Instalación de visual code

Instalación de node js

Npm init para crear el package.json

Se configuran los datos iniciales del package.json

Instalación de express npm i express

Script del package.json “start”:” “Index.js”

Instalación de nodemon npm i –D nodemon como paquete de desarrollo

Crear servidor en index.js

Importar las librerías, en este caso express; para utilizar la función import, es necesario en el package.json anexar el type después del main

"name": "bienes\_raices",

  "version": "1.0.0",

  "description": "Mi segundo proyecto",

  "main": "index.js",

  "type": "module",

El require se utiliza para common js

Import se utiliza para Emacscript modules

import express from "express";

crear una variable y llamar la función exportada

const app =express();

Asignación del puerto y en la variable de express, llamado de la función listen

const port=3000;

app.listen(port,()=>{

    console.log(`El servidor esta funcionando en el puerto ${port}`)

});

Se crean 3 carpetas

Controllers

Routes

Views

1. En la carpeta controllers van las funciones que se requieren y se guardan en variables

const formularioLogin=(req,res)=>{

res.render('auth/login',{

    pagina: 'Iniciar Sesión'

})

}

Estas se deberán exportar para poder usarlas

export{

    formularioLogin, formularioRegistro,formularioOlvidePassword

}

Res.render para redenrizar una pagina

Res.send para enviar un mensaje

1. En la carpeta de routes se colocarán las rutas hacia cada vista

Lo primero será importar las librerías necesarias

import express from "express";

import {formularioLogin,formularioRegistro,formularioOlvidePassword} from '../controllers/usuarioController.js'

la importación puede darse de librerías de paquetes o importación de funciones creadas para el caso en los controladores, donde se deberá indicar la ruta de acceso.

router.get('/login',formularioLogin);

router.get('/registro',formularioRegistro);

router.get('/olvide-password',formularioOlvidePassword);

Se deberá exportar la variable de las rutas.

export default router

1. En la carpeta views se crearán las vistas, layout que corresponde a los parciales de las vistas.

El template engine será Pug, se deberá instalar con el comando npm i pug

En el index.js se debe habilitar el template engine con las líneas, especificar el template y la ruta a buscar para rendereizar la vista.

app.set('view engine','pug')

app.set('views', './views')

en layout se puede crear el header normalmente

doctype html

html(lang="es")

    head

        meta(charset="UTF-8")

        meta(http-equiv="X-UA-Compatible", content="IE=edge")

        meta(name="viewport", content="width=device-width, initial-scale=1.0")

        title Bienes Raices | #{pagina}

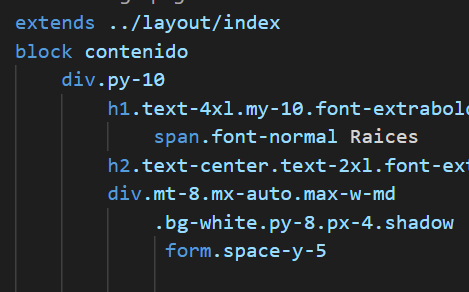
        link(rel="stylesheet", href="/css/app.css")

    body.min-h-screen.bg-gray-50

        main(class="mx-auto container mt-10 px-2")

            block contenido

la línea block contenido indica que traerá el bloque de código con el mismo titulo



Para enviar datos a la vista se podrá realizar a través de las funciones del controlador

const formularioRegistro=(req,res)=>{

res.render('auth/registro',{

    pagina: 'Crear Cuenta'

})

}

En el caso anterior llevara el titulo página.

**TAILWIND**

Para la instalación de tailwind se utilizara el comando npm i –D tailwindcss autoprefixer postcss-cli

Se deberá especificar en el index.js donde se encontrarán los archivos de css

app.use(express.static('public'))

se deberá crear una carpeta public y dentro de ella deberán estar 3 carpetas css,img,js

en la carpeta css se creará un archivo tailwind.css con las siguientes líneas

@tailwind base;

@tailwind components;

@tailwind utilities;

Se ingresa el comando npx tailwindcss init-p el cual creara el app.css, postcss.config y el tailwind.config para todos los estilos que se crearan.

En el tailwind.config se deberá indicar donde están ubicados los estilos

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

  content: ["./views/\*\*/\*.pug"],

  theme: {

    extend: {},

  },

  plugins: [],

}

En el package.json se deberá crear el script para correr los estilos

"scripts": {

    "start": "node index.js",

    "server": "nodemon index.js",

    "css": "postcss public/css/tailwind.css -o public/css/app.css --watch"

  },

La línea –watch es para poder ver los cambios en tiempo real, se deberá correr en una segunda terminal para evitar estar parando el servidor.

Si el npm run css no corre la extensión de los archivos postcss.config y el tailwind.config deberán quedar cjs.

**ORM**

Para la instalación de la base de datos se deberán instalar sequalize y mysql con el comando

Npm i sequalize mysql2

Se deberá crear una carpeta con el nombre config para la conexión a la base de datos

Dentro de la carpeta config se creará el archivo db.js

import Sequalize, { Sequelize } from 'sequelize'

const db = new Sequelize('bienesraices','root','' ,{

    host:'localhost',

    port:3306,

    dialect:'mysql',

    define:{

        timestamps:true

    },

    pool:{

        max:5,

        min:0,

        acquire:30000,

        idle:10000

    },

    operatorAliases:false

});

export default db;

Importación de sequalize

Nombre de la base de datos, usuario y contraseña

Host donde se hospeda el proyecto

Dialect el servidor de base de datos

Timestamps para agregar 2 columnas extra para creación y actualización de la tabla de usuarios

Max cantidad de conexiones, min 0, acquire tiempo para conectarse antes de un error, idle detecta que si no hay movimiento para finalizar conexión a base de datos.

Aliases capacidad mysql desactivar y finalmente exportar la variable de la conexión.

En el index.js se deberá configurar la conexión

Se importará la conexión

import db from './config/db.js'

La conexión con un try

try {

    await db.authenticate();

    console.log('Conexión correcta a la base de datos')

} catch (error) {

    console.log(error)

}

Se deberá proteger las credenciales de las bases de datos con variables de entorno

Se deberá instalar dotenv con el comando npm i dotenv

Se deberá crear un archivo .env , dentro del archivo se crean las variables y se reemplazan en el archivo config

BD\_NOMBRE=bienesraices

BD\_USER=root

DB\_PAASS=

DB\_HOST=localhost

import Sequalize, { Sequelize } from 'sequelize'

import dotenv from 'dotenv'

dotenv.config({path:'.env'})

const db = new Sequelize(process.env.BD\_NOMBRE,process.env.BD\_USER,process.env.DB\_PAASS ?? '' ,{

    host:process.env.DB\_HOST,

    port:3306,

    dialect:'mysql',

    define:{

        timestamps:true

    },

    pool:{

        max:5,

        min:0,

        acquire:30000,

        idle:10000

    },

    operatorAliases:false

});

export default db;

se debe importar dotenv y dar su ubicación.

**MODELOS**

Se creará una carpeta con el nombre models y se creará el primer modelo

import {DataTypes} from 'sequalize'

import db from '../config/db.js'

const Usuario = db.define('usuarios',{

nombre:{

    type:DataTypes.STRING,

    allowNull:false

},

email:{

    type:DataTypes.STRING,

    allowNull:false

},

password:{

    type:DataTypes.STRING,

    allowNull:false

},

token:DataTypes.STRING,

confirmado: DataTypes.BOOLEAN,

})

export default Usuario

Coumunicacion vista, routing y controller

1. Se creará la nueva ruta en routes y se importará de una vez
2. En el controlador se creará la función y se exportará para tomarla desde el router
3. En la vista se llama al método para invocar la ruta por post

(method='POST' action="/auth/registro")

Para capturar los datos de un formulario en el index.js se debe agregar

app.use(express.urlencoded({extended:true}))

**IMPLEMENTACION DE LOS MODELOS**

En el controlador se deberá importar el modelo a utilizar