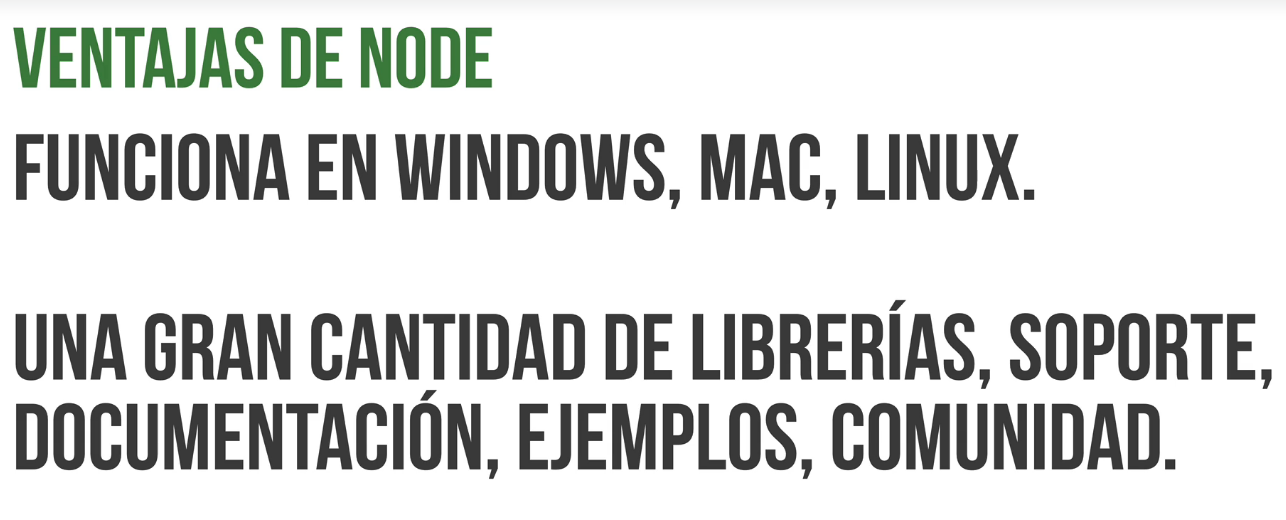
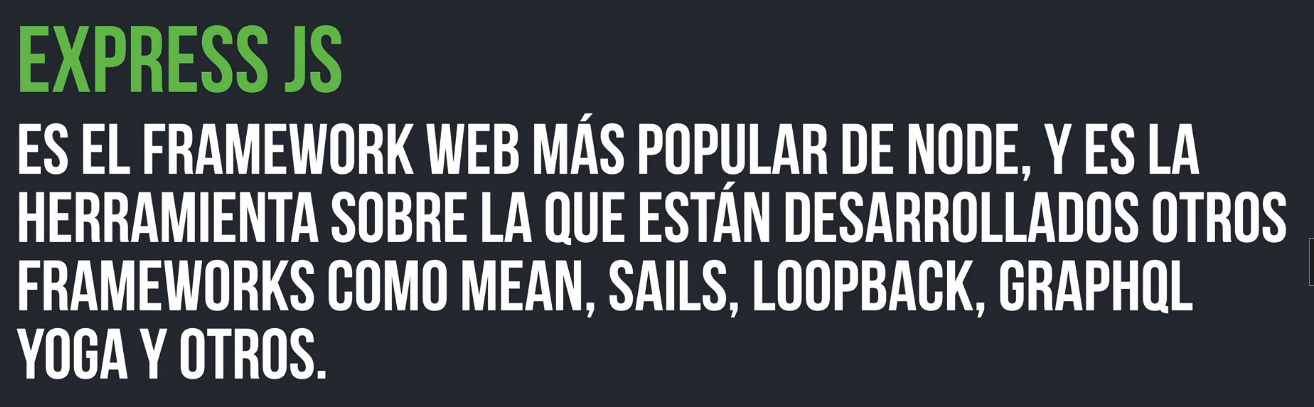
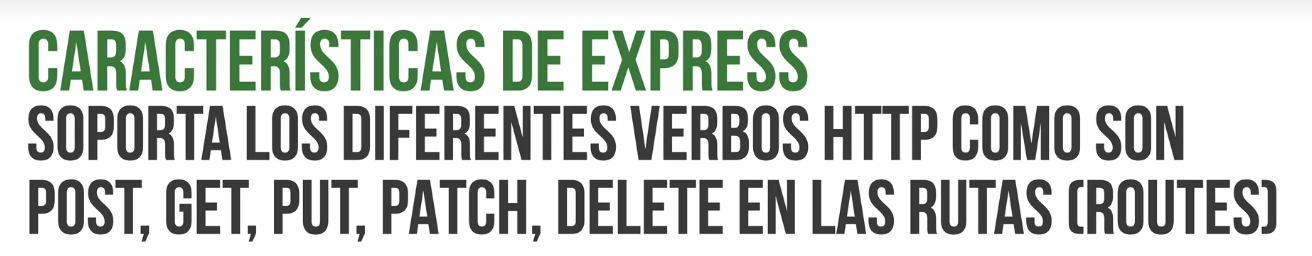


Npm install y el nombre del paquete para instalar un paquete

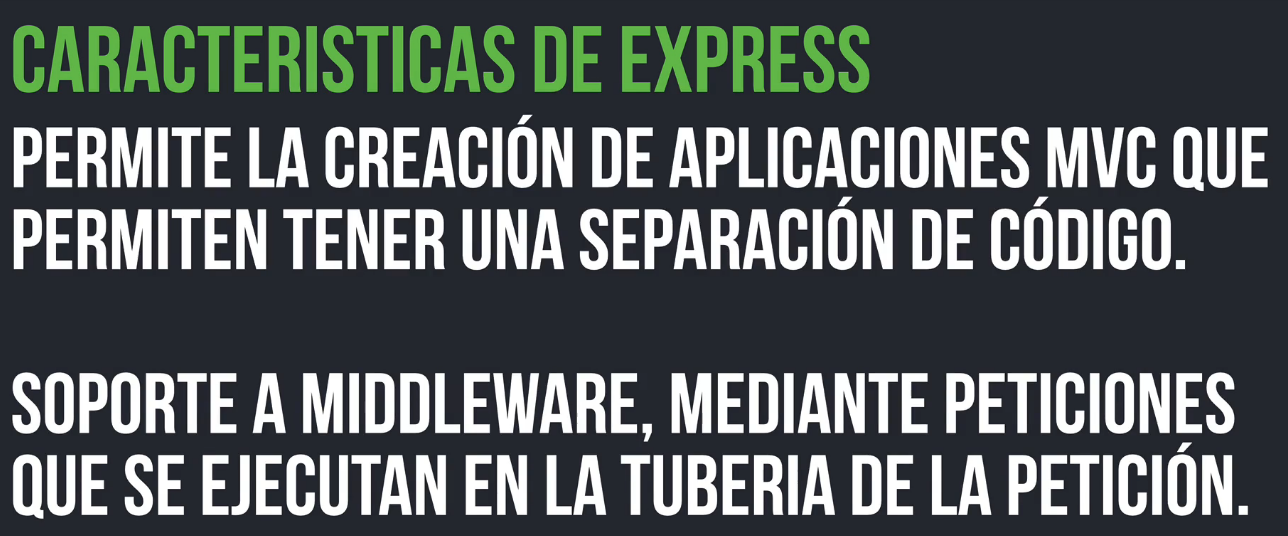


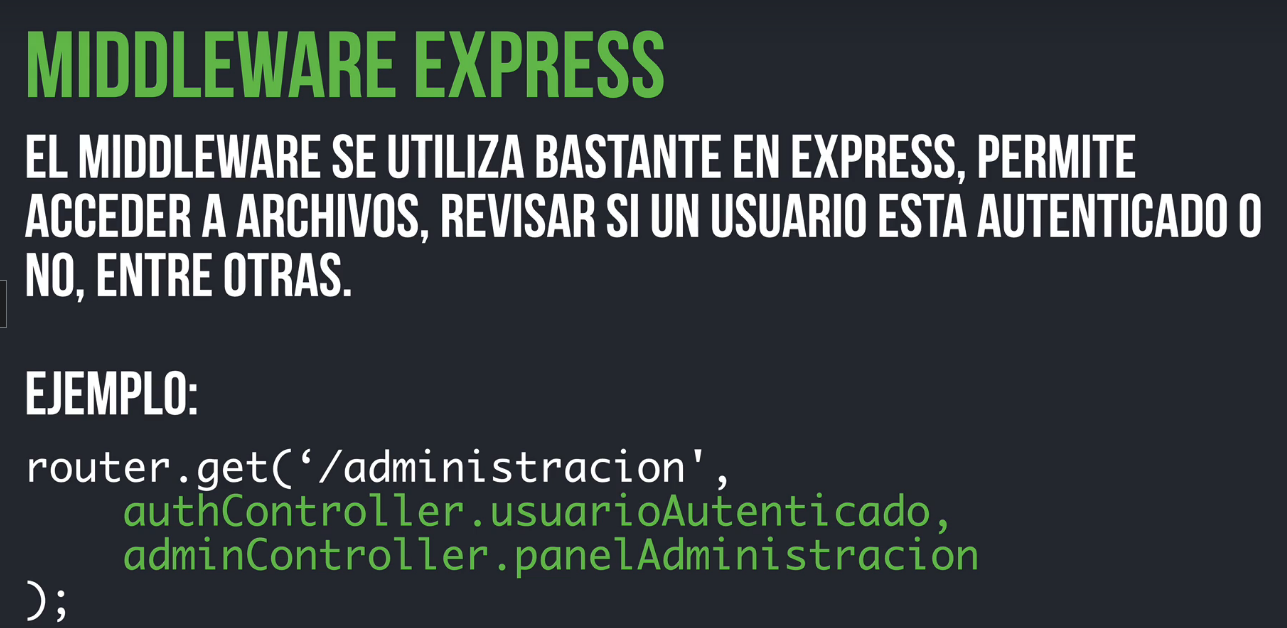
Express:

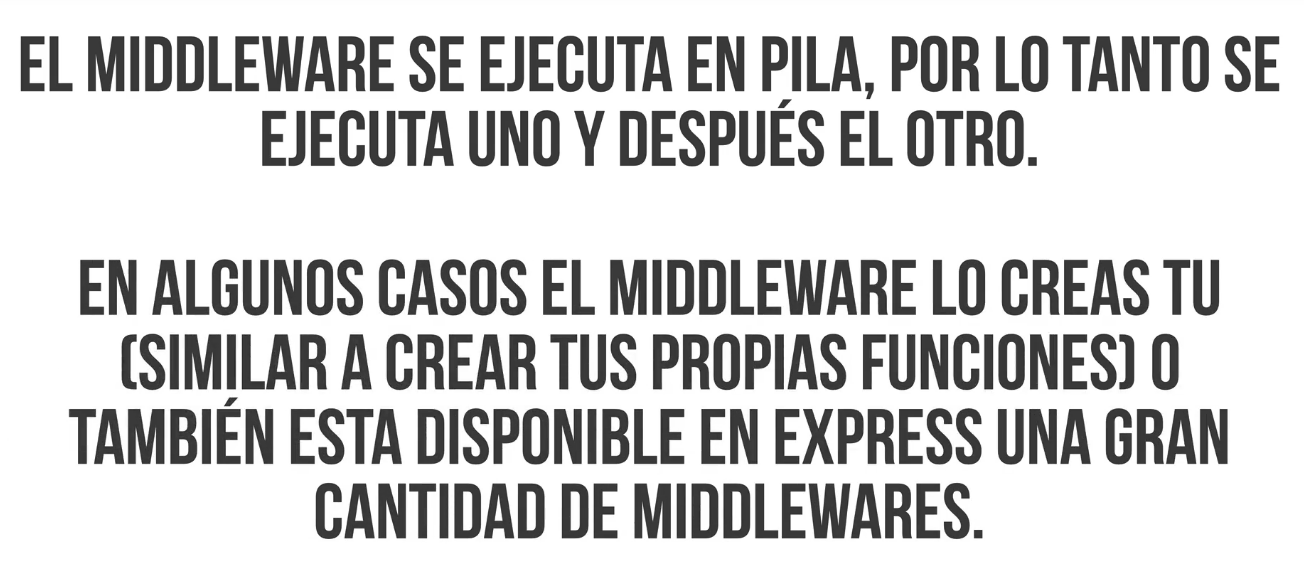
















**Instalamos:**

Abrimos cmd de Windows, vamos a la carpeta donde va a estar el proyecto y:

**Npm init**

Instalar la dependencia express

**Npm install express**

Si se instala así se va a necesitar la dependencia en producción

Con eso en el package.json vamos a ver esto ahora

  "dependencies": {

    "express": "^4.18.1"

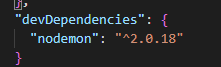
  }

Y se agrega un archivo package-lock

Ese nunca hay que modificarlo

Agregamos una dependencia de desarrollo que es **npm install --save-dev nodemon**

Se agrega esto

****

**Creamos un index.js**

Ahí vamos a tener nuestra aplicación.

En el index ponemos:

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

const app = express(); // funcion para ejecutar express

//Definir puerto

const port = process.env.PORT || 4000; //la primera variable en local no va a existir, va a tomar el segundo

//una vez que se hace el deployment esa variable de arriba si va a existir

app.listen(port, ()=>{ // arranca el servidor

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

})

Como ejecutamos el index.js para arrancar el servidor?

Eso lo haces desde el package.json

Nodemon habilita que cada vez que se hagan cambios en el index.js se reinice el servidor

En la parte de scripts se pone esto:

  "scripts": {

    "dev" : "nodemon index.js"

  },

Entonces ahí en el cmd podemos correr **npm run dev**

**Habilitar sintaxis de imports y exports:**

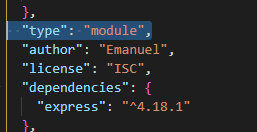
En vez de hacer esto

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

Podemos hacerlo con import

import  express  from 'express';

y en el package.json se agrega el type module



En el index.js:

Res.send es para imprimir algo en la pagina,

Res.json es para responder con un json

Res.render se usa para mostrar una vista

//express soporte, get, post, delete etc.

// la barra diagonal simboliza entiendo la pagina principal

// en el callback va el request (lo que yo envio) response es lo que express te envia

// el get se ejecuta cuando visitas una url

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

    res.send('Inicio');

})

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

    res.send('Nosotros');

})

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

    res.send('Contacto');

})

Entonces por ejemplo si poner en la url /nosotros se va a imprimir en la pantalla nosotros, si pones /contacto se va a imprimir contacto, y así

Igual, lo ideal es llevar estas rutas a otro archivo para un mejor orden.

En PUG para poner un div con una clase haces:

header

    div(class="navegacion")

o

header

    div.navegacion

o

header

    .navegacion

Ejemplo

header

    .navegacion

        .row.justify-content-center.justify-content-md-md-between.align-items-center.py-4

            .col-md-4

                img(src="/img/logo.svg", alt="sitio logo")

            .col-md-8

Se indenta para decir que esta dentro de el anterior

**MiddleWare:**

Vamos a crear nuestro propio middleware En el index.js y después de habilitar pug vamos a obtener el año actual.

//obtener el año actual

app.use( (req, res, next)=> { // req lo que se envia al servidor, res lo que responde, next para pasar al siguiente middleware

   res.locals.unaVariable = 'una variable'; // para usar una variable en una vista se usa res.locals

    return next();

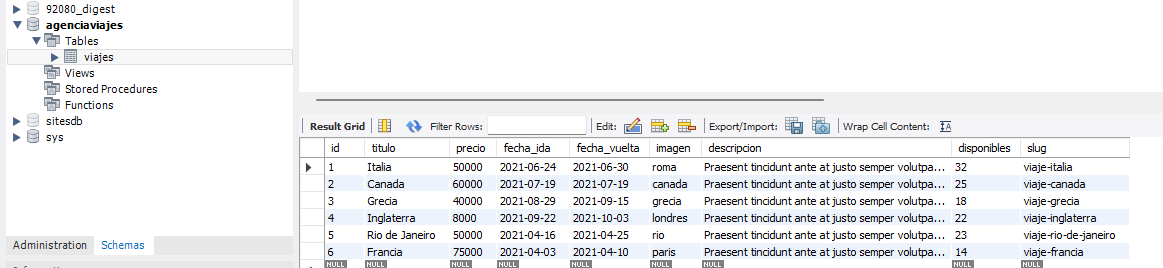
})

Para asignar en una vista se hace:

span= unaVariable

**Conexión a la base de datos:**

Tenemos creada la base de datos agenciaviajes, y la tabla viajes



En la consola en la ruta del proyecto ponemos

Npm install mysql2 sequelize

En el raíz creamos una carpeta llamada config y dentro creamos un archivo llamado db.js

En ese archivo hacemos esto:

import Sequelize from 'sequelize';

//primero va el nombre de la db, despues el nombre de usuario,

//despues la password, despues una serie de configuraciones

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

    host: '127.0.0.1',

    port: '3306',

    dialect: 'mysql',

    define: {

        timestamps: false

    },

    pool: {

        max: 5,

        min: 0,

        acquire: 30000,

        idle: 10000

    },

    operatorAliases: false

})

export default db;

y en el index.js principal ponemos

hacemos el import

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

y abajo de const app= express()

db.authenticate()

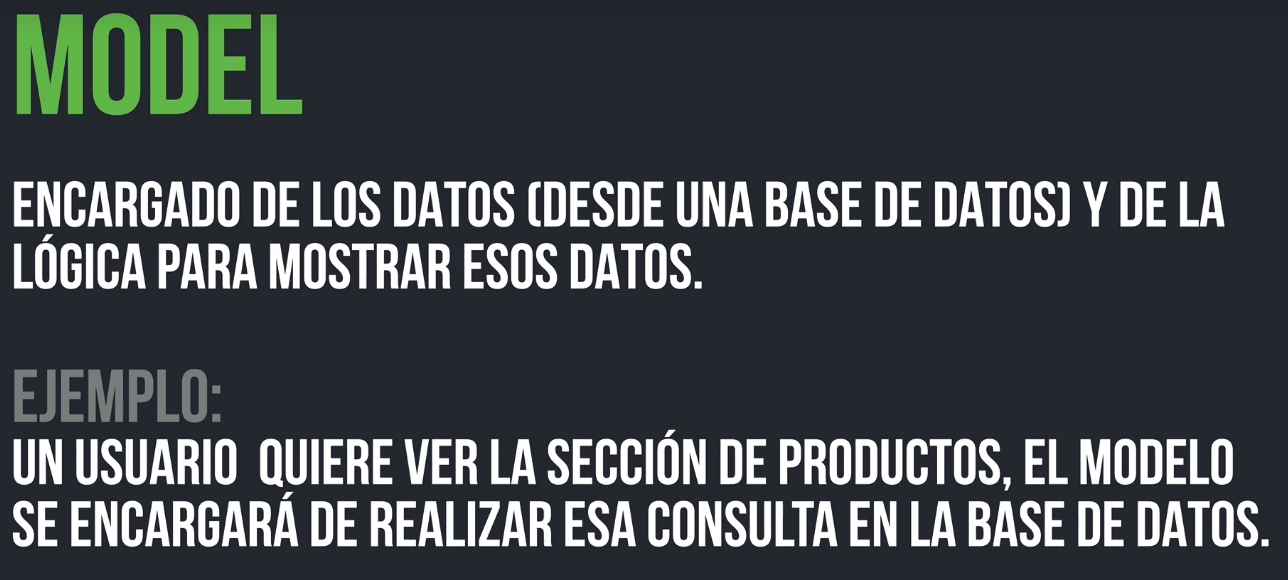
    .then(()=> console.log('Base de datos conectada'))

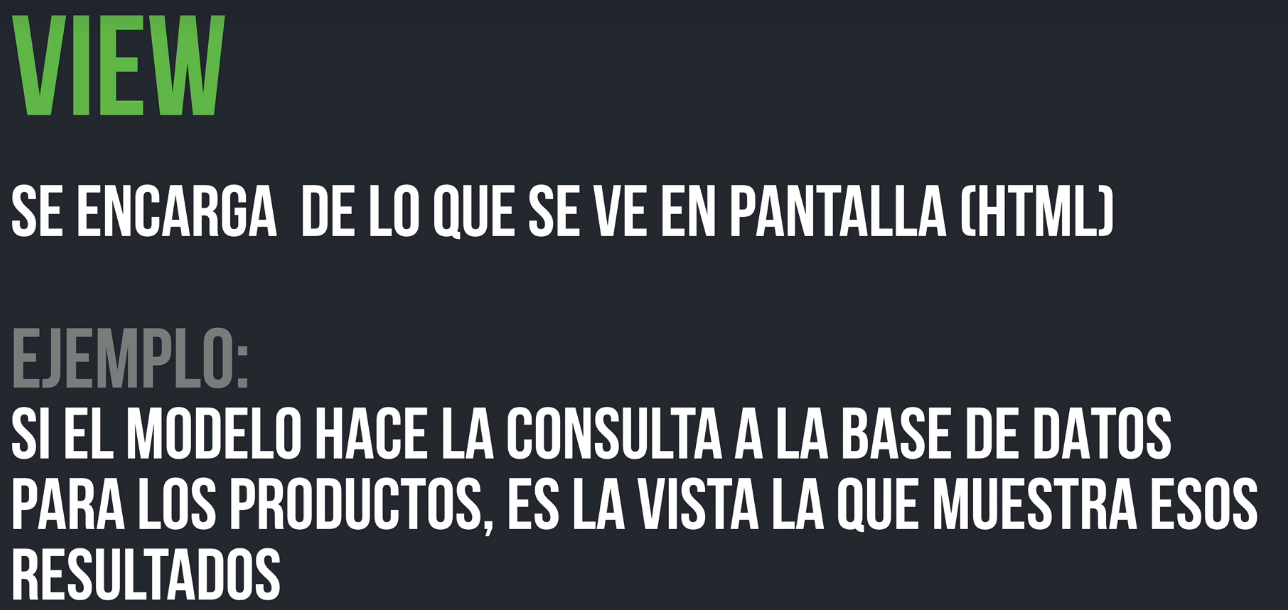
    .catch(error => console.log(error))

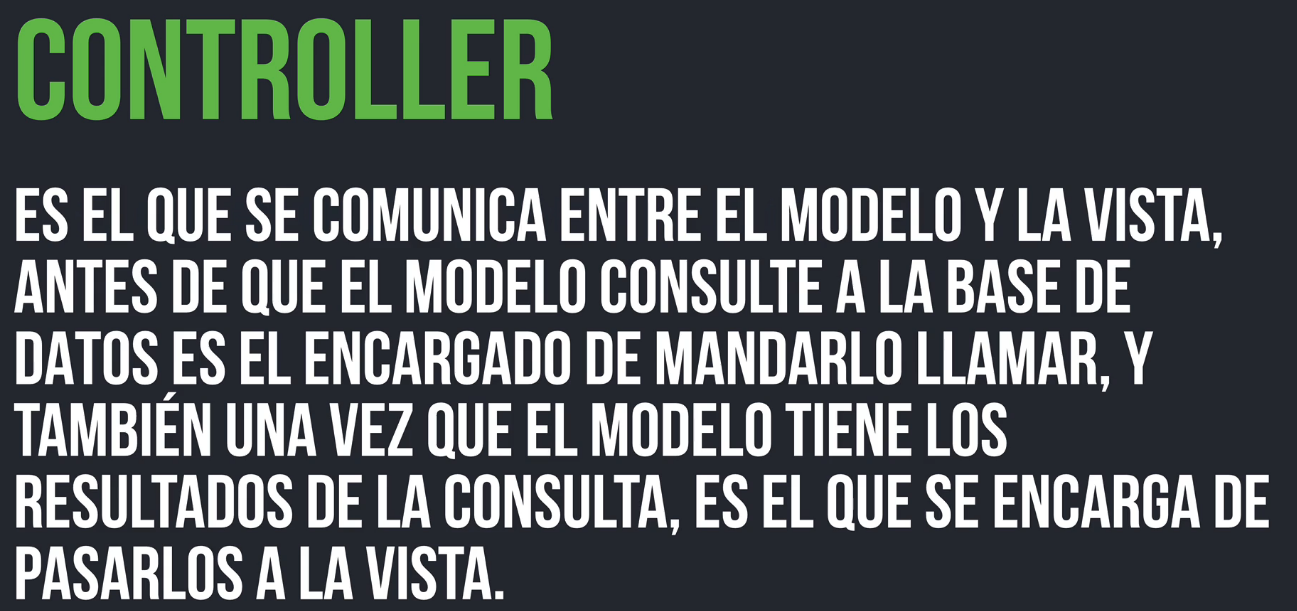
**MVC:**

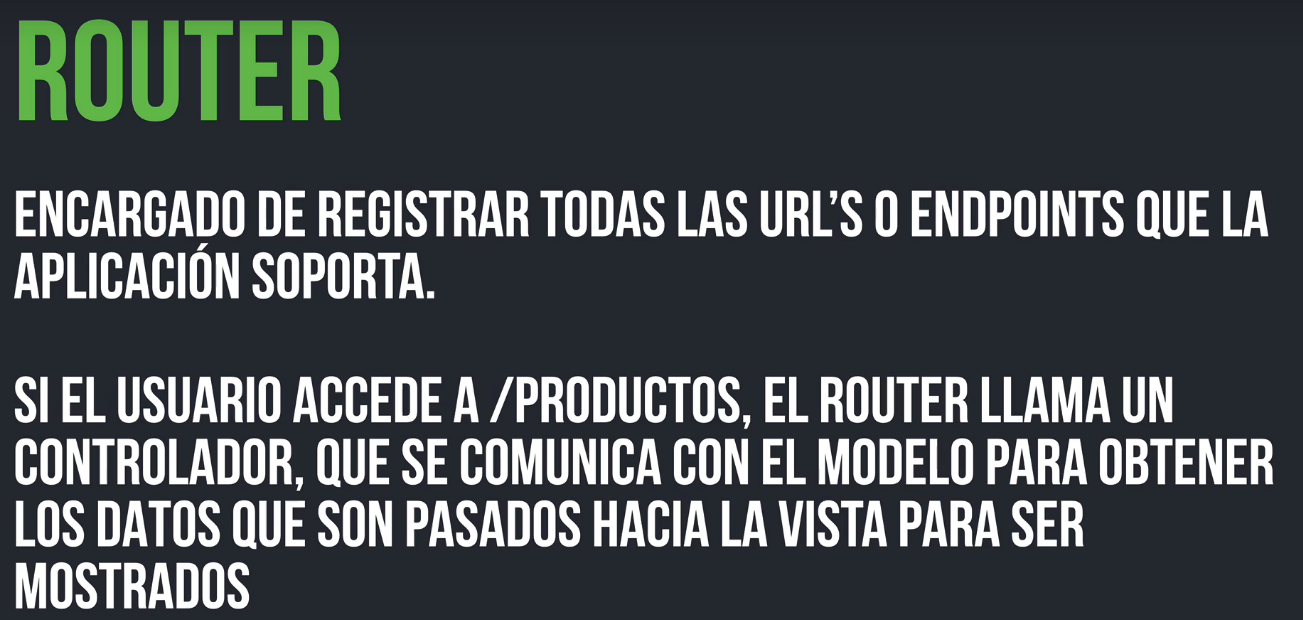
****

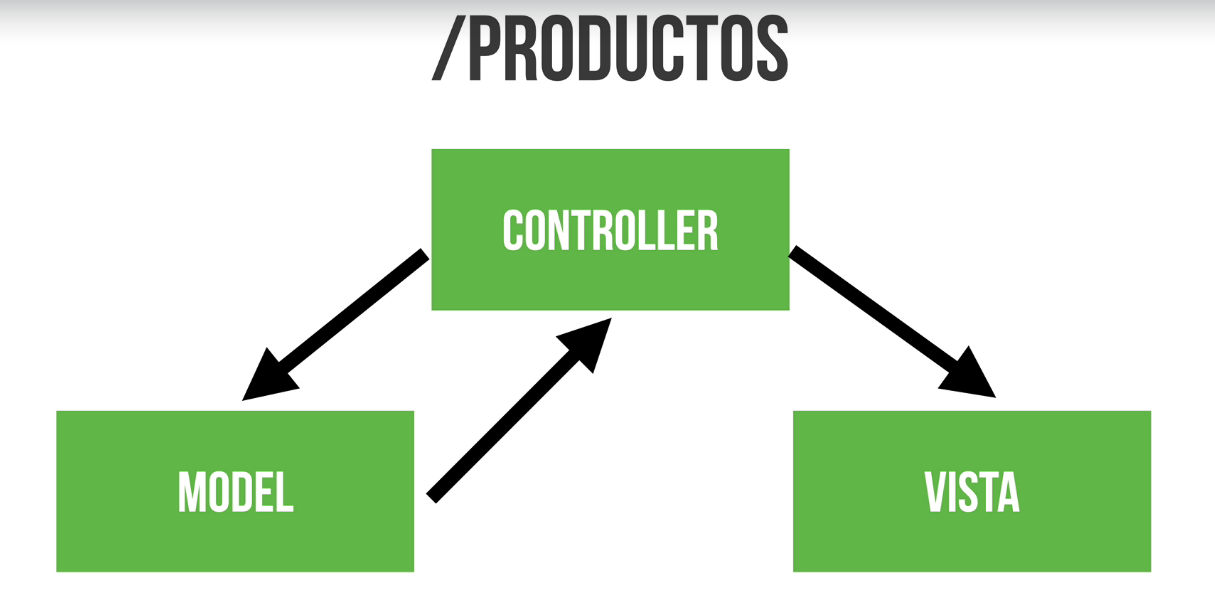
Cada parte de la aplicación hace una cosa, el modelo hace una cosa, la vista hace otra, el routing hace otra y el controlador hace otra











**Creamos un modelo:**

En la carpeta raíz se crea una carpeta llamada models

Y se recomienda los archivos de adentro comenzarlos con una letra mayúscula (Viajes.js)

Hacemos esto dentro de ese archivo:

import Sequelize from 'sequelize';

import db from '../config/db.js'; // la conexion que creamos

export const Viaje = db.define('viajes', { //viajes es el nombre de la tabla

    //ponemos todas las columnas y sus propiedades

    titulo: {

        type: Sequelize.STRING

    },

    precio: {

        type: Sequelize.STRING

    },

    fecha\_ida: {

        type: Sequelize.DATE

    },

    fecha\_VUELTA: {

        type: Sequelize.DATE

    },

    imagen: {

        type: Sequelize.STRING

    },

    descripcion: {

        type: Sequelize.STRING

    },

    disponibles: {

        type: Sequelize.STRING

    },

    slug: {

        type: Sequelize.STRING

    },

})

**Creamos un controlador:**

En routes/index.js teníamos esto que lo de abajo lo haría el controlador

router.get('/', (req, res)=>{ // esto es el routing

    res.render('inicio', { // esto lo haría el controlador

        pagina: 'Inicio'

    });

})

Hacemos una carpeta en el raíz llamada controllers y dentro un file paginasControllers.js

Tomamos el render que teníamos en el index.js y lo guardamos en una variable en el paginasController.js y lo exportamos, queda así:

const paginaInicio =  (req, res)=>{ // esto es el routing

    res.render('inicio', { // esto lo haría el controlador

        pagina: 'Inicio'

    });

}

export {

    paginaInicio

}

Y en el index.js de routes no va a quedar así

import { paginaInicio } from '../controllers/paginasController.js';

router.get('/', paginaInicio)

y lo mismo con el resto, los hacemos todos en el paginasController

Entonces ahí ya cuando vas a la url llama al controlador,

**Consultar a la base de datos con sequilize:**

En paginasController importamos Viaje de viaje.js que es la tabla

import { Viaje } from '../models/Viaje.js'

const paginaViajes = async (req, res)=>{

    //consultar BD

    const viajes = await Viaje.findAll(); //trae todos los resultados de la tabla

    res.render('viajes', {

        pagina: 'Viajes',

        viajes // pasamos el resultado de la consulta a la vista

    });

}

Entonces ya con eso viajes lo podemos usar en la vista, como es un array habría que reccorrerlo, se podría hacer con each:

            each viaje in viajes

                .col-md-6.col-lg-4.mb-4

                    .card

                    .card-body

                        h2 = viaje.titulo // podes acceder a cada columna

**consultar bd de cada viaje según URL visitada:**

En el index.js de las rutas hacemos esto:

El :slug va a tomar como una variable las distintas url que le pasemos

router.get('/viajes/:slug', paginaDetalleViaje); // el :slug es un comodin

En el paginas controller

//Muestra un viaje por su slug

const paginaDetalleViaje = async (req, res) =>{

    // params trae lo que mandas como parametro, el .viaje es lo que pusimos despues de los dos puntos en la ruta viajes/:viaje

    console.log(req.params.viaje)

    const { slug } = req.params;

    try {

        const viaje = await Viaje.findOne({ where : { slug }})

        res.render('viaje', {

            pagina: 'Informacion viaje',

            viaje

        })

    } catch (error) {

        console.log(error)

    }

}

Creamos entonces la vista “viaje.pug” y podemos llamar x ejemplo desde ahí viaje.titulo

**Formularios y validacion:**

Formulario .pug

            .row.justify-content-center.col

                .col-md-8

                    form(action="/testimoniales" method="POST")

                        .form-group

                            label(for="nombre") Nombre

                            input#nombre(type="text", placeholder="Tu nombre", name="nombre", class="form-control")

                        .form-group

                            label(for="correo") Correo electronico

                            input#nombre(type="text", placeholder="Tu Correo", name="correo", class="form-control")

                        .form-group

                            label(for="mensaje") Mensaje

                            textarea#mensaje(name="mensaje", class="form-control", placeholder="Tu mensaje" rows=3)

                        input.btn.btn-success.btn-block(type="submit")

en el action ponemos a donde vamos a enviar los datos, lo ponemos para la misma url.

En las rutas hacemos:

router.post('/testimoniales', guardarTestimonial)

creamos un nuevo controlador en la carpeta llamado testimonialController.js ahí se puede hacer algo así:

const guardarTestimonial = (req, res) =>{

    console.log(req.body); //el body es lo que el usuario pone en el formulario

}

export {

    guardarTestimonial

}

En el index.js general hay que hacer:

//Agregar body parser para leer los datos del form

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

El controlador testimonial va quedando algo así:

const guardarTestimonial = (req, res) =>{

    console.log(req.body); //el body es lo que el usuario pone en el formulario

    //validar

    const {nombre, correo, mensaje} = req.body;

    const errores = [];

    if(nombre.trim() === ''){

        errores.push({mensaje: 'El nombre esta vacio'})

    }

    if(correo.trim() === ''){

        errores.push({mensaje: 'El correo esta vacio'})

    }

    if(mensaje.trim() === ''){

        errores.push({mensaje: 'El mensaje esta vacio'})

    }

    if(errores.length > 0){

        //mostrar la vista con errores

        res.render('/testimoniales', {

            pagina: 'Testimoniales',

            errores,

            nombre, // de este para abajo se envia para que no pierda los datos el usuario si hay un error

            correo,

            mensaje

        })

    }

}

export {

    guardarTestimonial

}

Y en el .pug , el value=algo es para que se vea lo que había escrito si hay error

            .col-md-12

                h2.d-block.w-100.text-center Agrega un testimonial

                if(errores)

                    each error in errores

                        .alert.col.alert-danger.text-center= error.mensaje

            .row.justify-content-center.col

                .col-md-8

                    form(action="/testimoniales" method="POST")

                        .form-group

                            label(for="nombre") Nombre

                            input#nombre(type="text", placeholder="Tu nombre", name="nombre", class="form-control", value=nombre)

                        .form-group

                            label(for="correo") Correo electronico

                            input#nombre(type="text", placeholder="Tu Correo", name="correo", class="form-control", value=correo)

                        .form-group

                            label(for="mensaje") Mensaje

                            textarea#mensaje(name="mensaje", class="form-control", placeholder="Tu mensaje" rows=3)= mensaje

                        input.btn.btn-success.btn-block(type="submit")

**Guardar los datos de el formulario en la BD:**

Hay que crear la tabla donde se van a guardar los testimoniales en la base de datos y el modelo. Vamos a la carpeta models y creamos Testimoniales.js

Nos queda así:

import Sequelize from 'sequelize';

import db from '../config/db.js'; // la conexion que creamos

export const Testimonial = db.define('testimoniales', { //testimoniales es el nombre de la tabla

    //ponemos todas las columnas y sus propiedades

    //poner el ID no es necesario

    nombre: {

        type: Sequelize.STRING

    },

    correo: {

        type: Sequelize.STRING

    },

    mensaje: {

        type: Sequelize.DATE

    }

})

Ese modelo lo importamos en el controlador. Y nos quedaría así completo. Ya con esto se insertan los datos en la BD

import { Testimonial } from '../models/Testimoniales.js';

const guardarTestimonial = async (req, res) =>{

    console.log(req.body); //el body es lo que el usuario pone en el formulario

    //validar

    const {nombre, correo, mensaje} = req.body;

    const errores = [];

    if(nombre.trim() === ''){

        errores.push({mensaje: 'El nombre esta vacio'})

    }

    if(correo.trim() === ''){

        errores.push({mensaje: 'El correo esta vacio'})

    }

    if(mensaje.trim() === ''){

        errores.push({mensaje: 'El mensaje esta vacio'})

    }

    if(errores.length > 0){

        //mostrar la vista con errores

        res.render('/testimoniales', {

            pagina: 'Testimoniales',

            errores,

            nombre, // de este para abajo se envia para que no pierda los datos el usuario si hay un error

            correo,

            mensaje

        })

    }else{

        //almacenarlo en la BD

        try {

            await Testimonial.create({

                nombre,

                correo,

                mensaje

            });

            res.redirect('/testimoniales')// para redirigir y que se borren los cambios

        } catch (error) {

            console.log(error)

        }

    }

}

export {

    guardarTestimonial

}

**Consultar testimoniales de la base de datos:**

Importa el modelo de testimonial a paginas controller

Donde hace el render a la pagina hacemos la consulta a la base de datos y lo guardamos en una variable

const paginaTestimoniales = async (req, res)=>{

    try {

        const testimoniales = await Testimonial.findAll();

        res.render('testimoniales', {

            pagina: 'Testimoniales',

            testimoniales

        });

    } catch (error) {

        console.log(error);

    }

}

En el pug los mostramos así:

        blockquote.text-center Lee sobre nuestros clientes y sus experiencias

        .row.testimoniales

            each testimonial in testimoniales

                .col-md-6.col-lg-4.mb-4

                    .card

                        .card-body

                            blockquote.blockquote

                            p.mb-0

                                | #{testimonial.mensaje}

                            footer.blockquote-footer

                                | #{testimonial.nombre}

**Problema de enfoque async await:**

Si tenemos que hacer dos consultas, por ejemplo a viajes y testimoniales para pasarlos, primero va a esperar que se termine el primer await y después espera a que se termine el segundo await. Entonces puede demorar bastante en mostrar todos los datos

    try {

        const viajes = await Viaje.findAll({limit: 3});

        const testimoniales = await Testimoniales.findAll({limit: 3});

        res.render('inicio', { // esto lo haría el controlador

            pagina: 'Inicio',

            clase: 'home',

            viajes,

            testimoniales

        });

    } catch (error) {

        console.log(error);

    }

Lo ideal seria que las dos consultas se ejecuten al mismo tiempo. Para que se ejecuten al mismo tiempo se puede hacer con un promise, creamos un array en donde guardamos cada una de las consultas, y después hacemos un await Promise.all( array )

Va a quedar así:

const paginaInicio =  async (req, res)=>{ // esto es el routing

    const promiseDB = [];

    promiseDB.push(Viaje.findAll({limit: 3}));

    promiseDB.push(Testimoniales.findAll({limit: 3}));

    //consultar 3 viajes del modelo viaje

    try {

        const resultado = await Promise.all( promiseDB );

        res.render('inicio', { // esto lo haría el controlador

            pagina: 'Inicio',

            clase: 'home',

            viajes: resultado[0],

            testimoniales: resultado[1]

        });

    } catch (error) {

        console.log(error);

    }

}

**DEPLOYMENT:**

**Que son y como usar las variables de entorno:**

E¿Una variable de entorno es distinta dependiendo si estas en desarrollo o producción

Para eso instalamos una librería que se llama dotenv

Hacemos

Npm i dotenv

Creamos un archivo llamado .env en el raíz. Dentro le ponemos:

DB\_HOST=127.0.0.1

Así pelado

En el db.js hacemos un import de dotenv

Haciendo esto ya va a leer el archivo .env

import dotenv from 'dotenv';

dotenv.config()

si lo haces asi: no hace falta poner el dotenv.config()

import dotenv from 'dotenv/config';

para llamar la variable ahi se hace:

console.log(process.env.DB\_HOST)

**Añadir las variables de entorno al proyecto:**

En el .env definimos las variables que necesitamos para la conexión a la BD

DB\_NAME=agenciaviajes

DB\_USER=root

DB\_PASSWORD=

DB\_HOST=127.0.0.1

Y las reemplazamos en el archivo de base de datos

//primero va el nombre de la db, despues el nombre de usuario,

//despues la password, despues una serie de configuraciones

const db = new Sequelize(process.env.DB\_NAME, process.env.DB\_USER, process.env.DB\_PASSWORD, {

    host: process.env.DB\_HOST,

    port: '3306',

    dialect: 'mysql',

    define: {

        timestamps: false

    },

    pool: {

        max: 5,

        min: 0,

        acquire: 30000,

        idle: 10000

    },

    operatorAliases: false

})

**Subir repo a github, crear gitignore:**

Se crea un repo nuevo de github

Git init

Cambia el nombre del Branch master por main

Crea el archivo .gitignore en el raíz

Dentro le pones los archivos o carpetas que queres que se ignoren

.env

node\_modules/

git add .

git commit -m “primer commit”

git branch -M main

para decirle que el código pertenece a ese repo:



Git push -u origin main

**Subir codigo a Heroku:**

Heroku es para subir codigo gratis

Hay que instalar heroku CLI

Después hay que enlazar la cuenta con la terminal:

En la terminal hay que poner

heroku login

presionar cualquier tecla y abre pestaña en el navegador para conectarse

en la terminal desde la carpeta del proyecto hacemos

git push heroku main

en heroku vamos a more- view logs y también a run console

nosotros teníamos un script de nodemon que no es para producción entonces vamos al package.json

  "scripts": {

    "dev": "nodemon index.js",

    "start": "node index.js" //agregamos esto

  },

Se hace el commit de nuevo a github

Y git push heroku main

En la consola de heroku se puede hacer **npm run start**

Y ahí te tira mas detallado si hay errores

Falta subir la base de datos

**Subir base de datos y configurar variables:**

Vas a la app en heroku y la pestaña resourses

En add-ons buscas mysql y seleccionamos clearDB la primera opción es gratis.

Ahora vamos a la pestaña settings y reveal config vars

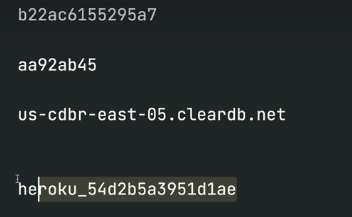
Copiamos el valor pintado ahí



Lo pego en un archivo nuevo sin nombre

Borra el mysql://

El primer valor es el usuario hasta los dos puntos ahí le damos enter borramos los dos puntos, hasya el arroba es la contraseña después esta el dominio, el arroba se puede borrar hasta la diagonal y lo ultimo es el nombre de la base de datos la diagonal la borramos al final el signo de interrogación y el reconnect lo borramos. queda así:



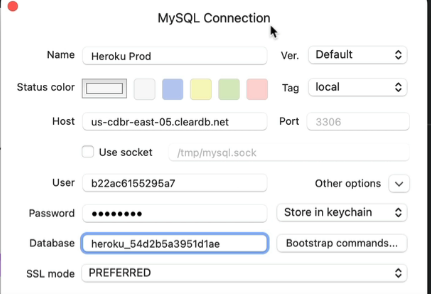
Tenemos que ir ingresando las variables de la base de datos según como esta en el archivo que definimos las variables de entorno (.env)

Queda así:



Desde la base de datos tenemos que exportar en sql las tablas

Creamos una conexión nueva desde mysql, le damos un nombre a la conexión : Heroku prod, pegamos el host que pusimos en heroku el puerto es 3306, nombre de usuario de heroku, password de heroku y base de datos



Se puede hacer en workbench

Entras a la conexión y le das a import from sql e importamos las tablas que habíamos exportado