Contenido

[Creación del proyecto 2](#_Toc42960321)

[CONFIGURACIÓN DEL MOTOR DE PLANTILLAS HANDLEBARS(HBS) 4](#_Toc42960322)

[CONFIGURACION DE MIDDLEWARES 7](#_Toc42960323)

[ROUTES 8](#_Toc42960324)

[CONFIGURACIÓN ARCHIVOS ESTATICOS 9](#_Toc42960325)

[CONEXIÓN CON MONGODB 10](#_Toc42960326)

# Creación del proyecto

Creamos un nuevo proyecto vacío.

Typamos npm init –y para crear un packpage json que describirá nuestro proyecto e iniciar el proyecto de node.

Iniciamos los siguientes módulos:

-express: para el servidor. npm i express

-express handlebars: motor de plantillas. npm i express-handlerbars

-express sessions: crea sesiones dentro del server (pj usuarios). npm i express-session

-method-override: extiende forms para añadir mas características ( methodos put & delete en forms). npm i method-override

-mongoose: modulo para database nosql. => driver para mongo db. npm i mongoose

-passport: para autentificación de usuarios. npm i passport

-passport-local: npm i passport-local

-bcrypjs: hash password. npm i bcryptjs

-connect-flash: mensajes diferentes vistas. (pj contraseña incorrecta). npm i connect-flash

-nodemon: refresco servidor en cambios automatico. npm i nodemon --D

npm i express express-handlebars express-session method-override mongoose passport passport-local bcryptjs connect-flash nodemon --D

Creamos la carpeta src con index.js y db.js para el código y la conexión con la base de datos respectivamente.

Creamos algunas carpetas dentro de la carpeta creada src:

Views: plantillas htmls

Routes: rutas del navegador url

Public: insertar archivos static. Imágenes,fuentes, css….

Models: definir la estructura de datos para la base de datos.

Helpers: Funciones que el server puede usar.

Config: Archivo de conexión, variables de entorno de configuración. De autenticación… config modulos…

mkdir config helpers models public routes views

En index.js requiero express.

const express = require('express'); // invocamos express

const app = express(); //lo inicializamos con app

Declaramos el puerto y realizamos el listen

//Settings

app.set('port',process.env.PORT || 3000); //inicializa el puerto del pc o el 3000

const puerto = app.get('port');

…

//Server listening

app.listen(puerto,() =>{

console.log('Server ON en puerto',puerto);

});

# PATH del proyecto

Una vez funcionando al tener la carpeta src fuera de node, hay que unir paths para que entienda que la carpeta de proyecto está dentro de una carpeta src.

Para ello inicializamos el modulo path interno de node (no hace falta instalarlo)

const path = require('path'); // modulo de enrutado interno de node.

Y le indicamos por ejemplo la carpeta view la ruta que tiene

app.set('views', path.join(\_\_dirname, 'views')); //une directorios mediante el modulo path. dirname devuelve donde se ejecuta index => src y la concadena con views. es lo mismo que poner /src.

# CONFIGURACIÓN DEL MOTOR DE PLANTILLAS HANDLEBARS(HBS)

const hbs = require('express-handlebars'); // Motor de plantillas html.

Una vez declarado el motor, lo configuramos de la siguiente forma:

app.engine('.hbs', hbs({

defaultLayout: 'main', //nombre de la plantilla principal.

layoutsDir:path.join(path\_view, 'layouts'), // direccion path para el proyecto.

partialsDir:path.join(path\_view, 'partials'), //pequeñas plantillas html para reutilizar en cualquier vista

extname:'.hbs' //extension de los archivos

}));

Creamos dentro de views una carpeta que se llame layouts con un archivo main.hbs que será la plantilla html principal definida con la propiedad defaultLayout.

Le indicamos donde está dicha carpeta con layoutsDir.

Creamos otra carpeta dentro de views que se llame partials donde guardaremos plantillas reutilizables.

Le indicamos donde esta al motor de plantillas con la propiedad de configuración partials  
Dir.

Con extname le indicamos la extensión que tendrán los archivos. En nuestro caso hbs.

Ahora creamos en la carpeta views 2 archivos : about.hbs y index.hbs que serán cargados en las [routes](#ROUTeS).

Modificamos dichos archivos, por ejemplo le ponemos un titulo diferente a about.hbs y a index.hbs.

Ahora en el archivo main.hbs de la carpeta layout montamos una plantilla html de prueba:

<!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>

    RUTA:

    {{{ body }}}

</body>

</html>

Al cambiar de ruta en el navegador entre la principal y la /about veremos que cambia el {{{body}}} por la plantilla index.hbs y about.hbs respectivamente.

Ahora crearemos en about un pequeño formulario para recoger datos

<h1>about</h1>

<form action="/usuario/newuser" method="POST">

    <div>

        <p>

            <label>Nombre:</label>

            <input id="name" name="nombre" type="text">

        </p>

        <p>

            <label>Apellido:</label>

            <input id="appellido" name="apellido" type="text">

        </p>

    </div>

    <button type="submit">Guardar</button>

</form>

Y en las routes en el index.js debemos crear esta nueva ruta

router.post('/usuario/newuser',(req,res)=>

    {

        console.log(req.body);

        res.send('ok');

    });

Cuando haya un post a esa dirección ( usuario/newuser) entrara en la ruta.

Con el console log vemos que le llega información.

# CONFIGURACION DE MIDDLEWARES

const methodOverride = require('method-override');  //añade funcinalidades al los formularios

const session = require('express-session'); //guarda sessiones usuarios

…

app.use(express.urlencoded()); // para recibir datos de formularios. sin ello el servidor no puede recibir formularios

app.use(methodOverride('\_method'));

app.use(session({ //configuracion para identificar al usuario

    secret:'secreto', //Palabra secreta

    resave: true,

    saveUninitialized:true

}))

# ROUTES

Creamos en la carpeta routes un archivo llamado index.js, users.js y notes.js

En cada uno iran las url del server.

Comenzamos por asignar en el index.js principal las rutas creadas

//Routes

app.use(require('./Routes/index'));

app.use(require('./Routes/notes'));

app.use(require('./Routes/users'));

Ahora definimos cada ruta. Comenzamos por routes/index.js

const router = require('express').Router(); //constructor de el enrutador de express

router.get('/',(req,res)=>

    {

    res.render(‘index’); //al iniciar en la pagina principal de la app usa la funcion

    });

router.get('/about',(req,res)=>

    {

        res.render('about');

    })

module.exports = router; // para que exporte la configuración.

Esto cargará el archivo requerido(index.hbs y about.hbs) cuando se visite las rutas.

Como en index.js de src le especificamos que la plantilla principal era el archivo [main](#defaultlayout) esta renderiza primero el archivo main.

# CONFIGURACIÓN ARCHIVOS ESTATICOS

En index.js de la carpeta src

app.use(express.static(path.join(\_\_dirname,'public'))); //config para archivos estaticos, le establece el path.

# CONEXIÓN CON MONGODB

Instalamos en la computadora mongodb.

Creamos en C una carpeta data que contenga otra carpeta db

En la ruta C:\Program Files\MongoDB\Server\4.2\bin ejecutamos un terminal como administrador ( si no no reconoce el comando) y ejecutamos el comando mongod

Para no tener que ir a esta ruta cada vez, entramos en variables del entorno del sistema y le añadimos una nueva variable llamada PATH y la ruta del C:\Program Files\MongoDB\Server\4.2\bin

Y ya podemos ejecutar mongod desde cualquier ruta.

En el archivo db.js en la carpeta src metemos el siguiente código:

const mongoose = require('mongoose');

//configuración mongoose.

mongoose.connect('mongodb://localhost/db-prueba',{ // si no existe la db la crea (local)

    useCreateIndex: true, //config necesaria para que no de error

    useNewUrlParser: true,

    useFindAndModify: false

}).then(db => console.log('DB Conected')).catch(err=> console.error(err)); //msg suceful o error

En index.js de src llamamos a esta configuración

require('./db'); //inicializacion de la base de datos (archivo db.js)

En models, establecemos los modelos que le llegan a la base de datos.

Para ello creamos en la carpeta models un archivo por ejemplo Usuario.js

En el archivo empleamos mongoose para hacer un esquema de datos

const mongoose = require('mongoose'); //req mongoose

const {Schema} = mongoose; //clase esquema

//declaramos una nueva clase esquema y sus propiedades

const schema\_nuevo\_usuario = new Schema({

    nombre:{type: String, required: true},

    apellido: {type: String, required: true},

    date: {type: Date, default: Date.now}

});

//lo exportamos con el nombre newuser

module.exports = mongoose.model('newuser',schema\_nuevo\_usuario);

Posteriormente tratamos estos datos

Para ello nos vamos a las index de las routes principales e importamos el esquema

const usuario = require('../models/Usuario');

esto lo usamos para guardar los datos en la base de datos

router.post('/usuario/newuser', async(req,res)=>

    {

        const {nombre,apellido}= req.body;

        const nuevo\_usuario = new usuario({nombre,apellido});

        console.log(req.body);

        await nuevo\_usuario.save();

        res.redirect('/');

    });

Cuando se haga post a esta dirección (lo hace el submit del formulario de la plantilla about)

Carga los names de los inputs (req.body) y los almacena en dos constantes (nombre,apellido).

Ahora les asigna el modelo usuario que definimos antes (creando los datos que faltan como la id y la date) .

Await es para hacerlo asíncrono y que el server no se quede pillado, hay que ponerle async a la function. Usando la función save() a la constante tratada se guarda automáticamente en la base de datos.

Después redirecciona al índice principal.

Modificamos el índice principal para que muestre la lista de todos los valores de la base de datos.

router.get('/', async (req,res) =>

    {

    const lista\_usuarios = await usuario.find().then(documentos=> //obtiene todos los datos de la base de datos (newuser import como usuario)

    {

        const datos =  // se crea una subvariable porque handlebars no se le pueden pasar datos directamente

        {

        listado: documentos.map(documento => //se almacena lista\_usuarios (variable con todos los datos de la db) en documentos

            { //se mapea documentos para obtener de cada documento un nombre y apellido (propiedades de lista\_usuarios)

                return{

                    nombre:documento.nombre,

                    apellido:documento.apellido

                }

            })

        }

    res.render('index', {lista\_usuarios:datos.listado}); //al iniciar en la pagina principal de la app usa la funcion se le pasa el objeto obtenido de la base de datos

    });

    });