



El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:

## Terminología básica

1. Que es Express JS?
2. Crear un servidor WEB.
3. Trabajando con método get.
4. Trabajando con método post.
5. Trabajando con método put.
6. Trabajando con método delete.
7. Herramienta postman.
8. Manejo de bases de datos NoSQL.
9. Manejo de bases de datos NoSQL - CRUD.



El futuro digital  
es de todos

MinTIC

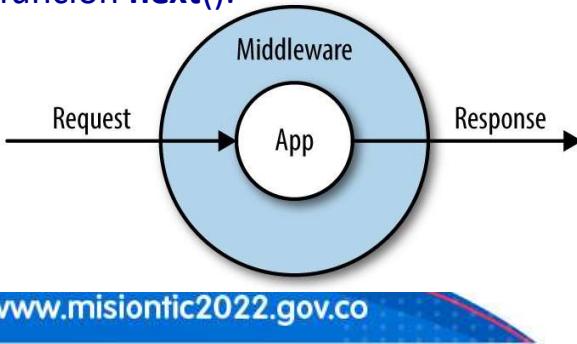


Misión  
TIC 2022

## ¿Qué es ExpressJS?

Expressjs es un framework rápido, minimalista y flexible de Node.js. Permite crear APIs y aplicaciones web fácilmente, provee un conjunto de características como manejo de rutas (direcciónamiento), archivos estáticos, uso de motor de plantillas, integración con bases de datos, manejo de errores, middlewares entre otras.

Un middleware es una función que se puede ejecutar antes o después del manejo de una ruta. Esta función tiene acceso al objeto **Request**, **Response** y la función **next()**.



node.js™  
express



El futuro digital  
es de todos

MinTIC

Misión  
TIC 2022

# Organizamos las carpetas de nuestra APP

as > Ciclos MinTic2022 UCaldas > Ciclo 4 > Deployments > Ejemplo UdeA > mern-commerce



backend



frontend

El **backend** es la parte del desarrollo web que se encarga de que toda la **lógica** de una página web funcione. Se trata del conjunto de acciones que pasan en una web pero que no vemos como, por ejemplo, la comunicación con el servidor.

El **frontend** es la parte del desarrollo web que se dedica a la parte frontal de un sitio web, en pocas palabras del **diseño** de un sitio web, desde la estructura del sitio hasta los estilos como colores, fondos, tamaños hasta llegar a las animaciones y efectos.



El futuro digital  
es de todos

MinTIC

Misión  
TIC 2022

# Iniciar el proyecto

Ejecutamos el comando: **npm init** para inicializar el proyecto, dentro de la carpeta **backend**.

```
o UdeA/mern-e-commerce/backend
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (backend)
version: (1.0.0)
description: App Ecommerce
entry point: (index.js)
test command:
git repository:
keywords:
author: MisionTic2022 ciclo 4A Desarrollo Aplicaciones WEB
license: (ISC)
About to write to D:\OneDrive\Trabajo Ivan Castro Ruiz\Docencia\1 - MisionTIC\Documentos ucaldas\Ciclos MinTic2022 ucaldas\ciclo 4\Deployments\Ejemplo UdeA\mern-e-commerce\backend\package.json:

{
  "name": "backend",
  "version": "1.0.0",
  "description": "App Ecommerce",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "MisionTic2022 ciclo 4A Desarrollo Aplicaciones WEB",
  "license": "ISC"
}

Is this OK? (yes) yes
```

Presionamos enter

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "App Ecommerce",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "MisionTic2022 Ciclo 4A Desarrollo Aplicaciones WEB",
  "license": "ISC"
}
```

Este proceso nos crea el  
archivo **package.json**.



El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Instalar ExpressJS en el proyecto.

Ejecutamos el comando: **npm i express cors dotenv mongoose** para instalar todas la dependencias de producción, necesaria para iniciar nuestro servidor WEB, revisamos que estemos dentro de la carpeta de **backend**.

```
PS D:\OneDrive\Trabajo Ivan Castro Ruiz\Docencia\1 - MisionTIC\Documentos UCaldas\Ciclos MinTic2022 UCaldas\Ciclo 4\Deployments\Ejemplo UdeA\mern-e-commerce\backend> npm i express cors dotenv mongoose
```

Ejecutamos el comando: **npm i --save-dev nodemon** para instalar todas la dependencias de desarrollo, necesarias para iniciar nuestro servidor WEB, revisamos que estemos dentro de la carpeta de **backend**.

```
PS D:\OneDrive\Trabajo Ivan Castro Ruiz\Docencia\1 - MisionTIC\Documentos UCaldas\Ciclos MinTic2022 UCaldas\Ciclo 4\Deployments\Ejemplo UdeA\mern-e-commerce\backend> npm i --save-dev nodemon
```

```
package.json x
@ package.json > {} devDependencies
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "description": "App Ecommerce",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\"$Error: no test specified\\" && exit 1"
8    },
9    "author": "MisionTic2022 Ciclo 4A Desarrollo Aplicaciones WEB",
10   "license": "ISC",
11   "dependencies": {
12     "cors": "^2.8.5",
13     "dotenv": "^16.0.3",
14     "express": "^4.18.2",
15     "mongoose": "^6.7.1"
16   },
17   "devDependencies": {
18     "nodemon": "^2.0.20"
19   }
20 }
```



# Modificamos el archivo package.json.

```
package.json x
package.json > {} dependencies
{
  "name": "backend",
  "version": "1.0.0",
  "type": "module",
  "description": "App Ecommerce",
  "main": "index.js",
  "scripts": {
    "test": "echo \\"$Error: no test specified\\" && exit 1",
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "author": "MisionTic2022 Ciclo 4A Desarrollo Aplicaciones WEB",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "mongoose": "^6.7.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

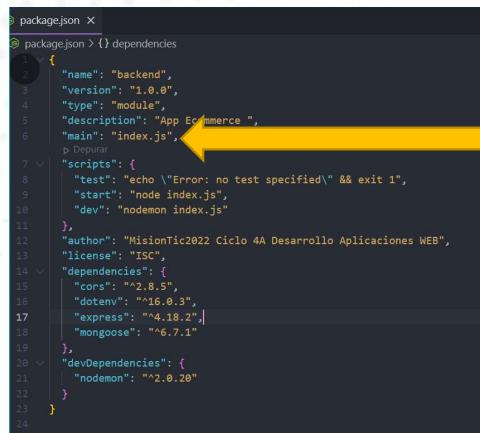
Para trabajar con  
sintaxis de import y  
export.

Para inicializar el servidor, y con **npm run dev** lo ejecutamos así cada cambio que hagamos de manera automática se detecta y el servidor se reinicia automáticamente.



# Creamos el archivo index.js

Este archivo es el punto de arranque de nuestro servidor, recordar que así lo definimos al inicializar el proyecto, revisamos que estemos dentro de la carpeta de **backend**.



```
package.json x
{
  "name": "backend",
  "version": "1.0.0",
  "type": "module",
  "description": "App E-commerce",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "author": "MisionTic2022 Ciclo 4A Desarrollo Aplicaciones WEB",
  "license": "ISC",
  "dependencies": {
    "cors": "2.8.5",
    "dotenv": "16.0.3",
    "express": "4.18.2",
    "mongoose": "6.7.1"
  },
  "devDependencies": {
    "nodemon": "2.0.20"
  }
}
```

```
// https://expressjs.com/es/
import express from "express";
import dotenv from 'dotenv';
import cors from 'cors';
import conectarDB from './config/db.js';
import usuarioRoutes from './routes/usuarioRoutes.js';
const PORT = process.env.PORT || 4000;
dotenv.config();

// Se le agrega toda la funcionalidad del servidor de express
const app = express();
app.use(express.json());

conectarDB();

// middlewares
// Se utiliza para realizar la comunicacion entre el servidor del frontend y el backend
const dominiosPermitidos = [process.env.FRONTEND_URL];
const corsOptions = {
  origin: function(origin, callback){
    if(dominiosPermitidos.indexOf(origin) !== -1){
      // El origen del Request esta permitido
      callback(null, true);
    }else{
      callback(new Error('No permitido por CORS'));
    }
  }
};

app.use(cors(corsOptions));

app.use('/api/usuarios', usuarioRoutes);

app.listen(PORT, () => {
  console.log(`Servidor funcionando en el puerto ${PORT}`);
});
```



# Creamos el archivo index.js

Explicación breve de cada instrucción.

```
// https://expressjs.com/es/
import express from "express";
import dotenv from 'dotenv';
import cors from 'cors';
import conectarDB from './config/db.js';
import usuarioRoutes from './routes/usuarioRoutes.js';
const PORT = process.env.PORT || 4000;
dotenv.config();

// Se le agrega toda la funcionalidad del servidor de express
const app = express();
app.use(express.json());

conectarDB();

// middlewares
// Se utiliza para realizar la comunicación entre el servidor del frontend y el backend
const dominiosPermitidos = [process.env.FRONTEND_URL];
const corsOptions = {
    origin: function(origin, callback){
        if(dominiosPermitidos.indexOf(origin) !== -1){
            // El origen del Request esta permitido
            callback(null, true);
        }else{
            callback(new Error('No permitido por CORS'));
        }
    }
};

app.use(cors(corsOptions));
app.use('/api/usuarios', usuarioRoutes);
app.listen(PORT, () => {
    console.log(`Servidor funcionando en el puerto ${PORT}`);
});
```

Importamos todas las funcionalidades necesarias.

Para que acepte variables de entorno.

Con esta instrucción, le indicamos a nuestro servidor que la información la vamos a recibir y enviar a través de JSON.

Función que se encargara de hacer la conexión con la BD

Solo permite el acceso a dominios que se establece en el frontend.

Para gestionar las rutas que definimos para la gestión de los usuarios, y el controlador que se encargara de gestionar la lógica de la petición y así mismo se encargara de direccionar la respuesta.



Mangosta es un Object Document Mapper (ODM). Esto significa que Mongoose le permite definir objetos con un esquema fuertemente tipado que se asigna a un documento MongoDB.



# Conexión a la base de datos

Creamos una carpeta llamada **config** y dentro de esta carpeta un archivo llamado **db.js**.

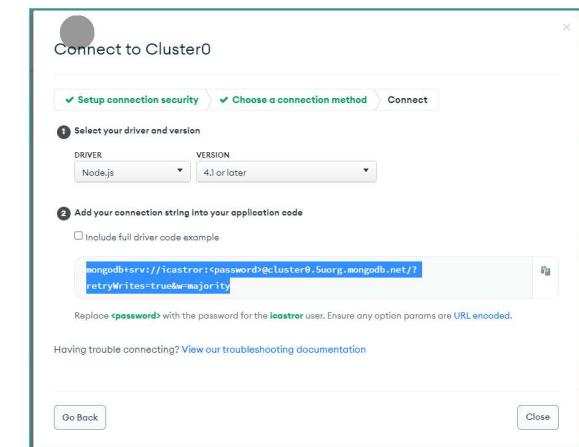
```
//https://mongoosejs.com/
// npm i mongoose
import mongoose from 'mongoose';

const conectarDB = async () =>{
    try {
        const db = await mongoose.connect(
            process.env.MONGO_URI,
            {
                useNewUrlParser: true,
                useUnifiedTopology: true,
            }
        );
        const url = `${db.connection.host}:${db.connection.port}`;
        console.log(`MongoDB conectado en: ${url}`);
    } catch (error) {
        console.log(`Error: ${error.message}`);
        process.exit(1);
    }
}
export default conectarDB;
```

Exportamos la función para invocarla en index.js

Mongoose es una librería para Node.js que nos permite escribir consultas para una base de datos de MongoDB, con características como validaciones, construcción de queries, middlewares, conversión de tipos y algunas otras, que enriquecen la funcionalidad de la base de datos.

Cadena de conexión de MongoDB Atlas.





# Conexión a la base de datos

```
//https://mongoosejs.com/
// npm i mongoose
import mongoose from 'mongoose';

const conectarDB = async () =>{
    try {
        const db = await mongoose.connect(
            process.env.MONGO_URI,
            {
                useNewUrlParser: true,
                useUnifiedTopology: true,
            }
        )
        const url = `${db.connection.host}:${db.connection.port}`;
        console.log(`MongoDB conectado en: ${url}`);
    } catch (error) {
        console.log(`Error: ${error.message}`);
        process.exit(1);
    }
}

export default conectarDB;
```

# Conexión a la base de datos

Como estamos trabajando con variables de entornos creamos el archivo `.env`

**MONGO\_URI** Es la cadena de conexión de MongoDB Atlas, recuerden trabajar con la que crearon su usuarios y contraseña de **MongoDB Atlas**



# Creación del archivo usuarioRoutes.js

Creamos una carpeta llamada **routes** y dentro de esta carpeta creamos un archivo llamado **usuarioRoutes.js**.

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with various icons for file operations like Open, Save, Find, and Refresh. The main area has a title bar with "Archivo", "Editar", "Selección", "Ver", "Ir", "Ejecutar", "Terminal", and "Ayuda". Below the title bar is a navigation bar with "EXPLORADOR" and a dropdown menu. The "EDITORES ABIERTOS" section shows a file named "usuarioRoutes.js" with a yellow icon. The "BACKEND" section contains "config", "controllers" (which is currently selected, indicated by a blue border), "node\_modules", and "routes". Inside the "routes" folder, there are files: "usuarioRoutes.js" (yellow icon), ".env" (grey icon), "index.js" (yellow icon), "package-lock.json" (grey icon), and "package.json" (grey icon). The central pane displays the code for "usuarioRoutes.js":

```
routes > JS usuarioRoutes.js > default
1 import express from 'express';
2 import {
3     prueba
4 } from '../controllers/usuarioController.js';
5
6 const router = express.Router();
7
8 // Rutas Publicas
9 router.get('/prueba', prueba);
10
11 export default router;
```

A yellow arrow points from the explanatory text below to the line "const router = express.Router();".

**express.Router**  
Utilice la clase `express.Router` para crear manejadores de rutas montables y modulares. Una instancia Router es un sistema de middleware y direccionamiento completo; por este motivo, a menudo se conoce como una "miniaplicación".

Los **verbos Http** involucrados en un sistema REST son GET, POST, PUT, PATCH y DELETE.



El futuro digital  
es de todos

MinTIC

Misión  
TIC 2022

# Creación del archivo usuarioRoutes.js

```
import express from 'express';
import {
    prueba
} from '../controllers/usuarioController.js';

const router = express.Router();

// Rutas Publicas
router.get('/prueba', prueba);

export default router;
```



# Creación del archivo usuarioController.js

Creamos una carpeta llamada **controllers** y dentro de esta carpeta creamos un archivo llamado **usuarioController.js**

```
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
usuarioController.js - backend - Visual Studio Code

EXPLORADOR ... JS usuarioController.js ×
EDITORES ABIERTOS controllers > JS usuarioController.js > ...
BACKEND
  config
  controllers
    usuarioController.js
  node_modules
  routes
    usuarioRoutes.js
  .env
  index.js
  package-lock.json
  package.json

1 2 const prueba = (req, res)=>{
3   res.send({
4     msg:"En esta ruta gestionaremos todas las peticiones correspondiente al modelo de Usuario"
5   })
6 };
7
8 export {
9   prueba
10 };
```

A través de estos middleware implementaremos la lógica para dar respuestas a las peticiones que lleguen en referencia al modelo de Usuarios.



El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Iniciar el servidor

```
PS D:\OneDrive\Trabajo Ivan Castro Ruiz\Docencia\1 - MisionTIC\Documentos UCaldas\Ciclos MinTic2022 UCaldas\Ciclo 4\Deployments\Ejemplo UdeA\mern-eCommerce\backend> npm run dev

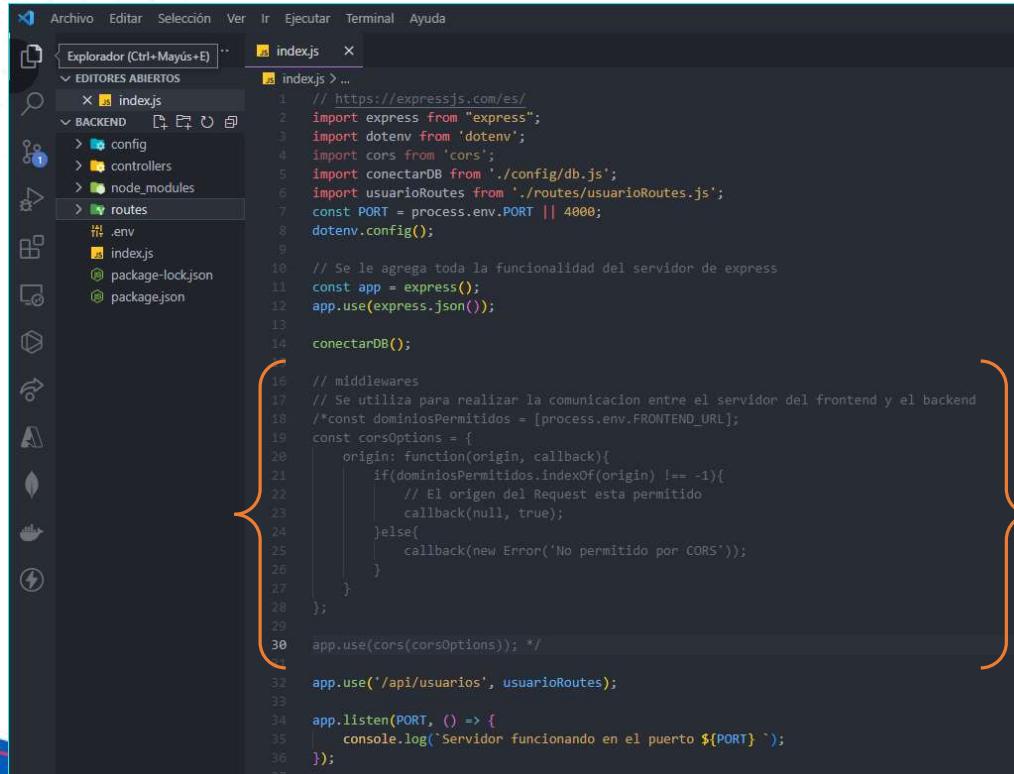
> backend@1.0.0 dev
> nodemon index.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Servidor funcionando en el puerto 4000
MongoDB conectado en: cluster0-shard-00-00.5uorg.mongodb.net:27017
```



# cors: Comentar la instrucción del cors

Para poder validar nuestras rutas con POSTMAN y el navegador temporalmente vamos a comentar las instrucciones del cors en el archivo **index.js**



```
// https://expressjs.com/es/
import express from 'express';
import dotenv from 'dotenv';
import cors from 'cors';
import conectarDB from './config/db.js';
import usuarioRoutes from './routes/usuarioRoutes.js';
const PORT = process.env.PORT || 4000;
dotenv.config();

// Se le agrega toda la funcionalidad del servidor de express
const app = express();
app.use(express.json());

conectarDB();

// middlewares
// Se utiliza para realizar la comunicación entre el servidor del frontend y el backend
//const dominiosPermitidos = [process.env.FRONTEND_URL];
const corsOptions = {
    origin: function(origin, callback){
        if(dominiosPermitidos.indexOf(origin) !== -1){
            // El origen del Request está permitido
            callback(null, true);
        }else{
            callback(new Error('No permitido por CORS'));
        }
    }
};

app.use(cors(corsOptions)); /*

app.use('/api/usuarios', usuarioRoutes);

app.listen(PORT, () => {
    console.log(`Servidor funcionando en el puerto ${PORT}`);
});
```

Para que Express y cors no bloquen las peticiones que vamos a realizar y probar nuestros **endpoints** desde **POSTMAN**



**El futuro digital  
es de todos**

MinTIC

Mision  
TIC 2022

Probamos la ruta en el navegador y POSTMAN

The screenshot illustrates a development environment for a REST API. At the top, a browser window shows the URL `localhost:4000/api/usuarios/prueba`. The main content area displays a JSON object with a single key-value pair: `msg: "En esta ruta gestionaremos todas las peticiones correspondiente al modelo de Usuario"`. Below this, the Postman application is open, showing a GET request to the same endpoint. The response body in Postman also contains the same JSON object, indicating a successful test run. A large yellow arrow points from the browser's response area down to the Postman response body, highlighting the consistency between the two environments.

Get por que así se definió en el routes.

localhost:4000/api/usuarios/prueba

{  
  msg: "En esta ruta gestionaremos todas las peticiones correspondiente al modelo de Usuario"  
}

+ - View source

Home Workspaces API Network Explore

GET http://localhost:4000/api/usuarios/prueba

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 19 ms Size: 329 B Save Response

1 {  
2 msg: "En esta ruta gestionaremos todas las peticiones correspondiente al modelo de Usuario"  
3 }

Respueta de nuestro endpoint, en formato JSON.

**Get por que así se definió en el routes.**

**Respuesta de nuestro endpoint, en formato JSON.**



El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Creamos las rutas para Producto y Venta

Modificamos el archivo **index.js**

```
indexjs  x
indexjs > ...
1 // https://expressjs.com/es/
2 import express from "express";
3 import dotenv from 'dotenv';
4 import cors from 'cors';
5 import conectarDB from './config/db.js';
6 import usuarioRoutes from './routes/usuarioRoutes.js';
7 import productoRoutes from './routes/productoRoutes.js';
8 import ventaRoutes from './routes/ventaRoutes.js';
9 const PORT = process.env.PORT || 4000;
10 dotenv.config();
11
12 // Se le agrega toda la funcionalidad del servidor de express
13 const app = express();
14 app.use(express.json());
15
16 conectarDB();
17
18 // middlewares
19 // Se utiliza para realizar la comunicacion entre el servidor del frontend y el backend
20 /*const dominiosPermitidos = [process.env.FRONTEND_URL];
21 const corsOptions = {
22   origin: function(origin, callback){
23     if(dominiosPermitidos.indexOf(origin) != -1){
24       // El origen del Request esta permitido
25       callback(null, true);
26     }else{
27       callback(new Error('No permitido por CORS'));
28     }
29   }
30 }*/
31 app.use(cors(corsOptions)); */
33
34 // Gestion Usuarios
35 app.use('/api/usuarios', usuarioRoutes);
36 // Gestion Productos
37 app.use('/api/productos', productoRoutes);
38 // Gestion Ventas
39 app.use('/api/ventas', ventaRoutes);
40
41 app.listen(PORT, () => {
42   console.log(`Servidor funcionando en el puerto ${PORT}`);
43 });
44
```

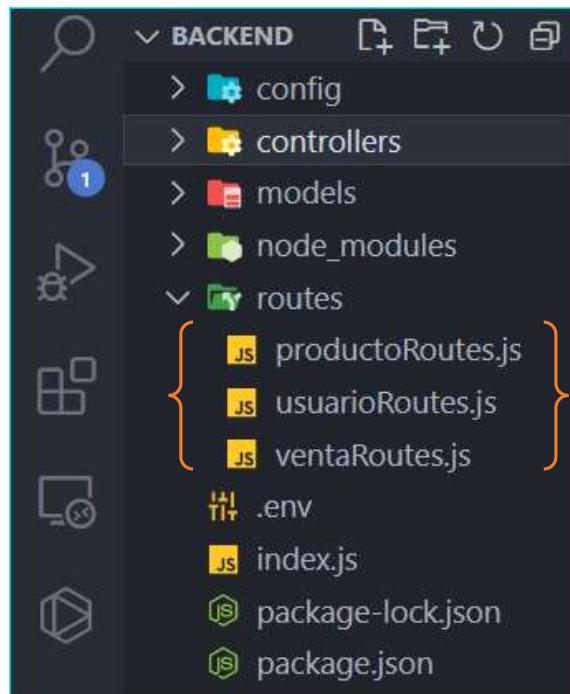


El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Creamos las rutas para Producto y Venta





El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Creamos los controladores para Producto y Venta

```
BACKEND
  config
  controllers
    productoController.js
    usuarioController.js
    ventaController.js
  models
  node_modules
  routes
  .env
  index.js
  package-lock.json
  package.json
```



El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Routes Producto y Venta

## productoRoutes.js

```
import express from 'express';
import {
  prueba
} from '../controllers/productoController.js';

const router = express.Router();

// Rutas Publicas
router.get('/prueba', prueba);

export default router;
```

## ventaRoutes.js

```
import express from 'express';
import {
  prueba
} from '../controllers/ventaController.js';

const router = express.Router();

// Rutas Publicas
router.get('/prueba', prueba);

export default router;
```



El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Controlador Producto y Venta

## productoController.js

```
const prueba = (req, res)=>{
    res.send({
        msg:"En esta ruta gestionaremos todas las
        peticiones correspondiente al modelo de Producto"
    })
};

export {
    prueba
};
```

## ventaController.js

```
const prueba = (req, res)=>{
    res.send({
        msg:"En esta ruta gestionaremos todas
        las peticiones correspondiente al modelo de
        Venta"
    })
};

export {
    prueba
};
```



El futuro digital  
es de todos

MinTIC

Mision  
TIC 2022

# Validar las rutas de Producto y Venta

GET http://localhost:4000/ε

http://localhost:4000/api/productos/prueba

GET http://localhost:4000/api/productos/prueba

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 "msg": "En esta ruta gestionaremos todas las peticiones correspondiente al modelo de Producto"
2
3
```

Status: 200 OK Time: 50ms

GET http://localhost:4000/ε

http://localhost:4000/api/ventas/prueba

GET http://localhost:4000/api/ventas/prueba

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 "msg": "En esta ruta gestionaremos todas las peticiones correspondiente al modelo de Venta"
2
3
```

Status: 200 OK Time: 50ms



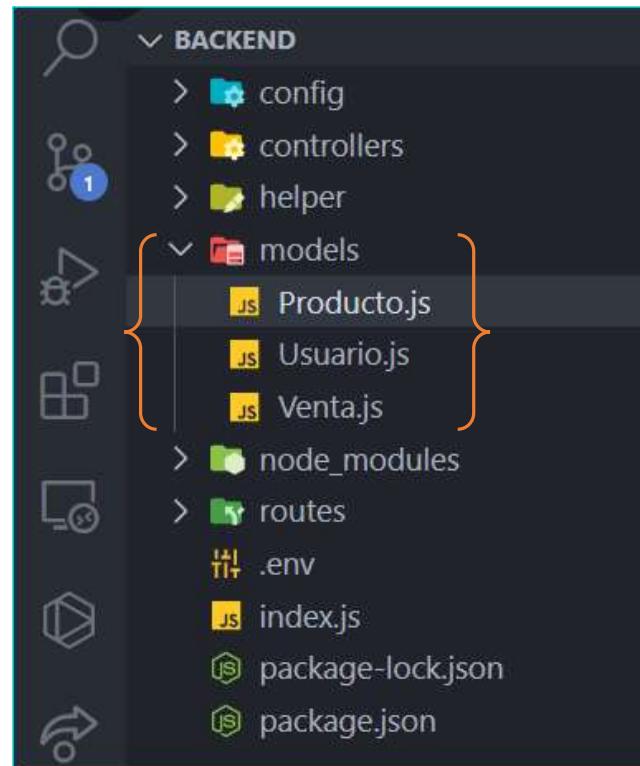
El futuro digital  
es de todos

MinTIC

Misión  
TIC 2022

# Modelo de Clases

Creamos una carpeta llamada **models** y dentro de esta carpeta creamos un archivo llamado **Producto.js**, **Usuario.js** y **Venta.js**





# Modelo de Usuario

Debemos crear una función **generarId** para crear un token dinámico el cual servirá para saber si es un usuario valido y .

En el modelo de Usuario se hace necesario instalar el paquete de **bcrypt** el cual nos permite hashear el password.

```
import mongoose from 'mongoose';
//https://www.npmjs.com/package/bcryptjs
//https://www.npmjs.com/package/bcrypt
import bcrypt from 'bcrypt';
import generarId from '../helper/generarId.js';

const usuarioShema = mongoose.Schema({
  nombre:{
    type: String,
    required: true,
    trim: true,
  },
  email:{
    type: String,
    required: true,
    unique: true,
    trim: true,
  },
  password:{
    type: String,
    required: true,
  },
  telefono:{
    type: String,
    default: null,
    trim: true,
  },
  direccion:{
    type: String,
    default: null,
    trim: true,
  },
  web:{
    type: String,
    default: null,
    trim: true,
  },
  token:{
    type: String,
    default: generarId(),
  },
  confirmado:{
    type: Boolean,
    default: false,
  },
  rol:{
    type: String,
    default: null,
    trim: true,
  }
});

// Antes de guardar el usuario Hashear el password
// https://www.npmjs.com/package/bcryptjs
// https://www.npmjs.com/package/bcrypt
usuarioShema.pre('save', async function(next) {
  if(!this.isModified('password')){
    next();
  };
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

// Confirmar password del usuario, esta funcion devuelve verdadero o falso
usuarioShema.methods.comprobarPassword = async function(passwordFormulario) {
  return await bcrypt.compare(passwordFormulario, this.password);
};

const Usuario = mongoose.model('Usuario', usuarioShema);
export default Usuario;
```



El futuro digital  
es de todos

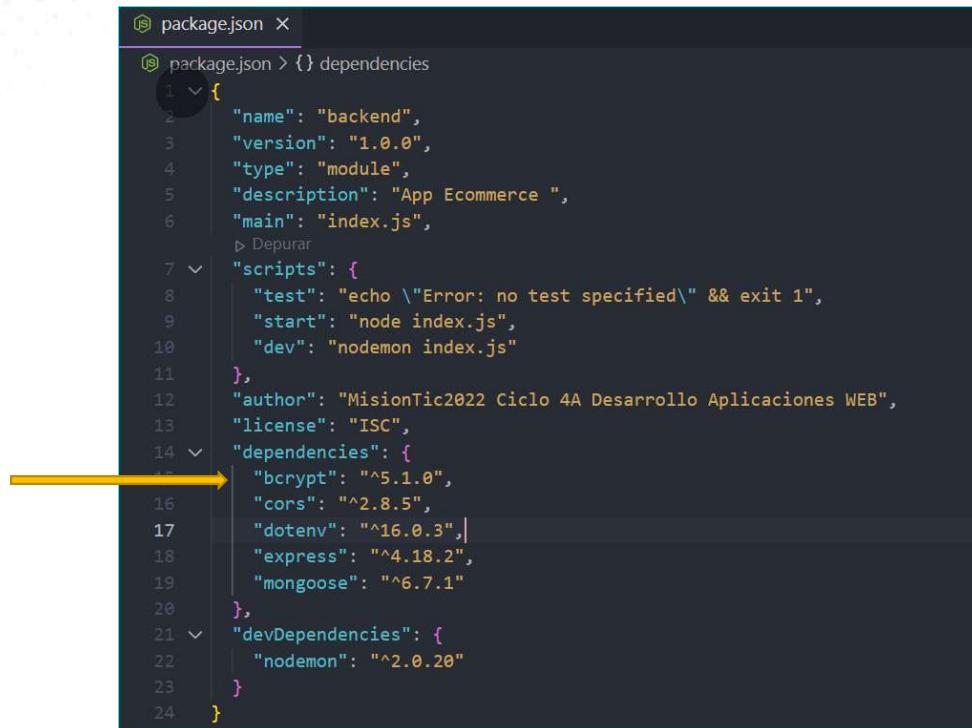
MinTIC

Mision  
TIC 2022

# Modelo de Usuario

Ejecutamos el comando: **npm i bcrypt**, revisamos que estemos dentro de la carpeta de **backend**.

```
PS D:\OneDrive\Trabajo Ivan Castro Ruiz\Docencia\1 - MisionTIC\Documentos UCaldas\Ciclos MinTic2022 UCaldas\Ciclo 4\Deployments\Ejemplo UdeA\mern-ecommerce\backend> npm i bcrypt
```



```
package.json ×
package.json > {} dependencies
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "type": "module",
5    "description": "App Ecommerce",
6    "main": "index.js",
7    "scripts": {
8      "test": "echo \\\"Error: no test specified\\\" && exit 1",
9      "start": "node index.js",
10     "dev": "nodemon index.js"
11   },
12   "author": "MisionTic2022 Ciclo 4A Desarrollo Aplicaciones WEB",
13   "license": "ISC",
14   "dependencies": {
15     "bcrypt": "^5.1.0",
16     "cors": "^2.8.5",
17     "dotenv": "^16.0.3",
18     "express": "^4.18.2",
19     "mongoose": "^6.7.1"
20   },
21   "devDependencies": {
22     "nodemon": "^2.0.20"
23   }
24 }
```



El futuro digital  
es de todos

MinTIC

# Modelo de Usuario

Mision  
TIC 2022

Creamos la carpetas **helper** y dentro el archivo llamado **generarId.js**

The screenshot shows a Visual Studio Code interface. On the left is the Explorer sidebar, which lists several files and folders under 'EDITORES ABIERTOS' (Editors Opened). Two specific paths are highlighted with yellow arrows: 'helper' and 'generarId.js'. The main editor area displays the code for 'generarId.js'.

```
helper > generarId.js > default
1 const generarId = () => {
2   return Date.now().toString(32) + Math.random().toString(32).substring(2);
3 }
4
5 export default generarId;
```

## generarId.js

```
const generarId = () => {
  return Date.now().toString(32) + Math.random().toString(32).substring(2);
}

export default generarId;
```



# Modelo de Producto

```
import mongoose from 'mongoose';

const productoSchema = new mongoose.Schema({
    nombre: {
        type: 'string',
        required: true,
        trim: true
    },
    description: {
        type: String,
        required: true,
        trim: true
    },
    precio: {
        type: Number,
        required: true,
        trim: true
    },
    image: {
        url: String,
        public_id: String,
    },
    stock: {
        type: Number,
        required: true,
        trim: true
    }
})

export default mongoose.model('Producto', productoSchema);
```



Referencia con la colección de Usuario.

Aquí vamos a guardar los artículos por cada venta.

# Modelo de Venta

```
import mongoose from 'mongoose';

const ventaShema = mongoose.Schema(
  {
    cliente: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Usuario"
    },
    articulos: {
      type: Array,
      required: true
    },
    confirmado: {
      type: Boolean,
      default: false
    },
    {
      timestamps: true,
    }
  }
);

const Venta = mongoose.model('Venta', ventaShema);
export default Venta;
```



El futuro digital  
es de todos

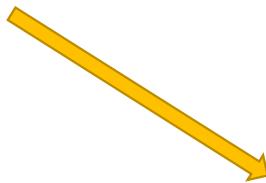
MinTIC

Misión  
TIC 2022

# Gestión Modelo de Usuario

En el archivo **usuarioRoutes.js**, creamos la siguiente ruta:

**Post** por que  
vamos a guardar  
un documento  
en la colección  
de Usuario.



```
import express from 'express';
import {
    prueba,
    registrar, ←
    confirmar ←
} from '../controllers/usuarioController.js';

const router = express.Router();

// Rutas Publicas
router.get('/prueba', prueba);
router.post('/', registrar );
router.get('/confirmar/:token', confirmar); ←

export default router;
```

**Get** Por que  
vamos a enviar  
un email de  
confirmación.



# Gestión Modelo de Usuario

En el archivo **usuarioController.js**, creamos la siguiente middleware (**registrar**):

```
1 < import Usuario from "../models/Usuario.js"; ←
2 < import emailRegistro from "../helper/emailRegistro.js"; ←
3
4 > const prueba = (req, res)=>{...
5
6
7 > const registrar = async (req, res)=>{
8
8   const { nombre, email, password, telefono, direccion, web } = req.body;
9
10  // Validar usuario duplicado
11  // findOne busca por los diferentes atributos de la colección
12  const existeUsuario = await Usuario.findOne({email});
13
14  if(existeUsuario){
15    const error = new Error("Usuario ya registrado");
16    return res.status(400).json({msg: error.message});
17  }
18
19  try {
20
21    const usuario = new Usuario(req.body);
22    const usuarioGuardado = await usuario.save();
23
24    {
25      // Enviar el email
26      emailRegistro({
27        email,
28        nombre,
29        token: usuarioGuardado.token
30      });
31
32      res.json(usuarioGuardado);
33    }
34
35    } catch (error) {
36      console.error(error.message);
37    };
38  }
39
40 > const confirmar = async (req, res)=>{...
41
42
43 export {
44   prueba,
45   registrar, ←
46   confirmar
47 },
```

En la carpeta de helper  
creamos el archivo  
**emailRegistro.js**.

```
const registrar = async (req, res)=>{
  const { nombre, email, password, telefono, direccion, web } = req.body;
  // Validar usuario duplicado
  // findOne busca por los diferentes atributos de la colección
  const existeUsuario = await Usuario.findOne({email}); ←
  if(existeUsuario){
    const error = new Error("Usuario ya registrado");
    return res.status(400).json({msg: error.message});
  }
  try {
    const usuario = new Usuario(req.body);
    const usuarioGuardado = await usuario.save(); ←
    // Enviar el email
    emailRegistro({
      email,
      nombre,
      token: usuarioGuardado.token
    });
    res.json(usuarioGuardado);
  } catch (error) {
    console.error(error.message);
  };
};
```

Busca en la colección  
de Usuario por email.

Guarda el Usuario en la  
colección.

Para agregar seguridad  
a nuestra aplicación.



# Gestión Modelo de Usuario

En el archivo **emailRegistro.js**, vamos a configurar el envío de un mensaje de confirmación:

Ejecutamos el comando: **npm i nodemailer**, revisamos que estemos dentro de la carpeta de **backend**.

PS D:\OneDrive\Trabajo Ivan Castro Ruiz\Docencia\1 - MisionTIC\Documentos UCaldas\Ciclos MinTic2022 UCaldas\Ciclo 4\Deployments\Ejemplo UdeA\mern-e-commerce\backend> **npm i nodemailer**

```
package.json > {} dependencies
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "type": "module",
5    "description": "App Ecommerce",
6    "main": "index.js",
7    > Depurar
8    "scripts": {
9      "test": "echo \\"$Error: no test specified\\" && exit 1",
10     "start": "node index.js",
11     "dev": "nodemon index.js"
12   },
13   "author": "MisionTic2022 Ciclo 4A Desarrollo Aplicaciones WEB",
14   "license": "ISC",
15   "dependencies": {
16     "bcrypt": "^5.1.0",
17     "cors": "^2.8.5",
18     "dotenv": "^16.0.3",
19     "express": "^4.18.2",
20     "mongoose": "^6.7.1",
21     "nodemailer": "^6.8.0" ←
22   },
23   "devDependencies": {
24     "nodemon": "^2.0.20"
25 }
```

**Nodemailer** es un módulo para aplicaciones Node.js que permite enviar correos electrónicos de forma sencilla. El proyecto comenzó en 2010 cuando no había una opción sensata para enviar mensajes de correo electrónico, hoy en día es la solución a la que recurren la mayoría de los usuarios de Node.js de forma predeterminada



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Misión  
TIC 2022

Vamos a usar esta pagina para gestionar los correos, debemos loguearnos yo creo el usuario con el correo de gmail:

<https://mailtrap.io/>

The screenshot shows the Mailtrap homepage. At the top, there's a banner with the text "Stand with Ukraine" and a "Donate to support" button. Below the banner, the main heading is "Email Sandbox Service". It features three main bullet points: "Capture SMTP traffic from staging and dev environments", "Automate test flows and scenarios with flexible API", and "Analyze email content for spam score and validate HTML/CSS". There are also links for "Sign Up For Free" and "Log In". Below the main heading, there are icons for various frameworks: Ruby on Rails, Python, .NET, Symfony, and Laravel. A small screenshot of the Mailtrap interface is shown at the bottom left, and a cookie consent banner is at the bottom right.

The screenshot shows the Mailtrap login page. It has three sign-in options: "Use Google account" (highlighted with a yellow arrow), "Use GitHub account", and "Use Office 365 account". Below these options is a "Next" button. To the right of the form, there's an illustration of a computer monitor displaying an envelope and a keyboard. A note below the illustration says "Set up a new inbox for each test server, so that it is clear where every email was sent from." At the bottom of the page, there's a cookie consent banner.



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Misión  
TIC 2022

The screenshot shows the Mailtrap.io home interface. On the left, there's a sidebar with links like Home, Email API (new), Sandbox, Billing, User Management, API, and Account Settings. The main area has two main sections: 'Sandbox' (Test Emails on Staging) and 'Email API' (Send Emails on Production). Each section has several sub-links: 'Fake SMTP Server', 'QA Automation', 'Preview and validate HTML/CSS', 'Inbox email address', 'Setup Inbox' (highlighted with a yellow arrow), 'Setup Domain', 'SMTP Service, API/SDK', 'SPF, DKIM, DMARC, Dedicated IP', 'Deliverability Alerts', and 'Email content and logs'. A 'Book a Demo' button is also visible.

The screenshot shows the Mailtrap.io inbox interface. The sidebar includes Home, Email API (new), Sandbox, Inboxes (highlighted with a yellow arrow), Billing, User Management, API, and Account Settings. The main area shows an 'Inboxes > My Inbox' view with a search bar and a 'How it works' section. It features a 'My Inbox' summary with SMTP Settings, Email Address, Auto Forward, Manual Forward, and Team Members. Below this is a 'SMTP / POP3' section with a 'Reset Credentials' link. A 'Show Credentials' dropdown is open, revealing a code snippet for Nodemailer configuration. A large orange curly brace groups this code block. A yellow arrow points from the 'Nodemailer' dropdown in the sidebar to this code block. A blue box at the bottom right contains the text: 'Estos datos los guardamos en variables de entorno.'

```
var transport = nodemailer.createTransport({  
  host: "smtp.mailtrap.io",  
  port: 2525,  
  auth: {  
    user: "0d362bbe57dce4",  
    pass: "cfaae7dd1a76d"  
  }  
});
```

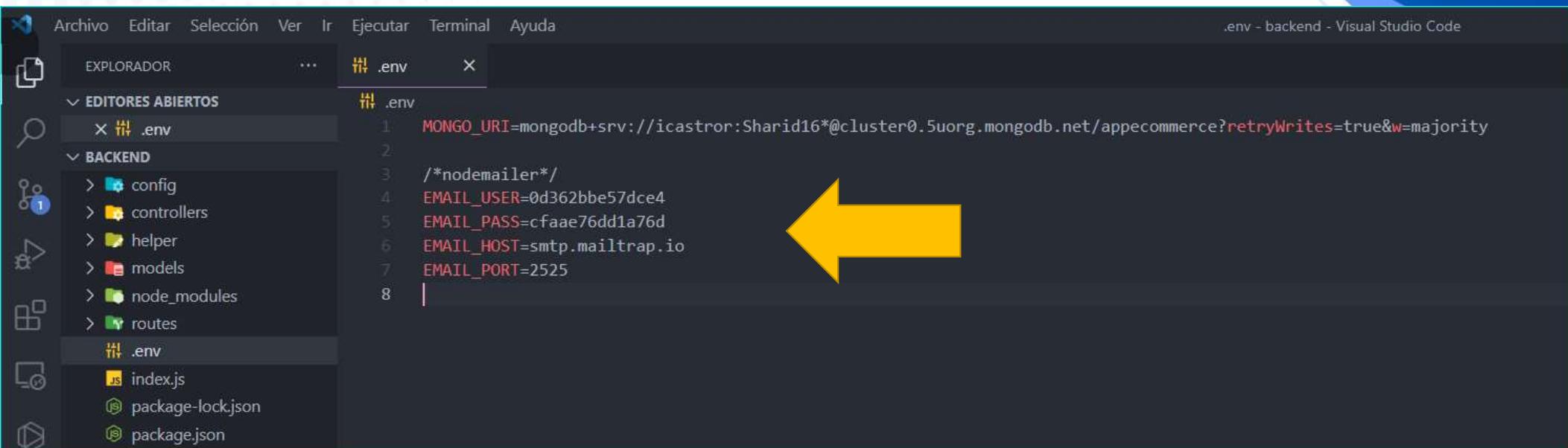


El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Mision  
TIC 2022



.env - backend - Visual Studio Code

Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda

EXPLORADOR ... .env X

EDITORES ABIERTOS .env X

BACKEND

- > config
- > controllers
- > helper
- > models
- > node\_modules
- > routes
- .env
- index.js
- package-lock.json
- package.json

```
1 MONGO_URI=mongodb+srv://icastror:Sharid16*@cluster0.5uorg.mongodb.net/appecommerce?retryWrites=true&w=majority
2
3 /*nodemailer*/
4 EMAIL_USER=0d362bbe57dce4
5 EMAIL_PASS=cfaae76dd1a76d
6 EMAIL_HOST=smtp.mailtrap.io
7 EMAIL_PORT=2525
8 |
```



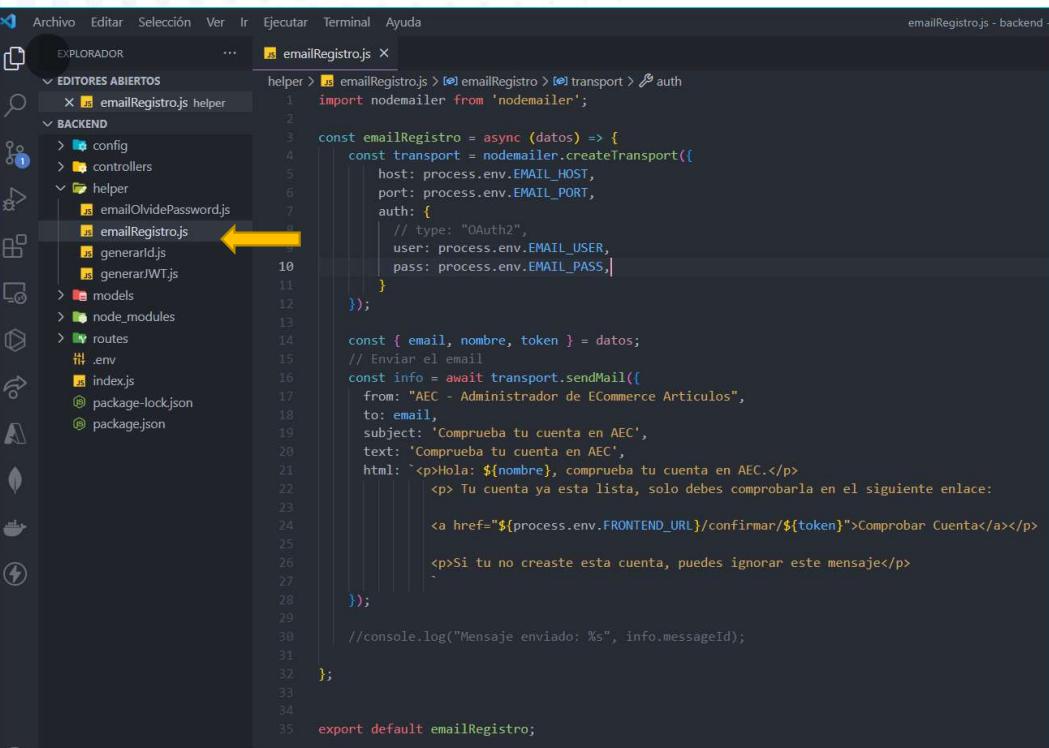
El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Mision  
TIC 2022

En el archivo **emailRegistro.js**, vamos a configurar el envío de un mensaje de confirmación:



The screenshot shows a code editor interface with a dark theme. On the left is a sidebar labeled 'EXPLORADOR' containing a tree view of files and folders. A yellow arrow points to the file 'emailRegistro.js' under the 'helper' folder. The main area displays the content of 'emailRegistro.js'. The code is written in JavaScript and uses the nodemailer library to send an email. It defines an asynchronous function 'emailRegistro' that creates a transport object and sends an email to a user with a confirmation link.

```
import nodemailer from 'nodemailer';

const emailRegistro = async (datos) => {
  const transport = nodemailer.createTransport({
    host: process.env.EMAIL_HOST,
    port: process.env.EMAIL_PORT,
    auth: {
      type: "OAuth2",
      user: process.env.EMAIL_USER,
      pass: process.env.EMAIL_PASS,
    }
  });

  const { email, nombre, token } = datos;
  // Enviar el email
  const info = await transport.sendMail({
    from: "AEC - Administrador de ECommerce Articulos",
    to: email,
    subject: 'Comprueba tu cuenta en AEC',
    text: 'Comprueba tu cuenta en AEC',
    html: `<p>Hola: ${nombre}, comprueba tu cuenta en AEC.</p>
           <p>Tu cuenta ya esta lista, solo debes comprobarla en el siguiente enlace:<br/>
           <a href="${process.env.FRONTEND_URL}/confirmar/${token}">Comprobar Cuenta</a></p>
           <p>Si tu no creaste esta cuenta, puedes ignorar este mensaje</p>`;
  });
  //console.log("Mensaje enviado: %s", info.messageId);
};

export default emailRegistro;
```

```
import nodemailer from 'nodemailer';

const emailRegistro = async (datos) => {
  const transport = nodemailer.createTransport({
    host: process.env.EMAIL_HOST,
    port: process.env.EMAIL_PORT,
    auth: {
      type: "OAuth2",
      user: process.env.EMAIL_USER,
      pass: process.env.EMAIL_PASS,
    }
  });

  const { email, nombre, token } = datos;
  // Enviar el email
  const info = await transport.sendMail({
    from: "AEC - Administrador de ECommerce Articulos",
    to: email,
    subject: 'Comprueba tu cuenta en AEC',
    text: 'Comprueba tu cuenta en AEC',
    html: `<p>Hola: ${nombre}, comprueba tu cuenta en AEC.</p>
           <p>Tu cuenta ya esta lista, solo debes comprobarla en el siguiente enlace:<br/>
           <a href="${process.env.FRONTEND_URL}/confirmar/${token}">Comprobar Cuenta</a></p>
           <p>Si tu no creaste esta cuenta, puedes ignorar este mensaje</p>`;
  });
  //console.log("Mensaje enviado: %s", info.messageId);
};

export default emailRegistro;
```



El futuro digital  
es de todos

MinTIC

Misión  
TIC 2022

# Gestión Modelo de Usuario

En el archivo **usuarioController.js**, creamos la siguiente middleware (**confirmar**):

```
controllers > usuarioController.js > confirmar
1 import Usuario from "../models/Usuario.js";
2 import emailRegistro from "../helper/emailRegistro.js";
3
4 > const prueba = (req, res)=>{ ...
5
6
7 > const registrar = async (req, res)=>{ ...
8
9
10 > const confirmar = async (req, res)=>{
11     // req.params para leer datos de la URL, en este caso token por que asi lo definimos en la ruta
12     const { token } = req.params;
13     const usuarioConfirmar = await Usuario.findOne({token});
14     // console.log(usuarioConfirmar);
15     // console.log(token);
16     if(!usuarioConfirmar){
17         const error = new Error("Token no valido");
18         // console.log("Token no valido");
19         return res.status(404).json({msg: error.message});
20     };
21     try {
22         usuarioConfirmar.token = null;
23         usuarioConfirmar.confirmado = true;
24         await usuarioConfirmar.save();
25         res.json({
26             msg: "Usuario confirmado correctamente"
27         });
28         // console.log("Usuario confirmado correctamente");
29     } catch (error) {
30         console.error(error.message);
31     }
32
33     export {
34         prueba,
35         registrar,
36         confirmar
37     };
38 }
```

```
const confirmar = async (req, res)=>{
    // req.params para leer datos de la URL, en este caso token por que asi lo definimos en la ruta
    const { token } = req.params;
    const usuarioConfirmar = await Usuario.findOne({token}); ← Busca en la colección de Usuario por Token.
    // console.log(usuarioConfirmar);
    // console.log(token);
    if(!usuarioConfirmar){
        const error = new Error("Token no valido");
        // console.log("Token no valido");
        return res.status(404).json({msg: error.message});
    };
    try {
        usuarioConfirmar.token = null;
        usuarioConfirmar.confirmado = true; ← Modificamos la propiedad token y confirmado, ya que si es un usuario valido.
        await usuarioConfirmar.save(); ← Guarda en la colección de Usuario los cambios realizados.
        res.json({
            msg: "Usuario confirmado correctamente"
        });
        // console.log("Usuario confirmado correctamente");
    } catch (error) {
        console.error(error.message);
    }
};
```



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Mision  
TIC 2022

Probamos en POSTMAN la implementación realizada:

The screenshot shows the Postman application interface. At the top, there are navigation tabs: Home, Workspaces, API Network, and Explore. On the right side, there are buttons for Invite, Settings, and Upgrade. Below the header, there's a search bar labeled "Search Postman". The main workspace shows a collection named "http://localhost:4000/api/usuarios". A specific POST request is selected, with the URL "http://localhost:4000/api/usuarios" in the request field. The "Body" tab is active, indicated by a green dot. To the right of the body tab, there are several options: Params, Authorization, Headers (9), Body (radio buttons: none, form-data, x-www-form-urlencoded, raw, binary, GraphQL, JSON), Tests, and Settings. The "raw" option is selected, and "JSON" is chosen from a dropdown menu. In the body editor, there is a JSON object with the following content:

```
1 {  
2     "nombre": "Ivan Castro Ruiz",  
3     "email": "icastro@hotmail.com",  
4     "password": "12345",  
5     "telefono": "3003162985",  
6     "direccion": "Calle 18B # 17F - 24",  
7     "web": "https://github.com/IvanCastroRuiz"  
8 }
```

A large orange curly brace groups the entire JSON object. To the right of the body editor, there are buttons for Save, Edit, and Delete. At the bottom right of the body editor, there are "Cookies" and "Beautify" buttons. A blue arrow points from the "Send" button to the "Beautify" button. Another blue arrow points from the "Send" button to the "Cookies" button.

Configuramos las opciones y le damos enviar.



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Mision  
TIC 2022

Probamos en POSTMAN la implementación realizada, de registro de un **Usuario**:

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:4000/api/usuarios`. The response status is 200 OK, and the response body is a JSON object containing user information and a token.

**Body (Pretty)**

```
1   {
2     "nombre": "Ivan Castro Ruiz",
3     "email": "icastroi@hotmail.com",
4     "password": "12345",
5     "telefono": "3003162985",
6     "direccion": "Calle 18B # 17F - 24",
7     "web": "https://github.com/IvanCastroRuiz"
8 }
```

**Response (JSON)**

```
1   {
2     "nombre": "Ivan Castro Ruiz",
3     "email": "icastroi@hotmail.com",
4     "password": "$2b$10sd07Yp1.t4APTT8TrQybk./@zc6UU5URtD8U4n0uFq3RlIW1uFI02",
5     "telefono": "3003162985",
6     "direccion": "Calle 18B # 17F - 24",
7     "web": "https://github.com/IvanCastroRuiz",
8     "token": "4gh7ruvc5cos17p4aj8",
9     "confirmado": false,
10    "rol": null,
11    "_id": "636863a6dd689e3d2e5d1294",
12    "__v": 0
13 }
```

A yellow arrow points from the text box to the URL in the Postman header. Two orange curly braces highlight the user input fields in the body and the generated token in the response.

Esta ruta fue la que definimos en el usuarioRoutes.js



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Mision  
TIC 2022

Validamos la BD de MongoDB Atlas:

The screenshot shows the MongoDB Atlas interface. On the left, the deployment sidebar lists 'Database' (selected), 'Data Lake' (PREVIEW), 'DATA SERVICES' (Triggers, Data API, Data Federation), and 'SECURITY' (Database Access, Network Access, Advanced). The 'Atlas' tab is selected in the top navigation bar. In the center, the 'appecommerce.usuarios' database is selected. The 'Find' section shows a query result for '\_id: ObjectId('636863a6dd689e3d2e5d1294')'. A callout box contains the text: 'Podemos notar que mongoose realizo la creación de la BD y la colección de Usuarios.' Below the interface, a footer bar includes links for System Status, Status, Terms, Privacy, Atlas Blog, Contact Sales, and the URL www.misiontic2022.gov.co.

System Status: All Good  
©2022 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

Podemos notar que mongoose realizo la creación de la BD y la colección de Usuarios.

```
_id: ObjectId('636863a6dd689e3d2e5d1294')
nombre: "Ivan Castro Ruiz"
email: "icastro@hotmail.com"
password: "$2b$10$07yyp1.t4APTT8TrQybK./0zc6Uu5URtD8U4n0uFq3RlW1uFID2"
telefono: "3003162985"
direccion: "Calle 18B # 17F - 24"
web: "https://github.com/IvanCastroRuiz"
token: "1gh7ruevc5cosl7p4aj8"
confirmado: false
rol: null
__v: 0
```

<https://www.youtube.com/watch?v=DJNAfQ3IgkI>

<https://www.youtube.com/watch?v=68GYqYeC6MQ>



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Mision  
TIC 2022

Probamos en **POSTMAN** y **NODEMAILER** la implementación realizada, para la confirmación del usuario:

The screenshot shows a browser window displaying the Mailtrap.io inbox at [mailtrap.io/inboxes/1917548/messages/3102708326](https://mailtrap.io/inboxes/1917548/messages/3102708326). The inbox lists an email from <> to <icastror@hotmail.com> sent 18 minutes ago. The subject of the email is "Comprueba tu cuenta en AEC". The message body contains a welcome message and a link to "Comprobar Cuenta". The interface includes a sidebar with options like Home, Email API, Sandbox, Inboxes (selected), Billing, User Management, API, and Account Settings. There are also tabs for HTML, HTML Source, Text, Raw, Spam Analysis, HTML Check, and Tech Info. Two yellow arrows are overlaid on the screenshot: one pointing up from the sidebar to the inbox header, and another pointing right from the sidebar to the message preview area.

El envío de email hasta esta altura esta configurado para desarrollo posteriormente se pase a producción se tiene que crear una correo por ejemplo gmail.

[www.misiontic2022.gov.co](http://www.misiontic2022.gov.co)

<https://www.youtube.com/watch?v=4nkzK34u1ks>



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Usuario

Mision  
TIC2022

Probamos en **POSTMAN** y **NODEMAILER** la implementación realizada, para la confirmación del usuario.

Extraemos de la BD el token generado.

Esta ruta fue la que definimos  
en el **usuarioRoutes.js**

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Project 0' selected. Under 'Database', 'PREVIEW' is chosen. In the main area, there's a list of namespaces: 'G69', 'api\_rest\_blog', 'appcommerce', and 'usuarios'. The 'usuarios' namespace is expanded, showing a single document. The document details are as follows:

```
_id: ObjectId('636863a6dd689e3d2e5d1294')
nombre: "Ivan Castro Ruiz"
email: "icastro@gmail.com"
password: "$2b$10$0d07Yp1.14APTT8TrQybk./0zc6Uu5URtDBU4n0uFq3RlWluFID2"
telefono: "3083162985"
direccion: "Calle 18B # 17F - 24"
web: "https://github.com/IvanCastroRuiz"
token: "igh7truevc5cosl7p4aj8"
confirmado: false
rol: null
__v: 0
```

The screenshot shows the Postman application. A GET request is made to the URL `http://localhost:4000/api/usuarios/confirmar/igh7truevc5cosl7p4aj8`. The response body is a JSON object:

```
{
  "msg": "Usuario confirmado correctamente"
}
```

Con esta confirmación estamos seguro que no es un bot  
el que esta tratando de crear cuentas en nuestra  
aplicación.



El futuro digital  
es de todos

MinTIC

## Gestión Modelo de Usuario

Misión  
TIC 2022

La gestión del modelo de **Usuario** la retomamos posteriormente creamos la rutas para la gestión del modelo de **Producto**.

**Pendiente:** perfil, auntenticar, olvidePassword, comprobarToken, nuevoPassword, usuarioRegistrados, actualizarPerfil, actualizarPassword



El futuro digital  
es de todos

MinTIC

Misión  
TIC 2022

# Gestión Modelo de Producto

Para el manejo de las imágenes vamos a utilizar **Cloudinary**:

**Cloudinary** es un servicio en la nube que nos permitirá almacenar nuestras imágenes en su servidor y disponer de ellas mediante una url personalizada.

<https://cloudinary.com/home-102622>

The Most Powerful Media API and Products

Cut the complexity involved in optimizing, editing, and managing media for your app, e-commerce store, marketplace, or website.

GET STARTED

Cloudinary products used by: AWS, Whole Foods, Virgin, Etsy, Tesla, Sony, Atlassian, Disney

Voy a crear una cuenta con mi usuario de **Gmail**.

SIGN UP TO CLOUDINARY

SIGN UP WITH EMAIL

Or

SIGN UP WITH GOOGLE

SIGN UP WITH GITHUB

Already have an account? [Log In](#)

Get Immediate Value

- Improve Visual Experiences
- Boost Productivity and Agility
- Enhance Performance

Cloudinary Products Solutions Developers Company Contact Us

Programmable Media Why Cloudinary Getting Started About Us Technical Support

DAM Video API Documentation Customers Contact Sales

Demos E-Commerce & Retail SDK Partners Education and Training

Pricing Media & Entertainment Add-Ons Events

f t in



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Producto

Misión  
TIC 2022

Para el manejo de las imágenes vamos a utilizar **Cloudinary**:

The screenshot shows the Cloudinary media library interface. At the top, there's a navigation bar with tabs like 'Tablero', 'Mediateca' (selected), 'Transformaciones', 'Informes', and 'Complementos'. A banner at the top says '¡Acabamos de lanzar una nueva experiencia de navegación!' with a 'Pruébalo ahora!' button. The main area is titled 'Mediateca' and shows a grid of six images. The images are labeled 'cid-muestra-5', 'cid-muestra-4', 'cid-muestra-3', 'cid-muestra-2', 'cid-muestra', and 'muestra'. Each image has a 'Nuevo' badge. Below the images, there's a message: 'Seleccione un elemento en la cuadrícula para obtener una vista previa'. On the left sidebar, there are sections for 'Carpetas (1)' and 'Activos (6)', with 'cid-muestra' being the active folder.

The screenshot shows the Cloudinary dashboard. At the top, it says 'Bienvenido a su panel de Cloudinary' and 'Encuentre toda la información que necesita sobre su plan, uso y cómo aprovechar al máximo las funciones de Cloudinary, o incluso influir en ellas.' Below this, there's a section for 'detalles de la cuenta' with fields for 'Nombre de la nube' (digi1aoxog), 'Clave API' (137277985668697), and 'Secreto de la API' (redacted). An orange bracket groups these three fields. Another bracket groups the 'Variable de entorno de la API' field (CLOUDINARY\_URL=cloudinary://\*\*\*\*\*@digi1aoxog) and the 'Uso actual del plan' section. The 'Uso actual del plan' section shows a credit usage of 25% (de 25%) and a status of 'Libre'. To the right, there's a large callout box with orange text: 'Estos datos los vamos a necesitar en el archivo de variables de entorno .env'. Below this, three environment variable names are listed in green: CLOUDINARY\_API\_KEY, CLOUDINARY\_API\_SECRET, and CLOUDINARY\_NAME.



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Producto

Misión  
TIC 2022

Actualizamos el archivo .env con la información de:

detalles de la cuenta

<b>CLOUDINARY_NAME</b> Nombre de la nube dgi1aoxog	<b>CLOUDINARY_API_KEY</b> Clave API 137277985668697	<b>CLOUDINARY_API_SECRET</b> Secreto de la API ***** **
--	---	--

Recuerda usar tus datos de acceso de Cloudinary

```
1 MONGO_URI=mongodb+srv://icastror:Sharid16*@cluster0.5uorg.mongodb.net/appecommerce?retryWrites=true&  
w=majority  
2  
3 /*nodemailer*/  
4 EMAIL_USER=0d362bbe57dce4  
5 EMAIL_PASS=cfaae76dd1a76d  
6 EMAIL_HOST=smtp.mailtrap.io  
7 EMAIL_PORT=2525  
8  
9 /*Manejo de las imágenes - CLOUDINARY*/  
10 CLOUDINARY_NAME=dgi1aoxog  
11 CLOUDINARY_KEY=137277985668697  
12 CLOUDINARY_SECRET=Z0vUAFrz_BIrWJCKaxMISjfMryw  
13
```



# Gestión Modelo de Producto

En el archivo **productoRoutes.js**, creamos la siguiente ruta:

```
import express from 'express';

{ import {
    prueba,
    deleteProductos,
    getProducto,
    getProductos,
    createProductos,
    updateProductos
} from '../controllers/productoController.js'; }

const router = express.Router();

// Rutas Publicas
router.get('/prueba', prueba);

{ // Rutas Gestión Producto
    router.get('/productos', getProductos);
    router.get('/productos/:id', getProducto);
    router.post('/productos', createProductos);
    router.put('/productos/:id', updateProductos);
    router.delete('/productos/:id', deleteProductos);
}

export default router;
```



# Gestión Modelo de Producto

Ejecutamos el comando: **npm i clodinary express-fileupload fs-extra**, revisamos que estemos dentro de la carpeta de backend.

PS D:\OneDrive\Trabajo Ivan Castro Ruiz\Docencia\1 - MisionTIC\Documentos UCaldas\Ciclos MinTic2022 UCaldas\Ciclo 4\Deployments\Ejemplo UdeA\mern-e-commerce\backend> **npm i clodinary express-fileupload fs-extra**

```
package.json M X
package.json > {} dependencies
1 {
2   "name": "backend",
3   "version": "1.0.0",
4   "type": "module",
5   "description": "App Ecommerce",
6   "main": "index.js",
7   > Depurar
8   "scripts": {
9     "test": "echo \\\"Error: no test specified\\\" && exit 1",
10    "start": "node index.js",
11    "dev": "nodemon index.js"
12  },
13  "author": "MisionTic2022 Ciclo 4A Desarrollo Aplicaciones WEB",
14  "license": "ISC",
15  "dependencies": {
16    "bcrypt": "^5.1.0",
17    "clodinary": "1.32.0",
18    "cors": "^2.8.5",
19    "dotenv": "16.0.3",
20    "express": "4.18.2",
21    "express-fileupload": "1.4.0",
22    "fs-extra": "10.1.0",
23    "mongoose": "6.7.1",
24    "nodemailer": "6.8.0"
25  },
26  "devDependencies": {
27    "nodemon": "2.0.20"
28 }
```

Realizamos el siguiente cambio en el archivo **index.js**

```
index.js > ...
1 // https://expressjs.com/es/
2 import express from "express";
3 import dotenv from 'dotenv';
4 import cors from 'cors';
5 import fileupload from 'express-fileupload';
6
7 import conectarDB from './config/db.js';
8 import usuarioRoutes from './routes/usuarioRoutes.js';
9 import productoRoutes from './routes/productoRoutes.js';
10 import ventaRoutes from './routes/ventaRoutes.js';
11
12 const PORT = process.env.PORT || 4000;
13 dotenv.config();
14
15 // Se le agrega toda la funcionalidad del servidor de express
16 const app = express();
17 app.use(express.json());
18
19 app.use(fileupload({
20   useTempFiles: true,
21   tempFileDir: './files'
22 }));
23
24
```

Como vamos a trabajar con el manejo de archivos se hace necesario instalar estas dependencias y hacer las configuraciones indicadas.



# Gestión Modelo de Producto

En el archivo **productoController.js**, creamos los siguiente endpoint:

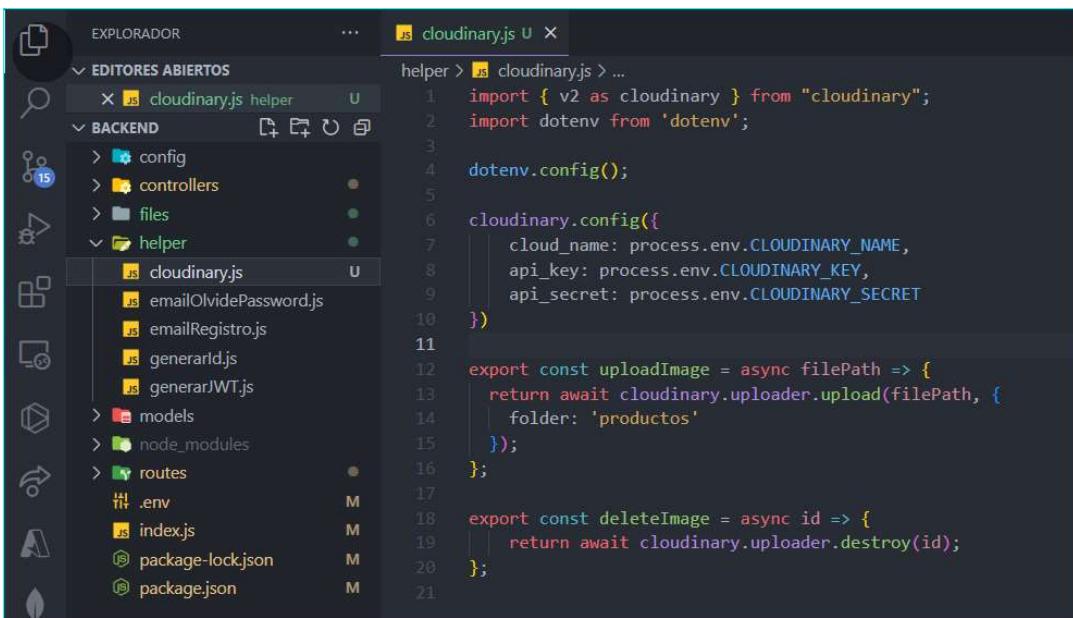
```
controllers > js productoController.js > (e) createProductos
  1 | import Producto from "../models/Producto.js";
  2 | import {
  3 |   uploadImage,
  4 |   deleteImage
  5 | } from '../helper/cloudinary.js';
  6 |
  7 |> const prueba = (req, res) => { ... };
  8 |
  9 |
 10 |> const createProductos = async (req, res) => { ... };
 11 |
 12 |> const getProductos = async (req, res) => { ... };
 13 |
 14 |> const updateProductos = async (req, res) => { ... };
 15 |
 16 |> const deleteProductos = async (req, res) => { ... };
 17 |
 18 |> const getProducto = async (req, res) => { ... };
 19 |
 20 |
 21 | export {
 22 |   prueba,
 23 |   getProductos,
 24 |   createProductos,
 25 |   updateProductos,
 26 |   deleteProductos,
 27 |   getProducto
 28 | };

 29 |
```

Importamos el modelo de Productos

Funciones para subir y eliminar en Cloudinary

Creamos la carpetas **helper** y dentro el archivo llamado **cloudinary.js**



The image shows a code editor interface with a sidebar labeled "EXPLORADOR" (Explorer) showing a file tree. The "EDITORES ABIERTOS" (Open Editors) section shows "clouidnary.js" and "helper". The "BACKEND" section shows various files like config, controllers, files, and helper. The "helper" folder contains several files: clouidnary.js, emailOlvidePassword.js, emailRegistro.js, generardId.js, generarJWT.js, models, routes, .env, index.js, package-lock.json, and package.json. The main editor window displays the content of "clouidnary.js".

```
helper > js clouidnary.js U X
helper > js clouidnary.js helper > ...
1  import { v2 as cloudinary } from 'cloudinary';
2  import dotenv from 'dotenv';
3  dotenv.config();
4
5  cloudinary.config({
6    cloud_name: process.env.CLOUDINARY_NAME,
7    api_key: process.env.CLOUDINARY_KEY,
8    api_secret: process.env.CLOUDINARY_SECRET
9  })
10
11  export const uploadImage = async filePath => {
12    return await cloudinary.uploader.upload(filePath, {
13      folder: 'productos'
14    });
15  }
16
17
18  export const deleteImage = async id => {
19    return await cloudinary.uploader.destroy(id);
20  }
21
```



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Producto

Misión  
TIC 2022

Creamos la carpetas **helper** y dentro el archivo llamado **Cloudinary.js**

```
import { v2 as cloudinary } from "cloudinary";
import dotenv from 'dotenv';

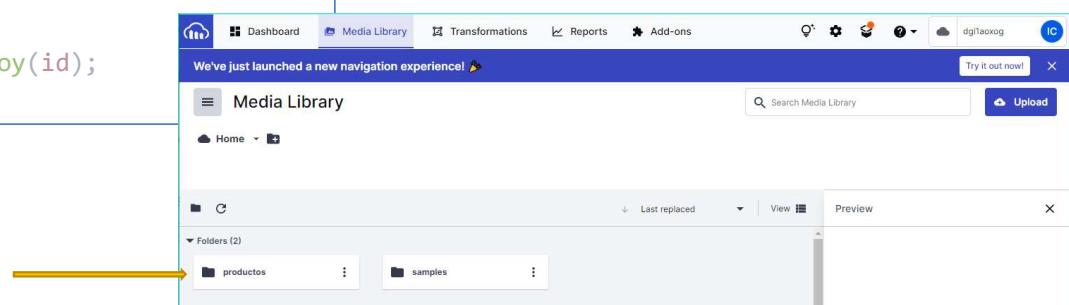
dotenv.config();

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_NAME,
  api_key: process.env.CLOUDINARY_KEY,
  api_secret: process.env.CLOUDINARY_SECRET
})

export const uploadImage = async filePath => {
  return await cloudinary.uploader.upload(filePath, {
    folder: 'productos'
  });
}

export const deleteImage = async id => {
  return await cloudinary.uploader.destroy(id);
}
```

En **Cloudinary**  
Creamos la  
carpeta  
**productos**.





# Gestión Modelo de Producto

En el archivo **productoController.js**, creamos la siguiente middleware (**createProductos**):

```
controllers > js productoController.js > ...
1 import Producto from "../models/Producto.js";
2 import fs from "fs-extra";
3 import {
4   uploadImage,
5   deleteImage
6 } from '../helper/cloudinary.js';
7
8 > const prueba = (req, res) => {
9 };
10
11 > const createProductos = async (req, res) => {
12 };
13
14 > const getProductos = async (req, res) => {
15 };
16
17 > const updateProductos = async (req, res) => {
18 };
19
20 > const deleteProductos = async (req, res) => {
21 };
22
23 > const getProducto = async (req, res) => {
24 };
25
26 export {
27   prueba,
28   getProductos,
29   createProductos,
30   updateProductos,
31   deleteProductos,
32   getProducto
33 };
34
35
36
```

```
const createProductos = async (req, res) => {
  try {
    const { nombre, description, precio, stock } = req.body;
    let image;

    if (req.files.image) {
      const result = await uploadImage(req.files.image.tempFilePath);
      await fs.remove(req.files.image.tempFilePath);
      image = {
        url: result.secure_url,
        public_id: result.public_id,
      };
      console.log(result);
    }

    const Newproducto = new Producto({ nombre, description, precio, image, stock });
    await Newproducto.save();
    return res.json(Newproducto);
  } catch (error) {
    console.log(error);
    return res.status(500).json({ msg: error.message });
  }
};
```



**El futuro digital  
es de todos**

MinTIC

# Gestión Modelo de Producto



Probamos la ruta de creación de un **Producto** en **POSTMAN** y **MongoDB Atlas**:

The screenshot shows the Postman interface with a successful POST request to `http://localhost:4000/api/productos/create`. The request body is set to `form-data` and contains the following fields:

KEY	VALUE
image	dBIVvKhF/cafeol.png
nombre	"Cafe Colombiano"
description	"El mejor cafe del mundo"
precio	3500
stock	50

The response status is `200 OK` with a time of `1466 ms` and a size of `539 B`. The response body is a JSON object representing the created product:

```
1 "nombre": "Cafe Colombiano",
2 "description": "El mejor cafe del mundo",
3 "precio": 3500,
4 "image": {
5     "url": "https://res.cloudinary.com/dsgiaoxog/image/upload/v1667855729/productos/jfirtukejq3qfsgrick7k.jpg",
6     "public_id": "productos/jfirtukejq3qfsgrick7k"
7 },
8 "stock": 50,
9 "_id": "63697565b97b66244687431a",
10 "_v": 0
```

The screenshot shows the Cluster0 interface for the 'appecommerce.products' database. The 'Collections' tab is selected. The left sidebar shows namespaces like 'G69', 'api\_rest\_blog', and 'appecommerce'. An orange arrow points to the 'productos' collection under 'appecommerce'. The right panel displays document details for an item with \_id: ObjectId('63697565bd7b66244607431a').

Cluster0

VERSION 5.0.13 REGION AWS N. Virginia (us-east-1)

DATABASES: 6 COLLECTIONS: 37

+ Create Database

Search Namespaces

G69

api\_rest\_blog

appecommerce

productos

usuarios

cms-tienda-virtual

ecommerceartmus

my-project

appecommerce.products

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 291B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 4KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

INSERT DOCUMENT

FILTER { field: 'value' } OPTIONS Apply Reset

QUERY RESULTS: 1-1 OF 1

{ \_id: ObjectId('63697565bd7b66244607431a')  
nombre: "Cafe Colombiano"  
descripcion: "El mejor cafe del mundo"  
precio: 3500  
image: Object  
url: "https://res.cloudinary.com/dgilaoxog/image/upload/v1667855729/pro  
public\_id: "productos/jfrtukejq3fsgrick7k"  
stock: 50  
\_\_v: 0 }



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Producto

Misión  
TIC 2022

Probamos que la imagen se halla guardado en **Cloudinary**:

Vamos a la carpeta  
productos

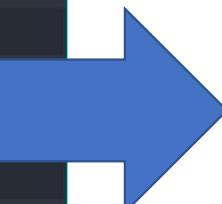
The screenshot shows the Cloudinary interface. At the top, there's a banner with the text "¡Acabamos de lanzar una nueva experiencia de navegación!" and a "¡Pruébalo ahora!" button. Below the banner, the "Mediateca" tab is selected. On the left, there's a sidebar with a "productos" folder icon. The main area displays a grid of three images. Each image is a white coffee cup filled with coffee, sitting on a saucer with coffee beans. The images are labeled with unique IDs: "jfrtukejq3qfsgrick7k", "obxxuvwtpsh9y3ulpbv", and "oigtwjzco4ajbmnpf1x1". Each image has a "products" tag and a file size of "9.17KB" and dimensions of "275 x 183". To the right of the images, there's a sidebar with the folder name "productos", a thumbnail of the images, and sections for "Ubicación" (set to "Hogar"), "Compartido con" (empty), and "Cuota" (empty). A note says "No compartido. Solo los administradores pueden administrar esta carpeta."



# Gestión Modelo de Producto

En el archivo **productoController.js**, creamos la siguiente middleware (**getProductos** y **updateProductos**):

```
controllers > js productoController.js > ...
1 import Producto from "../models/Producto.js";
2 import fs from "fs-extra";
3 import {
4   uploadImage,
5   deleteImage
6 } from '../helper/cloudinary.js';
7
8 > const prueba = (req, res) => {
9 };
10
11 > const createProductos = async (req, res) => {
12 };
13
14 > const getProductos = async (req, res) => {
15 };
16
17 > const updateProductos = async (req, res) => {
18 };
19
20 > const deleteProductos = async (req, res) => {
21 };
22
23 > const getProducto = async (req, res) => {
24 };
25
26 export {
27   prueba,
28   getProductos,
29   createProductos,
30   updateProductos,
31   deleteProductos,
32   getProducto
33 };
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
```



```
const getProductos = async (req, res) => {
  try {
    const productos = await Producto.find();
    res.send(productos);
  } catch (error) {
    console.log(error.message);
    return res.status(500).json({ message: error.message });
  }
};

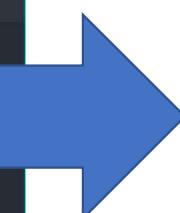
const updateProductos = async (req, res) => {
  try {
    const updatedProduct = await Producto.findByIdAndUpdate(
      req.params.id,
      req.body,
      {
        new: true,
      }
    );
    return res.send(updatedProduct);
  } catch (error) {
    return res.status(500).json({ message: error.message });
  }
};
```



# Gestión Modelo de Producto

En el archivo **productoController.js**, creamos la siguiente middleware (**deleteProductos** y **getProducto**):

```
controllers > js productoController.js > ...
1 import Producto from "../models/Producto.js";
2 import fs from "fs-extra";
3 import {
4   uploadImage,
5   deleteImage
6 } from '../helper/cloudinary.js';
7
8 > const prueba = (req, res) => { ... };
12
13 > const createProductos = async (req, res) => { ... };
37
38 > const getProductos = async (req, res) => { ... };
47
48 > const updateProductos = async (req, res) => { ... };
62
63 > const deleteProductos = async (req, res) => { ... };
81
82 > const getProducto = async (req, res) => { ... };
95
96 export {
97   prueba,
98   getProductos,
99   createProductos,
100  updateProductos,
101  deleteProductos,
102  getProducto
103};
```



```
const deleteProductos = async (req, res) => {
  try {
    const productRemoved = await Producto.findByIdAndDelete(req.params.id);

    if (!productRemoved) {
      const error = new Error("Token no valido");
      return res.sendStatus(404);
    } else {

      if (productRemoved.image.public_id) {
        await deleteImage(productRemoved.image.public_id);
      }
      return res.sendStatus(204);
    }
  } catch (error) {
    return res.status(500).json({ message: error.message });
  }
};

const getProducto = async (req, res) => {
  try {
    const OneProduct = await Producto.findById(req.params.id);

    if (!OneProduct) {
      return res.sendStatus(404);
    } else {
      return res.json(OneProduct);
    }
  } catch (error) {
    return res.status(500).json({ message: error.message });
  }
};
```



El futuro digital  
es de todos

MinTIC

# Gestión Modelo de Producto

Misión  
TIC 2022

## Tarea:

Validar con **POSTMAN** y **MongoDB Atlas** la gestión de las rutas implementadas para la ruta del modelo **Producto**.

**getProductos**, **updateProductos**, **deleteProductos** y **getProducto**