07-restserver

**Objetivos:**

* Configuraciones iniciales para poder crear el REST server
* Instalacion de la base de datos Mongo DB
* Crear un EXPRESS REST server
* Crear la configuracion global de la aplicacion

**CLASE 77 . Instalacion de MongDB**

Objetivo: Instalacion de MongoDB – base de datos no relacional

En navegador web

* Buscar y abrir pagina ‘mongodb.com’
* Click a boton verde ‘Try free’

Entra a pantalla ‘MongoDB Download Center’

* Ubicar menu horizontal a media pagina
* Click a opcion ‘Server’
* Entra a pantalla ‘MongoD Community Server’

Se muestra la version, el sistema operativo ‘Windows’ y el package ‘MSI’

* Clck en boton verde ‘Download’ para instalar y ‘next’ a todas las demas opciones
* En menu vertical de la derecha click a ‘Installation instructions’

Entra a pantalla de instrucciones de ‘Install MongoDB Community Edition’

* Localizar seccion

Run MongoDB Community Edition from the Command Interpreter

* Ubicar la seccion

### Create database directory

* Seleccionar las siguientes instrucciones, copiarlas y ejecutarlas en consola

**cd** C:\

**md** "\data\db"

* Ubicar la seccion

### Start your MongoDB database

* Seleccionar la siguiente instruccion, copiarla y ejecutarla en consola

"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="c:\data\db"

*Esta instrucción ‘levanta/arranca’ el servicio de MongoDB; Importante: Deben ponerse las comillas al inicio y al final de la instrucción;*

*Al ejecutarla se muestran una lista de mensajes relacionados con el arranque, en la segunda linea se muestra ‘port 27017’; este puerto sera utilizado despues*

*al final aparece el mensaje*

[initandlisten] waiting **for** connections on port 27017

*Que indica que esta corriendo el servicio de MongoDB*

* Cuando se quiera ‘bajar/cancelar’ el servicio de MongoDB teclear ‘Cntrl-C’

**CLASE 78 . Robo 3T – Herramienta para manejar Mongo**

Objetivo: Instalacion de Robo 3T y creacion de primera base de datos

En navegador web

* Buscar pagina ‘Robo 3T – formerly Robomongo’ (robomongo.org)
* Click a boton verde ‘download Robo 3T’

*Pasa a sig pantalla*

* Click a boton verde ‘download Robo 3T’
* Seleccionar sistema operativo
* Click en version correspondiente
* Click a todos los NEXT siguientes

En consola

* Teclear:

Cd 07-restserver

* Teclear:

"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="c:\data\db"

*Esta instrucción Levanta el servicio de ‘MongoDB’*

En escritorio

* Abrir app ‘robo 3T’ (con el icono creado cuando se instalo)

*Se abre pagina principal*

En app ‘Robo 3T’

CREAR CONEXIÓN A PUERTO 27017

*Se abre ventana ‘MongoDB Connections’*

* Click a opcion ‘Create’ en menu horizontal superior

*Se abre ventana ‘Connection Settings’*

* Parametros:

*Type: Direct connection*

*Name: Localhost*

*Address: Localhost*

*Puerto: 27017 (es el que se muestra cuando se levanta’Mongo’*

* Click a boton ‘save’

*Regresa a ventana ‘MongoDB Connections’*

*Se muestra la conexión creada ‘Localhost’*

* Seleccionar conexión ‘localhost’
* Click a boton ‘connect’ en la parte inferior de la ventana

*Regresa a pantalla principal ‘Robo 3T’*

CREAR BASE DE DATOS

* En menu vertical lado derecho, click boton derecho en item ‘Localhost (n)’

*Se abre menu qry*

* Click a opcion ‘Create database’

*Se abre ventana ‘database name’*

* Teclear database name: café

*Se muestra esa nueva base de datos en el menu vertical a la derecha*

* Abrir descendencia de base de datos ‘café’

*Se muestran folder descendientes*

* Click boton derecho en folder ‘collections (n)’

*Se abre menu qry vertical*

* Click a opcion ‘Create collection’
* Collection name: ususarios
* Click a boton ‘create’

*Se muestra la coleccion ‘ususarios’ bajo el folder ‘collections(n)’*

* Doble click a la colección ‘usuarios’

*Se muestra ventana ‘db.getcollection(‘usuarios’).find({}) sin ningun registro*

* Click boton derecho en colección ‘usuarios’

*Se abre menu qry*

* Click a opcion ‘Insert Document’

*Se abre pantalla ‘Insert Document’*

* Teclear entre los corchetes ‘ “nombre”:”Fernando”
* Click a boton ‘save’
* Doble click a la colección ‘usuarios’

*Se muestra ventana ‘db.getcollection(‘usuarios’).find({}) con este primer registro creado*

* Abrir este primer registro

*Se muestran dos campos, el primero es un ‘\_id’ creado automaticamente por ‘mongo’ y el segundo es el que se inserto recientemente*

**CLASE 79 . iniciando el proyecto - RESTServer**

Objetivo: crear un programa ‘server.js’ que despliegue el texto ‘Hello World’ usando ‘express REST’

En explorador de archivos

* Si no existe, crear el folder ‘07-restserver’
* Arrastrarlo a Visual Studio Code para que lo incorpore

En consola

* Teclear : Cd C:\Users\EG\_Lobo\Documents\Mis documentos\aa Major 1 EGB\EGB EDU\TAKEN\NODE\07-restserver
* Teclear : ‘Npm init’
* Responder si a todas los prompts

*Esta instrucción inicializa ‘npm’ en el folder y crea los archivos ‘.json’ para asi poder realizar instalaciones y disponer de todos los demas servicios necesarios*

INSTALAR EXPRESS

En navegador web

* Buscar ‘npm express’
* Abrir pagina ‘Express -npm’ - <https://www.npmjs.com/package/express>
* Ver instrucción ‘npm i express’

*Esta instrucción es para instalar ‘express’ en el folder*

En consola

* Teclear instrucción ‘npm i express --save’ (dos guiones)

*Se instalara express*

CREAR ARCHIVO JS PARA LEVANTAR SERVIDOR Y ESCUCHAR PETICIONES POR PUERTO

En visual studio code

* En folder ‘07-restserver’ crear folder ‘server’
* En folder ‘server’ crear archivo ‘server.js’

En navegador web

En pagina ‘npm express’

* Ver estas lineas para ponerlas en ‘server,js’

const express **=** require('express')

const app **=** express()

app.get('/', function (req, res) {

  res.send('Hello World')

})

app.listen(3000)

En visual studio code

En archivo ‘server.js’

* Poner / Modificar las lineas marcadas en azul

const express = require('express')

const app = express()

app.get('/', function(req, res) {

    res.send('Hello World')

})

app.listen(3000, () => {

console.log('Escuchando puerto: ', 3000);

});

LEVANTAR SERVIDOR CON NODE PARA ESCUCHAR PETICIONES POR PUERTO

En consola

* Teclear: ‘Node server/server’

*Se despliega el mensaje ‘Escuchando puerto: 3000’*

SIMULAR PETICIONES DE CLIENTE CON POSTMAN

En postman

En prompt para URL

* Seleccionar peticion ‘GET’
* En recuadro teclear ‘localhost:3000’
* Click a boton ‘SEND’

En ventana inferior

* En opcion ‘Body’ debe aparecer el texto ‘Hello World’
* En opcion ‘Headers’, en linea ‘Content type’ debe decir ‘text/html’

En visual studio code

En archivo ‘server.js’

* Modificar esta linea

app.get('/', function(req, res) {

    res.json('Hello World')

})

EN POSTMAN

En prompt para URL

* Selecciónar peticion ‘GET’
* En recuadro teclear ‘localhost:3000’
* Click a boton ‘SEND’

*En ventana inferior se despliega:*

*En opcion ‘Body’ debe aparecer el texto ‘Hello World’*

*En opcion ‘Headers’, en linea ‘Content type’ sigue mostrando ‘text/html’ (la razon es porque no se ha detectado que se hizo un cambio en el codigo porque el servidor fue ‘levantado’ con el comando ‘node’; (para que se esten detectando los cambios hechos a los programas debe utilizarse el comando ‘nodemon’)*

LEVANTAR SERVIDOR CON NODEMON

En consola

* Bajar el servidor con ‘cntrl-C’
* Teclear; ‘nodemon server/server’

*Con esta instrucción, el servidor estara detectando permanentemente cualquier cambio que se haga en el archivo server o cualquier otro programa llamado por este*

*Se despliegan varias lineas que indican que se arranco el archivo ‘server.js’, y se estaran monitoreando permanentemente cambios al mismo*

SIMULAR PETICIONES DE CLIENTE CON POSTMAN

En postman

En prompt para URL

* Seleccionar peticion ‘GET’
* En recuadro teclear: ‘localhost:3000’
* Click a boton ‘SEND’

En ventana inferior

*En opcion ‘Body’ debe aparecer el texto ‘Hello World’*

*En opcion ‘Headers’, en linea ‘Content type’ debe decir ‘application/json’*

Sever.js

**CLASE 80 . Peticiones HTTP – GET – PUT – POST - DELETE**

Objetivo:

* Modificar el programa ‘server.js’ para que haga peticiones ‘get’, ‘put’, ‘post’, ‘delete’;
  + En ‘get’ y en ‘delete’: recibir peticion de postman y solamente despligar mensaje en la ventana inferior de postman en la opcion ‘body’
  + En ‘post’: recibir peticion de postman y los datos de la seccion ‘Body ⚫’
  + En ‘put’: recibir peticion de postman, los datos de la seccion ‘Body ⚫’ y el dato despues del ‘/usuario/’ del URL

LEVANTAR SERVIDOR

En consola

* Teclear

Cd C:\Users\EG\_Lobo\Documents\Mis documentos\aa Major 1 EGB\EGB EDU\TAKEN\NODE\07-restserver

* Teclear: ‘nodemon server/server’

*Se desplegaran varias lineas*

*Se desplegara el mensaje ‘Escuchando puerto: 3000’*

En Visual Studio Code

En file server.js

* Modificar lineas para que quede como se indica la peticion ‘get’

app.get('/usuario', function(req, res) {

res.json('get usuario')

})

En postman

* Seleccionar peticion: GET
* En recuadro URL teclear <http://localhost:3000/usuario>
* Click a boton SEND

*En ventana inferior en opcion body se despliega*

*‘get usuario’*

En Visual Studio Code

En file server.js

* Poner estas lineas debajo de la peticion ‘get’

app.post('/usuario', function(req, res) {

res.json('post usuario')

})

En postman

* Seleccionar peticion: POST
* En recuadro URL teclear <http://localhost:3000/usuario>
* Click a boton SEND

*En ventana inferior en opcion body se despliega*

*‘post usuario’*

En Visual Studio Code

En file server.js

* Poner estas lineas debajo de la peticion ‘post’

app.put('/usuario', function(req, res) {

res.json('put usuario')

})

app.delete('/usuario', function(req, res) {

res.json('delete usuario')

})

En postman

* Seleccionar peticion: DELETE
* En recuadro URL teclear <http://localhost:3000/usuario>
* Click a boton SEND

*En ventana inferior en opcion body se despliega*

*‘delete usuario’*

* Seleccionar peticion: PUT
* En recuadro URL teclear <http://localhost:3000/usuario/8137982>
* Click a boton SEND

*En ventana inferior en opcion body se despliega*

*‘Cannot PUT /usuario/8137982’*

*La razon es que no se ha programado para que reciba un dato despues de* ‘/usuario/’

PROGRAMAR PARA RECIBIR UN ID DESPUES DE ‘/USUARIO/’ EN PETICION ‘PUT’

En Visual Studio Code

En file server.js

* Poner / Modificar las lineas marcadas en azul

app.put('/usuario/:id', function(req, res) {

let id = req.params.id;

res.json({

id

});

});

SIMULAR PETICION ‘PUT’

En postman

* Seleccionar peticion: PUT
* En recuadro URL teclear <http://localhost:3000/usuario/8137982>
* Click a boton SEND

*En ventana inferior en opcion body se despliega*

{

"id": "81379182"

}

En postman

en menu horizontal segunda linea,

* click a opcion ‘Body ⚫’
* click a boton-radio ‘x-www-form-urlencoded’
* teclear en recuadros debajo de columnas: ‘key’ y ‘value’

nombre | ernesto

edad | 59

mail | eglobo911@gmail.com

PROGRAMAR server.js PARA RECIBIR LOS DATOS ENVIADOS POR POSTMAN

INSTALAR BODYPARSER

En navegador web

* teclear ‘npm bodyparser’
* click a opcion ‘body-parser npm’

*se abre pagina ‘body-parser’*

* ver la instrucción

npm i body-parser

En consola

* teclear y ejecutar la instrucción ‘npm i body-parser - - save’

*ver que se instale exitosamente*

En navegador web

en pagina ‘npm-body-parser’

* ver instruccion

var bodyParser **=** require('body-parser')

*// parse application/x-www-form-urlencoded*

app.use(bodyParser.urlencoded({ extended**:** false }))

*// parse application/json*

app.use(bodyParser.json())

En visual studio

En file ‘server.js’

* Poner las lineas en azul

const app = express()

const bodyParser = require('body-parser')

app.use(bodyParser.urlencoded({ extended: false }))

*// parse application/json*

app.use(bodyParser.json())

en visual studio

en file ‘server.js’

* Poner/Modificar las lineas marcadase en azul

app.post('/usuario', function(req, res) {

let body = req.body;

res.json({

persona: body

});

});

En postman

* Seleccionar peticion: POST
* En recuadro URL teclear <http://localhost:3000/usuario>
* Click a boton SEND

*En ventana inferior en opcion body se despliega*

{

"persona": {

"nombre": "Ernesto",

"edad": "59",

"correo": "eglobo911@gmail.com"

}

}

*Lo que indica que ‘server.js’ recibio como el objeto ‘persona’ los datos enviados en el ‘Body ⚫’ de postman y le devolvio la instrucción de desplegar dicho objeto en formato ‘json’ y postman los desplego en la ventana inferior en la seccion ‘body’*

Sever.js

**CLASE 81 . Codigos de respuestas HTTP**

Objetivo: descargar pdf con los codigos



**CLASE 82 . Usando codigos de respuesta HTTP en Express**

Objetivo: usar en express los codigos de respuesta enviados por HTTP para ejectuar alguna accion

En visual studio code

En file ‘server.js’

* Poner las siguiente instrucción ‘if’ en la funcion ‘app.post’ despues de ‘let’

app.post('/usuario', function(req, res) {

let body = req.body;

*if* (body.nombre === undefined) {

res.status(400).json({

ok: false,

mensaje: 'El nombre es necesario'

});

} *else* {

res.json({

persona: body

});

}

});

*Esta instrucción checara si al recibirse del cliente una peticion ‘post’ , viene el nombre en el body de la peticion, esto se detecta si se recibio el codigo 400 desde HTTP; de no recibirse el nombre se enviara una respuesta indicandolo*

En postman

* En el body quitar el ‘check’ en el recuadro del ‘nombre’
* Click a ‘send’

*En el status aparece el codigo 400*

*En la ventana de body se despliegan los textos indicados en la instrucción ‘if’, en el objeto ‘res.status’*

Sever.js

**CLASE 83 . Creando un archivo de configuracion global**

Objetivo: crear un archivo que nos permita cambiar del ambiente de desarrollo al de produccion (cambiando el puerto a escuchar) sin tener que modificar el codigo en los programas que levantan el servidor en cada uno de los ambientes

En visual studio code

* En folder ‘server’ crear folder ‘config’
* En folder ‘config’ crear archivo ‘config.js’

En archivo ‘config.js’

* Poner esta linea

process.env.PORT = process.env.PORT || 3000;

En archivo ‘server.js’

* Al inicio poner esta linea

require('./config/config');

*Esta instrucción le indica al programa que lea y ejecute el archivo ‘config.js’ en el cual esta la instrucción de ‘process.env.PORT’ puesta para designar el puerto a escuchar*

* Modificar estas lineas como se indica

app.listen(process.env.PORT, () => {

    console.log('Escuchando puerto: ', process.env.PORT);

En consola

* Teclear: Cd C:\Users\EG\_Lobo\Documents\Mis documentos\aa Major 1 EGB\EGB EDU\TAKEN\NODE\07-restserver
* Teclear: ‘nodemon server/server’ (para levantar servidor)
* Se despliegan varias lineas
* Se despliega el mensaje ‘Escuchando puerto: 3000’

En visual studio code

En archivo ‘config.js’

* Modificar esta linea como se indica

process.env.PORT = process.env.PORT || 3001;

En consola

* Se despliegan varias lineas
* Se despliega el mensaje ‘Escuchando puerto: 3001’

En visual studio code

En archivo ‘config.js’

* Modificar esta linea como se indica

process.env.PORT = process.env.PORT || 3000;

En consola

*Se despliegan varias lineas*

*Se despliega el mensaje ‘Escuchando puerto: 3000’*

Config.js

Sever.js

**CLASE 84 . Respaldo del RESTserver a GitHub**

CREAR REPOSITORIO EN ‘GITHUB.COM’

En navegador web:

En pagina github <https://github.com>

* Sign in con usuario y contraseña
* Click en ‘Nuevo repositorio’ (boton verde)
* Dar nombre y descripcion a repositorio (07-restserver)
* Click a boton ‘Public’
* Click en ‘Crear repositorio’ (boton verde)

INICIALIZAR GIT

En consola:

* cd / … /07-restserver

*Ubicarse en el path que se desea respaldar*

* ‘git init’

*Inicializar el repositorio*

* ‘git status’

*Ver el status de git; aparecen todos los archivos grabados en el path*

INDICAR QUE ARCHIVOS NO RESPALDAR

En VS-Code:

* En el folder ‘07-restserver’, crear un archivo: ‘.gitignore’
* En este archivo teclear los archivos que NO se desea respaldar

*Ej ‘node\_modules/’, ‘tablas/\*.txt’*

* Grabar cambios y ver que dichos archivos se ponen en color gris

AGREGAR A ‘GIT’ ARCHIVOS A RESPALDAR

En consola:

* Git status

*Ya no debe aparecer el elemento ‘node\_modules’*

* Git add .

*Especificar que adicione los archivos que se van a respaldar*

* ‘git status’

*Deben aparecer los archivos adicionados en color verde*

* git commit -m “Commit inicial”

*Respaldar de los archivos; Se muestra la lista de todos los archivos que se respaldaran*

* git status

*debe decir que el respaldo del branch esta actualizado y no hay nada por respaldar, trabajando en arbol limpio*

AGREGAR EL REPOSITORIO A GIT

En navegador web

En pagina github <https://github.com>

* Click a HTTPS (boton gris)
* Seleccionar esta linea, copiarla a la consola y ejecutarla:

*git remote add origin* [*https://github.com/eglobo911/07-restserver.git*](https://github.com/eglobo911/07-restserver.git)

* Seleccionar esta linea, copiarla a la consola y ejecutarla:

*git push -u origin master*

*La primera vez pedira usuario y contraseña*

* F5
* Click a pestaña ‘Code’

Se despliegan todos los archivos respaldados

AGREGAR UN ‘README’

En Visual Studio Code

* Crear un nuevo archivo ‘README.md’
* Escribir el texto que se quiera mostrar y guardar
* El archivo ‘README.md’ se pone en verde (porque es nuevo y no se ha respaldado)

En consola:

* git status

*Se despliega el archivo ‘README.md’ en rojo*

* git add .

*se agrega lo que no esta respaldado, en este caso ‘README.md’*

* git status

*debe aparecer el archivo ‘README.md’ en verde*

* git commit -m “agregamos readme.md”

*se respalda el ‘README.md’*

* git push

*sube el ‘README.md’ a github*

En navegador web

En pantalla github.com:

* Refrescar pantalla y ver que aparezca el README.md

AGREGAR UN TAG DEL ‘RELEASE’

En consola

* git tag -a v0.0.1 -m "Alpha - inicio de proyecto"

crea el git tag

* git tag’

se despliega la version tecleada

* git push - -tags

sube el tag a ‘github’

En navegador

En pagina ‘github’

* En pestaña ‘releases’ debe aparecer el tag subido
* En pestaña ‘tags’ se pueden escribir notas – click en los 3 puntos; new release

*Las notas se publican con la opcion ‘Publish releases’ (boton verde)*

* En pestaña ‘releases’ debe aparecer el ‘tag’ y las ‘notas’ agregadas

**CLASE 85 . Subir el RESTserver a Heroku**

Objetivo: Subir la aplicación al servidor Heroku en la nube

CREAR REPOSITORIO EN EL SERVIDOR HEROKU

En consola teclear:

* Teclear: cd / …. /07-restserver

*ubicarse en el path del proyecto*

* Teclear: heroku login
* Teclear: heroku create -h

*se despliegan mensajes de la actividad*

* Teclear: heroku create

*se despliegan mensajes, en una linea aparece el url con el nombre del repositorio que se creo (este nombre lo asigna heroku)*

* Teclear: git remote

*se despliegan*

*‘heroku’*

*‘origin’*

* Teclear: git remote -v

*se despliegan urls de ubicaciones creadas de ‘fetch’ y ‘push’*

SUBIR APLICACIÓN AL SERVIDOR HEROKU

* Teclear: git push heroku master

*se despliegan mensajes, en una de las ultimas lineas se despliega el URL para accesar la aplicación creada* https://fast-scrubland-09379.herokuapp.com

* Teclear: heroku open

*se abre el navegador web*

En navegador web

*Intenta mostrar la pagina pero solo muestra el mensaje ‘Application error’; la razon es porque intenta ‘arrancar’ el servidor pero no puede porque no se ha puesto en el file ‘package.json’ la instrucción ‘start’ correspondiente*

PONER INSTRUCCIÓN PARA ‘LEVANTAR’ EL SERVIDOR DESDE ‘SERVER.JS’

En visual studio code

en file ‘package.json’

* Poner / modificar lineas en azul

    "scripts": {

        "start": "node server/server.js",

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

    }

SUBIR APLICACIÓN AL SERVIDOR

En consola

* Teclar: Git status

*Se despliegan mensajes, uno de ellos indica que fue modificado el ‘package.json’*

* Teclear: Git commit -am “Npm start listo”

*Se despliegan mensajes*

* Teclear: Git push heroku master

*Se despliegan mensajes de que se esta subiendo la aplicacion, casi al final se despliega el URL del path a donde fue subida la aplicación (es un nombre que asigna heroku)*

https://fast-scrubland-09379.herokuapp.com

En navegador web

En la pagina que se abrio

* Refrescar (F5)

*Se despliega ‘Cannot GET /’ (porque el file server esta programado para recibir ‘/usuario’)*

* En prompt de url teclear: <https://fast-scrubland-09379.herokuapp.com/usuario>

*se despliega el msg “get usuario” que es el que se programo en ‘server.js’ para una peticion ‘get’*

app.get('/usuario', function(req, res) {

    res.json('get usuario')

})

SIMULAR CON POSTMAN QUE SE RECIBEN PETICIONES DEL CLIENTE POR EL PUERTO DEFINIDO EN ‘CONFIG.JS’

En postman

* Seleccionar peticion ‘POST’
* en prompt para url, teclear: <https://fast-scrubland-09379.herokuapp.com/usuario>
* click a boton ‘SEND’

*Postman manda la peticion ‘POST’ al programa ‘server.js’ pero no le envia el ‘nombre’ por lo que ‘server.js’ le devuelve a Postman el mensaje ‘El nombre es necesario’ tal como fue programado;*

app.post('/usuario', function(req, res) {

    let body = req.body;

*if* (body.nombre === undefined) {

        res.status(400).json({

            ok: false,

            mensaje: 'El nombre es necesario'

        });

    } *else* {

        res.json({

            persona: body

        });

    }

});

*Postman lo recibe y lo despliega en la ventana inferior ‘body’*

{

"ok” : false

"El nombre es necesario"

}

}

en menu horizontal segunda linea,

* click a opcion ‘Body’
* click a boton-radio ‘x-www-form-urlencoded’
* en recuadros debajo de columnas: ‘key’ y ‘value’ teclear:

nombre | ernesto

* click a boton ‘SEND’

*postman envia al puerto 3000, (el cual lo esta escuchando ‘server.js’) la peticion ‘POST’ y el campo ‘nombre’ con el dato ‘ernesto’; ‘server.js’ valida lo recibido y dado que es correcto le devuelve a ‘postman’ el objeto ‘persona’ con el campo ‘nombre’ con el contenido ‘ernesto’; Postman lo despliega en la ventana abajo ‘body’*

{

"persona": {

"nombre": "ernesto"

}

}

En postman

* seleccionar peticion ‘DELETE’
* click a boton ‘send’

*se despliega el mensaje ‘delete usuario’ tal como se programo en ‘server.js*

app.delete('/usuario', function(req, res) {

    res.json('delete usuario')

})

En postman

* seleccionar peticion ‘PUT’
* en prompt para url, teclear: <https://fast-scrubland-09379.herokuapp.com/usuario/ernesto>
* click a boton ‘send’

*se despliega el mensaje “id” : “ernesto” tal como se programo en ‘server.js*

app.put('/usuario/:id', function(req, res) {

    let id = req.params.id;

    res.json({

        id

    });

});

**CLASE 86 . Pro Tip: Ambiente de produccion y desarrollo en Postman**

Objetivo: configurar postman para poder cambiar las peticiones a la aplicación en el servidor local (ambiente de ‘Desarrollo’) a la aplicación en la nube (ambiente de ‘Produccion’) y viceversa sin tener que teclear los ‘URL’ de cada uno cada vez que lo hagamos

SIMULAR PETICION ‘PUT’ A LA APLICACIÓN SUBIDA A LA NUBE

En Postman

* en recuadro para url, teclear: <https://fast-scrubland-09379.herokuapp.com/usuario/ernesto>
* Selecciónar opcion ‘PUT’
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto “id”: “ernesto” respondido por la aplicación en la nube (ambiente ‘Produccion’)

LEVANTAR SERVIDOR PUERTO 3000

En consola

* cd / …. /07-restserver
* nodemon server/server

se despliegan lineas indicando actividad, la ultima dice ‘escuchando puerto: 3000’

SIMULAR PETICION ‘GET’ A LA APLICACIÓN LOCAL PUERTO 3000

En Postman

* en recuadro para url, teclear: <http://localhost:3000/usuario>
* Selecciónar opcion ‘GET’
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto ‘get usuario’ respondida por la aplicación en el servidor local (ambiente ‘desarrollo’)

SIMULAR PETICION ‘GET’ A LA APLICACIÓN SUBIDA A LA NUBE

En Postman

* en prompt para url, teclear: <https://fast-scrubland-09379.herokuapp.com/usuario>
* Selecciónar opcion ‘GET’
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto ‘get usuario’ respondida por la aplicación en la nube

CONFIGURAR AMBIENTES EN POSTMAN

En postman

* En parte superior derecha ver recuadro de ‘Enviroment’ que muestra por default la opcion ‘No Environment’
* Click a ‘engrane’ en parte superior derecha

*Se abre ventana ‘MANAGE ENVIRONMENTS’*

CONFIGURAR AMBIENTE PRODUCCION

* Click en boton ‘Add’

*Se abre ventana ‘Add Evironment’*

* En recuadro ‘Environment Name’ teclear ‘Produccion’
* En recuadro ‘Variable’ teclear ‘url’

En recuadro ‘Initial value’ teclear: https://fast-scrubland-09379.herokuapp.com

* Click a boton ‘Add’

Se despliega en una linea el ambiente ‘Produccion’ creado

* Se despliegan dos botones en la parte inferior ‘Cancel’ y ‘Update’
* Click a boton ‘cancel’

CONFIGURAR AMBIENTE DESARROLLO

* Click a boton ‘Add’

Se abre ventana ‘Add Evironment’

* En recuadro ‘Environment Name’ teclear ‘Desarrollo’
* En recuadro ‘Variable’ teclear ‘url’

En recuadro ‘Initial value’ teclear: <http://localhost:3000>

* Click a boton ‘Add’

Se despliega en otra linea el ambiente ‘Desarrollo’ creado

* Cerrar ventana ‘MANAGE ENVIRONMENTS’
* en prompt para url, teclear: {{url}}/usuario

HACER PETICION A LA APLICACIÓN EN SERVIDOR LOCAL

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

Aparecen los dos ambientes creados

* Seleccionar ‘Desarrollo’
* Seleccionar opcion ‘GET’
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto ‘get usuario’ respondido por la aplicación en servidor local

HACER PETICION A LA APLICACIÓN EN LA NUBE

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

Aparecen los dos ambientes creados

* Seleccionar ‘Produccion’
* Seleccionar opcion ‘GET’
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto ‘get usuario’ respondido por la aplicación en la nube

COMPROBAR QUE FUNCIONA CORRECTAMENTE LA CONFIGURACION DE AMBIENTES

En visual studio code

En ‘server.js’

* Modificar esta funcion como se indica

app.get('/usuario', function(req, res) {

    res.json('get usuario LOCAL !!!')

})

En postman

HACER PETICION A LA APLICACIÓN EN SERVIDOR LOCAL

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

Aparecen los dos ambientes creados

* Seleccionar ‘Desarrollo’
* Seleccionar opcion ‘GET’
* Click a boton ‘SEND’

*Se simula la peticion a la aplicacion en el servidor local*

*En ventana ‘body’ abajo se despliega el texto ‘get usuario LOCAL !!!’ respondido por la aplicación en el servidor local*

HACER PETICION A LA APLICACIÓN EN LA NUBE

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

Aparecen los dos ambientes creados

* Seleccionar ‘Produccion’
* Seleccionar opcion ‘GET’
* Click a boton ‘SEND’

*Se simula la peticion a la aplicación en la nube*

*En ventana ‘body’ abajo se despliega el texto ‘get usuario’ respondido por la aplicación en la nube ya que no se ha subido a la nube el archivo ‘server.js’ modificado*

**SECCION 9: ALCANCES DEL REST SERVER Y MANTENIMIENTO**

**CLASE 91 . Mongoose – conectarnos a la base de datos**

Objetivo: conectarse a la base de datos ‘café ‘ usando la API ‘Mongoose’

LEVANTAR EL SERVICIO MongoDB

En consola

Abrir una primera sesion

* Teclear

cd 07-restserver

* Teclear

"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="c:\data\db"

*Esta instrucción ‘levanta/arranca’ el servicio de MongoDB; Importante: Deben ponerse las comillas al inicio y al final de la instrucción;*

*Al ejecutarla se muestran una lista de mensajes relacionados con el arranque, en la segunda linea se muestra ‘port=27017’; este puerto sera utilizado despues*

*al final se despliega el mensaje*

[initandlisten] waiting **for** connections on port 27017

*Que indica que esta corriendo el servicio de MongoDB*

* Cuando se quiera ‘bajar/cancelar’ el servicio de MongoDB teclear ‘Cntrl-C’

En consola

Abrir una segunda sesion

* Teclear: cd c:/…./07-restserver

ABRIR PAGINA MONGOOSE - NPM

En navegador web

* Teclear npm mongoose
* Click en Mongoose – npm

*Se abre la pagina*

* En la seccion ‘Documentation’ click en link ‘Monggosejs.com’

*Se abre la pagina*

* Click a boton ‘read the docs’

*Se abre pagina*

* Ver la version de Mongoose en la parte superior izquierda
* Regresar hasta pagina ‘Mongoose npm’
* Selecciónar la instrucción ‘npm i mongoose’y copiarla a la consola en la segunda sesion

INSTALAR MONGOOSE

En consola

En segunda sesion

* Completar la instruccion ‘npm i mongoose --save’

*Esta instrucción instala ‘Mongoose’; se despliegan varias lineas y al final se despliega la version que se instalo*

PROGRAMAR USO DE MONGOOSE Y CONEXIÓN A BASE DE DATOS ‘CAFÉ’

En navegador web

En pagina ‘Mongoose npm’

* Ir a la seccion ‘Importing’
* Seleccionar la instrucción const mongoose **=** require('mongoose'); y copiarla al file ‘server.js’ donde se indica

const express = require('express');

const mongoose = require('mongoose');

const app = express()

En navegador web

En pagina ‘Mongoose npm’

* Ir a la seccion ‘Overview’
* Seleccionar la instrucción

mongoose.connect('mongodb://localhost/my\_database', {

  useNewUrlParser**:** true,

  useUnifiedTopology**:** true

});

* copiarla a ‘server.js’ antes de la instrucción ‘app.listen’

En visual studio

En file ‘server.js’

* Hacer las modificaciones a la instrucción ‘mongoose.connect’ como se indica: (borrar las lineas ‘use..’ y poner la instrucción ‘if’)

mongoose.connect('mongodb://localhost:27017/cafe', (err, res) => {

*if* (err) *throw* err;

    console.log('Base de datos ONLINE');

});

app.listen(process.env.PORT, () => {

    console.log('Escuchando puerto: ', process.env.PORT);

});

*La instrucción ‘mongoose.connect’ hace conexión con la base de datos ‘café’ a traves del puerto ‘localhost:27017’ que fue ‘levantado’ en la consola en la primera sesion con la instrucción ‘mongod.exe’;*

*A continuacion checa si la conexión fue exitosa y despliega el mensaje ‘Base de datos ONLINE’, si no fue exitosa despliega el error tal como lo envia la consola y detiene el proceso*

PROBAR SI LA CONEXIÓN ES EXITOSA

En consola en segunda sesion

* Teclear ‘nodemon server/server’

*Despliega mensajes de actividad*

*Si la conexión es exitosa, al final se despliegan las lineas:*

*‘Escuchando puerto: 3000’*

*‘Base de datos ONLINE’*

*Si la conexión no es exitosa por alguna razon, (puerto incorrecto, etc) se despliegan las lineas:*

*‘Escuchando puerto: 3000’*

*‘(node:11028) UnhandledPromiseRejectionWarning: MongoNetworkError: failed to connect to server [localhost:2701] on first connect’ … y otras lineas relacionadas con el error*

*Notese que se esta haciendo conexión con la base de datos ‘café’ la cual no ha sido creada en este proceso de ‘mongoose’ aunque ya antes con ‘robo 3T’ se habia creado una base de datos con este mismo nombre*

* Teclear ‘cntrl-C’ para bajar el servidor

BORRAR LA BASE DE DATOS ‘CAFÉ’ CREADA ANTES CON ROBO 3T

En escritorio

* Abrir ‘robo 3T’ con el icono en escritorio

En ‘robo 3T’

*Se abre la pagina principal e inmediatamente se abre automaticamente la ventana ‘MongoDB connections’*

* Seleccionar la direccion ‘localhost: 27017’
* Click a boton ‘connect’
* En menu vertical a la izquierda, click derecho a base de datos ‘café’
* Click a ‘drop database’

*Ver que se borra esa base de datos del menu*

En consola en segunda sesion

* Teclear ‘nodemon server/server’

*Despliega mensajes de actividad*

*Si la conexión es exitosa, al final se despliegan las lineas:*

*‘Escuchando puerto: 3000’*

*‘Base de datos ONLINE’*

*Notese que la conexión es exitosa aunque la base de datos ‘café’ fue borrada con ‘robo 3T’*

En ‘robo 3T’

* Doble click a linea ‘localhost(n)’ en menu vertical a la izquierda
* Click a ‘refresh’

*La base de datos ‘café’ no se despliega aunque la conexión con ‘mongoose’ fue hecha exitosamente; esto es normal y mas adelante se explicara porque; por el momento es suficiente con que se desplieguen las dos lineas de conexión exitosa*

**CLASE 92 . Ordenar las rutas de usuario**

Objetivo: Mover todo el codigo para el manejo de las peticiones ‘app.’ del archivo ‘server.js’ a un archivo nuevo ‘usuario.js’ de manera que puedan ser invocadas por ‘server.js’ y por cualquier otro archivo y ademas si es necesario hacer una modificacion o actualizacion, solo se tenga que hacer una sola vez en este archivo ‘usuario.js’

En visual studio code

En folder ’07-RESTSERVER/server’

* Crear folder ‘routes’

En folder ’07-RESTSERVER/server/routes’

* Crear archivo ‘usuario.js’

En archivo ‘server.js’,

* seleccionar estas lineas y ‘moverlas’ (cut-paste) al archivo ‘usuario.js’

app.get('/usuario', function(req, res) {

    res.json('get usuario LOCAL !!!')

})

app.post('/usuario', function(req, res) {

    let body = req.body;

*if* (body.nombre === undefined) {

        res.status(400).json({

            ok: false,

            mensaje: 'El nombre es necesario'

        });

    } *else* {

        res.json({

            persona: body

        });

    }

});

app.put('/usuario/:id', function(req, res) {

    let id = req.params.id;

    res.json({

        id

    });

});

app.delete('/usuario', function(req, res) {

    res.json('delete usuario')

})

*Son las instrucciones para el manejo de las peticiones del cliente*

En archivo ‘usuario.js’

* poner estas lineas al inicio

const express = require('express');

const app = express()

*son las instrucciones para declarar el uso de la API ‘express’*

* poner esta linea al final

module.exports = app

*es la instrucción para permitir exportar el contenido de ‘app’ al archivo que invoco a ‘usuario.js’*

* modificar esta linea como se indica

    res.json('get usuario LOCAL with route/usuariojs !!!')

*para que cuando se simule la peticion, comprobar que se esta invocando correctamente al archivo ‘usuario.js’ y este es el que esta respondiendo a las peticiones*

En archivo ‘server.js’

* Poner esta linea

app.use(require('./routes/usuario'));

antes de esta linea

mongoose.connect('mongodb://localhost:27017/cafe', (err, res) => {

*es la instruccion para poder invocar al archivo ‘usuario.js’*

LEVANTAR SERVIDOR LOCAL PUERTO 3000 (AMBIENTE DESARROLLO)

En consola

* cd / …. /07-restserver
* nodemon server/server

se despliegan lineas indicando actividad, la ultimas dicen

‘Base de datos ON LINE’

‘escuchando puerto: 3000’

SIMULAR PETICIONES A LA APLICACIÓN LOCAL ‘PUERTO 3000’ (AMBIENTE DESARROLLO)

En postman

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

*Aparecen los dos ambientes creados*

* Seleccionar ‘Desarrollo’
* En recuadro para ‘url’ teclear

{{url}}/usuario

* Seleccionar opcion ‘GET’
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto

"get usuario LOCAL with route/usuariojs !!!"

* Seleccionar opcion ‘DELETE’
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto

"delete usuario"

* Seleccionar opcion ‘PUT’
* En recuadro para ‘url’ teclear

{{url}}/usuario

* Click a boton ‘SEND’

*Se despliega un mensaje de error del sistema*

*Al recibirse esta peticion esta programado que se reciba un dato despues de ‘*{{url}}/usuario’ *, al no recibirse se desplegara el mensaje de error del sistema ya que en el codigo no esta programado para validar si se recibio o no dicho dato y desplegar un texto por programa*

* En recuadro para ‘url’ teclear

{{url}}/usuario/lobo

* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto

{

"id": "lobo"

}

* Seleccionar opcion ‘POST’
* En recuadro para ‘url’ teclear

{{url}}/usuario

* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el texto

{

"ok": false,

"mensaje": "El nombre es necesario"

}

*Al recibirse esta peticion esta programado que valide que se recibe un objeto con los elementos ‘nombre’, ‘edad’, ‘correo’; el envio de estos datos se simula con postman con la opcion ‘body’, ‘x-www-form-urlencoded’ todos ellos con* ‘✓’ *(check); de no recibirse la informacion correctamente, se desplegara este mensaje de error del programa*

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘body’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear

nombre | ernesto

edad | 59

correo | eglobo911@gmail.com

* Todas las lineas deben tener el ‘✓’ en los recuadros del lado izquierdo
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega el objeto

{

"persona": {

"nombre": "Ernesto",

"edad": "59",

"correo": "eglobo911@gmail.com"

}

}

**CLASE 93 . Modelo de usuario**

Objetivo: Crear programa para definir un esquema de usuario con sus campos

En visual studio code

* En folder ‘server’ crear folder ‘models’
* En folder ‘models’ crear file ‘usuario.js’

En file ‘usuario.js’

* Poner el siguiente codigo

const mongoose = require('mongoose')

*Se utilizaran funcionalidades de moongose*

let Schema = mongoose.Schema;

*Obtener el ‘cascaron’ o ‘molde’ para crear esquemas de moongose*

let usuarioSchema = new Schema({

*Crear el ‘esquema’ con este nombre*

    nombre: {

        type: String,

        required: [true, 'El nombre es necesario']

    },

    email: {

        type: String,

        required: [true, 'El correo es necesario']

    },

    password: {

        type: String,

        required: [true, 'La contraseña es obligatoria']

    },

    img: {

        type: String,

        required: false

    },

    role: {

        type: String,

        default: 'USER\_ROLE'

    },

    estado: {

        type: Boolean,

        default: true

    },

    google: {

        type: Boolean,

        default: false

    }

});

*Definir los campos que tendra*

module.exports = mongoose.model('Usuario', usuarioSchema);

*Indicar que el esquema ‘usuariosSchema’ sera exportable al programa que invoco*

**CLASE 94 . POST: Creando un usuario en la coleccion**

Objetivo: Grabar un usuario en la base de datos ‘café en mongDB

ARRANCAR ‘MONGO DB’

En consola (primera sesion)

"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="c:\data\db"

*Esta instrucción ‘arranca’ el servicio de MongoDB; Importante: Deben ponerse las comillas al inicio y al final de la instrucción;*

*Al ejecutarla se muestran una lista de mensajes relacionados con el arranque, en la segunda linea se muestra ‘port 27017’; este puerto sera utilizado despues*

*al final aparece el mensaje*

[initandlisten] waiting **for** connections on port 27017

*Que indica que esta corriendo el servicio de MongoDB*

* Cuando se quiera ‘cancelar’ el servicio de MongoDB teclear ‘Cntrl-C’

LEVANTAR SERVIDOR LOCAL PUERTO 3000 (AMBIENTE DESARROLLO)

En consola (segunda sesion)

* cd / …. /07-restserver
* nodemon server/server

*se despliegan lineas indicando actividad, la ultimas dicen*

‘escuchando puerto: 3000’

‘Base de datos ON LINE’

SIMULAR PETICIONES A LA APLICACIÓN LOCAL ‘PUERTO 3000’ (AMBIENTE DESARROLLO)

En postman

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

*Aparecen los dos ambientes creados*

* Seleccionar ‘Desarrollo’
* Seleccionar opcion ‘POST’
* En recuadro para ‘url’ teclear

{{url}}/usuario

* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega lo siguiente:

{

"persona": {

"nombre": "Ernesto",

"edad": "59",

"correo": "eglobo911@gmail.com"

}

}

*Este objeto es el que se envia en el ‘body’ de postman (simulando una peticion del cliente)*

En Visual Studio

En file ‘routes/usuario.js’

* Poner el siguiente codigo en la segunda linea

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

*se crea el objeto ‘Usuario’ requiriendo el file ‘routes/usuario.js’ para utilizar el esquema definido ahi*

* en el bloque ‘app.post’ poner las lineas nuevas que se indican y quitar las lineas que ya no aparecen

app.post('/usuario', function(req, res) {

    let body = req.body;

    let usuario = new Usuario({

        nombre: body.nombre,

        email: body.email,

        password: body.password,

        role: body.role

    });

*Se crea el objeto ‘usuario’ con el molde ‘Usuario’ y se le asignan los valores del ‘body’*

    usuario.save((err, usuarioDB) => {

*if* (err) {

*return* res.status(400).json({

                ok: false,

                err

            });

        }

        res.json({

            ok: true,

            usuario: usuarioDB

        });

*grabar el objeto y poner una condicion para el control en caso de error*

    });

});

app.put('/usuario/:id', function(req, res) {

PROBAR LA GRABACION SIMULANDO UNA PETICION DE CLIENTE

En postman

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

*Aparecen los dos ambientes creados*

* Seleccionar ‘Desarrollo’
* Seleccionar opcion ‘POST’
* En recuadro para ‘url’ teclear

{{url}}/usuario

* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega lo siguiente:

{

"ok": false,

"err": {

"errors": {

"password": {

"message": "La contraseña es obligatoria",

"name": "ValidatorError",

"properties": {

"message": "La contraseña es obligatoria",

"type": "required",

"path": "password"

},

"kind": "required",

"path": "password"

}

},

"\_message": "Usuario validation failed",

"message": "Usuario validation failed: password: La contraseña es obligatoria",

"name": "ValidationError"

}

}

*Este objeto es el que se envia desde el ‘body’ de postman (simulando envio del cliente);*

*Notese que hay mensajes de error controlados por el codigo de ‘models/usuarios.js’*

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

email | eglobo911@gmail.com

password | 123456

* Todas las lineas deben tener el ‘✓’ en los recuadros del lado izquierdo
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega lo siguiente:

{

"ok": true,

"usuario": {

"role": "USER\_ROLE",

"estado": true,

"google": false,

"\_id": "5d9cd8efaf46b623688fb483",

"nombre": "Ernesto",

"email": "eglobo911@gmail.com",

"password": "123456",

"\_\_v": 0

}

}

*Ya no arroja ningun error*

*Se muestra el objeto con los datos de simulacion enviados por ‘postman’*

*Este objeto ya se debio haber grabado en la base de datos ‘café’ de ‘MongoDB’ la cual fue declarada con esta instrucción en ‘server.js’*

mongoose.connect('mongodb://localhost:27017/cafe', (err, res) => {

*if* (err) *throw* err;

    console.log('Base de datos ONLINE');

});

COMPROBAR SI SE GRABO EL USUARIO EN LA BASE DE DATOS MongoDB

* En robo 3T
* En ventana ‘MongoDB connections’, conectarse a la conexión ‘localhost:27017’
* En menu izquierdo vertical, click derecho a ‘localhost(n)’ – opcion ‘refresh’
* Doble click a base de datos ‘café’ (debe estar creada)
* Doble click a ‘collections’ (debe estar tabla ‘usuarios’)
* Doble click a tabla ‘usuarios’ – en ventana principal debe estar el documento ‘Objectid(…)’ que es el objeto recien grabado
* Doble click al documento ‘Object(..)’ – deben aparecer los campos grabados que fueron desplegados en la ventana de ‘postman’

GUARDAR EL SERVICIO ‘POST’ QUE SE ACABA DE UTILIZAR EN POSTMAN

En postman

* Click a boton ‘Save’ en parte superior derecha – se abre ventana ‘SAVE REQUEST’
* En recuadro ‘Request name’ se muestra ‘{(url)}/usuario’, teclear: ‘POST: Crear usuario’
* En recuadro ‘Request description (optional)’ teclear: ‘Creamos un usuario de BD’
* Click a opcion ‘create collection’ (color naranja)
* En recuadro ‘all collections’ teclear: ‘Café-Udemy’
* Click a boton naranja ‘Save to Café-Udemy’
* En pantalla principal, en parte izquierda, click a opcion ‘collections’ – debe mostrarse esta colección recien creada
* Abrir la colección ‘Café-Udemy’ – debe mostrarse el servicio recien grabado ‘POST: Crear usuario’

Sever.js (levantar servidor)

Config.js (configuracion de # de puerto)

Models/usuario.js (Definicion de Schema)

Routes/usuario.js (Manejo de Peticiones)

**CLASE 95 . Validaciones personalizadas – email y role**

Objetivo: Validar que no se graben registros con ‘email’ duplicado o con ‘role’ invalido

ARRANCAR ‘MONGO DB’

En consola (primera sesion)

"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="c:\data\db"

*Esta instrucción ‘arranca’ el servicio de MongoDB; Importante: Deben ponerse las comillas al inicio y al final de la instrucción;*

*Al ejecutarla se muestran una lista de mensajes relacionados con el arranque, en la segunda linea se muestra ‘port 27017’; este puerto sera utilizado despues*

*al final aparece el mensaje*

[initandlisten] waiting **for** connections on port 27017

*Que indica que esta corriendo el servicio de MongoDB*

* Cuando se quiera ‘cancelar’ el servicio de MongoDB teclear ‘Cntrl-C’

LEVANTAR SERVIDOR LOCAL PUERTO 3000 (AMBIENTE DESARROLLO)

En consola (segunda sesion)

* cd / …. /07-restserver
* nodemon server/server

*se despliegan lineas indicando actividad, las ultimas dicen*

‘escuchando puerto: 3000’

‘Base de datos ON LINE’

SIMULAR VARIAS PETICIONES CON LOS MISMOS DATOS Y VER QUE SE GRABAN REGISTROS DUPLICADOS EN LA BASE DE DATOS MongoDB

* Click a boton ‘SEND’ cinco veces

En ventana ‘body’ abajo se despliega lo siguiente:

{

"ok": true,

"usuario": {

"role": "USER\_ROLE",

"estado": true,

"google": false,

"\_id": "5d9cd8efaf46b623688fb483",

"nombre": "Ernesto",

"email": "eglobo911@gmail.com",

"password": "123456",

"\_\_v": 0

}

}

*Estos textos se desplegaran cada vez que se de click a ‘SEND’*

COMPROBAR SI SE GRABARON VARIOS DOCUMENTOS DE USUARIO DUPLICADOS EN LA BASE DE DATOS MongoDB

En robo 3T

* En ventana ‘MongoDB connections’, conectarse a la conexión ‘localhost:27017’
* En menu izquierdo vertical, click derecho a ‘localhost(n)’ – opcion ‘refresh’
* Click a base de datos ‘café’ (debe estar creada)
* Click a ‘collections’ (debe estar tabla ‘usuarios’)
* Doble click a tabla ‘usuarios’ – en ventana principal deben mostrarse cinco documentos ‘Objectid(…)’ que son los documentos recien grabados por el numero que las veces que se dio click a ‘SEND’
* Doble click a cada uno de los documentos ‘Object(..)’ – en todos deben aparecer los mismos campos grabados que son los que fueron desplegados en la ventana de ‘postman’

PROGRAMAR QUE NO PERMITA GRABAR REGISTROS CON EL ‘email’ DUPLICADO

En visual studio code

En archivo models/usuario.js

* Poner la linea marcada en azul

email: {

        type: String,

        unique: true,

        required: [true, 'El correo es necesario']

    },

BORRAR TODOS LOS REGISTROS EN MongoDB PARA HACER LA PRUEBA

En Robo 3T

* En ventana ‘MongoDB connections’, conectarse a la conexión ‘localhost:27017’
* En menu izquierdo vertical, click derecho a ‘localhost(n)’ – opcion ‘refresh’
* Click a base de datos ‘café’ (debe estar creada)
* Click a ‘collections’ (debe estar tabla ‘usuarios’)
* Doble click a tabla ‘usuarios’ – en ventana principal deben mostrarse cinco documentos con el mismo contenido ‘Objectid(…)’ que son los documentos recien grabados (el mismo numero que las veces que se dio click a ‘SEND’)
* Seleccionar y borrar todos los documentos

SIMULAR PETICION Y GRABAR UN PRIMER DOCUMENTO EN MongoDB

En postman

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

*Aparecen los dos ambientes creados*

* Seleccionar ‘Desarrollo’
* Seleccionar opcion ‘POST’
* En recuadro para ‘url’ teclear

{{url}}/usuario

* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliega

{

"ok": true,

"usuario": {

"role": "USER\_ROLE",

"estado": true,

"google": false,

"\_id": "5d9cd8efaf46b623688fb483",

"nombre": "Ernesto",

"email": "eglobo911@gmail.com",

"password": "123456",

"\_\_v": 0

}

}

*Se recibio la informacion correctamente y debe grabarse un documento en MongoDB*

CHECAR SI SE GRABO DOCUMENTO EN ‘MongoDB’

En Robo 3T

* En ventana ‘MongoDB connections’, conectarse a la conexión ‘localhost:27017’
* En menu izquierdo vertical, click derecho a ‘localhost(n)’ – opcion ‘refresh’
* Click a base de datos ‘café’ (debe estar creada)
* Click a ‘collections’ (debe estar tabla ‘usuarios’)
* Doble click a tabla ‘usuarios’ – en ventana principal debe mostrarse el documento ‘Objectid(…)’ con el contenido que se envio con Postman

SIMULAR PETICION Y PROBAR QUE NO SE GRABE OTRO DOCUMENTO EN MongoDB

* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliegan lineas de mensaje, las relevantes son:

"ok": false,

"message": "E11000 duplicate key error collection ….",

*Las cuales indican que hay un error porque se intenta grabar un registro con un ‘email’ duplicado; este msg lo envia el ‘postman’*

PROGRAMAR EL MANEJO DEL ERROR DESDE EL CODIGO

En navegador web

* Buscar el paquete ‘npm mongoose-unique-validator’
* Abrir ‘mongoose-unique-validator – npm’
* En la seccion ‘install’ ver la linea

‘npm i mongoose-unique-validator’

* En la seccion ‘usage’ ver las lineas

var mongoose **=** require('mongoose');

var uniqueValidator **=** require('mongoose-unique-validator');

*estas instrucciones seran utilizadas a continuacion*

En consola (en primera sesion)

* Teclear ‘npm i mongoose-unique-validator --save’

*Se hace la instalacion del paquete*

En visual studio code

* En archivo ‘models/usuario.js’
* Poner las lineas marcadas en azul

const mongoose = require('mongoose')

const uniqueValidator = require('mongoose-unique-validator')

*se crea esta constante para utilizar los recursos del paquete*

usuarioSchema.plugin(uniqueValidator, { message: '{PATH} debe ser unico' });

module.exports = mongoose.model('Usuario', usuarioSchema);

*se define el ‘plugin’ para la validacion de ‘email’ unico y el mensaje de error*

SIMULAR PETICION Y PROBAR QUE NO SE GRABE OTRO DOCUMENTO EN MongoDB Y QUE ENVIE MENSAJE PROGRAMADO EN EL CODIGO

En Postman

* En parte superior derecha, en recuadro de ‘Environments’, abrir menu vertical

*Aparecen los dos ambientes creados*

* Seleccionar ‘Desarrollo’
* Seleccionar opcion ‘POST’
* En recuadro para ‘url’ teclear

{{url}}/usuario

* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliegan las siguientes lineas

{

"ok": false,

"err": {

"errors": {

"email": {

"message": "email debe ser unico",

"name": "ValidatorError",

"properties": {

"message": "email debe ser unico",

"type": "unique",

"path": "email",

"value": "eglobo911@gmail.com"

},

"kind": "unique",

"path": "email",

"value": "eglobo911@gmail.com"

}

},

"\_message": "Usuario validation failed",

"message": "Usuario validation failed: email: email debe ser unico",

"name": "ValidationError"

}

}

*Las lineas marcadas en azul indican que hay un error porque se intenta grabar un registro con un ‘email’ duplicado; el programa se interrumpe y no se graba el documento*

SIMULAR ENVIO DE ‘ROLE’ Y HACER VALIDACION DEL MISMO AL RECIBIRLO

En postman

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

Email | test1@gmail.com

role| ADMIN\_ROLE

*se pone el dato ‘role’ y se pone otro correo para que no lo rechace como duplicado*

* Todas las lineas deben tener el ‘✓’ en los recuadros del lado izquierdo
* Click a boton ‘SEND’
* En ventana ‘body’ abajo se despliegan las siguientes lineas

{

"ok": true,

"usuario": {

"role": "ADMIN\_ROLE",

"estado": true,

"google": false,

"\_id": "5da61aaf8060651644571ad5",

"nombre": "Ernesto",

"email": "test1@gmail.com",

"password": "123456",

"\_\_v": 0

}

}

*Indicando que se acepto el registro y debio grabarse correctamente ya que el ‘email’ ya no esta duplicado*

* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

Email | test2@gmail.com

role| SUPER\_ROLE

* Click a boton ‘SEND’
* En ventana ‘body’ abajo se despliegan las siguientes lineas

{

"ok": true,

"usuario": {

"role": "SUPER\_ROLE",

"estado": true,

"google": false,

"\_id": "5da61aaf8060651644571ad5",

"nombre": "Ernesto",

"email": "test1@gmail.com",

"password": "123456",

"\_\_v": 0

}

}

*Indicando que tambien se acepto el registro y debio grabarse correctamente ya que el ‘email’ tampoco esta duplicado*

CHECAR QUE SI SE GRABARON LOS DOCUMENTOS EN ‘MongoDB’

En Robo 3T

* En ventana ‘MongoDB connections’, conectarse a la conexión ‘localhost:27017’
* En menu izquierdo vertical, click derecho a ‘localhost(n)’ – opcion ‘refresh’
* Click a base de datos ‘café’ (debe estar creada)
* Click a ‘collections’ (debe estar tabla ‘usuarios’)
* Doble click a tabla ‘usuarios’ – en ventana principal deben mostrarse los tres documentos ‘Objectid(…)’ que son los documentos enviados desde ‘Postman’

VALIDAR DESDE PROGRAMA QUE SOLO ACEPTE ‘ROLES’ PREDEFINIDOS

En visual studio

En archivo ‘models/usuario.js’

* Poner las lineas marcadas en azul

const uniqueValidator = require('mongoose-unique-validator')

let rolesValidos = {

    values: ['ADMIN\_ROLE', 'USER\_ROLE'],

    message: '{VALUE} no es un rol valido'

};

let Schema = mongoose.Schema;

*se crea un objeto para definir cuales roles seran validos y el mensaje en caso de que se reciba un rol invalido*

   role: {

        type: String,

        default: 'USER\_ROLE',

        enum: rolesValidos

    },

*Se indica que solo acepte los ‘roles’ definidos en el objeto ‘rolesValidos’*

En postman

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

Email | test3@gmail.com

role| SUPER\_ROLE

*se pone el dato ‘role’ y se pone otro correo para que no lo rechace como duplicado*

* Todas las lineas deben tener el ‘✓’ en los recuadros del lado izquierdo
* Click a boton ‘SEND’
* En ventana ‘body’ abajo se despliegan las siguientes lineas

{

"ok": false,

"err": {

"errors": {

"role": {

"message": "SUPER\_ROLE no es un rol valido",

"name": "ValidatorError",

"properties": {

"message": "SUPER\_ROLE no es un rol valido",

"type": "enum",

"enumValues": [

"ADMIN\_ROLE",

"USER\_ROLE"

],

"path": "role",

"value": "SUPER\_ROLE"

},

"kind": "enum",

"path": "role",

"value": "SUPER\_ROLE"

}

},

"\_message": "Usuario validation failed",

"message": "Usuario validation failed: role: SUPER\_ROLE no es un rol valido",

"name": "ValidationError"

*Las lineas en azul indican que no se recibio un role valido y no se debio grabar ningun registro*

SIMULAR ENVIO DE DATOS SIN EL NOMBRE Y EL EMAIL Y QUE SE DESPLIEGUEN LOS MENSAJES PROGRAMADOS EN EL CODIGO

En Postman

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* Quitar el ‘✓’ en los recuadros del lado izquierdo a la linea del ‘nombre’ y del ‘email’
* Click a boton ‘SEND’
* En ventana ‘body’ abajo se despliegan las siguientes lineas

{

"ok": false,

"err": {

"errors": {

"nombre": {

"message": "El nombre es necesario",

"name": "ValidatorError",

"properties": {

"message": "El nombre es necesario",

"type": "required",

"path": "nombre"

},

"kind": "required",

"path": "nombre"

},

"email": {

"message": "El correo es necesario",

"name": "ValidatorError",

"properties": {

"message": "El correo es necesario",

"type": "required",

"path": "email"

},

"kind": "required",

"path": "email"

},

"role": {

"message": "SUPER\_ROLE no es un rol valido",

"name": "ValidatorError",

"properties": {

"message": "SUPER\_ROLE no es un rol valido",

"type": "enum",

"enumValues": [

"ADMIN\_ROLE",

"USER\_ROLE"

],

"path": "role",

"value": "SUPER\_ROLE"

},

"kind": "enum",

"path": "role",

"value": "SUPER\_ROLE"

}

},

"\_message": "Usuario validation failed",

"message": "Usuario validation failed: nombre: El nombre es necesario, email: El correo es necesario, role: SUPER\_ROLE no es un rol valido",

"name": "ValidationError"

}

}

*Las lineas en azul indican cada uno los errores detectados en las validaciones y por los cuales no debio haberse grabado el registro; al final se muestra un resumen de todos los errores detectados*

config.js

Models/usuario.js

Routes/usuario.js

Server.js

**CLASE 96. Nota de actualizacion**

**CLASE 97 . Bcrypt – Encriptando la contraseña**

Objetivo: Encriptar la contraseña y hacer que no sea visible en los msgs enviados por sistema

ARRANCAR ‘MONGO DB’

ABRIR POSTMAN

LEVANTAR EL SERVIDOR PUERTO 3000

En consola sesion I

En MongoDB

* Abrir cualquiera de los documentos existentes, y ver que el password esta grabado como texto plano

BUSCAR EL PAQUETE ‘BCRYPT’

En navegador web

* Buscar ‘npm bcrypt’
* Doble click al paquete ‘bcrypt – npm’
* Localizar la instrucción

npm i bcrypt

* ver el contenido de este paquete especialmente la seccion ‘Usage’

INSTALAR EL PAQUETE ‘BCRYPT’

En consola (sesion II)

* Teclear ‘npm i bcrypt –save’
* Ver que se instale el paquete

CODIFICAR LA ENCRIPTACION DEL PASSWORD

En visual studio code

En ‘routes/usuario.js’

* Poner la linea marcada en azul

const express = require('express');

const bcrypt = require('bcrypt');

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

* modificar la linea marcada en azul

    let usuario = new Usuario({

        nombre: body.nombre,

        email: body.email,

        password: bcrypt.hashSync(body.password, 10),

        role: body.role

    });

PROBAR QUE EL PASSWORD SE ENCRIPTE Y SE GRABE ENCRIPTADO

En robo 3T

* Borrar todos los documentos existentes

En postman

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

Nombre | test1

email | [test1@gmail.com](mailto:test1@gmail.com)

password | 123456

role| SUPER\_ROLE

*son los datos para simular la primer peticion del cliente*

* Todas las lineas deben tener el ‘✓’ en los recuadros del lado izquierdo
* Quitar el ‘✓’ (check) en el recuadro de lado izquierdo de ‘role’
* Click a boton ‘SEND’
* En ventana ‘body’ abajo se despliegan las siguientes lineas

{

"ok": true,

"usuario": {

"role": "USER\_ROLE",

"estado": true,

"google": false,

"\_id": "5db48e37733ec8228c560669",

"nombre": "test1",

"email": "test1@gmail.com",

"password": "$2b$10$M.21bWGC6.6OXaFMf9RKkOhk.3ffmwq8U/WzQW1cd72gI5Z2nY3ou",

"\_\_v": 0

}

}

*La linea ‘ok’ indica que se hizo el proceso correctamente; La primera linea en azul muestra el ‘role’ por default, y la seguna linea azul muestra el ‘password’ ya encriptado; debio haberse grabado el regisro en MongoDB*

CHECAR QUE EN ‘MongoDB’ SE GRABO EL PASSWORD ENCRIPTADO

En Robo 3T

* Abrir el documento existente, y ver que el password esta grabado encriptado

CODIFICAR PARA QUE EN EL MSG DE POSTMAN, EN EL CAMPO PASSWORD SE MUESTRE ‘null’ EN VEZ DEL DATO CODIFICADO

En visual studio code

En ‘routes/usuario.js’

* Poner la linea marcada en azul

   usuario.save((err, usuarioDB) => {

*if* (err) {

*return* res.status(400).json({

                ok: false,

                err

            });

        }

        usuarioDB.password = null;

        res.json({

            ok: true,

            usuario: usuarioDB

        });

CHECAR QUE EN EL DATO DE PASSWORD SE MUESTRE EL TEXTO ‘null’

En postman

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

Nombre | test2

email | [test2@gmail.com](mailto:test2@gmail.com)

password | 123456

role| SUPER\_ROLE

*son los datos para simular la segunda peticion del cliente y que no se duplique el registro*

* Todas las lineas deben tener el ‘✓’ en los recuadros del lado izquierdo
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliegan las siguientes lineas

{

"ok": true,

"usuario": {

"role": "USER\_ROLE",

"estado": true,

"google": false,

"\_id": "5db48f98c5ac84059868025b",

"nombre": "test2",

"email": "test2@gmail.com",

"password": null,

"\_\_v": 0

}

}

*La linea ‘ok’ indica que se hizo el proceso correctamente; La linea en azul muestra el ‘password’ con el texto ‘null’ tal como se codifico; en MongoDB debio haberse grabado el regisro con el ‘password’ correctamente encriptado*

CHECAR QUE EN ‘MongoDB’ SE GRABO EL PASSWORD ENCRIPTADO

En Robo 3T

* Abrir el documento existente, y ver que el password esta grabado encriptado

CODIFICAR PARA QUE EN EL MSG DE POSTMAN NO SE MUESTRE NADA DEL CAMPO ‘PASSWORD’

En visual studio code

En ‘routes/usuario.js’

* ‘Comentar’ la linea marcada en azul

    usuario.save((err, usuarioDB) => {

*if* (err) {

*return* res.status(400).json({

                ok: false,

                err

            });

        }

*// usuarioDB.password = null;*

        res.json({

            ok: true,

            usuario: usuarioDB

        });

    });

En ‘models/usuario.js’

* Poner las lineas marcadas en azul

usuarioSchema.methods.toJSON = function() {

    let user = *this*;

    let userObject = user.toObject();

    delete userObject.password;

*return* userObject;

}

usuarioSchema.plugin(uniqueValidator, { message: '{PATH} debe ser unico' });

PROBAR QUE EN EL MSG DE POSTMAN NO SE MUESTRE NADA DEL CAMPO ‘PASSWORD’

En postman

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

Nombre | test3

email | [test3@gmail.com](mailto:test3@gmail.com)

password | 123456

role| SUPER\_ROLE

*son los datos para simular la tercera peticion del cliente y que no se duplique el registro*

* Todas las lineas deben tener el ‘✓’ en los recuadros del lado izquierdo
* Click a boton ‘SEND’

En ventana ‘body’ abajo se despliegan las siguientes lineas

{

"ok": true,

"usuario": {

"role": "USER\_ROLE",

"estado": true,

"google": false,

"\_id": "5db491f84d8ca636ccc0520e",

"nombre": "test3",

"email": "test3@gmail.com",

"\_\_v": 0

}

}

*Vease que ya no se muestra nada de informacion referente al ‘password’, ni el nombre del campo ni su contenido*

config.js

Models/usuario.js

Routes/usuario.js

Server.js

**CLASE 98 . PUT: Actualizar informacion de usuario**

Objetivo: Actualizar registros en la base de datos Mongo DB

LEVANTAR EL SERVIDOR PUERTO 3000

En consola sesion I

EN VISUAL STUDIO CODE

En ‘routes/usuario.js’

* Poner/Modificar las lineas marcadas en azul

app.put('/usuario/:id', function(req, res) {

    let id = req.params.id;

    let body = req.body;

*// recibir el contenido de ‘body’ proveniente de postman*

    Usuario.findByIdAndUpdate(id, body, { new: true }, (err, usuarioDB) => {

*// buscar en la tabla usuario el registro con el ‘id’ recibido y actualizarlo con el(los) datos que se reciban en ‘body’;*

*// la opcion ‘{new: true}’ es para que se despliegue(n) en la ventana de postman los nuevos datos recibidos y actualizados*

*if* (err) {

*return* res.status(400).json({

                ok: false,

                err

            });

        }

*// si el proceso no es exitoso, se recibe el status en ‘err’ y se codifica el* manejo del mismo

        res.json({

            ok: true,

            usuario: usuarioDB

*// si el proceso es exitoso, desplegar status y contenido de objeto ‘usuario’ el cual es recibido en ‘usuarioDB’*

        });

    })

});

EN POSTMAN

* Seleccionar el ambiente ‘Desarrollo’ en menu qry en parte superior derecha
* Abrir una pestaña nueva para simulaciones
* Seleccionar opcion ‘PUT’
* En cuadro para URL teclear:
* {{url}}/usuario/

EN MONGO DB

* En ventana ‘connections’

*Se muestra la conexión ‘Localhost’*

* Seleccionar conexión ‘localhost’
* Click a boton ‘connect’ en la parte inferior de la ventana

*Regresa a pantalla principal ‘Robo 3T’*

* Click a base de datos ‘café’
* Click a ‘collections’
* Doble click a tabla ‘usuarios’

*Se muestran los documentos ya grabados*

* Click a primer documento
* Se abren los campos del documento
* Click derecho a documento
* Opcion ‘Edit Document’
* Se abre ventana con lo datos del documento
* Seleccionar el contenido del campo ‘id’

*5db48e37733ec8228c560669*

* Pegarlo en Postman en cuadro para URL al final de la instrucción previamente tecleada

{{url}}/usuario/5db48e37733ec8228c560669

EN POSTMAN

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

nombre | Piojilla

* la linea debe tener el ‘✓’ en el recuadro del lado izquierdo
* Click a boton ‘SEND’

*En ventana ‘body’ abajo se despliega lo siguiente:*

{

"ok": true,

"usuario": {

"role": "USER\_ROLE",

"estado": true,

"google": true,

"\_id": "5db48e37733ec8228c560669",

"nombre": "Piojilla",

"email": "test1@gmail.com",

"\_\_v": 0

}

}

*Vease que status es ‘true’ y el nombre es el que fue enviado*

EN MONGO DB

* Doble click a tabla ‘usuarios’

*Se muestran los documentos ya grabados*

* Doble Click a primer documento
* Opcion ‘Edit Document’

*Se abre ventan ‘Edit’ donde se despliegan los campos del documento*

* Vease que ya se actualizo el campo nombre con el que fue enviado desde Postman

{

"\_id" : ObjectId("5db48e37733ec8228c560669"),

"role" : "USER\_ROLE",

"estado" : true,

"google" : true,

"nombre" : "Piojilla",

"email" : "test1@gmail.com",

"password" : "$2b$10$M.21bWGC6.6OXaFMf9RKkOhk.3ffmwq8U/WzQW1cd72gI5Z2nY3ou",

"\_\_v" : 0

}

config.js

Server.js

Routes/usuario.js

**CLASE 99 . Validaciones adicionales en el PUT**

Models/usuario.js

Objetivo: Poner validaciones al ejecutar el PUT para evitar que el usuario modifique campos que no se deben modificar desde el front end

LEVANTAR EL SERVIDOR PUERTO 3000

En consola sesion I

EN POSTMAN

SIMULAR ENVIO DE UN ROLE INVALIDO

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

role | laksdjflajksl

* la linea debe tener el ‘✓’ en el recuadro del lado izquierdo
* Click a boton ‘SEND’

*En la ventana inferior se despliega el mensaje de que el proceso fue exitoso y ese role se graba en el documento de MongoDB cuyo id fue enviado desde postman, aunque este sea un role no valido ya que no esta declarado en el objeto ‘RolesValidos’ del archivo ‘models/usuario.js’. Algunos piensan que este es un ‘bug’ de Mongoose*

EN VISUAL STUDIO CODE

En ‘routes/usuario.js’

* Poner/Modificar las lineas marcadas en azul

app.put('/usuario/:id', function(req, res) {

    let id = req.params.id;

    let body = \_.pick(req.body, ['nombre', 'email', 'img', 'role', 'estado']);

    Usuario.findByIdAndUpdate(id, body, { new: true, runValidators: true }, (err, usuarioDB) => {

*// El parametro ‘runvalidators: true’ indica que deben ejecutarse todas las validaciones codificadas en el file ‘models/usuario.js’*

*if* (err) {

*return* res.status(400).json({

                ok: false,

                err

            });

        }

        res.json({

            ok: true,

            usuario: usuarioDB

        });

    })

});

EN POSTMAN

SIMULAR ENVIO DE UN ROLE INVALIDO

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

role | laksdjflajksl

* la linea debe tener el ‘✓’ en el recuadro del lado izquierdo
* Click a boton ‘SEND’

*En la ventana inferior se despliega el mensaje de que el proceso NO fue exitoso; en el campo ‘role’ se despliega el mensaje de error declarado en el objeto ‘rolesvalidos’ del file ‘models/ususario.js’ con el texto ‘xxx no es un role valido’ y no se actualiza el documento en la base de datos MongoDB*

SIMULAR ENVIO DE UN ROLE VALIDO

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

role | USER\_ROLE l

* la linea debe tener el ‘✓’ en el recuadro del lado izquierdo
* Click a boton ‘SEND’

*En la ventana inferior se despliega el mensaje de que el proceso fue exitoso, no despliega mensaje de error ya que es un role valido que esta declarado en el objeto ‘RolesValidos’ del archivo ‘models/usuario.js’y se actualiza el documento en MongoDB*

SIMULAR ENVIO DE UN PASSWORD

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

password | 123 l

* la linea debe tener el ‘✓’ en el recuadro del lado izquierdo
* Click a boton ‘SEND’

*En la ventana inferior se despliega el mensaje de que el proceso fue exitoso, no despliega mensaje de error ya que no se ha codificado ninguna validacion para evitar que se modifique ese campo en el archivo ‘models/usuario.js’; se actualiza el documento en MongoDB con el password no encriptado*

INSTALAR UNDERCORE.JS

EN NAVEGADOR WEB

* Buscar y abrir el paquete ‘Underscore.js’
* Localizar la instrucción

*npm install underscore*

EN CONSOLA (en sesion II)

* Teclear la instrucción: npm install underscore –-save

*Ver que se ejecuta la instalacion del paquete*

DEFINIR CUALES CAMPOS SE PERMITIRA MODIFICAR

EN VISUAL STUDIO CODE

En ‘routes/usuario.js’

* Poner/Modificar las lineas marcadas en azul

const express = require('express');

const bcrypt = require('bcrypt');

const \_ = require('underscore');

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

app.put('/usuario/:id', function(req, res) {

    let id = req.params.id;

    let body = \_.pick(req.body, ['nombre', 'email', 'img', 'role', 'estado']);

*// la instruccion ‘pick’ indica que solo se podran modificar los campos incluidos en el arreglo; los demas campos no incluidos no deberan ser modificados aunque sean recibidos en el objeto ‘body’ desde postman*

    Usuario.findByIdAndUpdate(id, body, { new: true, runValidators: true }, (err, usuarioDB) => {

*if* (err) {

*return* res.status(400).json({

                ok: false,

                err

            });

        }

        res.json({

            ok: true,

            usuario: usuarioDB

        });

    })

});

EN POSTMAN

SIMULAR ENVIO DE CAMPOS ‘google’ y ‘password’

* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

google | false l

password | 123456 l

* la linea debe tener el ‘✓’ en el recuadro del lado izquierdo
* Click a boton ‘SEND’

*En la ventana inferior se despliega el mensaje de que el proceso fue exitoso, no despliega mensaje de error pero NO se actualizan estos campos en el documento en MongoDB ya qu no fueron incluidos en ‘models/usuario.js’ en el arreglo de la instrucción ‘-.pick’*

EN MONGO DB

* Borrar todos los documentos

*Dejar limpia la base de datos para pruebas posteriores*

EN POSTMAN

* Guardar el servicio request con el nombre ‘Usuario: PUT (Actualizar usuario)’ en el collection: ‘Café-Udemy’

*En el menu vertical a la izquierda en la opcion ‘collections’ debe aparecer esta colección y dentro de ella los dos servicios guardados; asi mismo cuando estan abiertos, se muestran en las pestañas de la parte superior*

config.js

Server.js

Routes/usuario.js

Models/usuario.js

**CLASE 100 . GET: Obtener todos los usuarios de forma paginada**

Objetivo: Obtener todos los usuarios de la base de datos MongoDB en forma paginada acorde a parametros ‘desde’ y ‘limite’ recibidos desde el ‘url’ de Postman

LEVANTAR EL SERVIDOR PUERTO 3000

En consola sesion I

EN POSTMAN

CREAR 16 REGISTROS DE USUARIOS EN MONGO DB

* Abrir la peticion ‘Usuario: POST (Crear usuario)’
* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

nombre | test1 l

email | test1@gmail.com l

password | 123456

* las lineas deben tener el ‘✓’ en el recuadro del lado izquierdo
* Click a boton ‘SEND’

*En la ventana inferior se despliega el mensaje de que el proceso fue exitoso, no despliega mensaje de errory se crea el documento del usuario ‘test1’ en MongoDB con el password encriptado y los demas campos según el valor default que se indico en el ‘Schema’ en ‘models/usuario.js’*

* Siguiendo estos mismo pasos, crear en total 16 usuarios con nombre: ‘test(n)’ y email: ‘Test(n)@gmail.com’ siendo ‘n’ el numero del usuario

*Ver que se hayan creado los 16 documentos en MongoDB*

EN VISUAL STUDIO CODE

PROGRAMAR EL SERVICIO A LA PETICION ‘GET’ PARA LEER LOS REGISTROS CREADOS EN MONGO DB

En ‘routes/usuario.js’

* Poner/actualizar las lineas en azul

app.get('/usuario', function(req, res) {

    let desde = req.query.desde || 0;

    desde = Number(desde);

    let limite = req.query.limite || 0;

    limite = Number(limite);

    Usuario.find({})

        .skip(desde)

        .limit(limite)

        .exec((err, usuarios) => {

*if* (err) {

*return* res.status(400).json({

                    ok: false,

                    err

                });

            };

            res.json({

                ok: true,

                usuarios

            });

        });

});

*Con el comando ‘let’ se indica que las variables ‘desde’ y ‘limite’ seran recibidas de Postman donde son tecleadas en la linea ‘url’ cuando se simula la peticion ‘GET’; a continuacion son convertidas a formato ‘numero’ ya que Postman las envia como ‘character’*

*La instrucción ‘Usuario.find()’ indica que se extraigan registros de MongoDB con las siguientes opciones:*

* *‘.skip’: saltar al registro ‘desde’ e iniciar la extraccion a partir del siguiente;*
* *‘.limit’: extraer este numero de registros*

*(el numero a la derecha en ambos comandos indica el default en caso de no recibir la variable ‘desde’ de Postman)*

* *‘. exec’: executar la extraccion de registros;* 
  + *en caso de no tener exito recibe el codigo de error en la variable ‘err’;*
  + *en caso de tener éxito recibe los datos de los registros extraidos en el objeto ‘usuarios’*
* *Se maneja el error con la instrucción ‘if’*

EN POSTMAN

SIMULAR LA PETICION ‘GET’

* Crear un nuevo servicio: click en ‘+’ en pestañas superiores
* Seleccionar peticion ‘GET’
* En recuadro URL teclear: ‘{(url)}/usuario’
* Seleccionar ambiente ‘Desarrollo’ en parte superior derecha
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan los datos de los 16 usuarios creados previamente en MongoDB*

* En recuadro URL teclear: ‘{(url)}/usuario?desde=5&limite=5’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan los datos de 5 usuarios iniciando con el usuario 6 o sea 6,7,8,9, y 10*

* En recuadro URL teclear: ‘{(url)}/usuario?desde=10&limite=10’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan los datos de solamente 6 usuarios iniciando con el usuario 11 o sea 11,12,13,14,15,16 ya que aunque se le dio un limite de 10 iniciando con el 11 solamente hay 6 registros a partir de este ultimo*

* En recuadro URL teclear: ‘{(url)}/usuario?desde=20&limite=10’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ no se despliega ningun usuario ya que aunque se indica que el limite es 10, se le indica que inicie en el 20 y solo hay 16 registros*

GUARDAR LA PETICION ‘GET’

* Cllick en boton ‘save’
* Pasa a ventana’SAVE REQUEST’
* Request name: Usuario: GET (Leer usuarios)
* Request description: Obtener usuarios acorde a parametros ‘desde’ y ‘limite’
* Select a collection or folder to save to: ‘Café–Udemy’

config.js

Server.js

Routes/usuario.js

Models/usuario.js

**CLASE 101 . GET: Retornar numero total de registros en una coleccion**

Objetivo: Contar y desplegar el conteo total de registros extraidos de una colección de datos

EN CONSOLA SESION I

levantar el servidor puerto 3000

EN VISUAL STUDIO CODE

En routes/usuario.js

* Poner / modificar las lineas en azul

app.get('/usuario', function(req, res) {

    let desde = req.query.desde || 0;

    desde = Number(desde);

    let limite = req.query.limite || 0;

    limite = Number(limite);

    Usuario.find({})

        .skip(desde)

        .limit(limite)

        .exec((err, usuarios) => {

*if* (err) {

*return* res.status(400).json({

                    ok: false,

                    err

                });

            };

            Usuario.count({}, (err, conteo) => {

                res.json({

                    ok: true,

                    usuarios,

                    cuantos: conteo

                });

            });

        });

});

EN POSTMAN

* Seleccionar la peticion ‘Usuarios: GET..’ en la ventana izquierda, opcion ‘collections’
* Seleccionar ambiente ‘Desarrollo’ en parte superior derecha
* En recuadro URL teclear: ‘{(url)}/usuario?desde=10&limite=10’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan los datos de 6 usuarios (del 11 al 16) y en la parte inferior se despliega el conteo total de usuarios en la colección en mongoDB*

{

    "ok": true,

    "usuarios": […

    ],

    "cuantos": 16

}

EN VISUAL STUDIO

En routes/usuarios.js

* Poner / modificar las lineas en azul

app.get('/usuario', function(req, res) {

    let desde = req.query.desde || 0;

    desde = Number(desde);

    let limite = req.query.limite || 0;

    limite = Number(limite);

    Usuario.find({google: true})

        .skip(desde)

        .limit(limite)

        .exec((err, usuarios) => {

*if* (err) {

*return* res.status(400).json({

                    ok: false,

                    err

                });

            };

            Usuario.count({google: true }, (err, conteo) => {

                res.json({

                    ok: true,

                    usuarios,

                    cuantos: conteo

                });

            });

        });

});

*Vease que en las dos instrucciones se puso el texto ‘google: true’ lo que indica que se extraigan solo los registros que tengan el status ‘true’ en el campo ‘google*

EN POSTMAN

* Seleccionar la peticion ‘Usuarios: GET..’ en la ventana izquierda, opcion ‘collections’
* Seleccionar ambiente ‘Desarrollo’ en parte superior derecha
* En recuadro URL teclear: ‘{(url)}/usuario?desde=10&limite=10’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan las siguientes lineas*

{

    "ok": true,

    "usuarios": [],

    "cuantos": 0

}

*Vease que no se extrajo ningun registro ya que ninguno cumple con la condicion. {google. True} y el resultado del conteo es ‘0’*

EN MONGO DB

* Editar el primer registro y modificar el campo: "google" a ‘true,’

{

"\_id" : ObjectId("5dd43fd01e86823ee0f8ce52"),

"role" : "USER\_ROLE",

"estado" : true,

"google" : true,

"nombre" : "test1",

"email" : "test1@gmail.com",

"password" : "$2b$10$LNulYmCxVXQAXJUBwZWTnehdcLvrWVXNhLc9/FalfH2BoDSR8NteS",

"\_\_v" : 0

}

EN POSTMAN

* Seleccionar la peticion ‘Usuarios: GET..’ en la ventana izquierda, opcion ‘collections’
* Seleccionar ambiente ‘Desarrollo’ en parte superior derecha
* En recuadro URL teclear: ‘{(url)}/usuario?desde=10&limite=10’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan las siguientes lineas*

{

    "ok": true,

    "usuarios": [],

    "cuantos": 1

}

*Vease que la cuenta es ‘1’ indicando que se conto un registro que cumple la condicion aunque no se extrajo dicho registro porque en la colección es el numero ‘1’ y la busqueda se inicio a partir del registro 11 porque el parametro ‘desde’ es igual a ‘10’*

* En recuadro URL teclear: ‘{(url)}/usuario?desde=0&limite=10’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan las siguientes lineas*

{

    "ok": true,

    "usuarios": [

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": true,

            "\_id": "5dd43fd01e86823ee0f8ce52",

            "nombre": "test1",

            "email": "test1@gmail.com",

            "\_\_v": 0

        }

    ],

    "cuantos": 1

}

*Vease que la cuenta es ‘1’ indicando que se conto un registro que cumple la condicion ‘google: true’ y se extrajo dicho registro porque ahora la busqueda se inicio a patir del registro ‘1’ porque ahora el argumento ‘desde’ es igual a ‘0’*

EN VISUAL STUDIO

En routes/usuarios.js

* Poner / modificar las lineas en azul

app.get('/usuario', function(req, res) {

    let desde = req.query.desde || 0;

    desde = Number(desde);

    let limite = req.query.limite || 0;

    limite = Number(limite);

    Usuario.find({})

        .skip(desde)

        .limit(limite)

        .exec((err, usuarios) => {

*if* (err) {

*return* res.status(400).json({

                    ok: false,

                    err

                });

            };

            Usuario.count({}, (err, conteo) => {

                res.json({

                    ok: true,

                    usuarios,

                    cuantos: conteo

                });

            });

        });

});

EN MONGO DB

* Editar el primer registro y modificar el campo: "google" a ‘false,’

{

"\_id" : ObjectId("5dd43fd01e86823ee0f8ce52"),

"role" : "USER\_ROLE",

"estado" : true,

"google" : false,

"nombre" : "test1",

"email" : "test1@gmail.com",

"password" : "$2b$10$LNulYmCxVXQAXJUBwZWTnehdcLvrWVXNhLc9/FalfH2BoDSR8NteS",

"\_\_v" : 0

}

EN POSTMAN

* Seleccionar la peticion ‘Usuarios: GET..’ en la ventana izquierda, opcion ‘collections’
* Seleccionar ambiente ‘Desarrollo’ en parte superior derecha
* En recuadro URL teclear: ‘{(url)}/usuario?desde=0&limite=10’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan las siguientes lineas*

{

    "ok": true,

    "usuarios": [...

    ],

    "cuantos": 16

}

*Vease que la cuenta es ‘16’ que son el total de registros en la colección dado que ya no hay ningun filtro en el codigo, y se extrajeron 10 registros iniciando con el 1, ya que el argumento ‘desde’ es igual a 0 y el argumento ‘limite’ es igual a 10*

config.js

Server.js

Routes/usuario.js

Models/usuario.js

**CLASE 102 . GET: Filtrando los campos de los resultados de un get**

Objetivo: Filtrar campos de un resultado ‘get’

EN CONSOLA SESION I

levantar el servidor puerto 3000

EN VISUAL STUDIO CODE

En routes/usuario.js

* Poner / modificar las lineas en azul

    Usuario.find({}, 'nombre email role estado google img')

        .skip(desde)

        .limit(limite)

        .exec((err, usuarios) => {

*if* (err) {

*return* res.status(400).json({

                    ok: false,

                    err

                });

            };

            Usuario.count({}, (err, conteo) => {

                res.json({

                    ok: true,

                    usuarios,

                    cuantos: conteo

                });

            });

        });

EN POSTMAN

* Seleccionar la peticion ‘Usuarios: GET..’ en la ventana izquierda, opcion ‘collections’
* Seleccionar ambiente ‘Desarrollo’ en parte superior derecha
* En recuadro URL teclear: ‘{(url)}/usuario?desde=10&limite=1’
* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se despliegan los datos de 1 usuario a partir del 11; solo se despliegan los campos indicados en la instrucción ‘find’; en la parte inferior se despliega el conteo total de usuarios en la colección en mongoDB*

{

    "ok": true,

    "usuarios": [

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd43fd01e86823ee0f8ce52",

            "nombre": "test1",

            "email": "test1@gmail.com"

        }

    ],

    "cuantos": 16

}

config.js

Server.js

Routes/usuario.js

Models/usuario.js

**CLASE 103 . DELETE: Borrando un usuario de la base de datos**

Objetivo: Borrar un usuario de la BD

EN CONSOLA SESION I

levantar el servidor puerto 3000

EN VISUAL STUDIO CODE

En routes/usuario.js

* Poner / modificar las lineas en azul

app.delete('/usuario/:id', function(req, res) {

    let id = req.params.id;

    Usuario.findByIdAndRemove(id, (err, usuarioBorrado) => {

*if* (err) {

*return* res.status(400).json({

                ok: false,

                err

            });

        };

*if* (!usuarioBorrado) {

*return* res.status(400).json({

                ok: false,

                err: {

                    message: 'Usuario no encontrado'

                }

            });

        }

*res.json({*

*ok: true,*

*usuario: usuarioBorrado*

*});*

*Se programa la respuesta a una peticion ‘DELETE’;*

*en la funcion ‘app.delete’ en el URL se recibe el ‘id’ del usuario que se desea borrar y se envia un callback para el manejo de la peticion; la instrucción ‘findByAndRemove’ envia el ‘id’ a borrar y un callback para el manejo de los errores en caso de presentarse con un ‘if’ para el caso de error al tratar de borrar y otro ‘if’ para el caso de que el registro no exista en la BD; si no se presenta ningun error, se despliega el msg de éxito ‘ok: true’ y los datos del usuario borrado; el registro debio ser borrado en la BD*

EN ROBO 3T

SELECCIONAR EL ‘ID’ DE UN REGISTRO PARA BORRARLO

* Editar el documento del usuario 16
* Seleccionar el contenido del campo ID que esta encriptado

*Este se va a pegar en la instrucción URL en postman*

EN POSTMAN

SIMULAR UNA PETICION ‘DELETE’ A UN REGISTRO

* Click a boton ‘+’ para crear una nueva peticion
* Seleccionar la peticion ‘DELETE’
* Seleccionar ambiente ‘Desarrollo’ en parte superior derecha
* En recuadro URL teclear: ‘{(url)}/usuario/5de593758119d33904687f50

*El ‘id’ es el que se seleccióno del registro 16 en Mongo DB*

* Click a boton ‘SEND’

*En la ventana inferior, opcion ‘Body’ se indica que el proceso fue exitoso y se despliegan los datos de del usuario 16; en mongo DB se borra el registro del usuario 16*

{

    "ok": true,

    "usuario": {

        "role": "USER\_ROLE",

        "estado": true,

        "google": false,

        "\_id": "5de595535693272ca061c9f2",

        "nombre": "test16",

        "email": "test16@gmail.com",

        "\_\_v": 0

    }

}

INTENTAR BORRAR EL REGISTRO QUE YA NO EXISTE POR QUE YA FUE BORRADO

* Click a boton ‘SEND’ (sin modificar nada mas)

*Se intenta borrar el mismo registro; En la ventana inferior, opcion ‘Body’ se indica que el proceso NO fue exitoso ‘ok: false’ y se despliega el mensaje ‘Usuario no encontrado’*

{

    "ok": false,

    "err": {

        "message": "Usuario no encontrado"

    }

}

GUARDAR LA PETICION ‘DELETE’

* Cllick en boton ‘save’
* Pasa a ventana’SAVE REQUEST’
* Request name: Usuario: DELETE (Borrar usuarios)
* Request description: Borrar usuarios en Mongo DB
* Select a collection or folder to save to: ‘Café–Udemy’
* Click a boton ‘Save’

config.js

Server.js

Routes/usuario.js

Models/usuario.js

**CLASE 104 . DELETE: Marcar una eliminacion en el mismo registro**

Objetivo: Hacer un borrado virtual de un registro lo cual significa que el registro no se elimine de la BD sino que solo se le cambie el valor al campo ‘estado’ de ‘true’ a ‘false’; a continuacion se debe hacer la programacion para la aplicación valide si el registro esta borrado virtualmente

EN CONSOLA SESION I

* Teclear: nodemon server/server

*levantar el servidor puerto 3000:*

EN VISUAL STUDIO

En routes/usuario.js

* Poner / modificar las lineas en azul

app.delete('/usuario/:id', function(req, res) {

    let id = req.params.id;

*// Usuario.findByIdAndRemove(id, (err, usuarioBorrado) => {*

    let cambiaEstado = {

        estado: false

    };

    Usuario.findByIdAndUpdate(id, cambiaEstado, { new: true }, (err, usuarioBorrado) => {

*if* (err) {

*return* res.status(400).json({

                ok: false,

                err

            });

        };

*if* (!usuarioBorrado) {

*return* res.status(400).json({

                ok: false,

                err: {

                    message: 'Usuario no encontrado'

                }

            });

        }

        res.json({

            ok: true,

            usuario: usuarioBorrado

        });

    });

EN ROBO 3T

* En BD ‘café/collections/usuarios’, editar usuario 15
* Seleccionar el ‘id’ encriptado para copiarlo en Postman

EN POSTMAN

PROBAR EL BORRADO VIRTUAL DEL REGISTRO 15

* Abrir la peticion ‘Usuario: DELETE’
* En cuadro URL, teclear: {{url}}/usuario/5dd440a71e86823ee0f8ce60

*El ‘id’ es el del regstro 15 que se selecciono en ROBO 3T*

* Click a SEND

*En ventana inferior, opcion ‘body’ se despliega:*

{

    "ok": true,

    "usuario": {

        "role": "USER\_ROLE",

        "estado": false,

        "google": false,

        "\_id": "5dd440a71e86823ee0f8ce60",

        "nombre": "test15",

        "email": "test15@gmail.com",

        "\_\_v": 0

    }

}

*Vease el campo ‘ok: true’ que indica que el proceso fue exitoso y el que el campo ‘estado: false’*

* Ver en ROBO 3T que se actualizo el campo ‘estado’ a ‘false’

FILTRAR REGISTROS BORRADOS VIRTUALMENTE

EN VISUAL STUDIO

En routes/usuario.js

* Poner / modificar las lineas en azul

app.get('/usuario', function(req, res) {

    let desde = req.query.desde || 0;

    desde = Number(desde);

    let limite = req.query.limite || 0;

    limite = Number(limite);

    Usuario.find({ estado: true }, 'nombre email role estado google img')

        .skip(desde)

        .limit(limite)

        .exec((err, usuarios) => {

*if* (err) {

*return* res.status(400).json({

                    ok: false,

                    err

                });

            };

            Usuario.count({ estado: true }, (err, conteo) => {

                res.json({

                    ok: true,

                    usuarios,

                    cuantos: conteo

                });

            });

        });

});

*Estas dos instrucciones indican que solo se obtengan y se cuenten registros activos (status: true)*

EN POSTMAN

PROBAR QUE NO OBTENGA REGISTROS BORRADOS VIRTUALMENTE

* Abrir peticion ‘Usuario: GET’
* En cuadro URL, teclear: {{url}}/usuario?limite=10&desde=14

*La peticion es obtener 10 registros a partir del 15 (14+1)*

* Click a SEND

*En ventana inferior, opcion ‘body’ se despliega:*

{

    "ok": true,

    "usuarios": [],

    "cuantos": 14

}

*Vease que no se despliega ningun registro ya que el registro 15 que es el unico que existe despues del 14 esta borrado virtualmente con el campo ‘estado’ con valor ‘false’; asi mismo la cuenta de registros activos es de solo 14*

* En cuadro URL, teclear: {{url}}/usuario?limite=10&desde=10

*La peticion es obtener 10 registros a partir del 15 (14+1)*

* Click a SEND

*En ventana inferior, opcion ‘body’ se despliega:*

{

    "ok": true,

    "usuarios": [

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd440671e86823ee0f8ce5c",

            "nombre": "test11",

            "email": "test11@gmail.com"

        },

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd4407e1e86823ee0f8ce5d",

            "nombre": "test12",

            "email": "test12@gmail.com"

        },

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd440881e86823ee0f8ce5e",

            "nombre": "test13",

            "email": "test13@gmail.com"

        },

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd4409a1e86823ee0f8ce5f",

            "nombre": "test14",

            "email": "test14@gmail.com"

        }

    ],

    "cuantos": 14

}

*Vease que se despliegan 4 registros a partir del 11, (del 11 al 14); asi mismo la cuenta de registros activos es de 14 que son los registros activos (estado: true)*

PROBAR PETICION ‘DELETE’ A UN REGISTRO BORRADO VIRTUALMENTE

* Abrir la peticion ‘Usuario: DELETE’
* En cuadro URL, teclear: {{url}}/usuario/5dd440a71e86823ee0f8ce60

*El ‘id’ es el del regstro 15 en ROBO 3T*

* Click a SEND

*En ventana inferior, opcion ‘body’ se despliega:*

{

    "ok": true,

    "usuario": {

        "role": "USER\_ROLE",

        "estado": false,

        "google": false,

        "\_id": "5dd440a71e86823ee0f8ce60",

        "nombre": "test15",

        "email": "test15@gmail.com",

        "\_\_v": 0

    }

}

*Vease que se muestra el registro 15 con el campo ‘ok: true’ que indica que el proceso fue exitoso y el que el campo ‘estado: false’; de hecho le re-actualizo el valor a ‘false’ ya que en esta peticion no se ha puesto ningun filtro para que detecte que el registro ya fue borrado virtualmente y mande el mensaje correspondiente*

FUERON ACTUALIZADOS/CREADOS LOS SIGUIENTES PROGRAMAS:

Routes/usuario.js

**CLASE 105/106 . MongoDBAtlas – Cloud Database – Configurar base de datos en la nube**

Objetivo: crear y configurar una base de datos en la nube con MongoDBAtlas, y crear un registro con esta herramienta

Nota:

La clase 105 aplica para quienes ya son usuarios de la aplicación ‘Mlab’, La clase 106 aplica para lo que no la tienen y por tanto utilizaran directamente ‘MongoDBAtlas’

Objetivo: con MongoDBAtlas, crear y configurar una base de datos en la nube y crear un registro desde esta aplicacion

En navegador web

* Buscar y abrir ‘www.mongodb.com’

*Se sugiere agregar esta pagina a favoritos*

En mongoDB Atlas

CREAR CUENTA DE USUARIO

* Click a boton ‘try free’
* Crear una nueva cuenta gratis

*User: egonzalez@idrall.com*

*Pwd: Aceg1357*

*Nota: se creo una nueva cuenta con el correo ‘egonzalez@idrall.com’ para trabajar en la nube (remoto); recuerdese que se habia creado la de ‘gmail’ anteriormente para trabajar localmente*

CREAR CLUSTER

* Click a boton ‘Create a new cluster’
* Seleccionar icono ‘AWS’, region ‘Oregon’, Cluster tier ‘M0 Sandbox’
* Click a boton ‘Create cluster’

*Se despliega caja ‘sandbox’ con la linea de contorno en movimiento, tarda varios minutos*

DAR ACCESO A BD EN LA NUBE

* En menu vertical a la izquierda, en seccion ‘SECURITY’, click a opcion ‘Network access’
* Click a opcion ‘IP whitelist’
* Click a boton ‘Add IP address’

*Se despliega ventana ‘Add Whitelist Entry’*

* Click a boton ‘allow access from anywhere’

*En cuadro ‘whitelist entry’ se despliega ‘0.0.0.0/0’ dejarlo asi*

* En cuadro ‘comment’ teclear ‘’cualquier lado’
* Click a boton ‘confirm’

*Se despliega la pantalla ‘Security’ y en la ventana abajo’IP address’ se despliega esta direccion adicionada*

CREAR USUARIO Y DAR ACCESO A LA BD EN LA NUBE

* En menu vertical a la izquierda, en seccion ‘Security’, click a opcion ‘Database access’
* Click a opcion ‘MongoDB Users’
* Click a boton ‘Add new user’
* En seccion ‘User Privileges’, click a icono ‘Atlas admin’
* En cuadro ‘Enter username’ teclear: strider
* Click a boton ‘autogenerate secure password’

*Se muestra un password creado por la aplicación,*

*w0RNQy3geNG6xKq2*

*Nota: se sugiere conservar el user y el pwd para futura utilizacion*

*Nota: se puede editar y modificar el pwd*

* Click a boton ‘Add User’

*Se despliega la pantalla ‘MongoDB users’ y en la ventana abajo este usuario creado*

CREAR BASE DE DATOS ‘café’ Y COLECCIÓN ‘clientes’

* En seccion ‘ATLAS’, click a opcion ‘Clusters’
* Click en boton ‘collections’

*Se despliega pantalla ‘collections’*

* Click a boton ‘create database’

*Se despliega ventana del mismo nombre*

* Teclear database name ‘café’
* Teclear collection name ‘clientes’
* Click a boton ‘create’

*Se despliega pantalla ‘collections’ y se muestra esta base de datos y colleccion creadas*

INSERTAR UN REGISTRO (DOCUMENTO)

* Click a boton ‘insert document’

*Se despliega la ventana del mismo nombre*

* Se muestra por default el campo 1 ‘\_id’ con el contenido ya creado
* En campo 2, teclear ‘nombre: “EG lobo”
* Click a boton ‘Insert’

*Se despliega la pantalla ‘collections’ y abajo en la base de datos café.clientes este registro insertado*

*{"\_id":{"$oid":"5df7d8fc1c9d440000af3aa0"},*

*"nombre":"EG Lobo Mongo Atlas"}*

**CLASE 107 . MongoDB Compass – UI para el manejo de MongoDB**

Objetivo: Crear y configurar una base de datos en la nube con MongoDBAtlas, y crear un registro con esta herramienta

En MongoDBAtlas

* Click a opcion ‘Clusters’ en menu vertical a la izquierda
* Click a boton ‘Connect’
* Click a opcion ‘Connect with MongoDB Compas’

*Se abre venta ‘Connect to Cluster0’*

DESCARGAR COMPASS

* Seleccionar sistema opertivo
* Descargar e instalar ‘Compass’
* Checar que se instalo correctamente
* Copiar y conservar el ‘connection string’

mongodb+srv://strider:w0RNQy3geNG6xKq2@cluster0-ehvg4.mongodb.net/cafe *Recuerdese que para conectarse se dio el permiso en ‘MongoDBAltas’ en la opcion ‘Security’ – ‘Whitelist’*

En MongoDB Compass

CONECTARSE A LA BASE DE DATOS

* Click a opcion ‘New Connection’
* Click a opcion ‘Fill in connection fields individually’
* Teclear en recuadros:
  + Hostname: cluster0-ehvg4.mongodb.net

*Vease que esto es una parte del ‘connection string’ que se indico conservar*

* + Port: 27017

*Es deplegado por default*

* + SRV Record: habilitar

*Debe estar habilitado si en el ‘Connection string’ esta el ‘+srv’*

* + Authentication: User name/Password
  + Username: Strider
  + Password: w0RNQy3geNG6xKq2

*Este es el password autogenerado por ‘MongoDBAtlas’ cuando se creo el usuario y se dio acceso a la nube*

* + Authentication Database: Admin

*Es deplegado por default*

* Click a boton ‘CONNECT’

*Se hace la conexión, se despliega la pantalla ‘Databases’ donde se muestran las bases de datos existentes en este host*

* Click a BD ‘café’

*Se despliegan las colecciones creadas en esta BD, en este caso ‘clientes’*

* Click a colección ‘clientes’

*Se despliega el unico registro creado en esta colección*

*{"\_id":{"$oid":"5df7d8fc1c9d440000af3aa0"},*

*"nombre":"EG Lobo Mongo Atlas"}*

HACER CAMBIO AL REGISTRO

* click a boton ‘Edit’ en el registro,
* Teclear ‘EG Lobo Mongo Atlas – Compass’
* Click a boton ‘update’

En MongoDBAtlas

CHECAR CAMBIO AL REGISTRO

* Click a boton ‘Collections’

*Se muestra la BD ‘café’, la colección ‘Clientes’ y el unico registro creado;*

*Checar que se grabo el cambio anterior*

**CLASE 108 . Actualizar todos los paquetes de mi aplicación NODE**

Objetivo: Idem

En visual studio

En ‘package.json’

* Ver las lineas de los paquetes utilizados

    "dependencies": {

        "bcrypt": "^3.0.6",

        "bcryptjs": "^2.4.3",

        "body-parser": "^1.19.0",

        "express": "^4.17.1",

        "mongoose": "^5.8.3",

        "mongoose-unique-validator": "^2.0.3",

        "node-pre-gyp": "^0.14.0",

        "underscore": "^1.9.1"

    }

En consola

* Teclear: cd C:\Users\EG\_Lobo\Documents\Mis documentos\aa Major 1 EGB\EGB /EDU\TAKEN\NODE\07-restserver

*Para Ir al dir del proyecto*

* Teclear: npm update

*Se despliegan mensajes indicando las actualizaciones y estas ultimas se despliegan en el ‘package.json’ en tiempo real, pero al final se muestran errores indicando que no se pudo actualizar el ‘bcrypt’; esto es por cambios en el paquete*

* Teclear: npm unistall bcrypt

*Se despliegan mensajes indicando la desinstalacion*

* Teclear: npm install [bcrypt@3.0.6](mailto:bcrypt@3.0.6) --save

*Es importante que se use la version indicada, ya que de otra manera desplegara errores y no se instalara; esto ocurre porque la version de ‘bcrypt’ debe ser compatible con la version de ‘node’; ver tabla en las preguntas y respuestas del curso*

* Teclear: nodemon server/server.js

*Para levantar el servidor; checar que no muestre ningun error*

*Si se muestran msgs de ‘deprecationWarning’ corregirlos en Visual Studio como lo indique el sistema para cada uno de ellos (ver indicaciones en bloque Visual Studio siguiente en este texto)*

En Visual studio

En ‘server.js’

* Teclear / modificar las lineas en azul

mongoose.connect(process.env.URLDB, { useNewUrlParser: true, useUnifiedTopology: true, useCreateIndex: true },

    (err, res) => {

*if* (err) *throw* err;

        console.log('Base de datos ONLINE');

    });

En consola

* Checar que ya no mande ningun msg de ‘deprecationWarning’

FUERON ACTUALIZADOS/CREADOS LOS SIGUIENTES PROGRAMAS:

package.json

Server.js

**CLASE 109 . Conectar mLab o Mongo Atlas con nuestra aplicación de Node**

Objetivo: Conectar mLab o Mongo Atlas con nuestra aplicación de Node en dos entornos: el local el remoto

CODIFICAR CRITERIO PARA DETERMINAR SI ESTA CORRIENDO ALGUN AMBIENTE REMOTO

En visual studio

En ‘config.js’

* Poner / actualizar las lineas en azul

*//*

*// Puerto*

*//*

process.env.PORT = process.env.PORT || 3000;

*//*

*// Entorno*

*//*

process.env.NODE\_ENV = process.env.NODE\_ENV || 'dev';

*con esta instrucción se detecta si esta corriendo un ambiente remoto; la variable ‘NODE\_ENV’ la establece ‘Heroku’ o cualquier otra aplicación cuando esta corriendo un proceso remoto; si dicha variable no existe significa que no esta corriendo un ambiente remoto y se asigna el valor ‘dev’ para trabajar en ambiente local*

*//*

*// Base de datos*

*//*

let urlDB;

*if* (process.env.NODE\_ENV === 'dev') {

*esta funcion ‘if’ checa si esta corriendo un ambiente remoto o no y según sea el caso asigna el URL correspondiente a la variable ‘urlDB’*

    urlDB = 'mongodb://localhost:27017/cafe';

*este URL es el que se utiliza para conectarse a la base de datos ‘café’ en el ambiente local; vease que es el que se codifico en ‘server.js’*

} *else* {

    urlDB = 'mongodb+srv://strider:w0RNQy3geNG6xKq2@cluster0-ehvg4.mongodb.net/cafe';

*este URL es el que se utiliza para conetarse a la base de datos ‘café’ en el ambiente remoto en la nube; vease qu es el que se obtuvo cuando se creo esta base de datos en la nube con Mongo Atlas (el cual se sugirio conservar); vease que tiene el usuario ‘strider’ y el password que genero la aplicación, y al final tiene el nombre de la base de datos ‘café’*

};

process.env.URLDB = urlDB;

*se crea la variable ‘URLDB’ a la cual se le asigna el valor que contenga la variable ‘urlDB’ la cual tiene el URL correspondiente según se este trabajando en ambiente local o ambiente remoto, lo cual se determino en la funcion ‘if’ anterior; (vease que son dos variables diferentes ya que una son solo mayusculas)*

CODIFICAR QUE EL URL SEA VARIABLE

En ‘server.js’

* Poner / actualizar las lineas en azul

mongoose.connect(process.env.URLDB, { useNewUrlParser: true, useUnifiedTopology: true, useCreateIndex: true },

    (err, res) => {

*if* (err) *throw* err;

        console.log('Base de datos ONLINE');

    });

*Vease que se cambIo el URL por la variable ‘process.env.URLDB’ creada en ‘config.js’; de esta manera se hara conexión ya sea al ambiente local o al ambiente remoto, según el valor que se le haya asignado a dicha variable en ‘config.js’*

LEVANTAR EL SERVIDOR

En consola

* Teclear: cd C:\Users\EG\_Lobo\Documents\Mis documentos\aa Major 1 EGB\EGB /EDU\TAKEN\NODE\07-restserver

*Para Ir al dir del proyecto*

* Teclear: nodemon server/server

*Para levantar el servidor*

HACER PETICIONES A BASE DE DATOS EN AMBIENTE LOCAL

En postman

* Seleccionar ambiente ‘desarrollo’
* Seleccionar peticion ‘GET’ (tal como fue grabada la ultima vez)
* Poner en recuadro para URL: {{url}}/usuario?limite=10&desde=10
* Click a boton ‘SEND’

*En ventana inferior ‘body’ se muestran los registros del 11 al 14 con los campos que se codificaron en la peticion GET en ‘/routes/usuario.js’*

{

    "ok": true,

    "usuarios": [

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd440671e86823ee0f8ce5c",

            "nombre": "test11",

            "email": "test11@gmail.com"

        },

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd4407e1e86823ee0f8ce5d",

            "nombre": "test12",

            "email": "test12@gmail.com"

        },

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd440881e86823ee0f8ce5e",

            "nombre": "test13",

            "email": "test13@gmail.com"

        },

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5dd4409a1e86823ee0f8ce5f",

            "nombre": "test14",

            "email": "test14@gmail.com"

        }

    ],

    "cuantos": 14

}

*Vease que la peticion se esta haciendo a la base de datos ‘café’ en el ambinte local*

CODIFICAR SIMULACION DE AMBIENTE REMOTO CORRIENDO

En visual studio

En ‘config.js’

* Poner / actualizar las lineas en azul

*//*

*// Base de datos*

*//*

let urlDB;

*// if (process.env.NODE\_ENV === 'dev') {*

*//     urlDB = 'mongodb://localhost:27017/cafe';*

*// } else {*

    urlDB = 'mongodb+srv://strider:w0RNQy3geNG6xKq2@cluster0-ehvg4.mongodb.net/cafe';

*// };*

*Vease que se estan comentando estas lineas para simular que esta corriendo un ambiente remoto asignando directamente a la variable ‘urlDB’ el URL de la base de datos remota ‘café’*

process.env.URLDB = urlDB;

*se comentan las lineas correspondientes en la funcion ‘if’ para que no cheque si esta corriendo algun ambiente remoto*

HACER PETICIONES A BASE DE DATOS EN AMBIENTE REMOTO

En postman

* Seleccionar ambiente ‘desarrollo’ (aunque el URL fue ‘forzado’ al ambiente remoto)
* Seleccionar peticion ‘GET’ (tal como fue grabada la ultima vez)
* {{url}}/usuario?limite=10&desde=10
* Click a boton ‘SEND’
* En ventana inferior ‘body’ no aparece ningun registro y la cuenta es ‘0’

{

    "ok": true,

    "usuarios": [],

    "cuantos": 0

}

*Esto es correcto ya que se esta haciendo la peticion a la base de datos ‘café’ a la colección ‘usuarios’ en el ambiente remoto la cual hasta este momento no tiene ningun registro*

CREAR UN REGISTRO EN BASE DE DATOS EN AMBIENTE REMOTO

* Seleccionar peticion ‘POST’
* En menu de barra horizontal, segunda linea seleccionar opcion: ‘Body ⚫’
* En menu de barra horizontal, tercera linea seleccionar opcion: ‘x-www-form-urlencoded’
* En recuadros debajo de columnas: ‘key’ y ‘value’ teclear/modificar:

nombre | test1 l

email | test1@gmail.com l

password | 123456

* poner a todas las lineas el ‘✓’ en el recuadro del lado izquierdo
* Click a boton ‘SEND’

*En la ventana inferior se despliega el mensaje de que el proceso fue exitoso, y los campos del registro creado con el campo “estado”: true*

{

    "ok": true,

    "usuarios": [

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5e0e82ef52623e3af4a11345",

            "nombre": "test1",

            "email": "test1@gmail.com"

        }

    ],

    "cuantos": 1

}

CHECAR EL REGISTRO CREADO DIRECTAMENTE EN LA BASE DE DATOS EN AMBIENTE REMOTO

En MongoDB Atlas

En base de datos ‘café’, en colección ‘usuarios’

* Ver que se ha creado el registro ‘test1’ con los campos enviados, los campos default, y el pwd encryptado

EXTRAER EL REGISTRO CREADO EN LA BASE DE DATOS EN AMBIENTE REMOTO

En postman

* Seleccionar la peticion ‘GET’
* Poner en el recuadro de URL: {{url}}/usuario?limite=10&desde=0
* Vease los parametros ‘limite’ y ‘desde’ … ya que solo hay un registro
* Click a boton ‘SEND’
* En la ventana inferior se muestra proceso existoso “ok”: true, el registro creado ‘test1’ con sus campos. y la cuenta “cuantos: 1”

{

    "ok": true,

    "usuarios": [

        {

            "role": "USER\_ROLE",

            "estado": true,

            "google": false,

            "\_id": "5e0e82ef52623e3af4a11345",

            "nombre": "test1",

            "email": "test1@gmail.com"

        }

    ],

    "cuantos": 1

}

*el registro esta siendo extraido exitosamente de la base de datos remota ‘café’ en la colección ‘usuarios’*

QUITAR SIMULACION DE AMBIENTE REMOTO CORRIENDO

En visual studio

En ‘config.js’

* Poner / actualizar las lineas en azul

let urlDB;

*if* (process.env.NODE\_ENV === 'dev') {

    urlDB = 'mongodb://localhost:27017/cafe';

} *else* {

    urlDB = 'mongodb+srv://strider:w0RNQy3geNG6xKq2@cluster0-ehvg4.mongodb.net/cafe';

};

process.env.URLDB = urlDB;

*se descomentan las lineas para quitar simulacion de ambiente remoto corriendo y dejar la condicion como debe estar para determinar si se esta corriendo un ambiente local o un ambiente remoto*

VERIFICAR QUE SE ESTE ACCESANDO BASE DE DATOS EN AMBIENTE LOCAL

En postman

* Seleccionar peticion ‘GET’
* Poner en cuadro de URL: {{url}}/usuario?limite=10&desde=0
* Click a boton ‘SEND’

*En ventana inferior ‘body’ se muestran los registros del 1 al 10 con los campos que se codificaron en la peticion GET en ‘/routes/usuario.js’ y la cuenta es 14*

{

    "ok": true,

    "usuarios": […

    ],

    "cuantos": 14

}

*Vease que al descomentar las lineas en ‘config.js’ se estan extrayendo los registros de la base de datos local ‘café’ ya que no esta corriendo ningun ambiente remoto*

FUERON ACTUALIZADOS/CREADOS LOS SIGUIENTES PROGRAMAS:

Config.js

Server.js

**CLASE 111 . Desplegando nuestro RESTServer en Heroku - Usuario**

Objetivo: Respaldar aplicación en GIT y subirla a la nube en HEROKU

**En consola**

RESPALDAR APLICACIÓN EN GIT

* Teclear: cd C:\Users\EG\_Lobo\Documents\Mis documentos\aa Major 1 EGB\EGB EDU\TAKEN\NODE\07-restserver
* Teclear: git status

*Se despliegan todos los archivos que fueron creados / modificados desde la ultima vez que se subio la aplicación a la nube*

* Teclear: ‘git add .’ (ojo con el punto al final)

*Se adicionan los archivos creados / modificados pero no se despliega nada*

* Teclear: git status

*Se despliegan en verde los archivos creados / modificados*

* Teclear. Git commit -m “RestServer – usuario”

*Se despliegan mensajes de la ejecucion de este proceso*

SUBIR LA APLICACION A HEROKU

* Teclear: git push heroku master

*Se despliegan mensajes del proceso y en la penultima linea se muestra el URL generado por Heroku para accesar la aplicación subida a la nube*

<https://fast-scrubland-09379.herokuapp.com/>

* Teclear: heroku open

*Se abre el navegador web en este URL y se despliega el mensaje ‘Cannot GET’; esto ocurre porque porque el file ‘server.js’ esta programado para recibir ‘/usuario’*

app.get('/usuario', function(req, res) {

**En postman**

PROBAR PETICIONES A LA APLICACIÓN EN LA NUBE

* Seleccionar ambiente ‘Produccion’
* *vease que el ambiente ‘Produccion’ esta configurado para hacer peticiones a la aplicación en la nube a traves del URL generado por Heroku*

OBTENER CONTENIDO DE BASE DE DATOS EN LA NUBE

* Seleccionar peticion: GET

*se despliega un solo registro que es el que se creo en la base de datos remota*

AGREGAR UN REGISTRO A LA DB EN LA NUBE

* seleccionar peticion POST
* en seccion ‘body’, ‘x-www-form-urlencoded’

nombre | test 2

email | [test2@gmail.com](mailto:test2@gmail.com)

* click a SEND

*en la ventana inferior en seccion ‘body’ se despliegan los campos (codificados en ‘/routes/usuario.js’) del nuevo registro creado en la base de datos remota*

OBTENER CONTENIDO DE BASE DE DATOS EN LA NUBE

* seleccionar peticion GET

*se despliegan los dos registros existentes*

**CLASE 112 . Subir los cambios a GitHub – Usuario y RESTServer**

**En consola**

RESPALDAR APLICACIÓN EN GIT

* Teclear: cd C:\Users\EG\_Lobo\Documents\Mis documentos\aa Major 1 EGB\EGB EDU\TAKEN\NODE\07-restserver
* Teclear: git status

*Se despliegan todos los archivos que fueron creados / modificados desde la ultima vez que se subio la aplicación a la nube*

*Dado que en la seccion anterior se respaldo, aquí no debe aparecer ningun archivo a respaldar*

* En caso de que hubiera algun archivo por respaldar
  + Teclear: ‘git add .’ (ojo con el punto al final)

*Se adicionan los archivos creados / modificados pero no se despliega nada*

* + Teclear: git status

*Se despliegan en verde los archivos creados / modificados*

* + Teclear: Git commit -m “RestServer – usuario”

*Se despliegan mensajes de la ejecucion de este proceso*

* Teclear: git push origin master

*Se despliegan mensajes indicando el numero de archivos que se estan respaldando*

**En navegador web**

VER URL QUE ACCESA A LA BD REMOTA EL CUAL PUEDE PROTEGERSE

En pagina github <https://github.com>

* Click a repositorio ‘eglobo911/07-restserver’ en menu vertical izquierdo

*Entra a la pagina de este repositorio; se muestran los archivos que contiene*

*Vease que se muestra el folder ‘server’*

* Click a folder ‘server’

*Se muestra el contenido en este folder*

*Vease que se muestra el folder ‘config’*

* Click a folder ‘config’

*Se despliega el contenido en este folder*

*Vease que se muestra el archivo ‘config.js’*

* Click a archivo ‘config.js’

*Se despliega el codigo de este archivo*

*Vease esta linea*

urlDB = 'mongodb+srv://strider:w0RNQy3geNG6xKq2@cluster0-ehvg4.mongodb.net/cafe';

*posteriormente se vera como puede protegerse este URL para que no sea visible publicamente*

* Click a eglobo911/07-restserver en parte superior

*Para volver a pagina inical de este repositorio*

* Click a opcion ‘releases’ en segundo menu horizontal

*Pasa a pagina ‘Releases/Tags’*

AGREGAR UN TAG AL RESPALDO

**En consola**

* Teclear: git tag

*Se despliegan las versiones*

* Teclear: git tag -a v0.0.3 -m “Usuario y RESTServer”

*Se agrega el tag; no se despliega nada*

* Teclear: git push -- tags

*Se despliegan mensajes de ejecucion*

AGREGAR AL TAG UN RELEASE NOTE

**En navegador web**

En pagina ‘www.github.com’

* Click en opcion ‘Tags’

*Pasa a pagina de ‘Tags’*

*Vease que ya se muestra la version recien agregada v0.0.3*

* Click a los tres puntos a la derecha
* Click a ‘add release notes’

*Pasa a pagina de ‘Releases’*

* Teclear el nombre: Usuario y RESTServer
* Teclear la descripcion
* Click a boton verde ‘Publish release’

*Se agrega el ‘release’*