Vamos a usar las tecnologias Express: Será nuestro servidor local. Y También Sequelize

Para manejar la base de datos

**INICIANDO PROYECTO EN NODEJS**

Vamos a comenzar instalar algunas dependencias, y el que vamos a usar es package .json con el comando

**npm init**

Nos hace un serie de preguntas, Name del packge, versión, description, autor, lciencia, Git Repository, etc.

Y con eso todos trabajamos con la misma versión y paquetes.(empresa)

Almacena las versiones de los demás programas también.

**INSTALANDO DEPENDENCIAS**

Para **instalar Express** con el comando

npm install --save nodemon body-parser express

body parser es requerido por node para poder presentar cierta info, nodemon es el servidor si detecta cambios en este lo recarga y express el el servidor que se ocupará localmente( como el apache en php) .

{

  "name": "intronodejs",

  "version": "1.0.0",

  "description": "NodeJs Express y Sequelize",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "Mario Garcia",

  "license": "ISC",

Dependencias

  "dependencies": {

    "body-parser": "^1.19.0",

    "express": "^4.17.1",

    "nodemon": "^2.0.4"

  }

}

Si borro la carpeta de node\_modules y pacagge-lick.jason , ejecutando

**Npm install** sobre el proyecto que tenga packge.json instalara todo nuevamente con las versiones y las dependencias que están en este.json

**CONFIGURANDO EL SERVIDOR EXPRESS**

Creamos dos caprtas nuevas, “public” para guardar los archivos dinámicos y otra “server”, en server vamos a colocar nuestros modelos, rutas y archivos de configuración asi como las vistas. Luego un archivo index,js dentro de esta ultima caprta que será la que tenga la configuración del servidor (donde van enlazadas las vistas, configuración, En este archivo vamos a crear nuestro servidor

Los vamos a hacer de una manera antigua ya hoy en dia solo se usan export e import. Antes se usaba require para importar.

// importar express

const express = require('express');

// configurar express

const app = express();

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

    res.send('Hola Mundo en NodeJs')

});

app.listen(3000);

express es donde se guarda el servidor que usaremos como una función.

App guarda el resultado de la función express,

Use indica que se usa cuando este cargado en el home ‘/’ , vamos atener un request y un response. Y el método send envia la respuesta de res.

Listen es el puerto donde se ejecutará el programa. (en este caso el puerto 3000)

Ahora ya tenemos el servidor pero hay que conectarlo con el proyecto que estamos llevando a cabo.

En el archivo package.json en los scripts vamos acrear uno para que inicie el servidor e indicarle en que carpeta se encuentra dicha configuración.

"scripts": {

    "start" : "nodemon server"

  },

Nodemos lo que hace es reiniciar el servidor, detecte cambios también y server es la carpeta donde se encuentra la configuración.

Lo ejecutamos con el comando

**npm run start**

\*\*\*Nota lo que hace **response y request** (es que request es lo que le vamos a pedir al servidor y response es precisamente lo que el servidor nos devuelve) y el método **send** es para poder imprimir algo en la pantalla. **Console.log** en este caso nos imprime en la consola de la computadora no en la del explorador.

La diferencia entre el método use, y get es que use responde a todos los métodos http

**CREANDO RUTAS**

En el archivo de index.js donde viene cargado el servidor express y la rita de inicio podemos crear las demás. Pero para no cragar tanto el archivo index.js con tantas rutas creamos una capeta dentro de server con el nombre de router y vamosa crear los achivos de las rutas ahí, importante el servidor y la función router para express, ya no se usa app.get si no router.(método http).

const express = require('express');

const router = express.Router();

//exporta las rutas

module.exports = function(){

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

    res.send('Hola Mundo en NodeJs')

});

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

    res.send('Hola Mundo en NodeJs')

});

return router; //para hacerlo disponible en el otro metodo

}

Y en el archivo index donde tenemos cargado el express importamos las rutas

const routes = require('./routes');

usamos use en este para que no tenmga problema en cada tipo http de las diferentes páginas. Como primer argumento la pagina principal y segunda por que export es un metodo

app.use('/',routes());

// importar express

const express = require('express');

const routes = require('./routes');

// configurar express

const app = express();

app.use('/',routes());

app.listen(3000);

**CREANDO VISTAS**

Ahora bien, temenos que introducer HTML, CSS, Baes de Datos etc. En exprees se usa lo que se llaman **TEMPLATES ENGINES**

These template engines work “out-of-the-box” with Express:

* [**Pug**](https://github.com/pugjs/pug): Haml-inspired template engine (formerly Jade).
* [**Haml.js**](https://github.com/tj/haml.js): Haml implementation.
* [**EJS**](https://github.com/tj/ejs): Embedded JavaScript template engine.
* [**hbs**](https://github.com/pillarjs/hbs): Adapter for Handlebars.js, an extension of Mustache.js template engine.
* [**Squirrelly**](https://github.com/squirrellyjs/squirrelly): Blazing-fast template engine that supports partials, helpers, custom tags, filters, and caching. Not white-space sensitive, works with any language.
* [**Eta**](https://github.com/eta-dev/eta): Super-fast lightweight embedded JS template engine. Supports custom delimiters, async, whitespace control, partials, caching, plugins.
* [**React**](https://github.com/reactjs/express-react-views): Renders React components on the server. It renders static markup and does not support mounting those views on the client.
* [**h4e**](https://github.com/louischatriot/h4e): Adapter for Hogan.js, with support for partials and layouts.
* [**hulk-hogan**](https://github.com/quangv/hulk-hogan): Adapter for Twitter’s Hogan.js (Mustache syntax), with support for Partials.
* [**combyne.js**](https://github.com/tbranyen/combyne): A template engine that hopefully works the way you’d expect.
* [**swig**](https://github.com/paularmstrong/swig): Fast, Django-like template engine.
* [**Nunjucks**](https://github.com/mozilla/nunjucks): Inspired by jinja/twig.
* [**marko**](https://github.com/marko-js/marko): A fast and lightweight HTML-based templating engine that compiles templates to CommonJS modules and supports streaming, async rendering and custom tags. (Renders directly to the HTTP response stream).
* [**whiskers**](https://github.com/gsf/whiskers.js): Small, fast, mustachioed.
* [**Blade**](https://github.com/bminer/node-blade): HTML Template Compiler, inspired by Jade & Haml.
* [**Haml-Coffee**](https://github.com/netzpirat/haml-coffee): Haml templates where you can write inline CoffeeScript.
* [**Webfiller**](https://github.com/haraldrudell/webfiller): Plain-html5 dual-side rendering, self-configuring routes, organized source tree, 100% js.
* [**express-hbs**](https://github.com/barc/express-hbs): Handlebars with layouts, partials and blocks for express 3 from Barc.
* [**express-handlebars**](https://github.com/ericf/express-handlebars): A Handlebars view engine for Express which doesn’t suck.
* [**express-views-dom**](https://github.com/AndersDJohnson/express-views-dom): A DOM view engine for Express.
* [**rivets-server**](https://github.com/AndersDJohnson/rivets-server): Render Rivets.js templates on the server.
* [**Exbars**](https://github.com/YoussefKababe/exbars): A flexible Handlebars view engine for Express.
* [**Liquidjs**](https://github.com/harttle/liquidjs): A Liquid engine implementation for both Node.js and browsers.
* [**express-tl**](https://github.com/Drulac/express-tl): A template-literal engine implementation for Express.
* [**vuexpress**](https://github.com/vuexpress/vuexpress): A Vue.js server side rendering engine for Express.js.
* [**Twing**](https://www.npmjs.com/package/twing): First-class Twig engine for Node.js.

(Los que vamos usar va hacer pug, EJS,hbs)

Instalamos la dependencia pug en la consola

npm install --save pug

Creamos una caprta para las vistas en la caprta server “views” esta tendrá todos los archivos html, dentro de la caprta creamos caprtas con el nombre de los archivos ejemplo.

Views/ index

Views / nosotros

Y cada caprta tendrá un archivo con el la terminación .pug

Index.pug

Nosotros.pug

Luego tenemos que decirle a Node que vamos a usar pug, vamos al archivo de configuración del servidor express (index.js)

Set permite agregar ciertas características. Y view engine por que es una configuración de express

//habilita pug

app.set('view engine','pug');

Luego temenos que indicar donde están los templates (en ese mismo archivo)

app.set('views');

Pero temenos que impoar una librería PATH para que entre al file system de nuestro proyecto

const path = require('path');

Para que al final quede así, \_\_dirname es para que acceda a la carpeta donde nos encontramos y como sewgundo parámetro dónde están los archivos de las vistas pug.

app.set('views',path.join(\_\_dirname,'./views'));

Para escribir en pug solo es necesario la etiqueta un espaico y poner el contenido,

Ejemplo en el archivo index.

h1 Inicio

Ahora en la carpeta de routes en el su archivo nosotros habíamos usado el método

Res.send(‘index) ahora hay que cambiar el método por **res.render(/index)**

Por que ahora toma como parámetro no el texto si no el nombre de la capeta de las vistas.(cuidar minúsculas y mayúsculas)

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

    res.render('index')

});

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

    res.render('nosotros')

});

\*\*en uso tuve que mover el archivo nosotros a la caprta raíz de views

**CREANDO UN MASTER PAGE**

Creamos una carpeta dentro de views que se llame layout, el archivo que tendrá se llama index.pug ahí vamos a importar boostrap

SI pongo la estructura del html en ese archivo me lo acomoda de la siguiente manera

Sin etiquetas de cierre y apertura, luego lo que este indentado debajo de la etiqueta padre será hijo de este.

<!DOCTYPE html>

html(lang="en")

    head

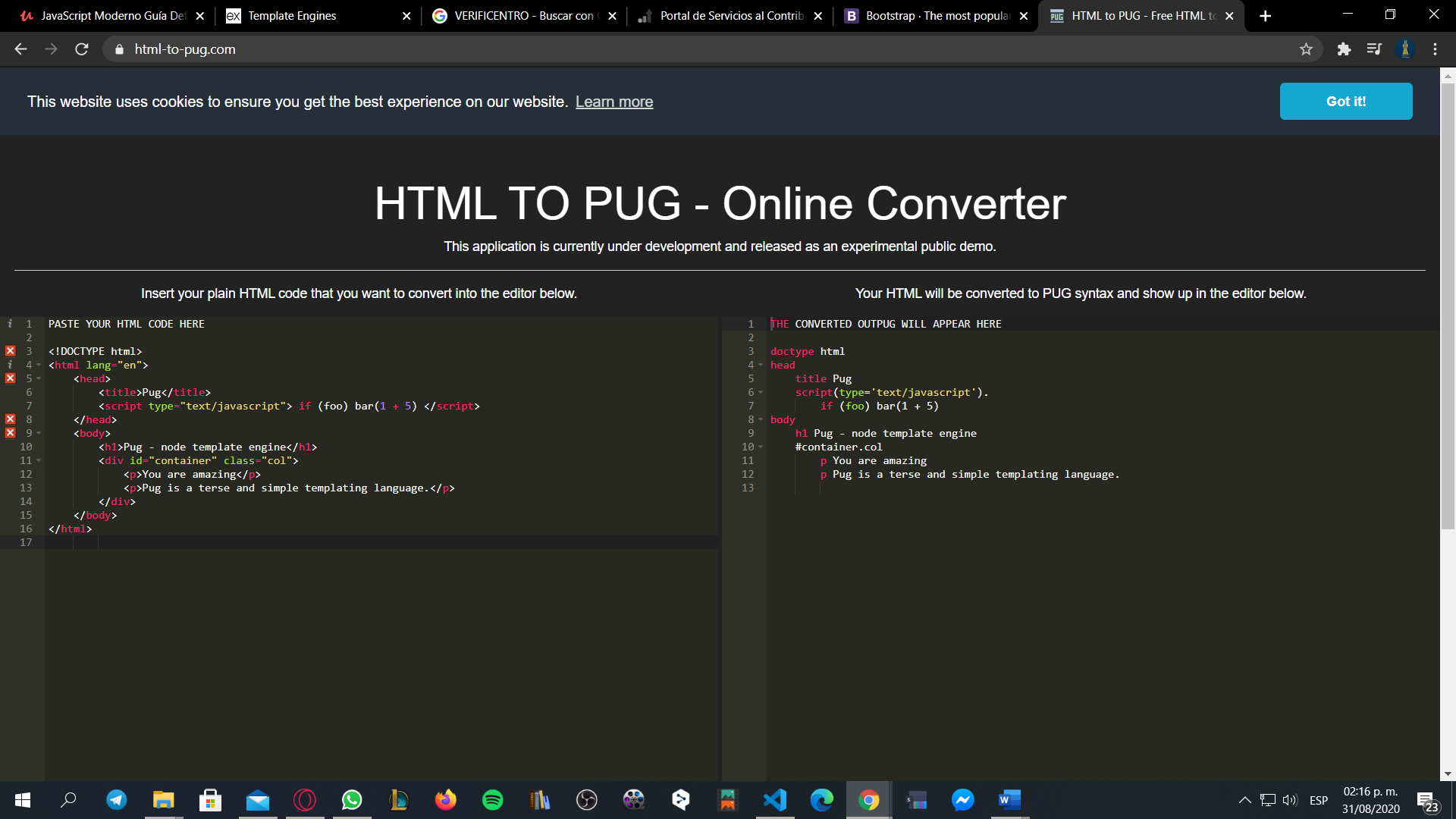
        meta(charset="UTF-8")

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

        title Document

    body

Por eso es que debo de entrar a una pagian que convierte código html a pug



Pegamos el CSS de boostrap y lo convertimos en formato pug (OJO QUE ESTE EN EL MISMO NIVEL)

Pero no basta con eso, tenemos que también importar boostrap para que lo reconozca en las vistas. Tenemos que hacer uso de herencia extends e indicar la ruta del layout para cada una de las vistas ( en sus respectivos archivos).

extends ./layout/index

Pero manda un error que dice “Solamente los bloques pueden aparecer como en el nivel más alto”

Debo de insertar en el layout una plabra reservada “block” y el nombre de la sección que le voy a poner para que se vincule con el archivo “parte” que queremos insertar.

Archivo layout

 link(rel='stylesheet' href='https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css')

        title Document

    body

        block contenido

archivo de nosotros (ojo con la indentacion)

extends ./layout/index

block contenido

      h1 Nosotros

Ya tenemos boostran en el proyecto.

Ahora en la capeta publica del proyecto estran los archivos CSS,JS,IMÁGENES ETC.

Tenemos que decirla Node que ahí va a encontrar esos recursos, tenemos que abrir el archivo de configuración index.js en la capreta raíz para indicárselo.

**CREANDO UN DIRECTORIO PARA LOS ARCHIVOS ESTATICOS**

Pegamos los css y las imágenes en la carpeta public

Usamos el código para cargar es carpeta al proyecto. En el archivo index de configuración del servidor

//carga la carpeta estatica public

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

Lo cargamos en nuestro archivo layout (por que estamos en la capeta publica)

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

**AGREGANDO GOOGLE FONTS AL PROYECTO**

Buscamos en la pagina de fuentes estas 3 fuentes requeridas;

Staatliches

Covered By Your Grace

Open Sans

Las agregamos las 3, y vamos a converetir el código a pug

Y lo pegamos en el layout.(antes de la carga de archivos css)

link(href='https://fonts.googleapis.com/css2?family=Covered+By+Your+Grace&family=Open+Sans:wght@300&family=Staatliches&display=swap' rel='stylesheet')

**Creando un Header**

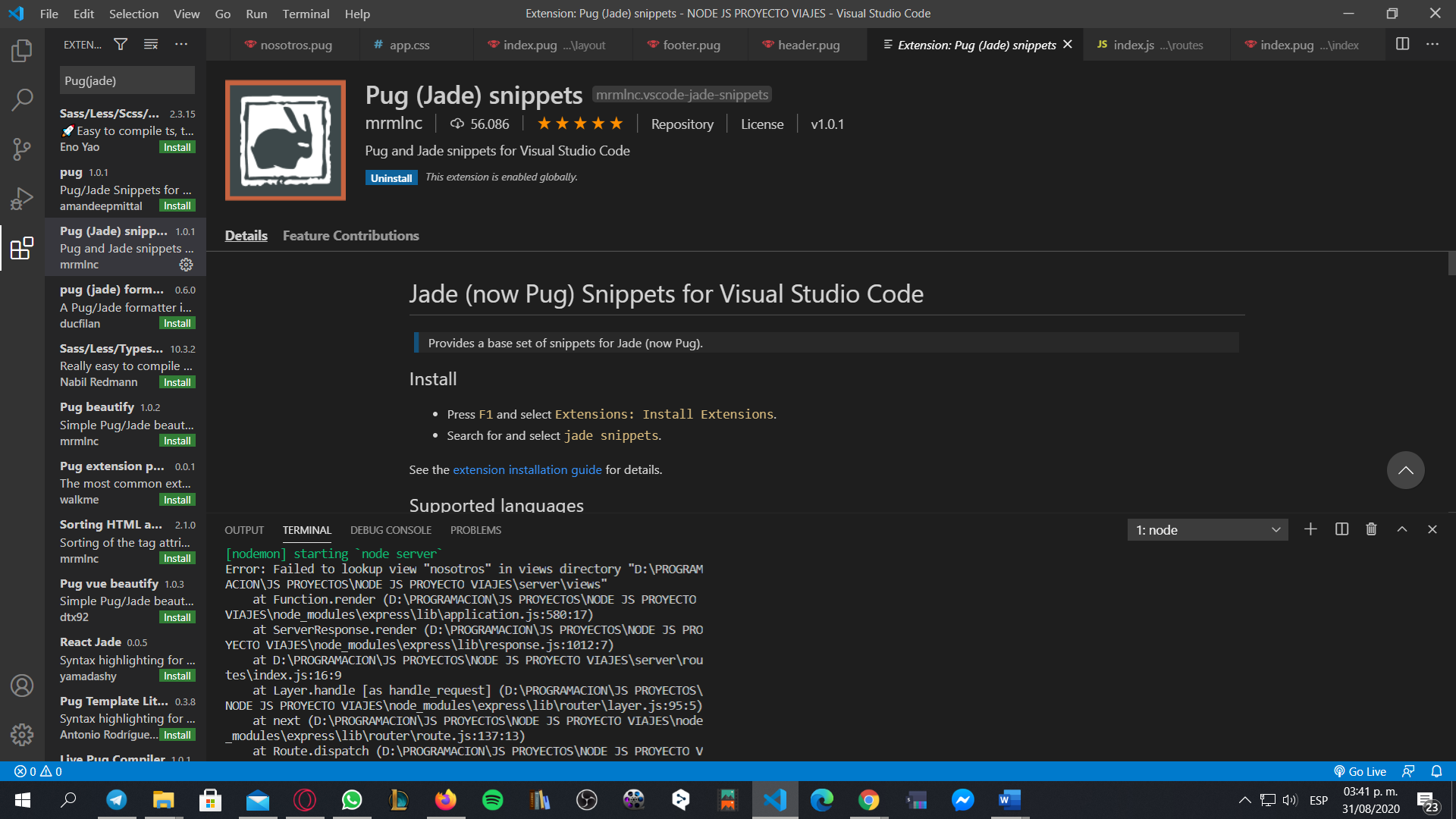
Este es el header que vamos a usar.



Vamos a crear un archive que se llama parshall PARA DIVIDIR MAS MI PROYECTO y el layout no quede tan cargado.

Dentro de la carpeta layout creo otra carpeta con el nombre “includes” que tendrá los archivos header.pug y footer.pug para importarlos en el layout.

Vamos a instalar la extensión en Visual estududio code Pug(Jade) snippets



header

  .navegacion

    .container

        .row

            .col-6.col-md-4

                a(href="/")

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

            .col-6. ol-md-8

                nav.nav.justify-content-center.justify-content-md-end

                    a(href="/nosotros",class="nav-link") Nosotros

                    a(href="/viajes",class="nav-link") Viajes

                    a(href="/testimoniales",class="nav-link") Testimoniales

Ahora lo vamos a poner en el archivo pug layout

La palabra reservada include es para incluir archivos, en archivos pug

 body

        include includes/header

**CREANDO EL FOOTER**

footer.container.py-5

    .row

        .col-md-6

            nav.nav.justify-content-center.justify-content-md-start

                a(href="/nosotros",class="nav-link") Nosotros

                a(href="/viajes",class="nav-link") Viajes

                a(href="/testimoniales",class="nav-link") Testimoniales

        .col-md-6

            p.copyright.text-center.text-md-right

                  | Todos los Derechos Reservados

Asi se manejan los párrafos en pug

Y colocamos el footer en layout

 body

        include includes/header

        block contenido

        include includes/footer

para eliminar el scroll de abajo

meta(name="viewport", content="width=device-width, initial-scale=1.0,shrink-to-fit=no")

Ahora vamos a hacer que se mesutre el año en el footer. En el archivo de configuración del servidos (index,js) vamos a usar la función de obtener año.

//muestra el año actual

app.use((req,res,next) =>{

    const fecha = new Date();

    fecha.getFullYear();

})

Ahora ¿cómo lo paso a los templates? Tenemos que crear variables locales o (locals)para quer estas variables se pasen entre los diferentes archivos

Nombre que queramos

res.locals.fechaActual = fecha.getFullYear();

Next nos sirve para que continue ejecutando la función siguiente

app.use((req,res,next) =>{

    const fecha = new Date();

    res.locals.fechaActual = fecha.getFullYear();

    return next();

})

Para ponerlo en el teamplate de footer se hace de la siguiente forma.

p.copyright.text-center.text-md-right

                  | Todos los Derechos Reservados

                  span= fechaActual

Por que ya es código dinamico con su signo de igual, espacio y el nombre de la variable locals

**PASANDO VARIABLES HACIA LAS VISTAS**

Podemos pasar variables desde el archivo de rutas hacia los templates. Se psasn como un objeto.

router.get('/nosotros', (req,res)=>{

    res.render('nosotros',{

        pagina: 'Sobre Nosotros'

    });

1. Forma; poner como lo hicimos antes

  h1= pagina

1. Forma:

  h1 #{pagina}

Podemos poner el titulo de la pagina de forma dinámica también, para que cada pagina tengo un titulo diferente

 title= pagina

Creamos una carpeta nueva en server, que se llama config y dentro un archivo js index, que va a tener un objeto para exportar a los demas documentos para su titulo.

module.exports = {

    development: {

        nombresitio: 'Agencia de Viajes [Desarrollo]'

    },

    production: {

        nombresitio: 'Agencia de Viajes'

    }

}

Lo importamos en el archivo de configuración del servidor

const configs = require('./config');

Luego le diremos a node que estamos en modo de desarrollo

//validacion de desarrollo o production

const config = configs[app.get('env')]

\*\*después del código de que carga la carpeta publica estatica…

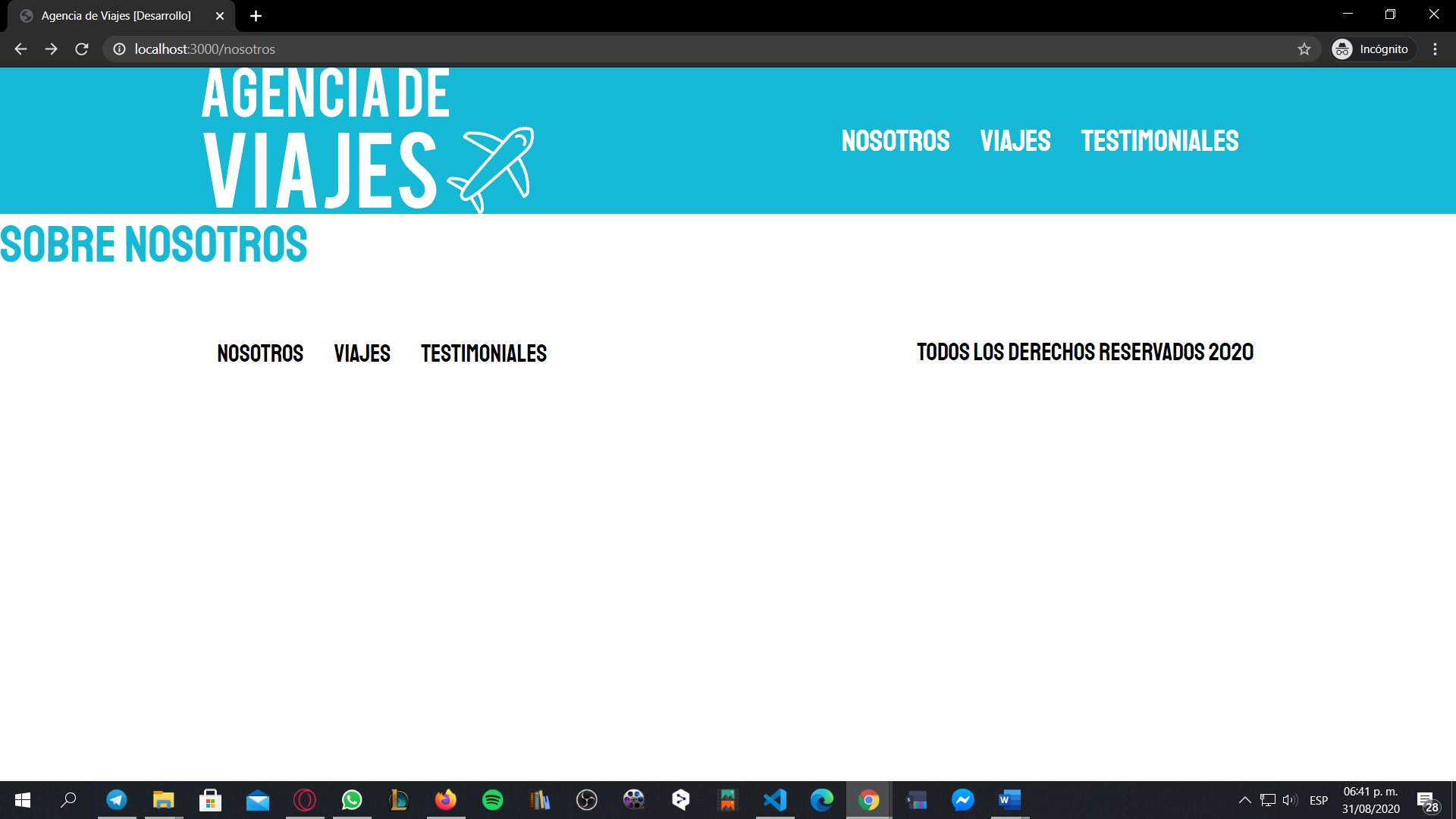
Creamos la variable locals y el valos del objeto.

//creamos la variable para el sitio web

app.locals.titulo = config.nombresitio

Y ya lo podemos poner en los layouts

 title= titulo



Y si escribimos el comando npm start env –production

Cambiará el titulo automáticamente por el de desarrollo

CREANDO LA PAGINA NOSOTROS

Esta página es mas estática por que es pura información

\*\*La class img-fluid hace la imagen responsive

extends ./layout/index

block contenido

   main.container.mt-5

      .row

         .col-md-5

            img(src="img/nosotros.jpg", alt="Imagen de Nosotros", class="img-fluid")

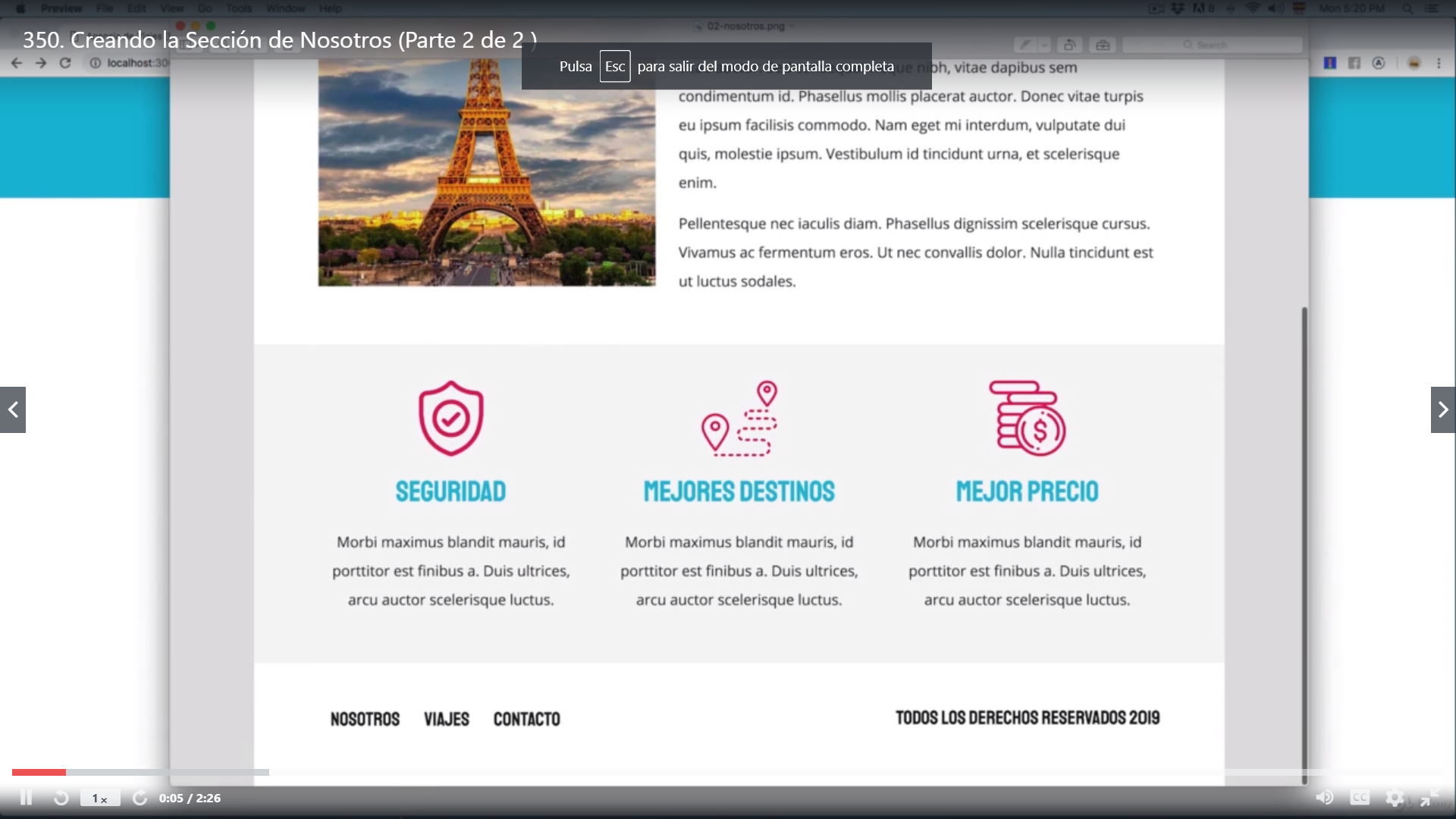
         .col-md-7

            h1 #{pagina}

            blockquote Pellentesque arcu metus, euismod et tincidunt sit amet, interdum vel felis. Quisque molestie eget lectus non bibendum. Fusce ut dignissim massa. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aliquam suscipit sollicitudin efficitur. Sed aliquet interdum nisl a bibendum. Nullam vitae enim eget nulla interdum feugiat ac vel augue.

            p Fusce scelerisque nunc quis augue rutrum, non aliquet ipsum posuere. Praesent sed nunc ac purus elementum porttitor et at mauris. Curabitur fringilla ultricies justo, sed mollis nulla dictum sit amet. Sed dolor sapien, aliquet sed venenatis id, eleifend at tellus.

            p Nullam ut ante efficitur, egestas ex tempus, aliquam eros. Curabitur aliquam, ligula et dapibus vulputate, tellus ipsum ornare urna, non suscipit magna dolor a libero.

CREANDO LOS ICONOS DE LA PARTE INFERIOR

Tenemo que hacerlo en la indentación de la etiquema main por aquí ahí pondremos el nuevo elemento.

**CONECTANDO UNA BASE DE DATOS CON SEQUELIZE**

Vamos a conectar una base de datos a NODE JS, y vamos a trabjar la base con la vista de viajes (todos esos viajes estarán guardados en una base de datos) Es un ORM, es más fácil hacer CRUDS, ya tiene sus propios métodos soporta conexión hacia MySQL,PostgreSQL,SQL Lite

Vamos a instalar unas dependencias

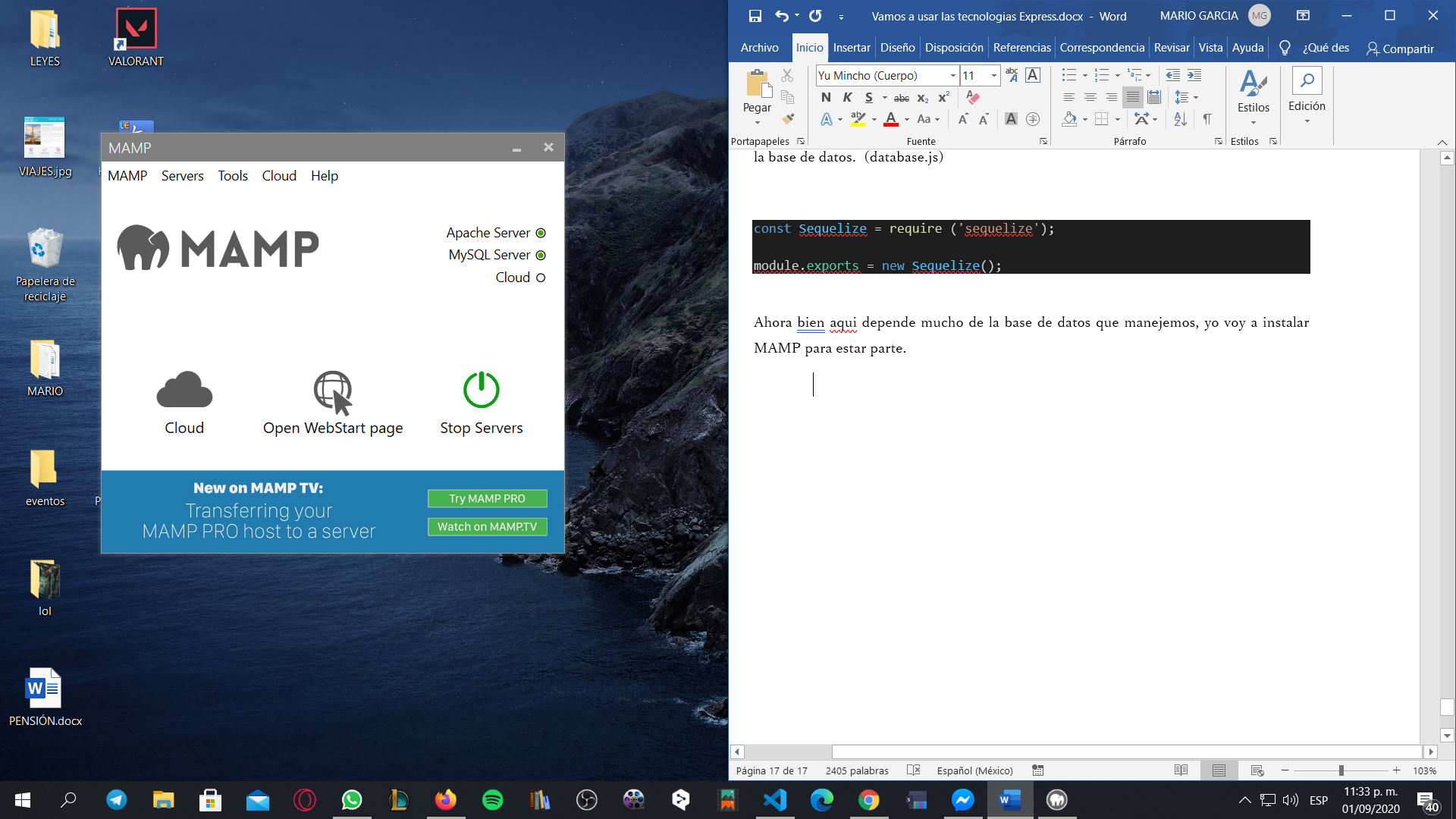
Npm install –save mysql2 sequelize

En nuestra carpeta config vamos crear un archivo que contendrá toda la configuración de la base de datos. (database.js)

const Sequelize = require ('sequelize');

module.exports = new Sequelize();

Ahora bien aqui depende mucho de la base de datos que manejemos, yo voy a instalar MAMP para estar parte.



Voy a darle Open WebStar Page para tener la base en línea, después en la pestaña de tolos y phpmyAdmin. Creamos la base de datos “agenciadeviajes” con utf8-general

Iniciamos con a la conexión (root como usuario y root como contraseña por que por defaoutl mampm nos pone eso)

module.exports = new Sequelize('agenciadeviajes','root','root');

Luego temenos que agregar los datos (opciones) especificas de la base de datos como un objeto

const Sequelize = require ('sequelize');

module.exports = new Sequelize('agenciadeviajes','root','root',{

    host:'127.0.0.1',

    port:'3306',

    dialect:'mysql',

    //dialectModule:'mysql2',

    define:{

        timestamps: false

    },

    pool: {

        max:5,

        min:0,

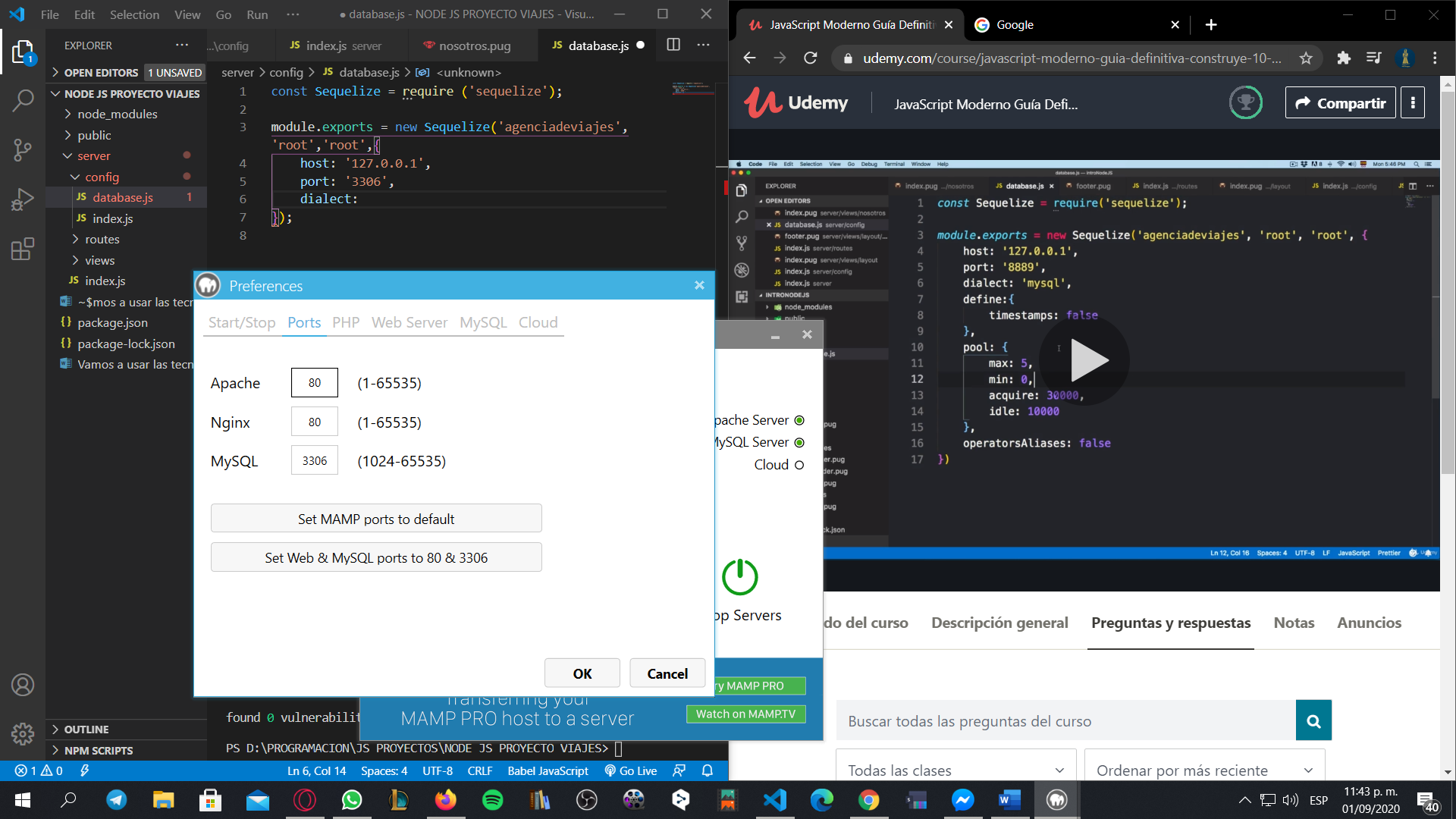
        acquire:30000,

        idle:10000

      }

});

El host se recomienda poner en número, el puerto es el que nos da mamp para MySQL. El dialecto podemos escoger (MySQL,MariaDB,SQLite,PostgreSQL,MSSQL) define es una configuración propia de sequelize, pool también así como operatorsAliases

  
Lo importamos en el archivo de config del servidor

// importar express

const express = require('express');

const path = require('path');

const routes = require('./routes');

const configs = require('./config');

const db = require('./config/database')

\*Este paso nos da la conexión

Vamos a probar la conexión ahí mismo unas líneas de código más abajo.

db.authenticate()

    .then(()=> console.log('DB CONECTADA'))

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

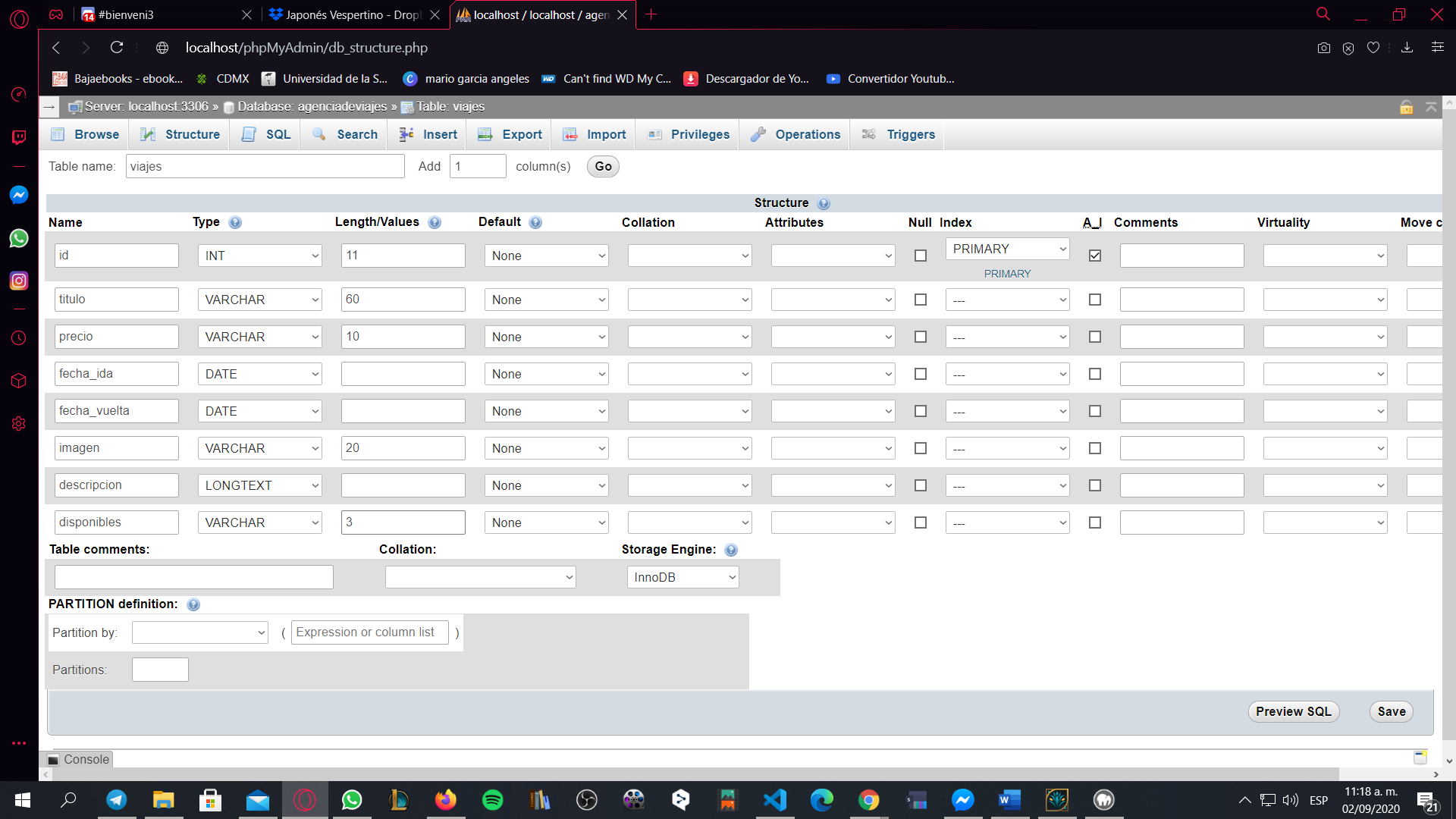
Devuelve una promesa (authenticate)

Ojo, no olvidar prender el MAPM para conectarse a la base de datos

**CREANDO LA TABLA DE VIAJES Y EL MODELO**

Solamente tendremos dos tablas una de viajes y la otra de testimoniales, 8las imágenes no se guardan en SQL)

La primera tabla se llamara viajes y tendrá 8 campos



La creamos y después vamos a construir los modelos, vamos a la capeta server, creamos una nueva carpeta “models” y el primer archivo será “Viajes.js”

Importamos sequelize, improtamos la conexicon del archivo database, y definimos el modelo en una variable con los campos de la tabla, excepto el id. Defvine pide el nombre del modelo y los campos como un objeto.

Tenemos que saber como son los dataTypes en squelize

DataTypes.STRING // VARCHAR(255)

DataTypes.STRING(1234) // VARCHAR(1234)

DataTypes.STRING.BINARY // VARCHAR BINARY

DataTypes.TEXT // TEXT

DataTypes.TEXT('tiny') // TINYTEXT

DataTypes.CITEXT // CITEXT PostgreSQL and SQLite only.

**Boolean**

DataTypes.BOOLEAN // TINYINT(1)

**Numbers**

DataTypes.INTEGER // INTEGER

DataTypes.BIGINT // BIGINT

DataTypes.BIGINT(11) // BIGINT(11)

DataTypes.FLOAT // FLOAT

DataTypes.FLOAT(11) // FLOAT(11)

DataTypes.FLOAT(11, 10) // FLOAT(11,10)

DataTypes.REAL // REAL PostgreSQL only.

DataTypes.REAL(11) // REAL(11) PostgreSQL only.

DataTypes.REAL(11, 12) // REAL(11,12) PostgreSQL only.

DataTypes.DOUBLE // DOUBLE

DataTypes.DOUBLE(11) // DOUBLE(11)

DataTypes.DOUBLE(11, 10) // DOUBLE(11,10)

DataTypes.DECIMAL // DECIMAL

DataTypes.DECIMAL(10, 2) // DECIMAL(10,2)

**Unsigned & Zerofill integers - MySQL/MariaDB only**

In MySQL and MariaDB, the data types INTEGER, BIGINT, FLOAT and DOUBLE can be set as unsigned or zerofill (or both), as follows:

DataTypes.INTEGER.UNSIGNED

DataTypes.INTEGER.ZEROFILL

DataTypes.INTEGER.UNSIGNED.ZEROFILL

// You can also specify the size i.e. INTEGER(10) instead of simply INTEGER

// Same for BIGINT, FLOAT and DOUBLE

**Dates**

DataTypes.DATE // DATETIME for mysql / sqlite, TIMESTAMP WITH TIME ZONE for postgres

DataTypes.DATE(6) // DATETIME(6) for mysql 5.6.4+. Fractional seconds support with up to 6 digits of precision

DataTypes.DATEONLY // DATE without time

Asi se contruye el modelo

const Sequelize = require('sequelize');

const db = require('../config/database');

const Viaje = db.define('viaje',{

    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

    }

});

module.exports = Viaje; //para usarlo en otro metodos

**CREANDO VISTA DE LOS PRÓXIMOS VIAJES**

Hacemos la ruta con una variable interna

router.get('/viajes', (req,res)=>{

    res.render('viajes',{

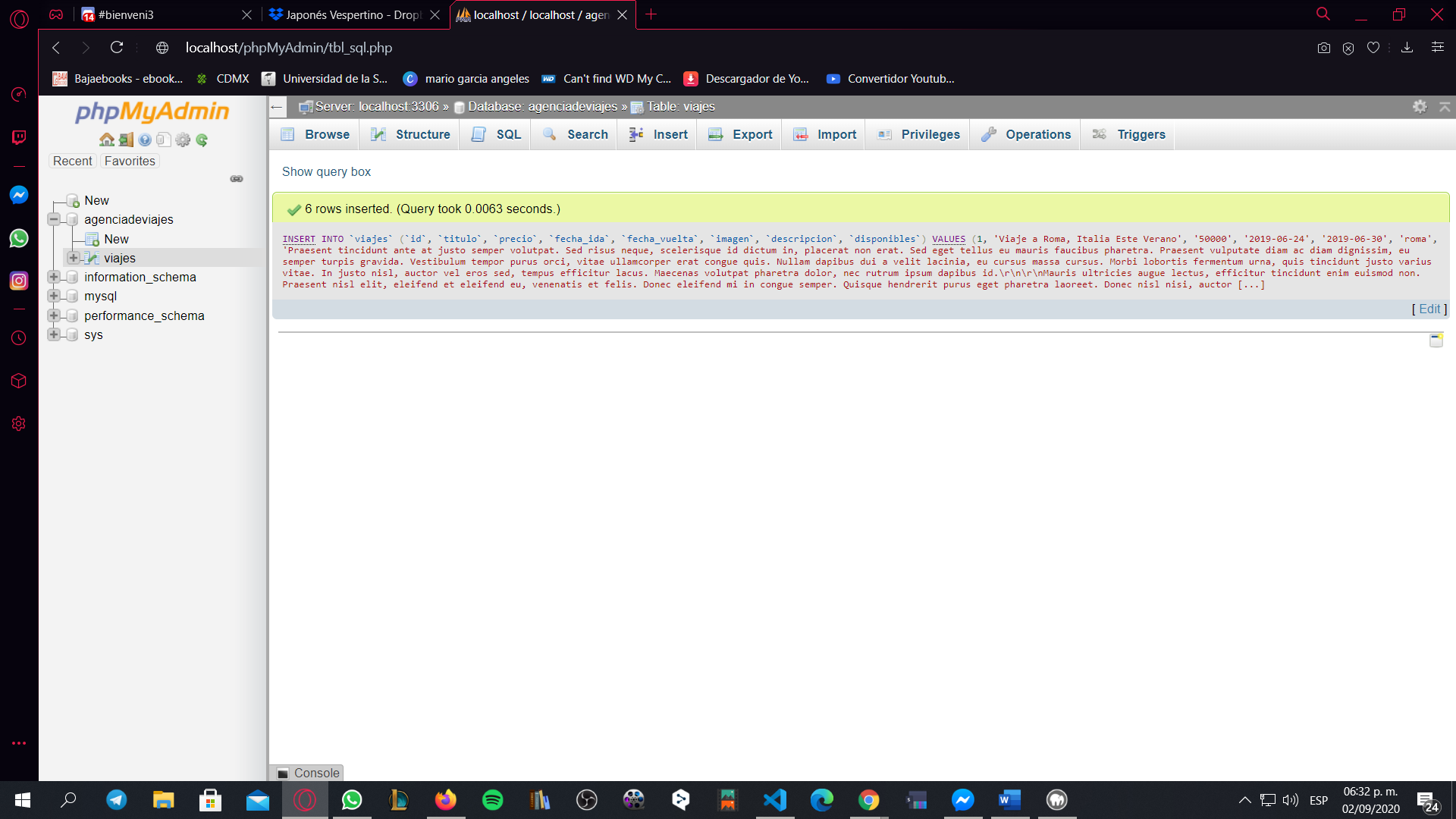
        pagina: 'proximos viajes'

    });

});

Hacemos su carpeta respectiva con su archivo pug (tuve que moverlo a fuera de su capeta como nosotros.

En los archivos encontramos los datos para insertar en la base de datos



CONSULTANDO BASE DE DATOS Y PASANDO EL RESULTADO A LA VISTA.

En la pagina de Sequelize en Quering puedo encontrar todos los métodos para hacer consulta la base de datos.

Nosotros vamos a usar el que dice findAll

## Specifying attributes for SELECT queries

To select only some attributes, you can use the attributes option:

Model.findAll({

attributes: ['foo', 'bar']

});

SELECT foo, bar FROM ...

Attributes can be renamed using a nested array:

Model.findAll({

attributes: ['foo', ['bar', 'baz'], 'qux']

});

SELECT foo, bar AS baz, qux FROM ...

You can use [sequelize.fn](https://sequelize.org/master/class/lib/sequelize.js~Sequelize.html" \l "static-method-fn) to do aggregations:

Model.findAll({

attributes: [

'foo',

[sequelize.fn('COUNT', sequelize.col('hats')), 'n\_hats']

'bar'

]

});

SELECT foo, COUNT(hats) AS n\_hats, bar FROM ...

When using aggregation function, you must give it an alias to be able to access it from the model. In the example above you can get the number of hats with instance.n\_hats.

Sometimes it may be tiresome to list all the attributes of the model if you only want to add an aggregation:

// This is a tiresome way of getting the number of hats (along with every column)

Model.findAll({

attributes: [

'id', 'foo', 'bar', 'baz', 'qux', 'hats', // We had to list all attributes...

[sequelize.fn('COUNT', sequelize.col('hats')), 'n\_hats'] // To add the aggregation...

]

});

// This is shorter, and less error prone because it still works if you add / remove attributes from your model later

Model.findAll({

attributes: {

include: [

[sequelize.fn('COUNT', sequelize.col('hats')), 'n\_hats']

]

}

});

SELECT id, foo, bar, baz, qux, hats, COUNT(hats) AS n\_hats FROM ...

Similarly, it's also possible to remove a selected few attributes:

Model.findAll({

attributes: { exclude: ['baz'] }

});

-- Assuming all columns are 'id', 'foo', 'bar', 'baz' and 'qux'

SELECT id, foo, bar, qux FROM ...

En el archivo de rutas importo el modelo de la base

const Viaje = require('../config/models/Viajes')

Siempre me retornan un Promise

Con el metodo findAll retorna todo los datos de la base de datos, y los guardamos dentro del render de la vista como propiedad viajes que se hace por el promise de viajes (asi le llamamos)

router.get('/viajes', (req,res)=>{

    Viaje.findAll()

        .then(viajes => res.render('viajes',{

            pagina: 'Proximos viajes',

            viajes: viajes

        }))

        .catch(error=>{

            console.log(error);

        })

    });

MOSTRANDO EL RESULTADO EN LA VISTA

Ahora nosotros vamos a trabajar en el arreglo para recoores todos esos datos, en la pagina de pug encontramos deocumentacion para iterar en la vista.

ul

each val in [1, 2, 3, 4, 5]

li= val

<ul>

<li>1</li>

<li>2</li>

<li>3</li>

<li>4</li>

<li>5</li>

</ul>

You can also get the index as you iterate

ul

each val, index in ['zero', 'one', 'two']

li= index + ': ' + val

<ul>

<li>0: zero</li>

<li>1: one</li>

<li>2: two</li>

</ul>

ul

each val, key in {1: 'one', 2: 'two', 3: 'three'}

li= key + ': ' + val

<ul>

<li>1: one</li>

<li>2: two</li>

<li>3: three</li>

</ul>

- var values = [];

ul

each val in values.length ? values : ['There are no values']

li= val

<ul>

<li>There are no values</li>

</ul>

Es la vista de viajes, recorre con un each viaje es como le vamos a poner, in viajes la variable que esta dentro de la promesa,

extends ./layout/index

block contenido

   main.container.mt-5

            h1.text-center.mb-5 #{pagina}

            .row.proximos-viajes

               each viaje in viajes

Para poner las tarjetas de viajes

 h1.text-center.mb-5 #{pagina}

            .row.proximos-viajes

               each viaje in viajes

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

                     .card

                        .card-body

                           h2=viaje.titulo

\*\*titulo por que son los mismo campos que están en el modelo de datos..

const Viaje = db.define('viaje',{

    titulo : {

        type: Sequelize.STRING

    },

    precio:{

        type: Sequelize.STRING

    },

    fecha\_ida:{

        type: Sequelize.DATE

    },//etc.

Vamos a poner los demás datos en la viaje, en las imágenes son las mismas que encontramos en la base de datos .

IMAGENES(IMG/DESTINOS\_x) Es el nombre de la caprta en la capeta publica donde están las imágenes.

     h1.text-center.mb-5 #{pagina}

            .row.proximos-viajes

               each viaje in viajes

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

                     .card

                        img(src=`img/destinos\_${viaje.imagen}.jpg`,class="img-fluid")

                        .card-body

                           h2=viaje.titulo

Para los pequeños logos vamos a usar FONT AWESOME

Usamos el link de fontawesome, y lo convertyimos a pug

Lo pegamos en el layout general

script(src='https://kit.fontawesome.com/149a950f34.js' crossorigin='anonymous')

\*antes del archive css

COLOCANDO EL TITULO, FECHAS Y PRECIO

            h1.text-center.mb-5 #{pagina}

            .row.proximos-viajes

               each viaje in viajes

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

                     .card

                        img(src=`img/destinos\_${viaje.imagen}.jpg`,class="img-fluid")

                        .card-body

                           h2=viaje.titulo

                           p

                               i.fas.fa-calendar-alt

                               |  #{viaje.fecha\_ida} - #{viaje.fecha\_vuelta}

                           p

                               i.fas.fa-dollar-sign

                               |  #{viaje.precio}

Como la descripción es larga podemos cortarlo con el metogo substr(0,100) parámetro 1 inicio parámetro 2 cuantos caracteres necesitamos

   p

                               i.fas.fa-dollar-sign

                               |  #{viaje.precio}

                           p= viaje.descripcion.substr(0,100)

Agregamos el botón que nos lleve a la pagina de cada viaje con su id.

a(href=`viajes/${viaje.id}`, class="btn btn-success btn-block") Más información

Ya con este id Podemos hacer un filtro a la base de datos son squelize

// search for known ids

Project.findByPk(123).then(project => {

// project will be an instance of Project and stores the content of the table entry

// with id 123. if such an entry is not defined you will get null

})

**CREANDO LA VISTA PARA CADA VIAJE Y CONSULTANDO LA BD**

Vamos a iniciar en el archivo de rutas para poner las correspondientes al ID, se usa en las rutas un “COMODIN”

Comodin

 router.get('/viajes/:id', (req,res)=>{

        });

    });

Con podemos leer la url de cada uno y leer su respectivo id con el comodin req.params.id

    router.get('/viajes/:id', (req,res)=>{

      res.send(req.params.id)

        });

Vamos a usar el método que encuentra el id de sequelize de forma dinámica. Pero tenemos que crear una caprta nueva en vista con el viaje en singular. (Viaje por que es del modelo de datos esportado)

router.get('/viajes/:id', (req,res)=>{

      Viaje.findByPk(req.params.id)

        .then(viaje => res.render('viaje',{

            viaje

        }))

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

        });

\*\*Esa función findByPk es la verison 6 ya no es mas findById

Hacemos la vista de viaje

extends ./layout/index

block contenido

   main.container.mt-5

      .row

         .col-md-5

            img(src=`/img/destinos\_${viaje.imagen}\_ln.jpg`,class="img-fluid")

         .col-md-7

            h1 #{viaje.titulo}

\*\*usamos otras imágenes de mayor tamaño, y como tenemos problemas con las rutas todas las dejamos en la vista views afuera de sus capetas.

**IMPRIMIENDO EL RESTO DE LOS CAMPOS**

extends ./layout/index

block contenido

   main.container.mt-5

      .row

         .col-md-5

            img(src=`/img/destinos\_${viaje.imagen}\_ln.jpg`,class="img-fluid")

         .col-md-7

            h1 #{viaje.titulo}

            p

               i.fas.fa-calendar-alt

               |  #{viaje.fecha\_ida} - #{viaje.fecha\_vuelta}

            p

               i.fas.fa-dollar-sign

                 |      #{viaje.precio}

               |

            p

               i.fas.fa-users

                 |      #{viaje.disponibles} Disponibles

            p= viaje.descripcion

CREANDO LA VISTA DE TESTIMONIALES

Vamos a usar un formulario que se guardaran los comentarios en la BD mediante sequelize.

Creamos su ruta

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

        res.render('testimoniales',{

            testimonales : 'Testimoniales'

        });

    });

Su vista

extends ./layout/index

block contenido

   main.container.mt-5

            h1.text-center.mb-5 #{testimoniales}

extends ./layout/index

block contenido

   main.container.mt-5

            h1.text-center.mb-5 #{testimoniales}

            .row

                .col-md-12

                    .row.justify-content-center

                        .col-md-8

                            form

                                .form-group

                                    label(for="nombre") Nombre

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

                                .form-group

                                    label(for="correo") Nombre

                                    input(id= "correo", class="form-control", type="email",placeholder="Correo Electrónico", name="correo")

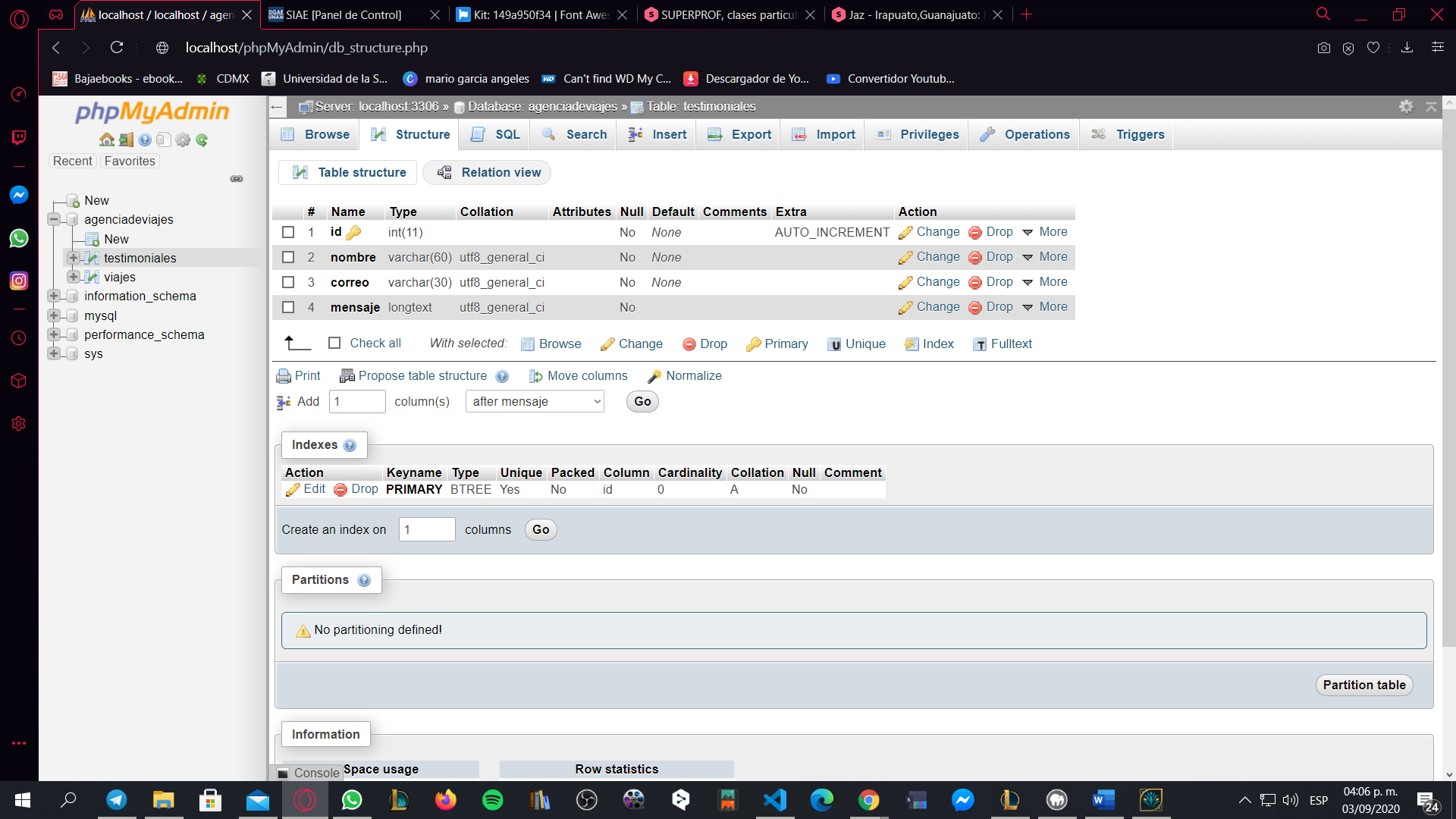
                                .form-group

                                    label(for="mensaje") Mensaje

                                    textarea#mensaje(class="form-control",name="mensaje",  rows="10", placeholder="Tu mensaje")

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

**CREANDO LA TABLA Y EL MODELO PARA LOS TESTIMONIALES**



Creamos el archivo Testimoniales en la carpeta modelos a la base de datos

const Sequelize = require('sequelize');

const db = require('../config/database');

const Testimonial = db.define('testimoniales',{

    nombre : {

        type: Sequelize.STRING

    },

    correo : {

        type: Sequelize.STRING

    },

    mensaje : {

        type: Sequelize.STRING

    }

})

module.exports = Testimonial;

**VALIDANDO EL FORMULARIO EN NODE**

Ahora bien en la el documento de las rutas nosotros estamos usando el método get, pero cuando se llenan datos se usa el método post.

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

        res.render('testimoniales',{

            testimoniales : 'Testimoniales'

        });

    });

    //para el formulario

    router.post('/testimoniales',(req,res)=>{

        console.log(req.body);

    })

En la vista pug, debo de poner los atributos a form para que funcione como post

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

esto me mandara un undefined…

Esto nos remonta a un paquete que instalamos al principio del proyecto “body-parser”

"body-parser": "^1.19.0",

Tenemos que usarlo en el archivo configuración y puerto del servidor

const bodyParser = require('body-parser');

//ejecutamos body

app.use(bodyParser.urlencoded({extended: true}));

//carga las rutas

app.use('/',routes());

con eso ya puedo leer en la consola los datos que se enviaron en el formulario en la consola, ahora vamos a usar destructe para usar esos datos, y con puras condionales vamos a almacernar los datos o sacar una vista si hay errores

router.post('/testimoniales',(req,res)=>{

        //valida campos

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

        let errores =[];

        if(!nombre){

            errores.push({'mensaje':'agrega tu nombre'})

        }

        if(!correo){

            errores.push({'mensaje':'agrega tu correo'})

        }

        if(!mensaje){

            errores.push({'mensaje':'agrega un asunto'})

        }

        //revisa cada error

        if(errores.length >0){

            //muestra los errores

        }else{

            //almacena en la BD

        }

    })

**LLENANDO LOS CAMPOS QUE EL USUARIO HA LLENADO CORRECTAMENTE**

Como hemos llenado los campos de Name en el formulario es como los tenemos que usar, como en PHP. Asi se tiene que aplicar el destructuring en la ruta

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

Hagamos que la info del formulario se guarde, si hay algún error que no lo borre.

U le damos un value a cada cmapo del formulario para que se guarde

placeholder="Tu Nombre", name="nombre" , value=nombre)

para el de text área va fuera

textarea#mensaje(class="form-control",name="mensaje",  rows="10", placeholder="Tu mensaje")=mensaje

Y ya se va guarda y no se borran los campos

MOSTRANDO LOS MENSAJES DE ERROR

Con un if en pug que recoora con un each

            .row

                .col-md-12

                    h2.text-center Agrega un Testimonial

                    if(errores)

                        each error in errores

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

                    .row.justify-content-center

                        .col-md-8

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

**ALMACENANDO LOS DATOS DEL FOMRULARIO EN LA BD**

Importamos el modelo de datos en el archivop de enlaces

const Testimonial = require('../models/Testimoniales');

Trabajamos la segunda parte del if, créate es para ingresar un nuevo dato al modelo, y redirect hacia donde nos enviará cuando terminemos el formulario.

else{

            //almacena en la BD

            Testimonial.create({

                nombre,

                correo,

                mensaje

            })

            .then(testimonial => res.redirect('/testimoniales'))

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

        }

**MOSTRANDO LOS TESTIMONIALES EN EL DOM**

Con esto ya podemos obtener los datos de la base

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

        Testimonial.findAll()

        .then(testimoniales => res.render('testimoniales',{

            pagina: 'testimoniales',

            testimoniales : testimoniales

        }))

        });

Vamos a hacer el DOM

h1.text-center.mb-5 #{testimoniales}

            .row

                .col-md-12

                    blockquote.text-center Lee Sobre Nuestros clientes y sus Experiencias

                    .row.testimoniales

                        each testimonial in testimoniales

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

                                .card

                                    .card-body

                                        blockquote.blockquote

                                        p.mb-0

                                            | #{testimonial.mensaje}

                                        footer.blockquote-footer

                                            | #{testimonial.nombre}

Asi quedaría la tarjeta con el nombre



CREANDO LA PAGINA PRICIPAL

Hacemos un extends para el layout en la vista de inicio

extends ../layout/index

Luego en el archivo layout definimos un nuevo bloque

extends ../layout/index

block slider

    .slider

        h1 日本　/　メキシコ　/　中国

        em Adventure

block contenido

Los estilos definidos son los siguientes

.slider {

    background: url(../img//superior.jpg);

    min-height: 600px;

    height: 100vh;

    background-size: cover;

    display: flex;

    flex-direction: column;

    align-items: center;

    justify-content: center;

    color: white;

}

.slider h1 {

    color: white;

}

.slider em {

    font-family: 'Covered By Your Grace', cursive;

    font-size: 4rem;

    order: -1;

    line-height: .8;

    color: rgba(255, 255, 255, .5);

}

Ahora vamos darle tranparencia al header en el slider, para que no afecte los colores.

En el archivo de rutas pasamos una nueva clase al index

router.get('/', (req,res)=>{

    res.render('index',{

        clase: 'home'

    });

});

Y en el layout

title= titulo

    body(class=clase)

estilo es

.home {

    position: relative;

}

.home .navegacion {

    background: transparent;

    position: absolute;

    top: 0;

    left: 0;

    width: 100%;

}

CREANDO LA SECCIÓN DE NOSOTROS Y DESCUENTO

extends ../layout/index

block slider

    .slider

        h1 日本　/　メキシコ　/　中国

        em Adventure

block contenido

    main.container.mt-5

        h2.text-center.mb-5 Sobre Nosotros

        .row.mb-5

            .col-md-6

                p ed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

                p Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

            .col-md-6

                img(src="/img/inicio\_nosotros.jpg", alt="img-fluid")

Ahora el banner de descuento

.descuento {

    height: 400px;

    background-image: linear-gradient( rgba(0, 0, 0, .5) 0%,  rgba(0, 0, 0, .5) 100%),  url(../img/cupon.jpg);

    background-repeat: no-repeat;

    background-size: cover;

}

@media (min-width:768px) {

    .descuento {

        height: 400px;

        background-image:  linear-gradient(

            to right,

            rgba(0, 0, 0, .5) 0%,

            rgba(0, 0, 0, .5) 50%,

            transparent 50%,

            transparent 100%

        ), url(../img/cupon.jpg);

    }

}

.descuento .contenido {

    display: flex;

    flex-direction: column;

    justify-content: center;

    height: 400px;

    color: white;

}

.descuento p {

    font-family: 'Staatliches', cursive;

    line-height: .8;

}

.descuento .titulo {

    font-size: 2.4rem;

}

.descuento .precio {

    color: #15B9D5;

    font-size: 3rem;

} ///////código en la vista

    .descuento

        .container

            .row

                .col-md-6

                    .contenido

                        h3 5% de descuento

                        p Viaje a Canada

                        p.fecha 25 de octubre 2020 - 25 de noviembre 2020

                        p.precio $5,000 MXN

h3 {

    font-size: 3rem;

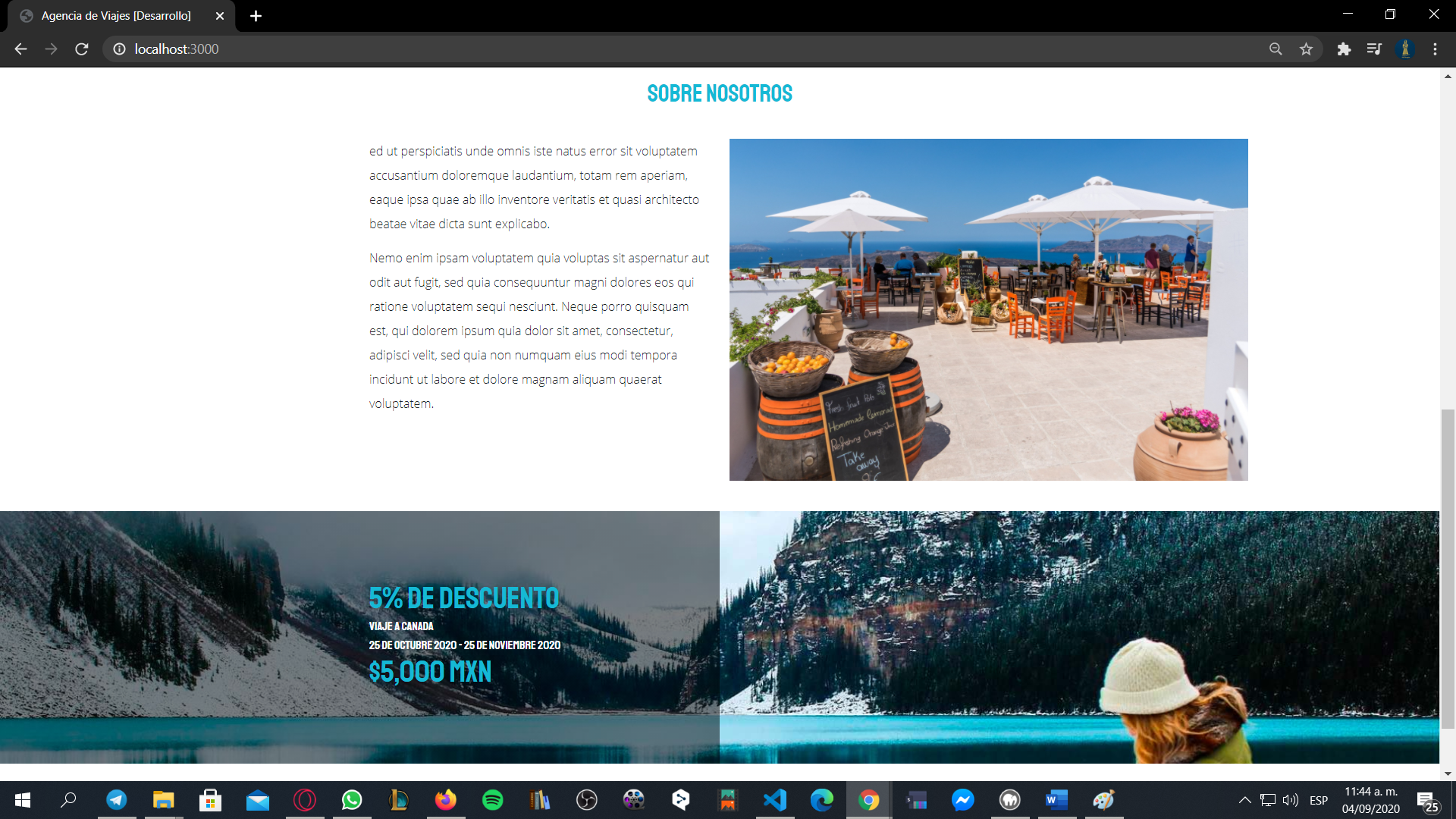
}

p {

    font-size: 1.2rem;

    line-height: 2;

}



Ahora tenemos que implementar la parte dinámica los viajes con la BD y las testimoniales

Tenemos que modificar la ruta para obtener lo datos de la base de datos

router.get('/', (req,res)=>{

    Viaje.findAll({

        limit: 3 //aquí modificamos el query

    })

        .then(viajes => res.render('index',{

            pagina: 'Próximo viajes',

            clase: 'home',

            viajes: viajes

        }))

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

});

Pego el código que ya hicimos en la vista de viajes

.row.proximos-viajes

               each viaje in viajes

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

                     .card

                        img(src=`img/destinos\_${viaje.imagen}.jpg`,class="img-fluid")

                        .card-body

                           h2=viaje.titulo

                           p

                               i.fas.fa-calendar-alt

                               |  #{viaje.fecha\_ida} - #{viaje.fecha\_vuelta}

                           p

                               i.fas.fa-dollar-sign

                               |  #{viaje.precio}

                           p= viaje.descripcion.substr(0,150)

                           a(href=`viajes/${viaje.id}`, class="btn btn-success btn-block") Más información

que este dentro del main o del row de arriba, depende el gusto

Pero ahora si corrue un cambio en la base tenemos que hacer dos cambios y para evitarnos eso vamos a conectar esta parte

En layout creamos una capreta que se llama partials y el nombre del archivo es viajes.pug

Cortamos el código de viajes que estaba en el index

Lo pego en el mismo lugar en la vista de inicio y de igual manera con la vista de viajes para que estén conectados.

 include ../layout/partials/viajes

\*2

include ./layout/partials/viajes

asi quedo para la vista viajes

Y así ambias vistas quedaron conectadas

**PASANDO LA PARTE DE LOS TESTIMONIALES A LA VISTA DE INICIO**

Se nos ocurrió crear un arreglo de modelos para no trabajar por separados las consultas, ósea que en una consulta se traiga los modelos de testimoniales y de viajes que hacemos a la base de datos.

router.get('/', (req,res)=>{

    const promises = [];

    promises.push(Viaje.findAll({

        limit: 3

    })

    )

    promises.push(Testimonial.findAll({

        limit: 3

    })

    )

    //pasarlo al promise y ejecutarlo

    const resultado = Promise.all(promises);

Promise.all ejecuta todos las promesas que haya, por el método de findAll que devueleve promises. Y las va air ejecutando de una en una.

   resultado.then(resultado => res.render('index',{

            pagina: 'Próximo viajes',

            clase: 'home',

            viajes: resultado[0],

            testimoniales: resultado[1]

        }))

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

});

Esto recordamos es solo para la vista de inicio, (el valor del array es la posición de cada modelo que hicimos se tiene que hacer conforme a ese orden)

Vamos a crear un partial para las testimoniales

De igual manera en partial creamos un nuevo archivo para testitmonial.pug

Partial de testimonial

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}

index

.descuento

        .container

            .row

                .col-md-6

                    .contenido

                        h3 5% de descuento

                        p Viaje a Canada

                        p.fecha 25 de octubre 2020 - 25 de noviembre 2020

                        p.precio $5,000 MXN

    .container

        h2.text-center.my-5 Testimoniales

        include ../layout/partials/testimoniales

Vista de testimonials

            h1.text-center.mb-5 #{testimonial}

            .row

                .col-md-12

                    include ./layout/partials/testimoniales

                .col-md-12

                    h2.text-center Agrega un Testimonial

**VAMOS A RESULTAR EN EL HEADER CUANDO ESTAMOS EN UNA PÁGINA EN CONCRETO**

Podemos hacer un nuevo use en la pagina de configuración del servidor, con los mimos parámetros pero para ahorrarnos código lo metemos dentro del que da la fecha actual

app.use((req,res,next) =>{

    const fecha = new Date();

    res.locals.fechaActual = fecha.getFullYear();

    res.locals.ruta = req.path;

    return next();

})

Lo que hace req.path nos retorna la url donde nos encontramos

Ahora modificamos el header, para que quede de esta forma, agregamos un operador ternario para su comprobación y agregamos la clase nav-link en vada enlace

  a.nav-link(href="/nosotros",class=(ruta === '/nosotros' ? 'active' :'')) Nosotros

                    a.nav-link(href="/viajes",class=(ruta === '/viajes' ? 'active' :'')) Viajes

                    a.nav-link(href="/testimoniales",class=(ruta === '/testimoniales' ? 'active' :'')) Testimoniales

y este es el estilo css

.nav-link.active {

    border-bottom: 5px solid white;

}

LISTO!!!Ya quedó la pagina

AGREGAR UN CONTROLADOR

Lo que pasa ahora es que nuestro archivo de rutas esta actuando como router y controlado al mismo tiempo

Modelo: Como ya vimos es el que tiene la configuración de la BD y la conexión a la misma.

Vistas: Lo que ve el usuario

Controlador: Lo que une el modelo con la vista.

En nuestro archivo estamos haciendo lo mismo

const Viaje = require('../models/Viajes');

const Testimonial = require('../models/Testimoniales');

Cramos una nueva carpeta en serve ‘controllers’ y va a tenerlo por cada pagina, cortamos el código del archivo de rutas para tenerlo por separada en cada archivo.

exports.infoNosotros = (req,res)=>{

    res.render('nosotros',{

        pagina: 'Sobre Nosotros'

    });

}

Y en el archivo de rutas lo exportamos como siempre

const nosotrosController = require('../controllers/nosotrosController');

y ahora quedara el código solo en una linea

router.get('/nosotros', nosotrosController.infoNosotros);

Lo mismo para la pagina de inicio en el nuevo documento homeaController

const Viaje = require('../models/Viajes');

const Testimonial = require('../models/Testimoniales');

exports.consultasHome = (req,res)=>{

    const promises = [];

    promises.push(Viaje.findAll({

        limit: 3

    })

    )

    promises.push(Testimonial.findAll({

        limit: 3

    })

    )

    //pasarlo al promise y ejecutarlo

    const resultado = Promise.all(promises);

        resultado.then(resultado => res.render('index',{

            pagina: 'Próximo viajes',

            clase: 'home',

            viajes: resultado[0],

            testimoniales: resultado[1]

        }))

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

}

Pero tenemos que pasarle los modelos para que se conecte, que no se olvide

const homeController = require('../controllers/homeController');

router.get('/', homeController.consultasHome);

Lo mismo hago con viajes y sus respectivos id

const Viaje = require('../models/Viajes');

exports.mostrarViajes = (req,res)=>{

    Viaje.findAll()

        .then(viajes => res.render('viajes',{

            pagina: 'Proximos viajes',

            viajes: viajes

        }))

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

    }

    exports.mostrarViaje =  (req,res)=>{

        Viaje.findByPk(req.params.id)

          .then(viaje => res.render('viaje',{

              viaje

          }))

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

          }

rutas

router.get('/viajes',viajesController.mostrarViajes );

router.get('/viajes/:id',viajesController.mostrarViaje);

controlador de testiomoniales

const Testimonial = require('../models/Testimoniales');

exports.mostrarTestimoniales = (req,res)=>{

    Testimonial.findAll()

    .then(testimoniales => res.render('testimoniales',{

        testimonial: 'testimoniales',

        testimoniales : testimoniales

    }))

    }

exports.agregarTestimonial = (req,res)=>{

    //valida campos

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

    let errores =[];

    if(!nombre){

        errores.push({'mensaje':'agrega tu nombre'})

    }

    if(!correo){

        errores.push({'mensaje':'agrega tu correo'})

    }

    if(!mensaje){

        errores.push({'mensaje':'agrega un asunto'})

    }

    //revisa cada error

    if(errores.length >0){

        //muestra los errores

        res.render('testimoniales' ,{

            errores,

            nombre,

            correo,

            mensaje

        })

    }else{

        //almacena en la BD

        Testimonial.create({

            nombre,

            correo,

            mensaje

        })

        .then(testimonial => res.redirect('/testimoniales'))

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

    }

}

Index

router.get('/testimoniales', testimonialesController.mostrarTestimoniales);

 //para el formulario

router.post('/testimoniales',testimonialesController.agregarTestimonial)

return router; //para hacerlo disponible en el otro metodo

}

**COMO HACER ASYN AWAIT EN EL PROYECTO**

**viajes**

const Viaje = require('../models/Viajes');

exports.mostrarViajes = async(req,res)=>{

   const viajes = await  Viaje.findAll()

    res.render('viajes',{

            pagina: 'Proximos viajes',

            viajes: viajes

        });

    }

    exports.mostrarViaje = async (req,res)=>{

      const viaje =  await Viaje.findByPk(req.params.id)

      res.render('viaje',{

        viaje

    })

          }

Home controller

const Viaje = require('../models/Viajes');

const Testimonial = require('../models/Testimoniales');

exports.consultasHome = async (req,res)=>{

    const viajes = await Viaje.findAll({

        limit: 6

    });

    const testimoniales = await Testimonial.findAll({

        limit: 3

    })

    res.render('index',{

        pagina: 'Próximo viajes',

        clase: 'home',

        viajes: viajes,

        testimoniales: testimoniales

    })

}

PUNTOS FINALES MIDDLEWARE USADOS

app.use((req,res,next) =>{

    const fecha = new Date();

    res.locals.fechaActual = fecha.getFullYear();

    res.locals.ruta = req.path;

    return next();   //middleware para que se pase a al otro middleware

})

**PREPARANDO LAS VARIABLES DE ENTORNO**