Node Js

Introduction:

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009. Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run with in the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js is an open-source and cross-platform runtime environment for executing JavaScript code outside a browser. **NodeJS is not a framework and it's not a programming language**. Node.js is used to build back-end services like **APIs** like Web App or Mobile App.

Node.js = Runtime Environment + JavaScript Library

Features of Node.js

1. **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.

2. **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

3. **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.

4. **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.

Working of Node.js: Node.js accepts the request from the clients and sends the response, while working with the request node.js handles them with a single thread. To operate I/O operations or requests node.js use the concept of threads. Thread is a sequence of instructions that the server needs to perform. It runs parallel on the server to provide the information to multiple clients. Node.js is an event loop single-threaded language. It can handle concurrent requests with a single thread without blocking it for one request.

Node.js basically works on two concept

1. Asynchronous
2. Non-blocking I/O

Non-blocking I/o: Non-blocking i/o means working with multiple requests without blocking the thread for a single request. I/O basically interacts with external systems such as files, databases. Node.js is not used for CPU-intensive work means for calculations, video processing because a single thread cannot handle the CPU works.

Asynchronous: Asynchronous is executing a callback function. The moment we get the response from the other server or database it will execute a callback function. Callback functions are called as soon as some work is finished and this is because the node.js uses an event-driven architecture. The single thread doesn't work with the request instead it sends the request to another system which resolves the request and it is accessible for another request.
To implement the concept of the system to handle the request node.js uses the concept of Libuv.

Libuv is an open-source library built-in C. It has a strong focus on asynchronous and I/O, this gives node access to the underlying computer operating system, file system, and networking.

Libuv implements two extremely important features of node.js

1. Event loop
2. Thread pool

Event loop: The event loop contains a single thread and is responsible for handling easy tasks like executing callbacks and network I/O. When the program is to initialize all the top-level code is executed, the code is not in the callback function. All the applications code that is inside callback functions will run in the event loop. EventLoop is the heart of node.js. When we start our node application the event loop starts running right away. Most of the work is done in the event loop.
Nodejs use event-driven-architecture.

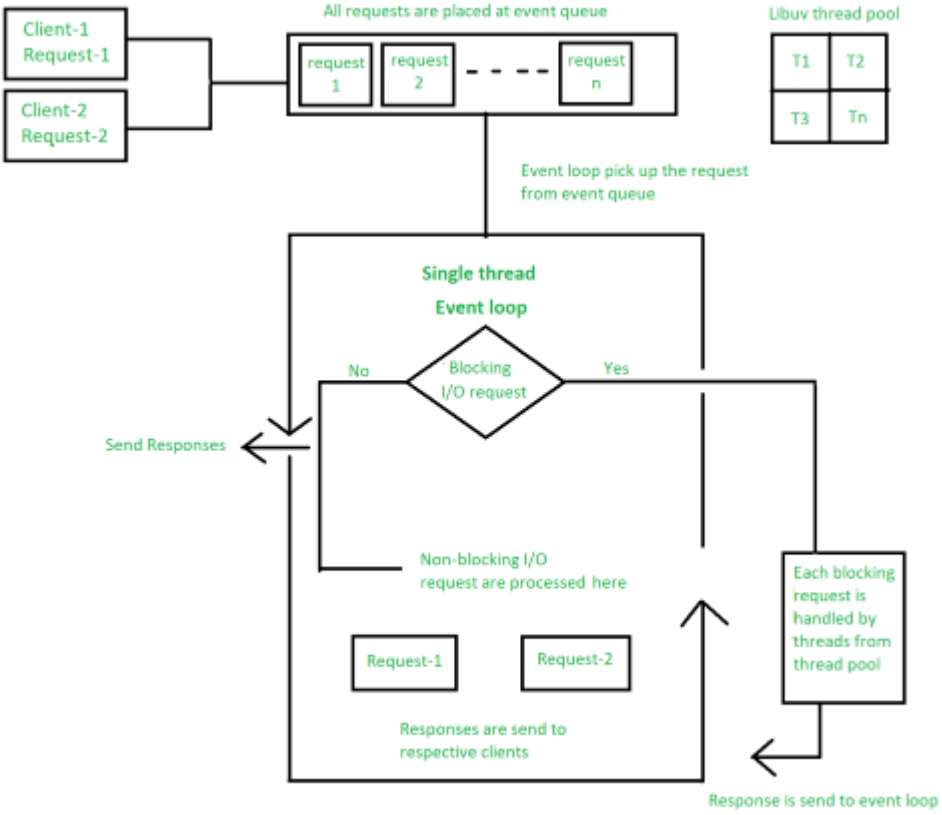
1. Events are emitted.
2. Event loop picks them up.
3. Callbacks are called.

Event queue: As soon as the request is sent the thread places the request into a queue. It is known as an event queue. The process like app receiving HTTP request or server or a timer will emit event as soon as they are done with the work and event loop will pick up these events and call the callback functions that are associated with each event and response is sent to the client.
The event loop is an indefinite loop that continuously receives the request and processes them. It checks the queue and waits for the incoming request indefinitely.

Thread pool: Though node.js is single-threaded it internally maintains a thread pool. When non-blocking requests are accepted there are processed in an event loop, but while accepting blocking requests it checks for available threads in a thread pool, assigns a thread to the client's request which is then processed and send back to the event loop, and response is sent to the respective client.

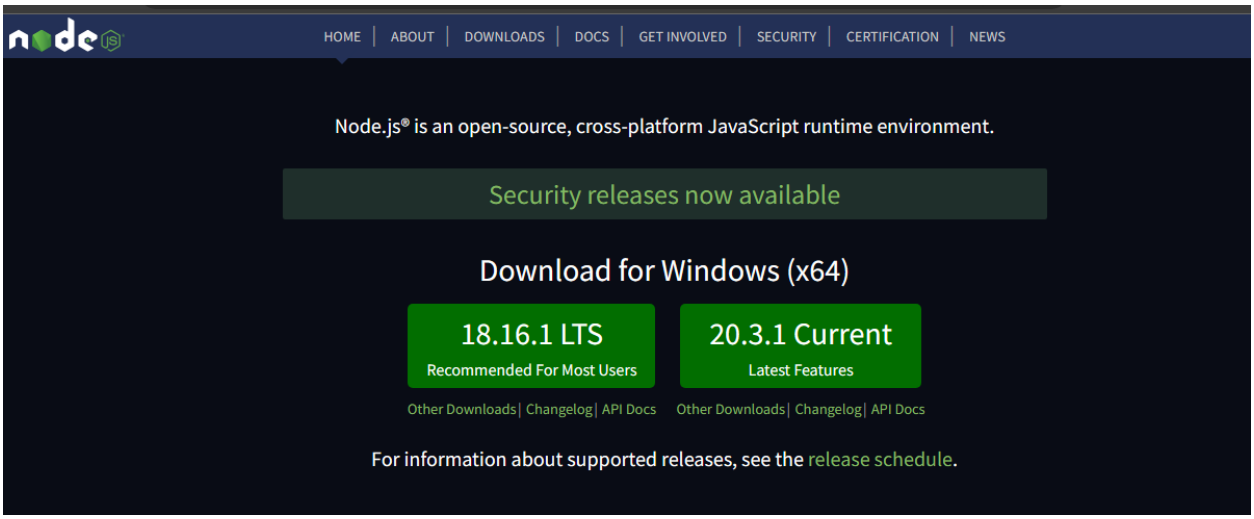
The thread pool size can be change:

process.env.UV_THREADPOOL_SIZE = 1;



Installation:

Download latest Node.js from
https://nodejs.org



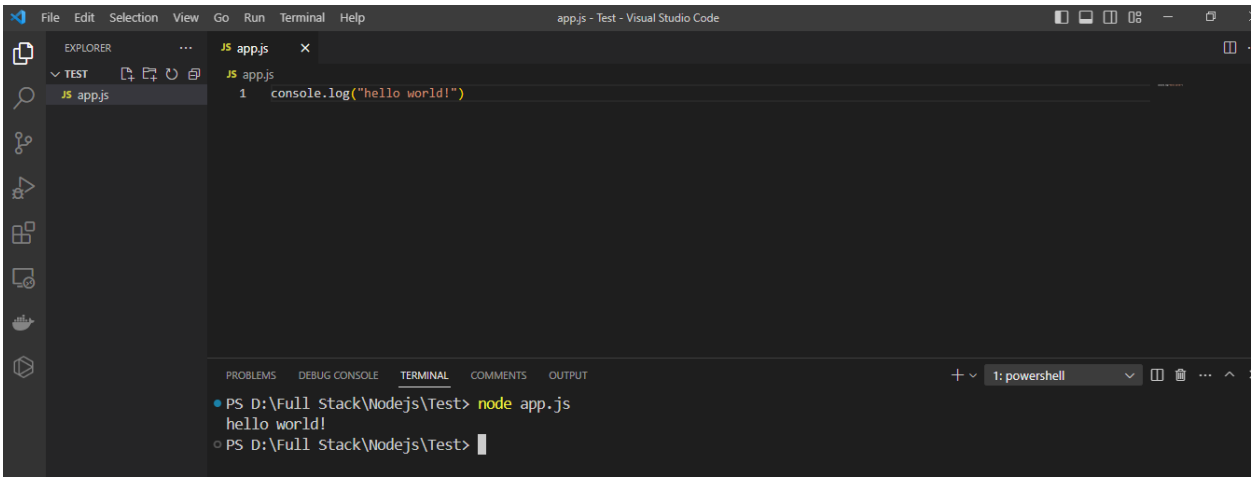
Run:

1. **Command Prompt(cmd):**

```
C:\Users\KHAN>node -v
v16.15.0

C:\Users\KHAN>
```

```
C:\Users\KHAN>node
Welcome to Node.js v16.15.0.
Type ".help" for more information.
> 5+6
11
>
```

2. VS-Code:



Status-Code

The Status-Code element in a server response, is a 3-digit integer where the first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

S.N.	Code and Description
1	**1xx: Informational** It means the request has been received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

First server app

```
const http =require('http');
const hostname ='127.0.0.1';
const port =3000;

const server=http.createServer((req,res)=>{
    res.setHeader('Content-Type','text/html');
    res.statusCode=200;
    res.end("hello world") // res.write("hello world")

});

server.listen(port, hostname,()=>{
    console.log(`server is listening at http://${hostname}:${port}`)
});
```

```
const http =require('http');
const hostname ='127.0.0.1';
const port =3000;

const server=http.createServer((req,res)=>{
    res.writeHead(200,{'Content-Type':'text/html'})
    res.end("hello world") // res.write("hello world")

});

server.listen(port, hostname,()=>{
    console.log(`server is listening at http://${hostname}:${port}`)
});
```

Difference between response.setHeader and response.writeHead:

response.setHeader() allows you only to set a singular **header.**

```
const http =require('http');
const hostname ='127.0.0.1';
const port =3000;

const server=http.createServer((req,res)=>{
    var body = "hello world";
    res.setHeader("Content-Length", body.length);
    res.setHeader("Content-Type", "text/plain");
    res.setHeader("Set-Cookie", "type=ninja");
    res.statusCode=200;
    res.write('hello worlddddddd');

});

server.listen(port, hostname,()=>{
    console.log(`server is listening at http://${hostname}:${port}`)
});
```

response.writeHead() will allow you to set pretty much everything about the response head including status code, content, and multiple **headers.**

```
const http =require('http');
const hostname ='127.0.0.1';
const port =3000;

const server=http.createServer((req,res)=>{
    var body = "hello world";
    res.writeHead(200, {
    "Content-Length": body.length,
    "Content-Type": "text/plain",
    "Set-Cookie": "type=ninja" });
    res.write('hello world123');

});
```

```
server.listen(port, hostname,()=>{
    console.log(`server is listening at http://${hostname}:${port}`)
});
```

Serving html :

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

const app =http.createServer((req,res)=>{

  // res.setHeader('Content-Type','text/html')
  // res.statusCode=200;
  //res.status(200);
  res.writeHead(200,{'Content-Type':'text/html'})

  res.end('<h1>hello world</h1>')

})

app.listen(3000, ()=>{
    console.log('server listen at : http://localhost:3000')
})
```

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

const app =http.createServer((req,res)=>{

  res.setHeader('Content-Type','text/html')
  res.statusCode=200;
  res.write(`<!DOCTYPE html>
  <html lang="en">
  <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
      <title>Document</title>
  </head>
  <body>
      <h1>hello</h1>
  </body>
  </html>`)
})

app.listen(3000, ()=>{
    console.log('server listen at : http://localhost:3000')
})
```

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

const app =http.createServer((req,res)=>{

  res.setHeader('Content-Type','text/html')
  res.statusCode=200;
  res.write('<!DOCTYPE html>');
  res.write('<html lang="en">');
  res.write('<head>');
  res.write('<meta charset="UTF-8">');
  res.write(' <meta name="viewport" content="width=device-width,
initial-scale=1.0">');
  res.write('<title>Document</title>');
  res.write('</head>');
  res.write('<body>');
  res.write('<h1>hello</h1>');
  res.write('</body>');
  res.end('</html>');

})

app.listen(3000, ()=>{
    console.log('server listen at : http://localhost:3000')
})
```

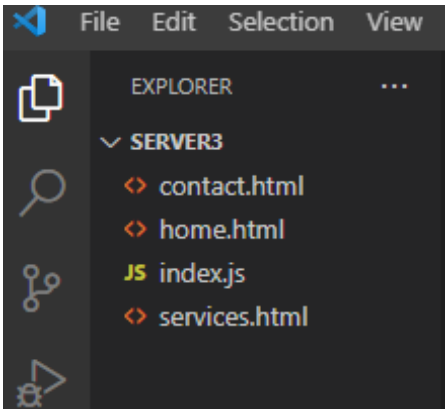
Serving External html File :

Index.html file present in same folder

```
const http = require('http');
const fs=require('fs');

const port = 3000;

const home = fs.readFileSync('index.html')
const server = http.createServer((req, res) => {
  res.statusCode = 200;

  res.setHeader('Content-Type', 'text/html');
  res.end(home);
});

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

Routing though http module:

1. **Folder structure**



Sample home.html file

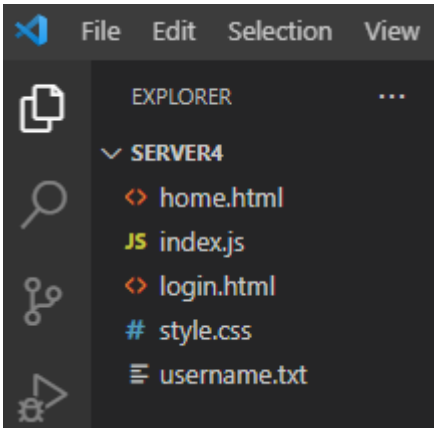
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
    <style>
          ul{
            display: flex;

          }

          li{
             list-style: none;
            margin: 20px;
             cursor: pointer;
          }


    </style>
</head>
<body>
    <nav>
         <ul>

            <li class="item"> <a href="/">Home</a>  </li>
            <li class="item"><a href="/contact">Contact </a> </li>
            <li class="item"> <a href="/services">Services</a> </li>
         </ul>


    </nav>

     <h1>Home</h1>
</body>
</html>
```

Index.js or server file

```
const http = require('http');
const fs=require('fs');
const port = 3000;

const home=fs.readFileSync('home.html');
const contact=fs.readFileSync('contact.html');
const services=fs.readFileSync('services.html');


const server = http.createServer((req, res) => {
  res.statusCode = 200;
  const url=req.url;

  res.setHeader('Content-Type', 'text/html');
  if (url=='/'){
      res.end(home);
  }

  else if (url=='/contact'){
    res.end(contact);
  }

  else if (url=='/services'){
    res.end(services);
  }
  else{
    res.statusCode=404;
    res.end('<h1> file not found</h1>');
  }


});

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

Sending data from client side to server side using form data

1. Folder structure



2.Home.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>

</head>
<body>
    <h1>Home</h1>
</body>
</html>
```

3.login.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login page</title>
</head>
<body>
    <h1>Login</h1>
    <form action="/login" method="post">
        <label for="name"> Name</label>
```

```
    <input type="text" name="name" placeholder="Enter your name">
    <br></br>
    <label for="password"> Password</label>
    <input type="password" name="password" placeholder="Enter your password">
    <br></br>
    <button type="submit">Submit</button>

   </form>
</body>
</html>
```

3.Index.js or server

```
const http = require('http');
const fs = require('fs')
const port = 3000;
const home = fs.readFileSync('home.html')
const login = fs.readFileSync('login.html')

const server = http.createServer((req, res) => {
    res.setHeader('Content-Type', 'text/html')
    res.statusCode = 200;
    //    console.log(req.body)
    if (req.url === '/') {
        res.end(home);
    }
    else if (req.url === '/login') {
        res.end(login);
    }

    if ((req.url === '/login') && (req.method === 'POST')) {

        const body = [];
        req.on('data', (data) => {
            body.push(data);
        })

        req.on('end', () => {
            // console.log(body);
            const requestbody = Buffer.concat(body).toString()
            // console.log(requestbody);
            const list = requestbody.split('&')
            const name = list[0].split('=')[1]
            const pws = list[1].split("=")[1]
```

```
            console.log(name);
            console.log(pws);
            fs.writeFileSync('username.txt',`name=${name}& pws=${pws}`)
        })
    }
})

server.listen(port, 'localhost', () => {
    console.log(`server listening at http://localhost:${port}`);
})
```

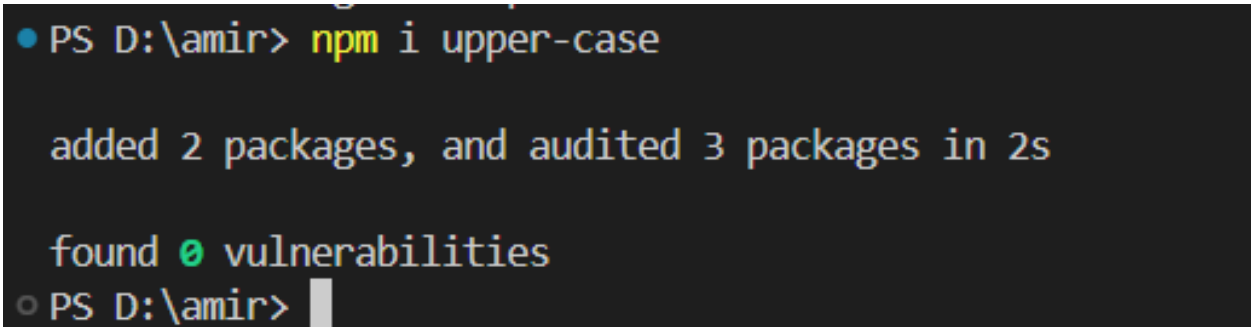
Node Package Manager (NPM):

What is NPM?

NPM is a package manager for Node.js packages, or modules,www.npmjs.com hosts thousands of free packages to download and use. The NPM program is installed on your computer when you install Node.js

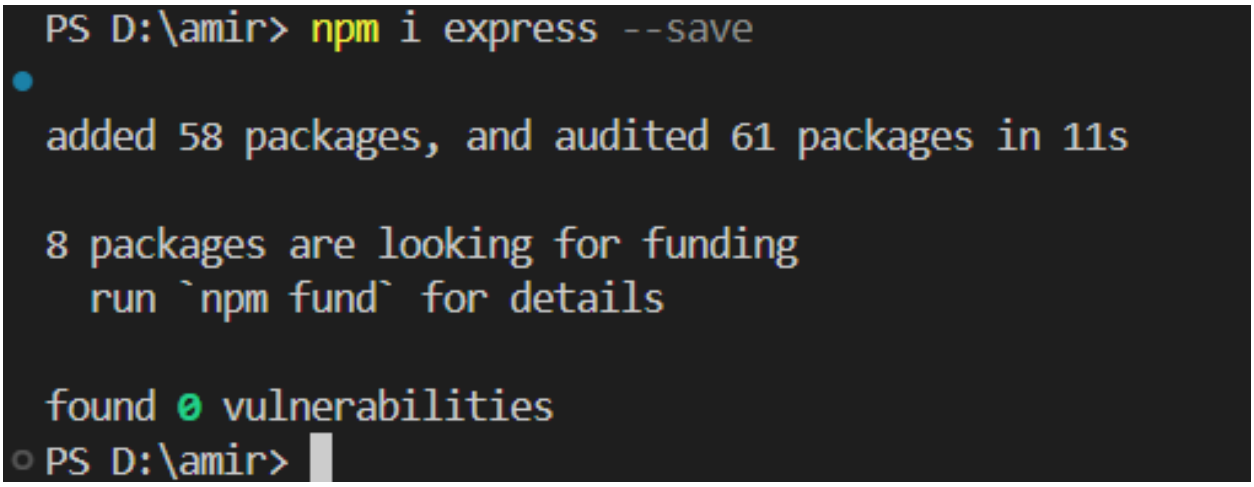
Download a Package

1. upper-case




```
var http = require('http');
var uc = require('upper-case');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc.upperCase("Hello World!"));
  res.end();
}).listen(8080,'localhost');

console.log('server running at http://localhost:8080/');
```

2. express

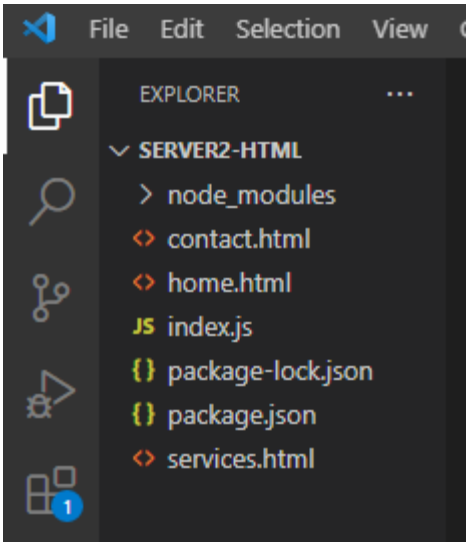


First server app using express

```
const express = require('express')
const app = express()
const port = 3000

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

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Serving html file

Folder structure



Index.js or server file

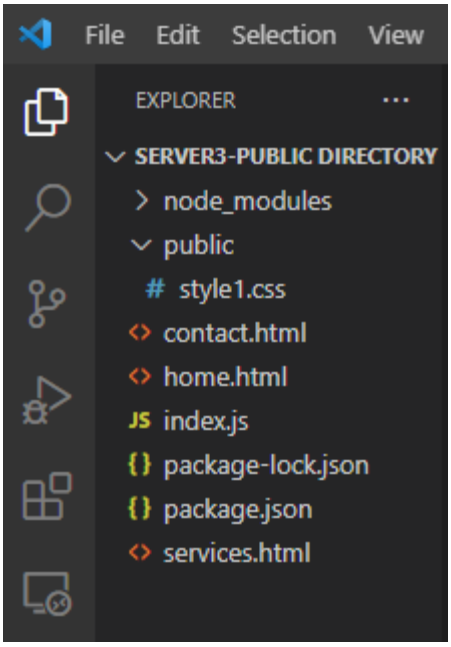
```
const express = require('express');
const path =require('path');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname,'./home.html'))

})

app.get('/contact', (req, res) => {
  res.sendFile(path.join(__dirname,'./contact.html'))

})

app.get('/services', (req, res) => {
  res.sendFile(path.join(__dirname,'./services.html'))

})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Middleware:

The middleware in node. js is a function that will have all the access for requesting an object, responding to an object, and moving to the next middleware function in the application request-response cycle.

Serving public file (Static file)

Folder structure



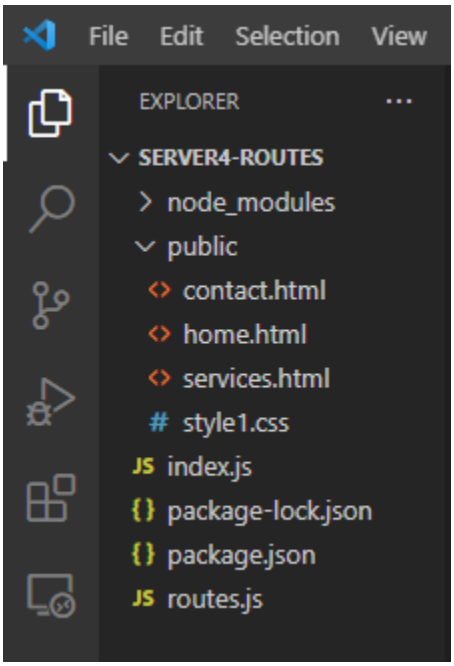
Index.js or server file

```
const express = require('express');
const path =require('path');
const app = express();
const port = 3000;

// middleware to access static file
app.use(express.static(path.join(__dirname,'public')))

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname,'home.html'))

})

app.get('/contact', (req, res) => {
  res.sendFile(path.join(__dirname,'contact.html'))

})
```

```
app.get('/services', (req, res) => {
  res.sendFile(path.join(__dirname,'services.html'))

})

console.log(__dirname)

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Separating all routes (Routes)

Folder structure



routes.js

```
const express =require('express');
const path =require("path");
const route =express.Router();

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

       res.sendFile(path.join(__dirname,'public/home.html'))

})

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

    res.sendFile(path.join(__dirname,'public/contact.html'))

})

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

    res.sendFile(path.join(__dirname,'public/services.html'))

})

module.exports =route;
```

index.js

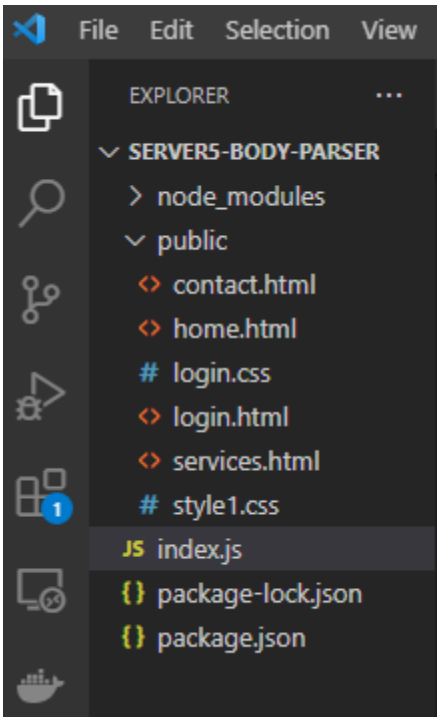
```
const express = require('express');
const path =require('path');
const app = express();
const port = 3000;

app.use(express.static(path.join(__dirname,'public')))

app.use(require(path.join(__dirname,'routes.js')))


app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Sending data from client side to server side using form data (using body-parser middleware)

Install body-parser

npm i body-parser

Folder structure



Index.js or server file

```
const express = require('express');
const path =require('path');
const bodyParser=require('body-parser');
const app = express();
const port = 3000;

// app.use(express.static(path.join(__dirname,'public')))
app.use(express.static('./public'))

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))

// parse application/x-www-form-urlencoded

// app.use(express.urlencoded({extended: false }));
```

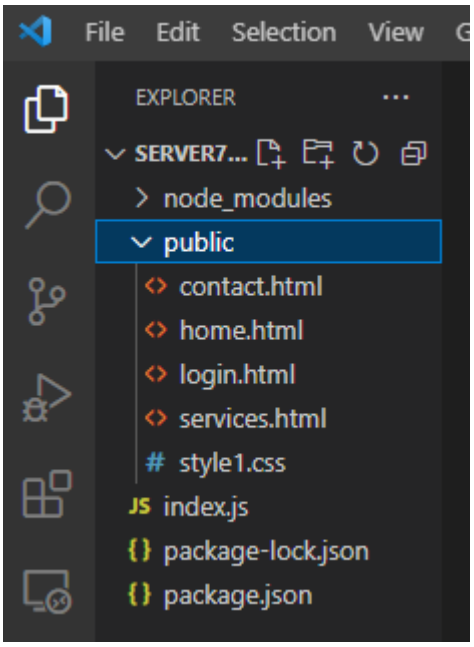
```
app.get('/',(req,res)=>{

   res.sendFile(path.join(__dirname+'/public/home.html'));

})

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

  res.sendFile(path.join(__dirname+'/public/contact.html'));

})

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

  res.sendFile(path.join(__dirname+'/public/services.html'));

})

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

  console.log(req.body.name);

})

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

  res.sendFile(path.resolve(__dirname+'/public/login.html'));
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Store cookies data in browser (using cookie-parser middleware)

Install cookie-parser

npm i cookie-parser

folder structure



Index.js or server file

```
const express = require('express');
const path =require('path');
const cookieParser = require('cookie-parser');
const app = express();
const port = 3000;

// app.use(express.static(path.join(__dirname,'public')))
app.use(express.static('./public'))

// middleware for cookie
app.use(cookieParser());

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

   res.sendFile(path.join(__dirname+'/public/home.html'));
```

```
   res.cookie('name', 'GeeksForGeeks');
   console.log(req.cookies.name)

})

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

  res.sendFile(path.join(__dirname+'/public/contact.html'));

})

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

  res.sendFile(path.join(__dirname+'/public/services.html'));

})

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

  console.log(req.body.name);

})

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

  res.sendFile(path.resolve(__dirname+'/public/login.html'));
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```