Servidor Web con NodeJS

Objetivo

En esta guía aprenderemos a crear un servidor web para manejar solicitudes http y https

Introducción

Para acceder a cualquier tipo de aplicación web, se necesita un servidor web. Un servidor web maneja todas las solicitudes http de la aplicación web, por ejemplo Apache es el servidor web que se usa para manejar peticiones de aplicaciones de tipo PHP o Java.

A diferencia de otros lenguajes Node.js proporciona las capacidades para crear nuestro propio servidor web que controlará las peticiones de forma asíncrona.

Crear Servidor Web

Para crear un servidor web que procese solicitudes entrantes de forma asíncrona se hace uso del módulo core `http`:

app.js

```
var http = require("http");
var server = http.createServer(function(req,res){
  //aqui manejamos las peticiones entrantes
});
server.listen(3000);
console.log("El servidor se encuentra escuchando por el puerto 3000 ");
```

Gestionar solicitudes HTTP

El método `http.createServer()` incluye los parámetros de solicitud (`req`) y respuesta (`res`). El objeto request se puede utilizar para obtener información sobre la solicitud HTTP actual, por ejemplo, la URL, el encabezado de la solicitud y otros. El objeto de respuesta se puede utilizar para enviar una respuesta a la solicitud actual. Ejemplo:

app.js

```
var http = require("http");
var server = http.createServer(function(req,res){
```

```
    res.writeHead(200,{"Content-Type":"text/html"});

    if(req.url == "/"){
        //cabecera de respuesta
        res.write("<html><body>Hola desde la pagina index!</body></html>");
    }else if(req.url == "/admin"){
        res.write("<html><body>Hola desde la pagina de
administracion!</body></html>");

    }else if(req.url == "/blog"){
        res.write("<html><body>Hola desde la pagina del blog!</body></html>");

    }else{
        res.write("Pagina no encontrada");
    }

}).listen(3200);
console.log("El servidor fue iniciado correctamente");
```

Envío de respuestas en formato JSON

Modificando la cabecera de respuesta podemos enviar datos en formato json:

app.js

```
var http = require("http");
var server = http.createServer(function(req,res){
    if(req.url == "/estudiantes"){
        var estudiantes = [{nombre:"JOSE LUIS GARCIA",notafinal:76},{nombre:"YALO
CUELLAR",notafinal:90},{nombre:"MIREYA TORREJON",notafinal:82},{nombre:"ALFREDO
OVANDO",notafinal:46},];

        res.writeHead(200,{"Content-Type":"application/json"});
        res.write(JSON.stringify(estudiantes));
        res.end();
    }else{
        res.writeHead(200,{"Content-Type":"text/plain"});
        res.end("Archivo no encontrado");
    }
}).listen(3200);
```

A medida que vayamos agregando funcionalidades, nuestra web app se irá haciendo cada vez mas grande y complejo de mantener. Existe un framework que hace que el desarrollo de aplicaciones web sea más rápido y fácil conocido como express

Framework Express

Express es un Framework que facilita la creación de aplicaciones web con NodeJS, proporciona muchas funcionalidades como en el enrutamiento, gestión de peticiones GET, POST, PUT, sesiones, cookies, etc.

Express está basado en el módulo `connect` y `http`. Entre las ventajas que proporciona este framework podemos mencionar:

- Hace que el desarrollo de aplicaciones web .js node sea rápido y fácil.
- Fácil de configurar y personalizar.
- Le permite definir rutas de la aplicación basadas en métodos HTTP y direcciones URL.
- Incluye varios módulos de middleware que puede usar para realizar tareas adicionales a petición y respuesta.
- Fácil de integrar con diferentes motores de plantilla como Jade, Vash, EJS, etc.
- Le permite definir un middleware de control de errores.
- Fácil de servir archivos estáticos y recursos de su aplicación.
- Le permite crear un servidor de API rest.
- Fácil de conectar con bases de datos como MongoDB, Redis, MySQL

Para usar express en nuestro proyecto debemos instalarlo:

```
npm install express
```

Una vez añadido express a nuestro sistema ya podremos usarlo de la siguiente forma:

app.js

```
var express = require("express");
var app = express();
var server = app.listen(3200,function(){
    console.log("el servidor express se encuentra en ejecucion");
});
```

Si el código anterior funciona correctamente al ejecutar `node app.js` significa que ya podemos definir las diferente rutas HTTP de nuestra aplicación:

app.js

```
var express = require("express");
var app = express();
app.get("/",function(req,res){
    res.send("<html><body><h1>Pagina principal</h1></body></html>");
});

app.get("/posts/listar",function(req,res){
    res.send(`LISTAR POSTS`);
});
app.post("/posts/nuevo",function(req,res){
    res.send(`POST REQUEST`);
```

```
});

app.put("/posts/update",function(req,res){
    res.send(`PUT REQUEST`);
});
app.delete("/posts/delete",function(req,res){
    res.send(`DELETE REQUEST`);
});

var server = app.listen(3200,function(){
    console.log("el servidor express se encuentra en ejecucion");
});
```

Analizar el cuerpo del request

Para analizar los datos enviados mediante la solicitud HTTP POST, se debe instalar el módulo `body-parser`. Una vez instalado el módulo podemos analizar los datos de un formulario (formLogin.html) de la siguiente forma:

formLogin.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>Formulario</title>
</head>
<body>
    <form action="/submit-student-data" method="post">
        First Name: <input name="firstName" type="text" /> <br />
        Last Name: <input name="lastName" type="text" /> <br />
        <input type="submit" />
    </form>
</body>
</html>
```

app.js

```
var express = require("express");
var app = express();
var bP = require("body-parser");
app.use(bP.urlencoded({extended:false}));
app.get("/",function(req,res){
    res.sendFile(__dirname+"/formLogin.html");
});
app.post("/login",function(req,res){
    var usuario = req.body.txtUsuario;
```

```
    var clave = req.body.txtClave;

    res.send(`Datos recibidos usuario: <b>${usuario}</b> contraseña:
<b>${clave}</b>`);

});

app.listen(3200,function(){
    console.log("el servidor express se encuentra en ejecucion");
});
```

Una vez añadida el módulo `body-parser` podremos acceder a los datos del post mediante el objeto `req.body`

Soporte HTTPS

Para que nuestro servidor soporte el protocolo cifrado https, se necesita dos cosas: un certificado y el módulo core `https`.

Existen dos tipos de certificados: los firmados por una entidad certificadora o CA, y los certificados autofirmados.

Generalmente los certificados autofirmados se usan en ambiente de desarrollo y los certificados validados por una CA en producción.

Para generar un certificado necesitamos instalar la herramienta OpenSSL. Al momento de generarl los certificados se te pedirá una serie de datos como ser: pais, departamento, ciudad, organización,FQDN, email. Para generar los archivos ejecutamos el siguiente comando:

```
openssl req -nodes -new -x509 -keyout server.key -out server.cert
```

En la carpeta donde ejecutó el comando anterior, se generaran dos archivos:

- server.cert es el archivo de certificado autofirmado
- server.key es la clave privada del certificado

Estos archivos se necesitan para establecer una conexión https y los debemos colocar en una ruta accesible por la aplicación.

apphttps.js

```
const https = require('https')
var express = require("express");
const fs = require('fs');
const app = express()

app.get('/', (req, res) => {
    res.send('Hola desde https')
})
```

```
https.createServer({
    key:fs.readFileSync("./server.key"),
    cert: fs.readFileSync("./server.cert"),

}, app).listen(3000, () => {
    console.log('Listening...')
})
```