



Guia CRUD de Usuarios

Hecho por: Carlos Estrella

Introducción:

En esta guía se podrá crear un CRUD de usuarios usando Express y Node.js así como para nivel de base de datos Mogodb.

Creación del entorno de la aplicación.

- 1.- Para iniciar con esta guía crearemos una carpeta la cual contendrá nuestro proyecto con el nombre guía-crud. Esta carpeta se abrirá con VS code y ahí se iniciará la terminal.
- 2.- El primer comando a ejecutar en la terminal será para instalar el framework Express que es el siguiente: **npm i express**.

```
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\CE\guia-crud>npm i express
npm WARN saveError ENOENT: no such file or
directory, open 'C:\Users\CE\guia-crud\
package.json'
npm notice created a lockfile as package-
lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or d
irectory, open 'C:\Users\CE\guia-crud\pac
kage.json'
npm WARN guia-crud No description
npm WARN guia-crud No repository field.
npm WARN guia-crud No README data
npm WARN guia-crud No license field.

+ express@4.16.3
added 50 packages in 18.159s
```

- 3.- Ya teniendo instalado el framework, ahora procederemos a crear un proyecto Express con el siguiente comando: **npm i express-generator -g**.

```
C:\Users\CE\guia-crud>npm i express-generator -g
C:\Users\CE\AppData\Roaming\npm\express -> C:\Users\CE\AppData\Roaming\npm\node_mo
dules\express-generator\bin\express-cli.js
+ express-generator@4.16.0
updated 1 package in 9.794s
```

- 4.- A continuación preparemos la estructura del sitio web con el siguiente comando: **express - -view =ejs**.

```

C:\Users\CE\guia-crud>express --view=ejs
destination is not empty, continue? [y/N] y

  create : public\
  create : public\javascripts\
  create : public\images\
  create : public\stylesheets\
  create : public\stylesheets\style.css
  create : routes\
  create : routes\index.js
  create : routes\users.js
  create : views\
  create : views\error.ejs
  create : views\index.ejs
  create : app.js
  create : package.json
  create : bin\
  create : bin\www

```

Ahora corremos la aplicación con el comando **npm start**, en caso de que nos salte algún error, se podrá solucionar instalando alguno de los siguientes módulos: **npm i "masNombredependencia"**.

Para finalizar instalaremos el motor ejs para codificar nuestras vistas con el comando **npm i ejs**.

```

C:\Users\CE\guia-crud>npm i ejs
+ ejs@2.5.9
added 1 package in 2.782s

```

Consejo: como vamos a estar haciendo constantes modificaciones, recomiendo instalar en este momento **nodemon** que es un servicio que mantiene en ejecución nuestra aplicación. El comando **npm i nodemon** para instalarlo y para iniciarlo con el comando **nodemon start**.

```

C:\Users\CE\guia-crud>npm i nodemon

> nodemon@1.18.3 postinstall C:\Users\CE\guia-crud\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
> https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

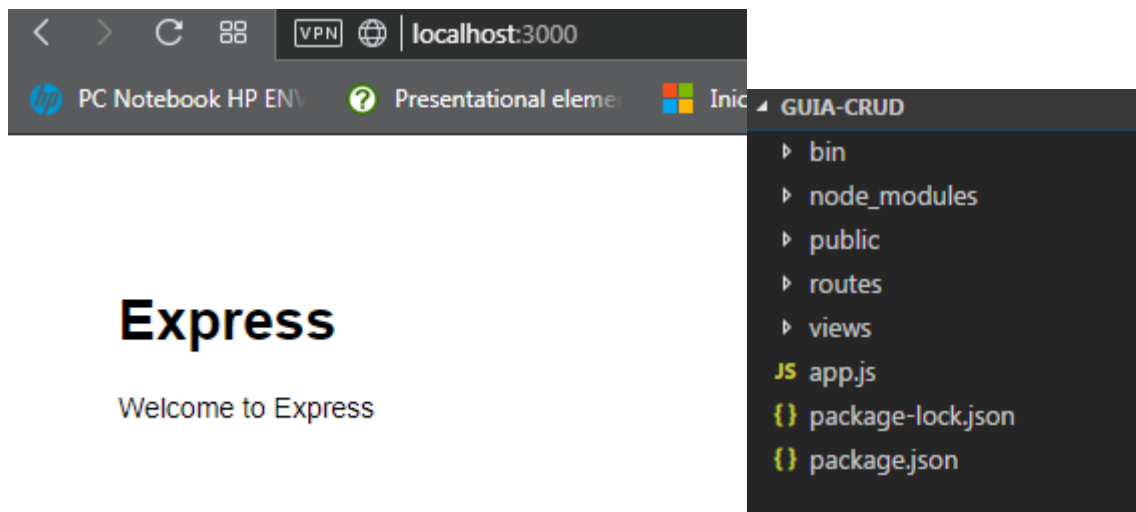
+ nodemon@1.18.3
added 226 packages in 77.719s

```

```
C:\Users\CE\guia-crud>nodemo start
"nodemo" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\CE\guia-crud>nodemon start
[nodemon] 1.17.5
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node ./bin/www start`
GET / 304 53.084 ms - -
GET /stylesheets/style.css 304 6.568 ms - -
```

Resultado: como primer resultado tendremos ya corriendo nuestra página de inicio y la estructura del proyecto es la siguiente.



Estructura de base de datos con Mogodb.

Antes de comenzar con la estructura de la base de datos tenemos que instalar el modulo para hacer la conexión con mongo debe, este se instala con el siguiente comando **npm i mongoose**.

```
C:\Users\CE\guia-crud>npm i mongoose
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ mongoose@5.2.4
added 18 packages in 34.406s
```

El siguiente paso es crear una carpeta con el nombre **lib** y dentro de ella crearemos un archivo llamado **db-connection.js**. Luego empezaremos por escribir en el archivo el siguiente código.

```

1  const mongoose = require('mongoose');
2
3  let db;
4
5  module.exports = function Connection(){
6    if(!db){
7      db = mongoose.createConnection('mongodb://localhost/crud-guia');
8    }
9    return db;
10 }

```

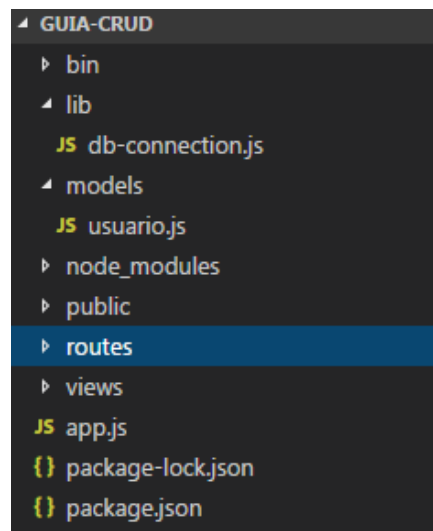
Una vez finalizada esta parte crearemos una carpeta **models** y dentro de esta un archivo el cual llamaremos **usuario.js** y tendrá el siguiente código.

```

1  module.exports = function() {
2    var db = require('../lib/db-connection')();
3    var Schema = require('mongoose').Schema;
4
5    var Usuario = Schema({
6      nombre: String,
7      apellidos: String,
8      foto: String,
9      sexo: String
10   });
11
12   return db.model('Usuario', Usuario);
13 }

```

Al finalizar esto tendremos la siguiente estructura del proyecto:



Complementos para la manipulación de datos entre el formulario y la aplicación.

1.- Antes de iniciar con la manipulación de los datos a través del patrón de diseño MVC debemos instalar los siguientes módulos:

Body-Parser: **npm i body-parser**

Express fileupload: **npm i express-fileupload**

Luego de la instalación dentro del archivo **app.js** se deben incluir de la siguiente forma:

```
var bodyParser = require('body-parser'); //modulo para recuperar
const fileUpload = require('express-fileupload'); //modulo para archivos

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

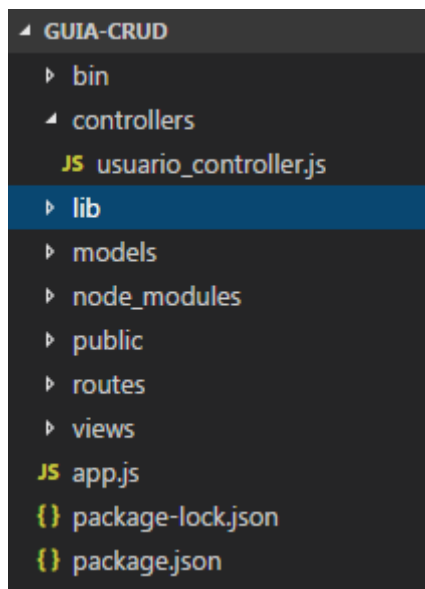
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

//Pasar datos del fomr en post.
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

//para archivos
app.use(fileUpload());
```

Operaciones en la base de datos.

Ahora crearemos una carpeta la cual llamemos **controllers** y crearemos dentro el archivo **usuario_controller.js** como se muestra en la imagen. Este archivo contendrá toda la lógica que nos permitirá hacer las operaciones de la base de datos y obtener los datos del formulario y tratarlos.



Comenzaremos por importar la estructura de la base datos con la siguiente línea del código que ira en el archivo **usuario_controller.js**

```
1  const model = require('../models/usuario')();  
2
```

Luego crearemos la función que nos permitirá cargar en la **página index** los elementos registrados en la base de dato.

```
exports.usuario= function(req,res){  
  model.find().  
  then(function(doc){  
    res.render('index', { title: 'CRUD Usuarios', usuarios:doc });  
  });  
};
```

Después de esto crearemos la función que nos llevara a la página para dar de alta al nuevo usuario.

```
exports.nuevoUsuario= function(req,res){  
  res.render('agregarUsuario', { title: 'Agregar Usuario' });  
};
```

Teniendo la función que nos llevara la página del formulario agregar usuario, ahora crearemos la función que nos permitirá insertar los datos del formulario a la base de datos.

```
exports.crearUsuario = function(req, res, nex){  
  var Usuario = {  
    nombre: req.body.nombre,  
    apellidos: req.body.apellidos,  
    foto: req.files.foto.name,  
    sexo: req.body.sexo,  
  };  
  
  //validar si hay archivo  
  if (!req.files){  
    return res.status(400).send('No hay archivos.');  }  
  
  //seccion subir archivo  
  let filToUpload = req.files.foto;  
  filToUpload.mv('public/images/'+req.files.foto.name, function(err) {  
    if (err)  
      return res.status(500).send(err);  
  });  
  
  var data = model(Usuario);  
  data.save();  
  console.log(req.body.nombre);  
  res.redirect('/');  
}
```

Ahora el siguiente método es crear la función que nos permitirá ir a la página modificar usuario. En este caso al seleccionar un usuario de la lista nos llevara a la página modificar usuario y se cargaran los datos actuales del usuario.

```
exports.cargarModificar = function(req, res){  
  var id = req.params.id;  
  console.log(id);  
  model.findById(id, function (err, doc) {  
    res.render('modificarUsuario', { title: 'Modificar Usuario', usuario:doc });  
  });  
}
```

Ahora si crearemos la función modificar datos del usuario con las siguientes líneas de código:

```
exports.modificarUsuario = function(req, res){
  var id = req.body.id;

  model.findById(id,function(err, doc){
    doc.nombre = req.body.nombre;
    doc.apellidos = req.body.apellidos;
    if(req.files.foto != null){
      doc.foto = req.files.foto.name;
      //validar si hay archivo
      if (!req.files){
        return res.status(400).send('No hay archivos.');
```

Para finalizar con esta sección el último método sería el de eliminar usuario y está compuesto por el siguiente código:

```
exports.eliminarUsuario = function(req, res){
  var id = req.params.id;
  console.log(id);
  model.findByIdAndRemove(id).exec();
  res.redirect('/');
```

Creación de vistas

Dentro de la carpeta **view** tendremos la siguiente estructura:

```
├── routes
├── views
│   ├── layout
│   ├── agregarUsuario.ejs
│   ├── error.ejs
│   ├── index.ejs
│   └── modificarUsuario.ejs
├── app.js
├── package-lock.json
└── package.json
```


Comenzaremos por definir una carpeta que tendrá el nombre **layout**, dentro estarán dos archivos uno con el nombre de **footer.ejs** y el otro con el nombre de **header.ejs**. El objetivo de estos dos archivos es tener una plantilla reutilizable dentro de las demás vistas que crearemos. Nos ahorra espacio en el código del HTML.

Dentro del primer archivo llamado **header.ejs** llevara el siguiente código HTML: aquí incluiremos la librería de **bootstrap**.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>UTM-<%= title %></title>
5     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" in
6   </head>
7   <body>
8
9
```

Dentro del segundo archivo footer.ejs incluiremos las siguientes líneas:


```
1   <div>UTM - 9C</div>
2 </body>
3 </html>
```

Ahora comenzaremos por modificar la página **index** de tal forma que el resultado final sea algo como esto:

CRUD Usuarios

Welcome to CRUD Usuarios

[Agregar Usuario](#)

Nombre	Apellido	Foto	Genero	
test foto	test		test	Modificar Eliminar

Ahora se anexa el código del **index** a utilizar:

```
<%- include layout/header.ejs%>

<div class="container">
  <h1><%= title %></h1>
  <p>Welcome to <%= title %></p>
  <a class="btn btn-primary" href="/agregarUsuario">Agregar Usuario</a>
  <hr>
  <table class="table table-striped ">
    <thead>
      <tr>
        <th>Nombre</th>
        <th>Apellido</th>
        <th style="width:20%;">Foto</th>
```

```

        <th>Genero</th>
        <th></th>
        <th></th>
    </tr>
</thead>
<tbody>
    <% for(var i=0; i < usuarios.length; i++){ %>
        <tr>
            <td><%= usuarios[i].nombre %></td>
            <td><%= usuarios[i].apellidos %></td>
            <td></td>
            <td><%= usuarios[i].sexo %></td>
            <td><a href="/modificarUsuario/<%= usuarios[i].id %>" class="btn btn-
success">Modificar</a></td>
            <td><a id="eliminar" class="btn btn-danger" href="/eliminarUsuario/<%=
usuarios[i].id %>" class="btn btn-primary">Eliminar</a></td>
        </tr>
    <% }%>
</tbody>

</table>
</div>
<!--Script validar eliminar-->
<script>
    document.getElementById('eliminar').addEventListener('click',function(e){
        let response = confirm('¿Desea eliminar este registro?');
        if(!response){
            e.preventDefault();
        }
    })
</script>
<%- include layout/footer.ejs%>

```

Como podemos observar se agregó una función script para poder validar la acción de eliminar.

Ahora procederemos a crear el formulario para agregar un nuevo usuario el cual tendrá la siguiente apariencia y crearemos el archivo **agregarUsuario.ejs**.

Agregar Usuario

Nombre

Apellidos

Foto

Ningún archivo seleccionado

Genero

©UTM - 9C

El código es el siguiente:

```
<%- include layout/header.ejs%>
<h1><%= title %></h1>
<center>
  <form action="/agregarUsuario" method="post" enctype="multipart/form-data">
    <div class="col-md-4">
      <div class="form-group">
        <label for="nombre">Nombre</label>
        <input type="text" class="form-control" id="nombre" name="nombre"
placeholder="Nombre">
      </div>
      <br>
      <div class="form-group">
        <label for="apellidos">Apellidos</label>
        <input type="text" class="form-control" id="apellidos" name="apellidos"
placeholder="Apellidos">
      </div>
      <br>
      <div class="form-group">
        <label for="foto">Foto</label>
        <!--<input type="text" class="form-control" id="foto" name="foto"
placeholder="Foto">-->
        <input type="file" class="form-control-file" id="foto" name="foto">
      </div>
      <br>
      <div class="form-group">
        <label for="sexo">Genero</label>
        <input type="text" class="form-control" id="sexo" name="sexo"
placeholder="Genero">
      </div>
      <br>
      <div class="form-group ">
        <button type="submit" class="btn btn-dark">Agregar usuario</button>
```

```

        </div>
    </div>

    </form>
</center>
<%- include layout/footer.ejs%>

```

Para finalizar con esta sección de vistas ahora crearemos la vista de modificar usuario con el archivo por nombre **modificarUsuario.ejs** con el siguiente código:

```

<%- include layout/header.ejs%>
<h1><%= title %></h1>
<center>
    <form action="/modificarUsuario" method="post" enctype="multipart/form-data">
        <input type="hidden" name="id" value="<%= usuario.id%>">
        <div class="col-md-4">
            <div class="form-group">
                <label for="nombre">Nombre</label>
                <input type="text" class="form-control" id="nombre" name="nombre"
value="<%= usuario.nombre %>" placeholder="Nombre">
            </div>
            <br>
            <div class="form-group">
                <label for="apellidos">Apellidos</label>
                <input type="text" class="form-control" id="apellidos" name="apellidos"
value="<%= usuario.apellidos %>" placeholder="Apellidos">
            </div>
            <br>
            <div class="form-group">
                <label for="foto">Foto</label>
                <input type="file" class="form-control-file" id="foto" name="foto">
            </div>
            <br>
            <div class="form-group">
                <label for="sexo">Genero</label>
                <input type="text" class="form-control" id="sexo" name="sexo" value="<%=
usuario.sexo %>" placeholder="Genero">
            </div>
            <br>
            <div class="form-group ">
                <button type="submit" class="btn btn-dark">Modificar usuario</button>
            </div>
        </div>

    </form>
</center>
<%- include layout/footer.ejs%>

```

Como se dieron cuenta esta última parte se reutilizar el mismo formulario que el de agregar solo que en este se cargan los inputs con los valores del usuario seleccionado.

Al final tendremos algo como esto:

Modificar Usuario

Nombre
prueba controller 2

Apellidos
estrella

Foto
Seleccionar archivo Ningún archivo seleccionado

Género
masculino

Modificar usuario

Adaptación del archivo router

Para que nuestras vistas y el controlador se unan y funcionen en debemos emplear los métodos creados en el archivo **usuario_controller.ejs** con el archivo **index.js** que está dentro de la carpeta routes

En este archivo incluiremos las siguientes líneas de código:

Es importante agregar el módulo de controlados-

```
var usuarioController = require('../controllers/usuario_controller');

//Aplicacion de Controllers //obtener lista usuarios
router.get('/', usuarioController.usuario);

//Aplicacion de Controllers //ir a pagina crear usuario
router.get('/agregarUsuario', usuarioController.nuevoUsuario);

//Aplicacion de Controllers //accion crear usuario
router.post('/agregarUsuario', usuarioController.crearUsuario);

//Aplicacion de Controllers //eliminar usuario
router.get('/eliminarUsuario/:id', usuarioController.eliminarUsuario);

//Aplicacion de Controllers //cargar usuario seleccionado
router.get('/modificarUsuario/:id', usuarioController.cargarModificar);

//Aplicacion de Controllers //accion modificar libro
router.post('/modificarUsuario', usuarioController.modificarUsuario);
```

Con esto concluimos esta guía básica de crud con Express, node.js y mongo db.