CRUD = **CREATE READ UPDATE DELETE**

**MVC = MODELO VISTA CONTROLADOR**

**Instalación**

1-Creamos una carpeta **“crud-nodejs-mongodb”**

2-arrastramos al workspace del editor de código

3- ctrl + shift + p 🡪 terminal 🡪 **npm init –yes** (Para iniciar un package json)

4-INSTALAR 🡪 **npm install express mongoose ejs morgan**

**5- INSTALAR 🡪 npm i nodemon -D (En DEV DEPENDENCIES)**

**express:** framework de node.js que nos permite escribir código del servidor de node.js de manera mas resuelta

**mongoose:** modulo que conecta express con mongodb y nos permite modelar los datos, interactuar con la base.

**ejs**: motor de plantilla, dentro del html podremos ejecutar pequeños javascript para diseñar el HTML

**(JAVASCRIPT EMBEBIDO EN HTML)**

**Morgan:** Modulo que ayuda en el desarrollo, nos muestra por consola que nos piden los usuarios

**nodemon estará en dev dependencies** (modulos que solo funcionan en el desarrollo pero no en la app)

**Servidor del proyecto**

**1-** creamos una folder “src” para poner el código del proyecto 🡪 dentro creamos “app.js”

**2**- Dentro de **src** creamos 3 carpetas🡪 **MVC (MODEL VIEW CONTROLLER)**

**“routes”:** para las rutas del servidor

**“views”:** vistas que **HTML** que envio al navegador (para archivos “**ejs**”)

**“models”:** almacena archivos de como lucen los datos dentro de la base de datos (que campos y métodos tienen)

**App.js**

const path = require ('path');

const express = require('express');

const morgan = require('morgan');

const app = express();

const mongoose = require('mongoose');

**//Connecting to DB**

mongoose.connect('mongodb://localhost/crud-mongo') //conexion del localhost a mongo db

.then(res => console.log('Data Base Connected'))  //promise muestra por consola la respuesta

.catch(err => console.log(err))

**//importing routes**

const indexRoutes = require('./routes/index'); //desde la carpeta routes archivo index

**//settings**

app.set('port', process.env.PORT || 3000); //Asi definimos 'port' como una variable

app.set('views', path.join(\_\_dirname, '/views'))  //decimos al servidor que la carpeta views esta en este mismo path para luego al hacer render la encuentre directo

app.set('view engine', 'ejs'); //con esto llamo al motor de plantillas ejs

**//middlewares** (funcion que se ejecuta antes que lleguen a las rutas)

//Da permiso de ir a la ruta o no, alterar y procesar antes que lleguen a la ruta

app.use(morgan('dev'));

app.use(express.urlencoded({extended: false})); //extended: true --> para archivos pesados

//Entiende datos que envía un formulario html, envía al servidor y traduce a .json

**//routes**

app.use('/', indexRoutes);  //cuando alguien entre a mi ruta inicial 🡪 indexRoutes

**//starting the server**

app.listen(app.get('port'), () =>{

    console.log(`Server on port ${app.get('port')}` );

});

**Unificar formato de rutas**

**A veces los símbolos de las rutas difieren dependiendo el sistema operativo /views \views**

1. **Importamos modulo preisntalado 🡪**

const path = require('path');

1. **Ejecutamos de esta manera 🡪**

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

**ROUTE**

1. Dentro de **/routes** 🡪 creamos **index.js**
2. **Index.js**

const express = require('express');

const router = express.Router(); //metodo de express (enrutador en blanco)

//almacenamos lo que devuelve

router.get('/', (req,res)=>{

    res.render('index'); //aca le decimos que responda la vista de /views/index.ejs

    //Aclaramos todo esto en app.js //settings

})

module.exports = router; //exporto para que otras carpetas lo requieran

1. H0la

**ESQUEMA**

1. Dentro de /models 🡪 creamos **tasks.js**
2. Creamos un esquema de mongoose y lo exportamos

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const TaskSchema = new Schema({ //

    tittle: String,

    description: String,

    status: {

        type: Boolean,

        default: false

    }

})

module.exports = mongoose.model('task',TaskSchema); // "nombre de la coleccion" , documento

**VIEW**

**Ejs: motor de plantilla que da funcionalidad extra al html**

**El navegador siempre recira HTML ignora el motor de plantilla utilizado**

1. Dentro de la carpeta **views** creamos 🡪 **index.ejs**
2. Vamos a 🡪 [www.getboostrap.com](http://www.getboostrap.com) 🡪 **get started 🡪 copiamos CSS y JS**
3. **CSS** debajo del titulo / **JS** al final del body para que muestre primero la pagina

**NO ME METO A FONDO CON BOOTSTRAP, TAL VEZ LUEGO DEL VIDEO REVEA LAS CLASES**

**Para reutilizar código de HTML que va a desplegarse igual en las diferentes paginas**

1. Dentro de **Views** creamos carpeta 🡪 **partials** (aquí guardamos estas vistas que se repiten)
2. Dentro de **partials 🡪 \_header.ejs** (por ejemplo) 🡪 **Pegamos esta parte del head que se repita**
3. Dentro de **partials 🡪 \_footer.ejs**  (El html va a quedar como desmembrado en varios ejs)
4. Para que función debo importar estos **partials 🡪 en index.js (con sintaxis de ejs)**

<%- include('partials/\_header.ejs') %>

<%- include('partials/\_footer.ejs') %>

Toda el codigo de **.ejs** y la explicacionesta escrito en el la carpeta del proyecto

HTML COMMENTS SHORTCUT 🡪 **CTRL + K + U**

**BOOTSTRAP SE CONFORMA DE 12 COLUMNAS**

**Ruta que sigue el input del usuario**

1. **El input que ingresa el usuario, es enviado desde ejs a la ruta add con un método post**

<form action="/add" method="POST">

1. **La ruta procesa el dato en index.js**

//Recibo los datos que vienen desde el formulario en index ejs

router.post('/add',async (req,res)=>{

    const task = new Task(req.body);

    //req.body es lo que nos envia el usuario y lo metemos dentro del esquema Task

    await task.save() //guardo lo que me devuelve en la BASE DE DATOS

    res.redirect('/') //Redirecciona a la ruta inicio

});

1. **Ya podemos ver este objeto en la base de datos**
2. **Para desplegar los datos del documento en la tabla del navegador**

router.get('/', async (req,res)=>{

    const tasks= await Task.find();

//aca le decimos que responda la vista de /views/index.ejs

    res.render('index', { //lo envio a la vista de index.ejs

        tasks //un objeto con el valor de tasks

    });

});

1. En **index.ejs 🡪 Para interpretar una variable (exportada por router)**

<%= tasks %>

**Solo despliega la vista en pantalla**

1. **recorrer la lista de datos y ubicarlos en la tabla en el navegador**

                <!-- table body -->

                <tbody>

                    <!-- Recorremos cada elemento del objeto (leer como for comun)-->

                    <% for(var i = 0; i < tasks.length; i++) {%>

                    <tr>

                        <td><%= i + 1%></td>

                        <td><%= tasks[i].tittle%></td>

                        <td><%= tasks[i].description%></td>

                        <td>

                            <!-- Estos botones van a ser enlaces que vamos a utilizar -->

                            <a href="" class="btn btn-dark">Done</a>

                            <a href="" class="btn btn-danger">Delete</a>

                            <a href="" class="btn btn-info">Edit</a>

                        </td>

                    </tr>

                    <% } %>

**DELETE**

1. **En index.ejs 🡪 /delete/ 🡪 Para que nos envie a una ruta del servidor**
2. **Tasks[i] 🡪 índice de la tarea que esta recorriendo representa el \_id**

**(\_id generados automáticamente por mongo db)**

<a href="/delete/<%= tasks[i].\_id %>" class="btn btn-danger">Delete</a>

1. **En index.js 🡪 ruta para eliminar un elemento de la base de datos**

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

    //:loquesea le dice que hay una especie de variable (nro de objeto en este caso)

    const {id} = req.params; //esta constante va a ser el id que viene de req.params

    // console.log(req.params.id)

    await Task.remove({\_id: id});

//Task (importado).borrar de base({buscar por \_id : id que nos devuelve el req.params})

    res.redirect('/'); //luego de Borrar redirecciona a esta ruta

})

**CAMBIAR STATUS (de un objeto) ESTUDIARLO**

**1-En index.js­ 🡪 ruta turn (color)**

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

    const {id} = req.params; //recibo los parametros del id del objeto

    const task = await Task.findById(id); //con ese id busco el objeto especifico y lo guardo

    task.status = !task.status; //si estaba false colocando el contrario va a estar en true y viceceersa

    await task.save(); //guardo la accion

    res.redirect('/') //refresco

})

**2-En index.ejs 🡪 tasks[i].\_id se reemplaza con el id de mongodb**

**(ver el array al que se refiere i en Mongo DB)**

**La clase hace un condicional con ejs**

<a href="/turn/<%= tasks[i].\_id %>"

class="btn <%= tasks[i].status?'btn-success':'btn-dark' %> ">Done</a>

**Editar dato de la base y mostrarlo**

**EDIT.JS**

1. **Creamos un archivo en la carpeta views 🡪 edit.ejs**
2. **Pegamos el codigo del primer formulario dentro del archivo, importamos partials**
3. **A value le agregamos el valor de la propiedad tittle**

 <input type="text" name = "tittle" placeholder="Inserta un titulo" class="form-control" value="<%task.tittle%>">

1. **DENTRO DEL TEXTAREA agregamos 🡪 <%task.description>**

<textarea name="description" cols="80" class="form-control" placeholder="insert a description"><%= task.description %></textarea>

**En la ruta de index.js**

router.post('/edit/:id',async (req,res)=>{

    const {id} = req.params;

    await Task.update({\_id: id},req.body); //el segundo parametro es el que estamos actualizando

    res.redirect('/');

});

1. **Al presionar el botón edit en index.ejs nos envía a su ruta get(/edit y post(edit/**
2. **Este método get renderiza la nueva vista ‘edit’ en esta nueva ruta edit/:id**
3. **Luego gracias a <form action= “edit/:id” method = “POST”> hacemos submit al formulario con sus datos (al boton update le dimos un type=submit y submitea la ruta que le dimos en <form action=)**

**Quiero hacer un grafico o algo de como se relaciona el href de ejs con la ruta de index.js**

1. **Presionan el botón que tiene este href**

<a href="/edit/<%= tasks[i].\_id %>"

1. **El índice de tasks[i]**

**Dudas**

**//settings**

app.set('port', process.env.PORT || 3000);

router.get('/', async (req,res)=>{

    const tasks= await Task.find(); //aca le decimos que responda la vista de /views/index.ejs

    res.render('index', { //lo envio a la vista de index.ejs

        tasks //envio un objeto con el valor de tasks a la vista index

    });  //le devuelvo la respuesta al

});

**Resumen del camino de un dato dentro del servidor y como los archivos se relacionan con otros archivos**

1. **De entrada estamos conectados a determinada base de datos en mongoose (en app.js)**
2. **Y le dijimos que estamos utilizando como view engine ejs (en app.js)**
3. **Por tanto las colecciones y propiedades que nombre en cualquier archivo ejs se estarán relacionando directamente con la base de datos 🡪 ej: <% = tasks[i].\_id %>**

**Tasks 🡪 colección / \_id 🡪 propiedad**

1. **De entrada cuando entramos a la ruta inicial nos despliega la vista de index.ejs**
2. **Puedo entonces enviar desde un botón un req.params a la ruta, el valor de una propiedad en un índice especifico en el array que compone la coleccion y los datos de su propiedad \_id en este caso**

<a href="/delete/<%= tasks[i].\_id %>"

1. **Las rutas estaran disponibles para cualquier archivo dentro del proyecto ya que fueron exportadas**
2. **Cada ruta cumplirá una función especifica, ejemplo DELETE 🡪 recibir el id, buscar este objeto en la base de datos, eliminarlo y redireccionarnos a determinada pagina**