await productModel.find();
  res.status(200).json(items);
});
```

mongoose - add a product

```
app.post("/api/products/", async (req, res) => {
  const { name, price } = req.body;
  const newProduct = new productModel({
    name: name,
    price: price,
  });
  await newProduct.save();
  res.status(200).json(newProduct);
});
```

mongoose - update a product

```
app.patch("/api/products/:id", async (req, res) => {
  const id = req.params.id;
  const product = req.body
  const result = await productModel.findByIdAndUpdate(id,product);
  res.status(200).json(product)
});
```

mongoose - delete a product

```
app.delete("/api/products/:id", async (req, res) => {
  const id = req.params.id;
  const result = await productModel.findByIdAndDelete(id);
  res.status(200).json(result);
});
```

Mongoose - pagination

```
app.get("/:pid", async (req,res) => {
    const pid = req.params.pid
    const limit = 1;
    const skip = (pid - 1) * limit;
    const total = await productModel.countDocuments({});
    const products = await productModel.find({}).skip(skip).limit(limit);
    res.json({products,total})
})
```

Deploy in Vercel

Add vercel.json file in the root of node project and pust to github

Goto Network Access in Mongodb Atlas and update ip address 0.0.0.0/0

Create account in vercel

Goto Project tab > Add New > Select github repository

Create Env > MONGODB_URI and paste the mongodb atlas string without quotes

vercel.json

```
{
"version": 2,
"builds": [
{
"src": "index.js",
"use": "@vercel/node"
}
],
"routes": [
{
"src": "/(.*)",
"dest": "index.js"
}
]
}
```

Thank You

- PRAVEEN NAIR