

Confiabilidad, pruebas y análisis de la información

**IDES de desarrollo**

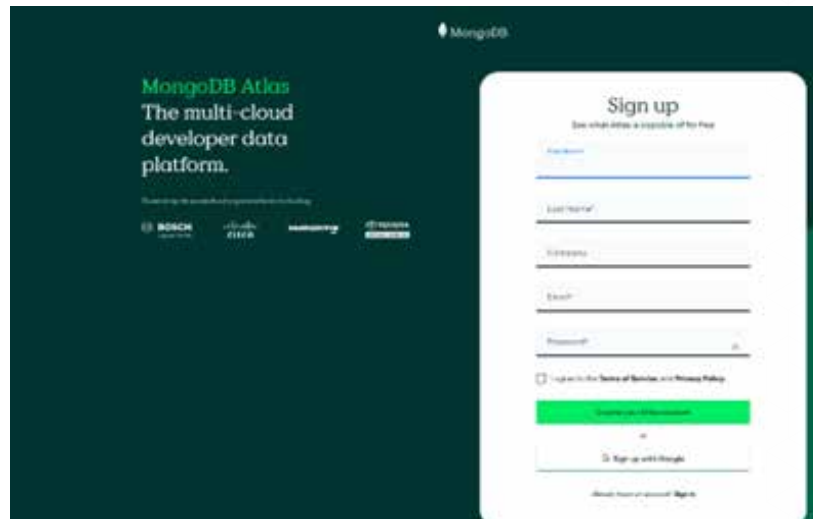
## Configuración del ambiente de Base de datos

La base de datos se realizará con el motor de MongoDB.

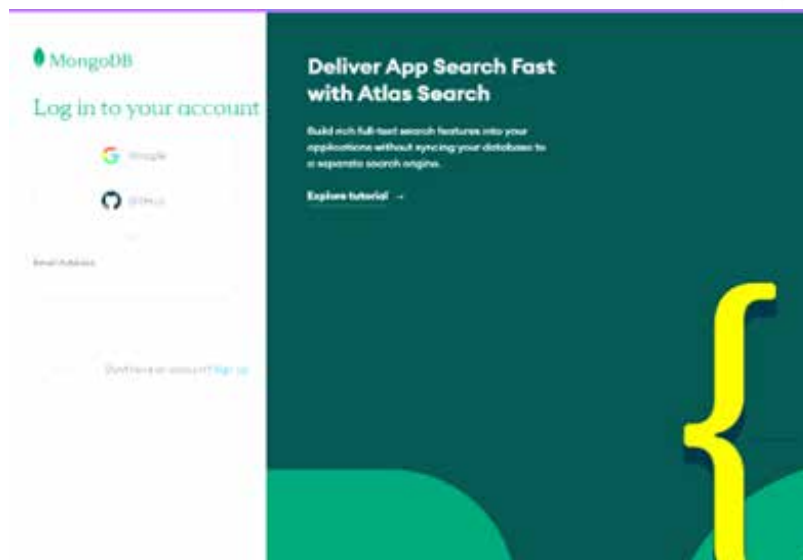
Para esta ocasión, se creará la base de datos con la herramienta Atlas, una herramienta cloud para alojar y gestionar la base de datos de MongoDB.

1. Cree una cuenta en la URL actual para registro: <https://www.mongodb.com/cloud/atlas/register>

Nota: puede crearla incluso con su cuenta de gmail en el botón de la parte final

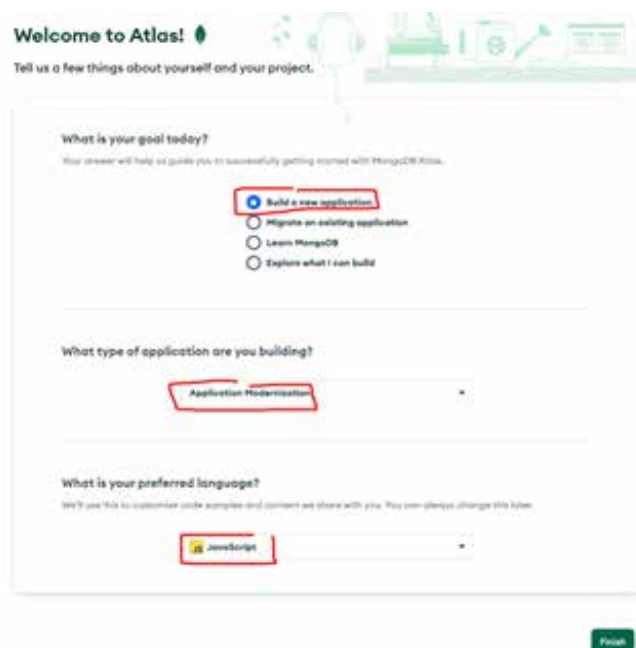


Una vez tenga la cuenta creada, ingrese a la plataforma desde: <https://account.mongodb.com/account/login?>

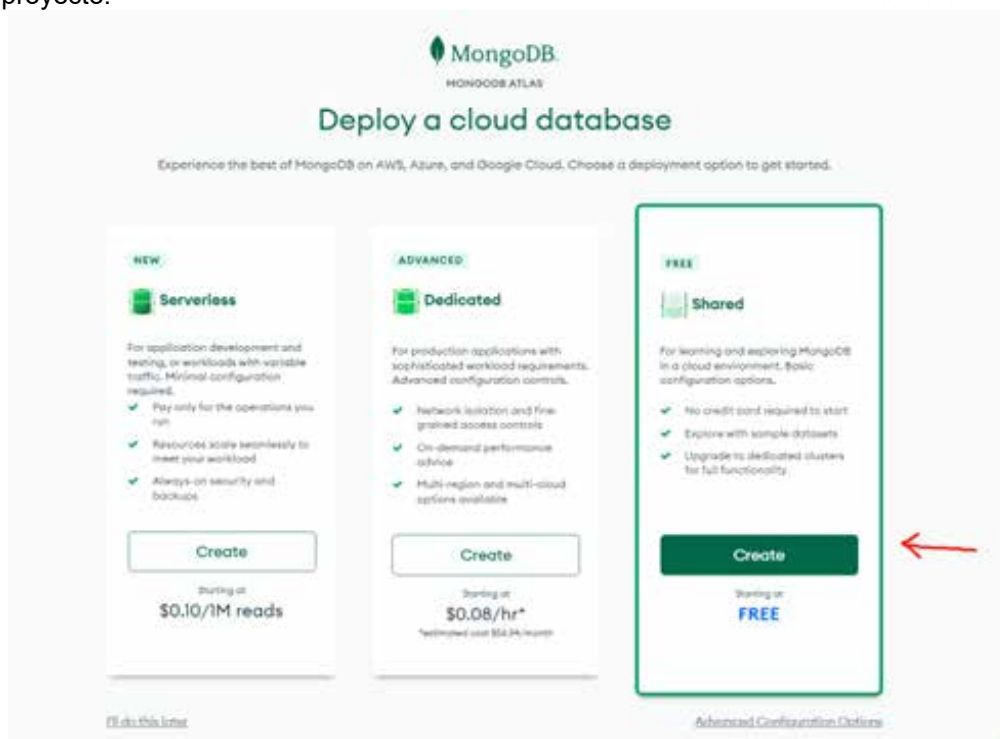


0. Una vez haya ingresado, realizará los siguientes pasos:

Nota: si le llega a aparecer la siguiente ventana, puede seleccionar de manera rápida, las opciones que se muestran en la siguiente imagen:

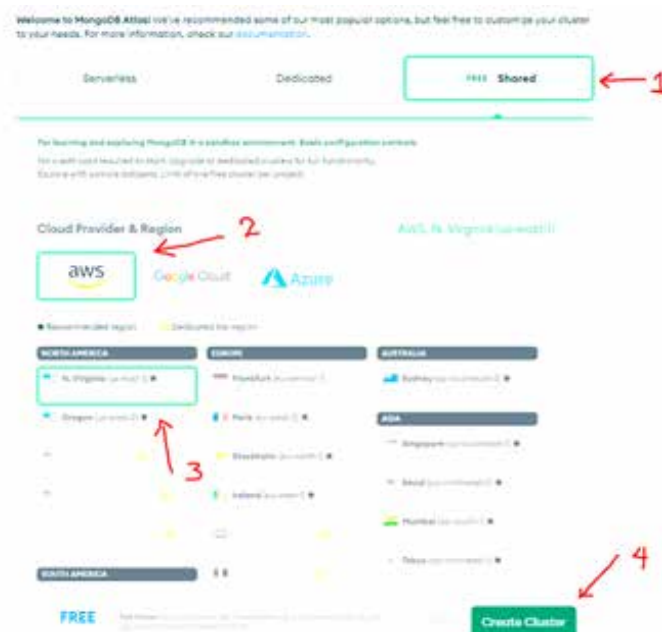


Luego, se mostrará la siguiente ventana, y seleccionará “free” que es para una capa gratuita, bastará para este proyecto.



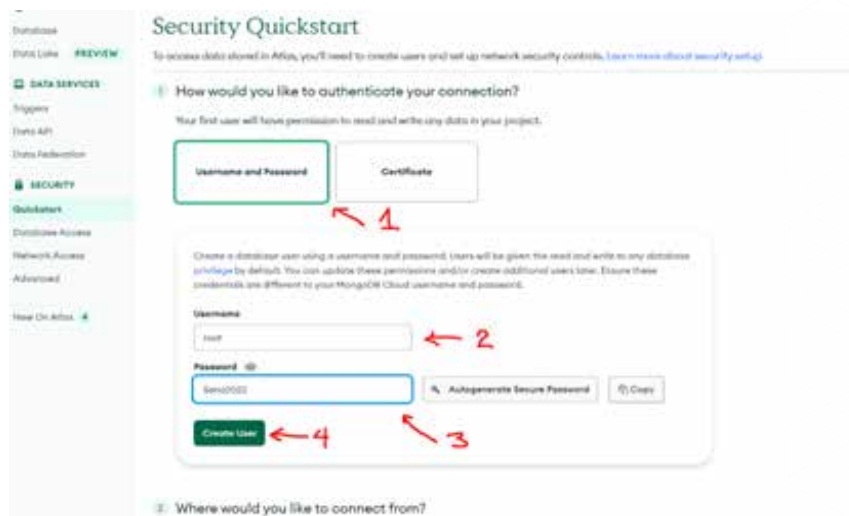
A continuación, seleccione las opciones que se indican a continuación y haga clic en el botón: “create cluster” (crear cluster):

En el 1, seleccione un plan gratuito (compartido); en el 2, seleccione el proveedor de AWS, aunque puede seleccionar el que desee; en el 3, una zona cercana, en este caso “N. Virginia”; por último (4), se crea el clúster con la infraestructura necesaria para su base de datos de MongoDB.



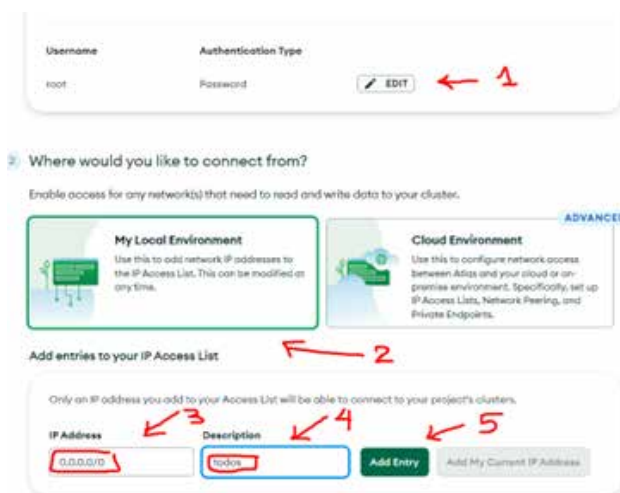
En la siguiente pantalla, se pedirán algunos datos de seguridad, puede asignar el que desee en usuario y contraseña.

Nota: Asegúrese de guardar bien estos datos, los necesitará a futuro.

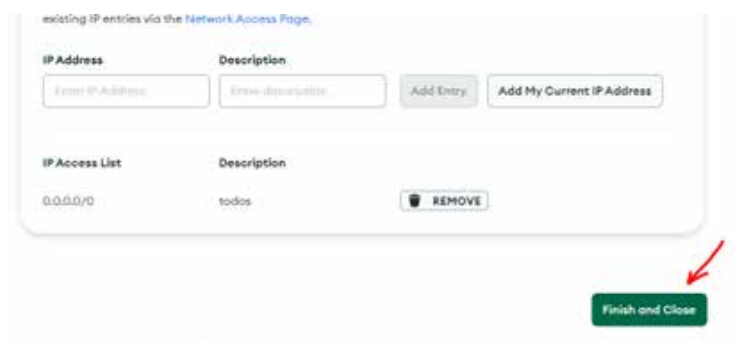


En el 1, seleccione la autenticación con usuario y contraseña, en el 2; digite el usuario deseado; en el 3, digite la contraseña deseada, incluso puede autogenerar una segura (Autogenerate Secure Password); click en el botón 4, para sus credenciales.

En la siguiente pantalla que se muestra:



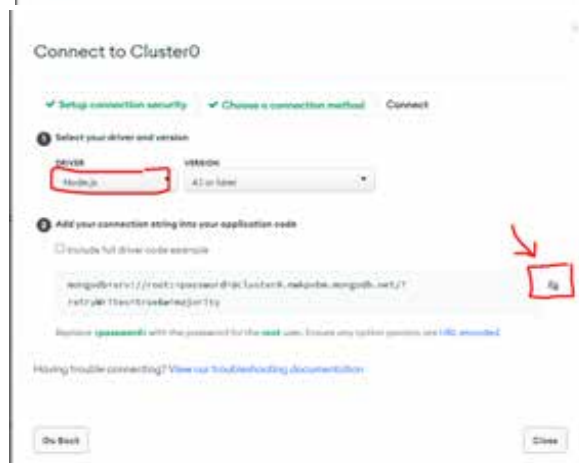
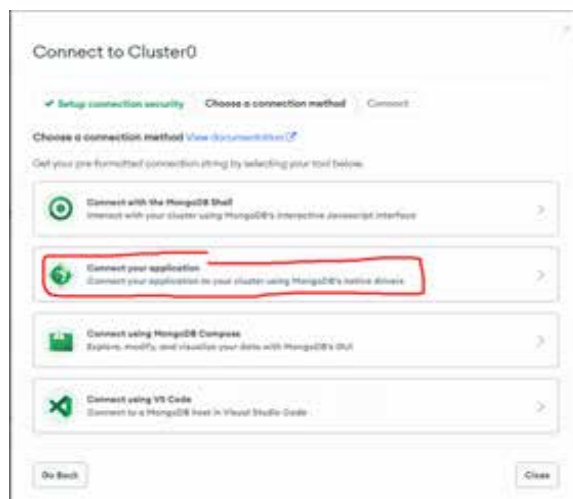
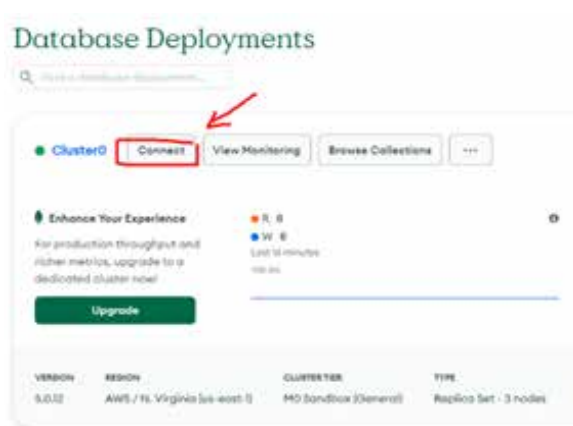
En 1, se muestra el usuario ya creado con el paso anterior; en 2, se selecciona el ambiente de trabajo local; en 3, se digita: 0.0.0.0/0, esto significa que se podrá acceder desde cualquier dirección IP con cualquier “máscara”, y en el campo 4, se coloca una descripción (opcional); por último (5), se agrega esta condición a las listas de acceso de direcciones IP. Haga clic en el botón “finish and close”



Finalmente, estará creado su clúster.



Haga clic sobre el botón “connect”, y seleccione “connect your application”, copie la URI como se muestra a continuación:



La URI: `mongodb+srv://root:<password>@cluster0.nwkpvtbm.mongodb.net/?retryWrites=true&w=majority`

En <password>, coloque la contraseña que asignó al momento de crear las credenciales de su usuario.

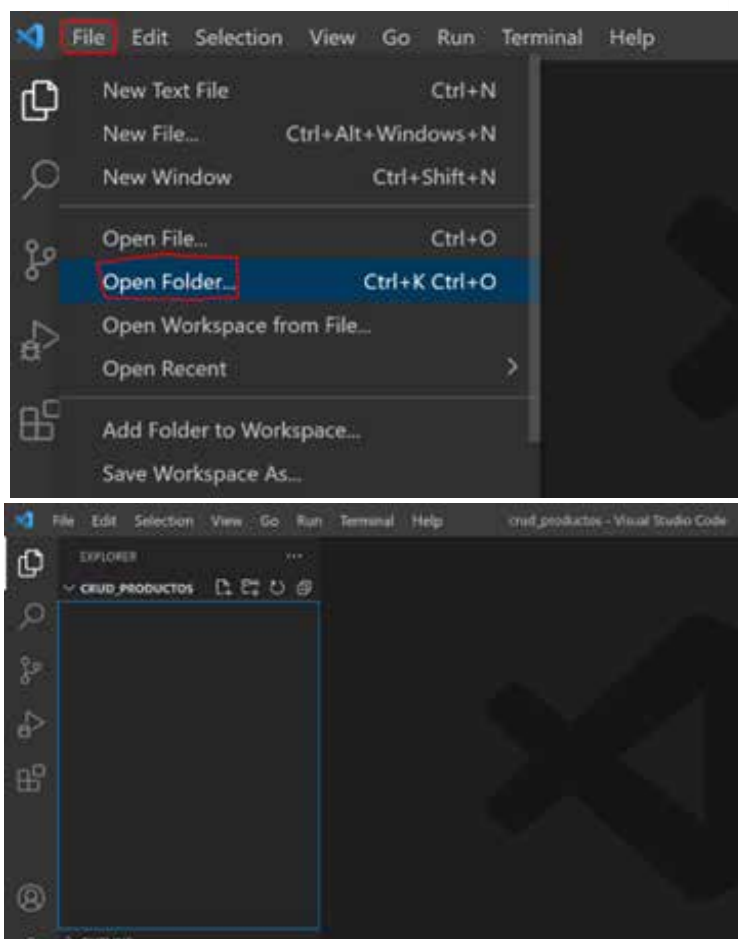


## Codificación para conexión con Base de datos y CRUD con NodeJS

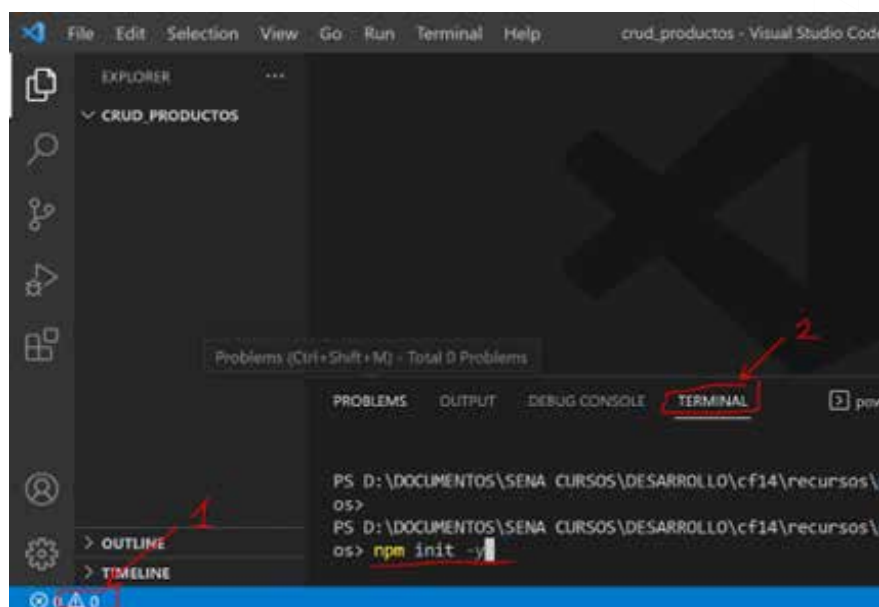
1. Cree un directorio (carpeta) en el lugar que prefieras de su disco duro “crud\_productos”



0. Abra el IDE de programación: Visual Studio Code y seleccione la carpeta creada o arrastra dicha carpeta a la ventana del IDE abierto



Desde la “terminal”, ejecute el comando: “npm init -y”, para inicializar el proyecto de Node.

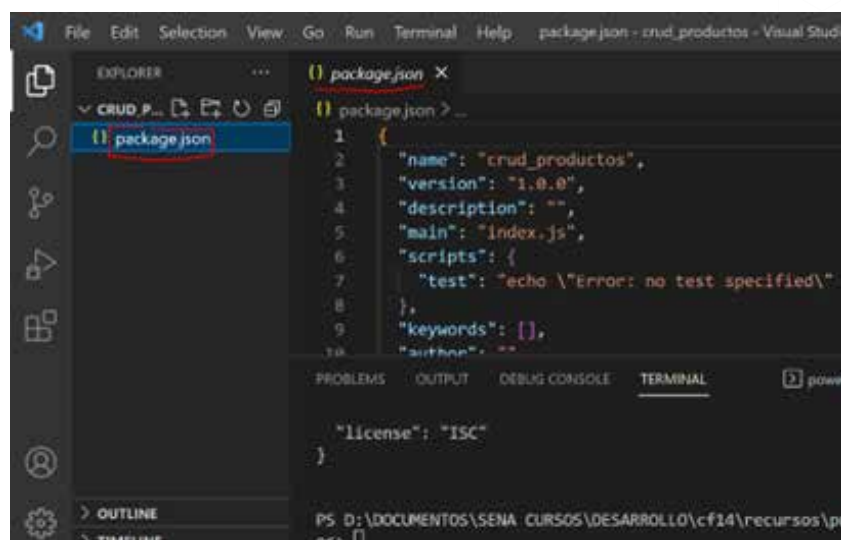


En 1, da clic para que aparezcan la opción de “terminal”, hace clic en esta y digita el comando indicado. Una vez digitado, aparecerá el fichero de “package.json”.

0. Cree un archivo llamado “app.js” y un archivo llamado “server.js” en la raíz del proyecto.

server.js: será el punto de ejecución de tu API REST

app.js: se colocará la configuración para el funcionamiento de la aplicación, se importan las rutas y se definen los **endpoints** para los productos.

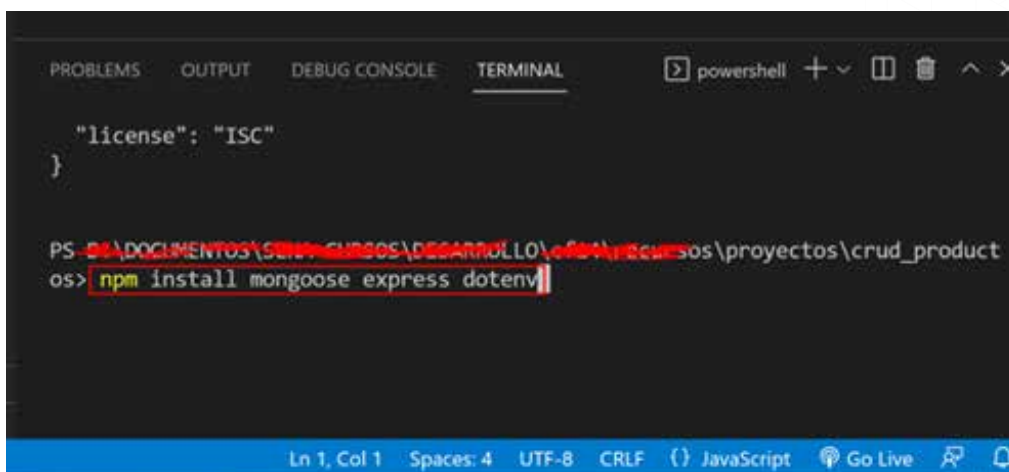




0. Instale las dependencias necesarias para desarrollar su proyecto BackEnd. Puede ejecutar en la terminal, los siguientes comandos:

```
npm install mongoose
npm install express
npm install dotenv
```

incluso, puede ejecutarlos todos al mismo tiempo: `npm install mongoose express dotenv`



```

"license": "ISC"
}

PS C:\DOCUMENTOS\SENA\CURSOS\DESARROLLO\curso-desarrollo\proyectos\crud_productos> npm install mongoose express dotenv

```

La librería de dotenv: lee las variables de entorno de la aplicación.

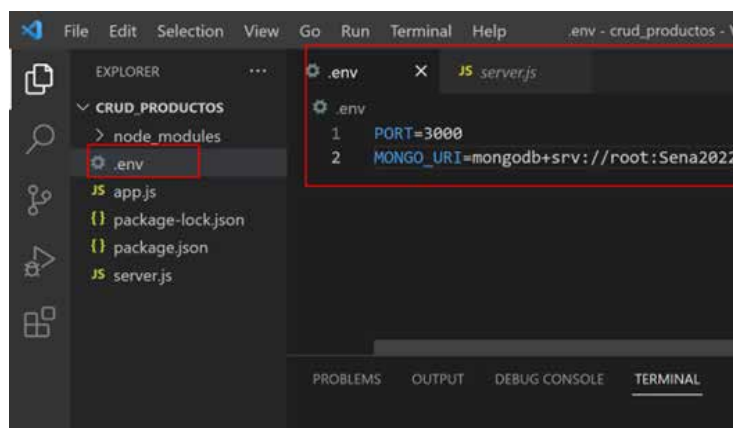
0. Cree en la raíz del proyecto, el archivo llamado: ".env" con el siguiente contenido:

0. PORT=3000

0. MONGO\_URI=mongodb+srv://root:Sena2022@cluster0.nwkpvt.mongodb.net/?retryWrites=true&w=--majority

**Recuerde:** el usuario y contraseña en este caso es root y Sena2022, respectivamente. Pero debe sustituirlos por los que se crearon como tus credenciales.

Este archivo (.env), corresponde a las variables de entorno de su aplicación que serán leídas con "dotenv".

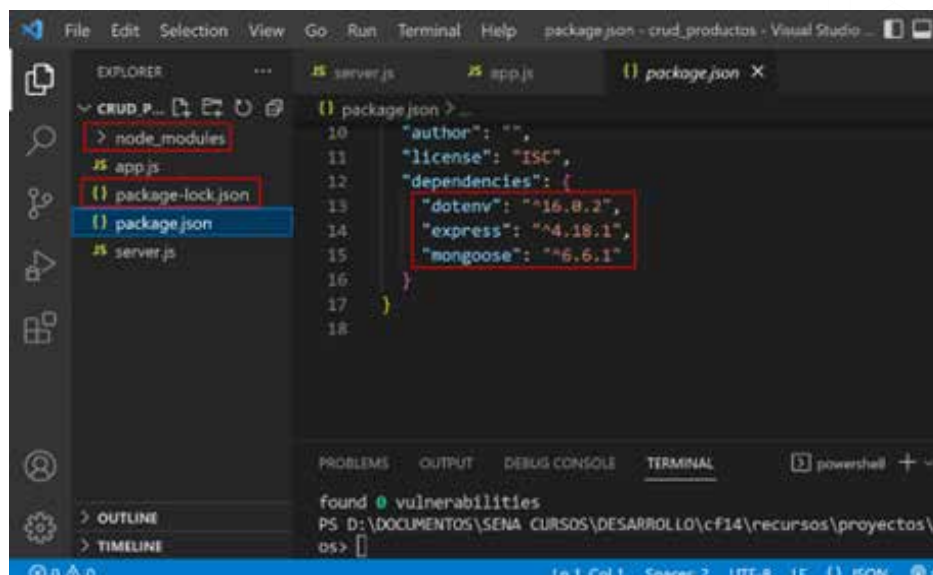


```

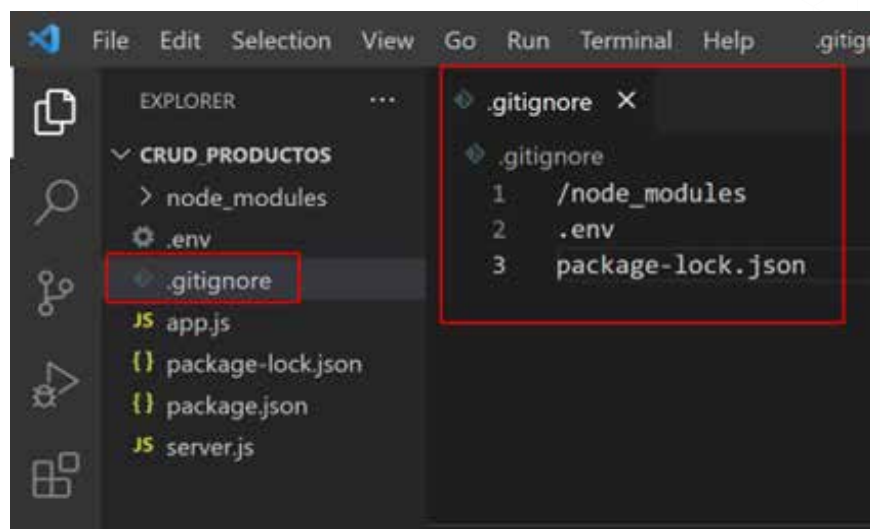
File Edit Selection View Go Run Terminal Help .env - crud_productos - V
EXPLORER
  CRUD_PRODUCTOS
    > node_modules
    .env
  JS app.js
  {} package-lock.json
  {} package.json
  JS server.js
.
1 PORT=3000
2 MONGO_URI=mongodb+srv://root:Sena2022@cluster0.nwkpvt.mongodb.net/?retryWrites=true&w=--majority

```

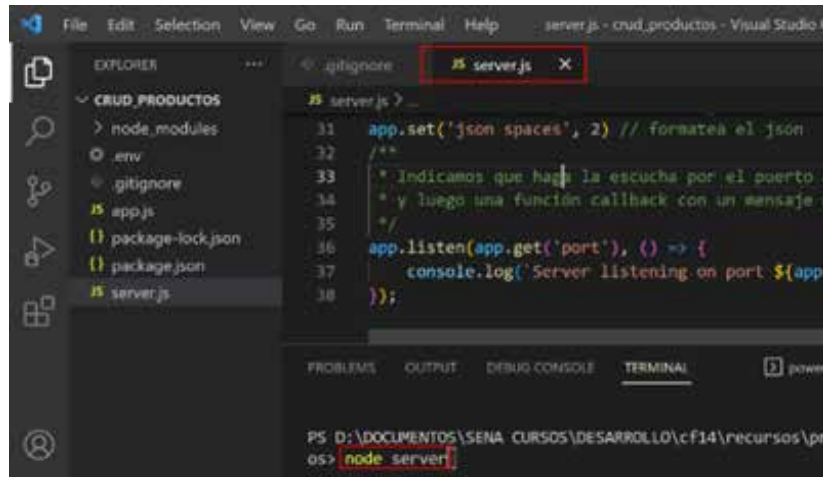
Debe esperar que finalice la instalación. Si revisa el archivo `package.json`, podrá ver que se encuentran las extensiones que acaba de instalar con las versiones correspondientes, se creará el directorio `node_modules` y el archivo `package-lock.json` de manera automática.



Se recomienda que cree el archivo llamado `.gitignore` para excluir de subir a su repositorio GIT: `.env`, `node_modules` y `package-lock.json`. Al final, subirá tu proyecto a un repositorio GIT.



0 Pruebe que funciona su servidor colocando inicialmente el código del proyecto realizado en el componente formativo 10 y posteriormente ejecute `server.js` con el comando `node server` en la terminal



```

11 app.set('json spaces', 2) // formatea el json
12
13 /**
14  * Indicamos que haga la escucha por el puerto
15  * y luego una función callback con un mensaje
16  */
17 app.listen(app.get('port'), () => {
18   console.log('Server listening on port ${app.get('port')}');
19 });

```

PS D:\DOCUMENTOS\SENA CURSOS\DESARROLLO\cf14\recursos\pro...  
os> node server

Abra el navegador y escriba: localhost:3000/clients



```

[
  {
    "id": 1,
    "name": "Maria",
    "email": "mariadoe@mail.com"
  },
  {
    "id": 2,
    "name": "Juan",
    "email": "juandoe@mail.com"
  }
]

```

De esta manera podrá probar inicialmente que todo funcione de manera adecuada.

0. Cree la estructura de directorios, para organizar su proyecto, de manera que sea más entendible, mejor organizado y mantenible a futuro para modificar o agregar nuevos módulos. Simplemente va a crear cuatro (4) directorios en la raíz de su proyecto.

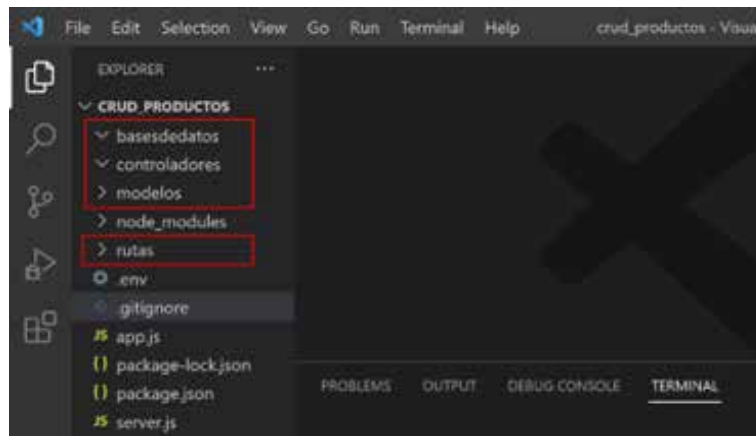
Nota: el presente proyecto tiene un único módulo que sería el de “productos”.

controladores: Aquí organizarás la lógica de negocios de tu módulo de productos

basededatos: Colocará la configuración de las conexiones de la base de datos con Node, en este caso, con MongoDB

modelos: Creará los “modelos” de su módulo

rutas: configurará y asignará las rutas de los endpoints de tu CRUD



0. Crea dentro del directorio “basesdedatos” el archivo “configuración.js” y en este, escriba “línea a línea” el siguiente código. Y no olvides guardar los cambios del archivo.

Nota: recuerde que cuando se recomienda escribir “línea a línea” el código, es para aportar y garantizar su aprendizaje, evitar copiar y pegar código simplemente. Esperamos que tome nuestro consejo.

```
const mongoose = require('mongoose') // se importa mongoose

const conexion = async () => { // async convierte a promise de JS
  try{
    // se usa el método connect para realizar la conexión
    // con process.env.MONGO_URI, se llama desde .env la variable
    await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true
    });
    console.log('Conexión satisfactoria!')
  }catch(e){
    // si ocurre algún error: mal URI, error de internet
    // u otro error, entra por este catch y muestra el porqué
    console.log('Error al conectar...', e);
    throw new Error('Error al conectar...');
  }
}
```

```
module.exports = { mongoConn } // se exporta el módulo
```

El código lleva comentarios para que comprenda mejor en qué consiste cada cosa. Investigue lo que no entienda y no esté comentado. En la siguiente imagen, puede ver cómo va quedando la construcción de tu proyecto al momento:

0. Ahora va a probar si la conexión funciona correctamente. Realizará un refactoring del código ac-



tual del archivo “server.js”, lo reorganizará. Los archivos “app.js” y “server.js”, quedarán como se muestra a continuación, digite línea a línea y analice al mismo tiempo lo que ocurre.

#### “app.js”

```
const express = require('express');// se importa express

const app = express(); // se crea nueva instancia de express

// middlewares
app.use(express.json()) // se coloca el middleware para formatear a json

module.exports = app; // se exporta la instancia app
```

#### “server.js”

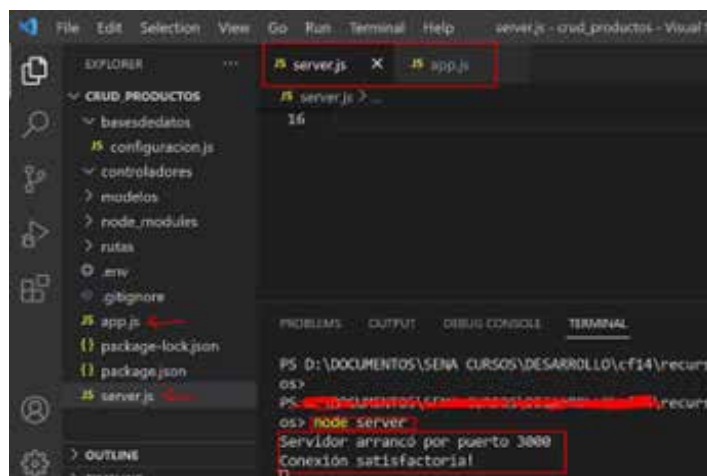
```
const app = require('./app')// se importa el módulo app
const { Conexion } = require('./basesdedatos/configuracion')// se importa la conexión
const dotenv = require('dotenv').config()// se importa dotenv y configura para que tome las variables del .env
// se asigna el puerto de escucha que se puso en .env
// si no se encuentra toma 3000 por defecto
app.set('puerto', process.env.PORT || 3000)

// se crea una nueva instancia de conexion
const conn = Conexion()

// se coloca como puerto de escucha el asignado
// aqui arranca el servidor
app.listen(app.get('puerto'), () => {
  console.log(' Servidor arrancó por puerto ${app.get('puerto')}');
});
```

Guarde todos los cambios.

Debe parar el proyecto, en caso que esté “arrancado” y ejecutar nuevamente con “node server”. Asegúrese de que indique que el servidor arrancó correctamente.



0. Cree el modelo. En la carpeta “modelos”, crear un el archivo “producto.js”. Aquí creará un Schema de su colección productos de MongoDB. Digite línea a línea el siguiente código:

```
const { Schema, model } = require("mongoose");// Se importa Schema y model de mongoose
```

```
/**
 * se crea un Schema de mongoose
 * como parámetro de Schema, se colocan los atributos
 * se colocan las restricciones correspondientes de cada uno
 * Los atributos se definieron al inicio del problema
 */

const ProductoSchema = Schema({
  serial: {
    type: String,
    required: [true, 'Serial requerido']
  },
  nombre:{
    type: String,
    required: [true, 'Serial requerido']
  },
  precioUnitario:{
    type: Number,
    default: 0
  },
  fechaCreacion:{
    type: Date,
    default: new Date()
  },
  fechaActualizacion:{
    type: Date,
    default: new Date()
  }
})

/**
 * Finalmente, se exporta el modulo como el nombre del modelo
 * Producto asignado al esquema ProductoSchema
 */
module.exports = model('Producto', ProductoSchema)
```



0. Cree un archivo llamado “producto.js” dentro de la carpeta “controladores”, con el siguiente código, en este van los comentarios para que digite y vaya analizando y entendiendo para qué es cada cosa:

```
/**
 * si te fijas el en lo que está dentro del require
 * es la ruta del archivo sin la extensión
 * esto se hace para módulos del proyecto
 */
const Producto = require('../modelos/producto')// se importa el modelo anterior creado

/**
 * Se importan estos atributos desde express
 * para colocarlo por defecto en los parámetros
 */
const { request, response } = require('express')

/**
 * método para crear producto nuevo
 */
const crearProducto = async (req = request,
  res = response) => { // pámetros de petición y respuesta (request, response)
  const nombre = req.body.nombre // se obtiene el nombre del body enviado desde el cliente
  const productoBD = await Producto.findOne({ nombre })// se consulta el producto con este nombre
  if(productoBD){ // si existe el nombre arroja un error de que existe ya, para evitar crear nombres repetidos
    return res.status(400).json({msg: 'Ya existe nombre'})
  }
  const producto = new Producto(req.body)// se crea un nuevo producto del Schema Producto
  await producto.save() // se guarda en base de datos el producto
  res.status(201).json(producto) // una vez guardado, se retorna respuesta http 201 con el producto creado en formato JSON
}

/**
 * método para consultar todos los productos
 */
const obtenerProductos = async (req, res = response) => {
  const productos = await Producto.find(); // se obtienen todos los productos
  res.status(200).json(productos) // se retorna respuesta http 200 con todos los productos en formato JSON
}

/**
 * método para consultar un producto por su ID
 */
const obtenerProductoPorID = async (req = request,
  res = response) => {
  const {id} = req.params; // se obtiene el parametro de la URL llamado id
  const productoBD = await Producto.findOne({ id });
  res.json(productoBD) // se muestra el producto encontrado en formato JSON con respuesta Http 200
}

/**
```

```

* método para Actualizar un producto por su ID
*/
const actualizarProductoPorID = async (req = request,
  res = response) => {
  const { id } = req.params; // obtiene el id pasado por parámetro
  const data = req.body; // obtiene todo el body de los datos a actualizar
  const productoBD = await Producto.findOne({ id }); // consulta el producto por su id
  // si no encuentra el producto, retorna error
  if(!productoBD){
    return res.status(400).json({
      msg: 'No existe el producto'
    });
  }
  data.fechaActualizacion = new Date();
  // encuentra producto con ese id y lo actualiza
  const producto = await Producto.findByIdAndUpdate(id, data, {new : true}); // new: true, retorna nuevo,
  // no el objeto original
  res.json(producto) // se muestra el producto actualizado en formato JSON con respuesta Http 200
}

/**
* método para borrar un producto por su ID
*/
const borrarProductoPorID = async (req = request,
  res = response) => {
  const { id } = req.params; // obtiene el id pasado por parámetro
  const productoBD = await Producto.findOne({ id }); // consulta el producto por su id
  // si no encuentra el producto, retorna error
  if(!productoBD){
    return res.status(400).json({
      msg: 'No existe el producto'
    });
  }
  // encuentra producto con ese id y lo borra
  const producto = await Producto.findByIdAndDelete(id);
  res.status(204).json(producto) // una vez guardado, se retorna respuesta http 204 (no content) con el
  // producto borrado en formato JSON
}

module.exports = {
  crearProducto,
  obtenerProductos,
  obtenerProductoPorID,
  actualizarProductoPorID,
  borrarProductoPorID
}

```

Cuando se realicen las pruebas de los endpoints del API REST finalizada, comprenderá mejor el tema del “req.body”



0. Ahora, dentro del directorio “rutas”, cree un archivo “producto.js” y escriba la siguiente lógica línea a línea:

```
const { Router } = require('express');// se importa Router desde express
// se importan todos los métodos desde el controlador de productos creado
const {
  crearProducto,
  obtenerProductos,
  obtenerProductoPorID,
  actualizarProductoPorID,
  borrarProductoPorID
} = require('../controladores/producto');

const router = Router();// se crea una nueva instancia de Router

/**
 * ruta de endpoint de crear un producto
 */
router.post('/', crearProducto);// ruta con verbo POST: indicado para creación de recursos

/**
 * ruta de endpoint de obtener todos los productos
 */
router.get('/', obtenerProductos);// Verbo GET: indicado para obtener recursos

/**
 * ruta de endpoint de obtener un producto por su id
 */
// se pasa :id, el cual debe tener el mismo nombre al
// momento de hacer la petición desde el cliente
// se colocar en la URL con el mismo nombre
router.get('/:id', obtenerProductoPorID);// Verbo GET: indicado para obtener recursos

/**
 * ruta de endpoint de actualizar un producto por su id
 */
// se pasa :id, el cual debe tener el mismo nombre al
// momento de hacer la petición desde el cliente
// se colocar en la URL con el mismo nombre
router.put('/:id', actualizarProductoPorID);// Verbo PUT: indicado para actualizar recursos

/**
 * ruta de endpoint de borrar un producto por su id
 */
// se pasa :id, el cual debe tener el mismo nombre al
// momento de hacer la petición desde el cliente
// se colocar en la URL con el mismo nombre
router.delete('/:id', borrarProductoPorID);

module.exports = router;// se exporta la instancia router
```



0. Ahora actualizará los archivos de server.js y app.js respectivamente como sigue:

server.js

```
const app = require('./app')// se importa el módulo app
const { Conexion } = require('./basesdedatos/configuracion')// se importa la conexión
const dotenv = require('dotenv').config()// se importa dotenv y configura para que tome las variables del
.env
// se asigna el puerto de escucha que se puso en .env
// si no se encuentra toma 3000 por defecto
app.set('puerto', process.env.PORT || 3000)
app.set('json spaces', 2)// formato de json

// se crea una nueva instancia de conexion
const conn = Conexion()

// se coloca como puerto de escucha el asignado
// aqui arranca el servidor
app.listen(app.get('puerto'), () => {
  console.log(`Servidor arrancó por puerto ${app.get('puerto')}`);
});
```

app.js

```
const express = require('express');// se importa el express
const app = express();// se crea nueva instancia de express

const producto = require('./rutas/producto');// se importa el módulo de rutas de productos

// se configuran los middlewares de la aplicación
app.use(express.json());// sino se colocan, va a haber problema con el req.body

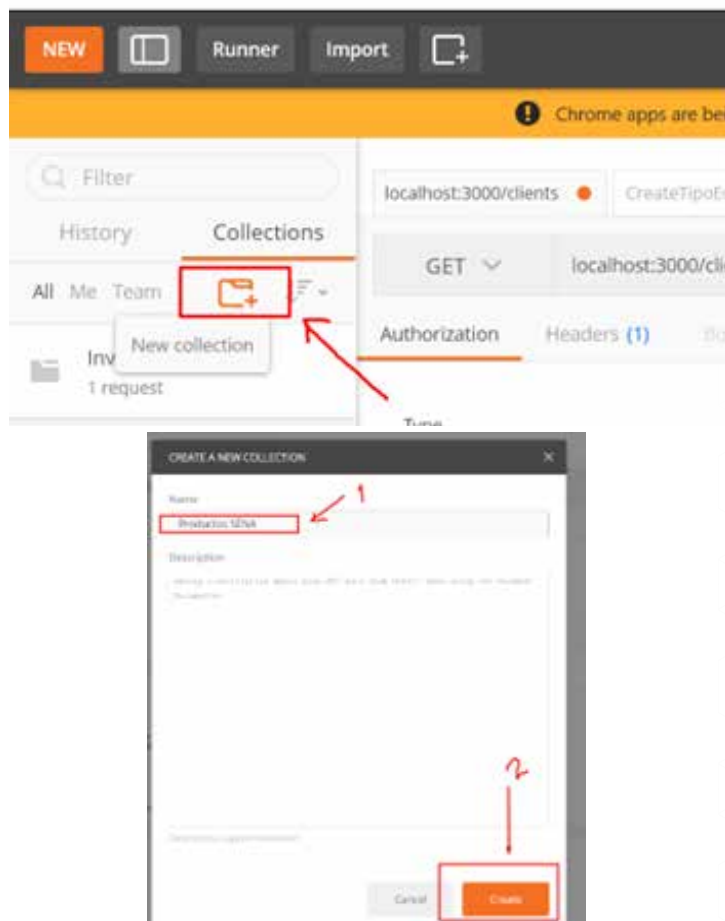
app.use('/api/productos', producto); // esta ruta se complementa con los endpoints de producto

module.exports = app;// se exporta módulo (instancia de express)
```

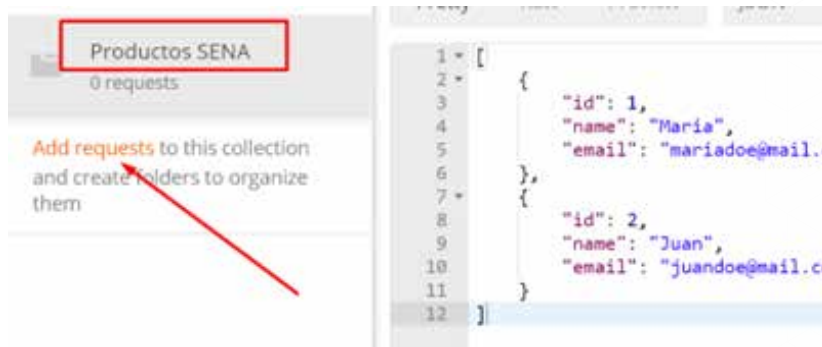
Ejecute la aplicación nuevamente con “node server” (pare la aplicación antes si está ejecutándose).

0. En este paso, probará el funcionamiento de su aplicación. Abra la herramienta de postman y siga los siguientes pasos:

Para organizar las pruebas funcionales en postman, cree una nueva colección, en la pestaña “Collections”, opción “New Collection”, y coloque un nombre



Puede ir agregando request.



Agregue el request de crear un producto nuevo.



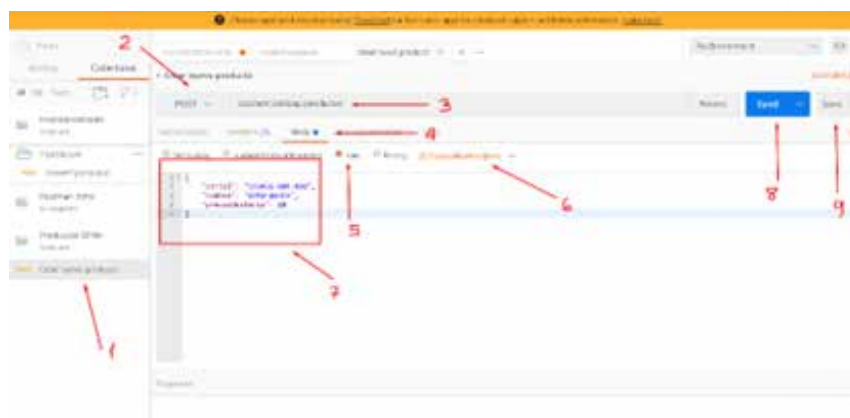
En 1, coloque el nombre deseado para el request; para el 2, seleccione la colección; por último, en 3 le da clic para crear el request.

A continuación, va a seleccionar el verbo de petición (POST para este caso de crear), colocar la URL y colocar el “body” en formato JSON, este body es el que recibe en Node con “req.body”. Los nombres de las claves del JSON del body, debe escribirlos con los mismos que tiene en el Schema que creó en el archivo producto.js de la carpeta modelos. Al final, guarda el request en su colección con los cambios hechos.

Este es el body:

```
{
  "serial": "123456-000-000",
  "nombre": "detergente",
  "precioUnitario": 10
}
```

Nota: como las fechas de creación y actualización por defecto la fecha actual, no tiene necesidad de colocarlas para la creación.

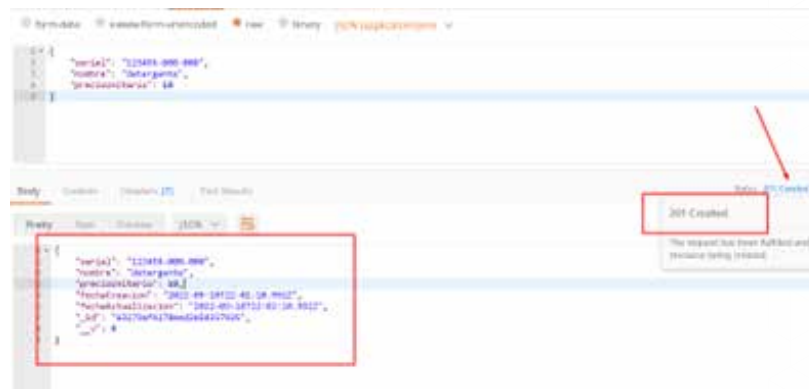




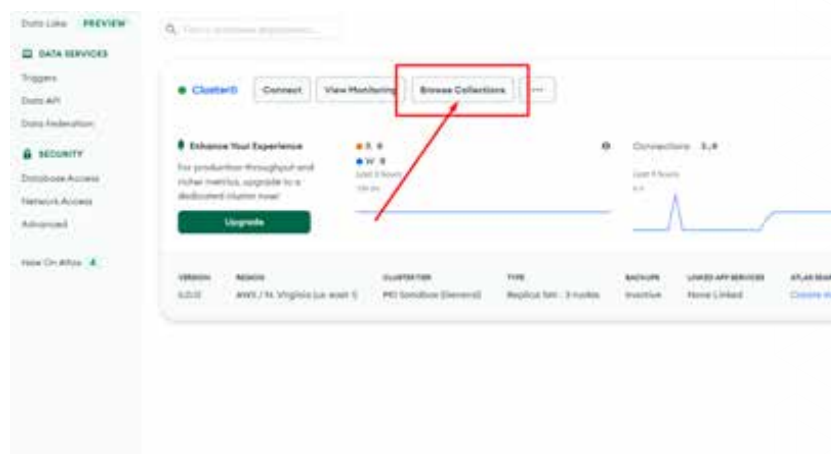
En 1, se muestra el request ya creado; en 2, se selecciona el verbo del endpoint; en 3, se escribe la URL del endpoint establecida; en 4, se selecciona la pestaña body para escribir el JSON; en 5, se selecciona el tipo y en 6 se selecciona el formato JSON; en 7, se coloca el JSON con los datos a guardar; en 8, se debe dar clic para enviar la petición; se da clic en 9 para guardar la petición (request) en la colección de postman creada.

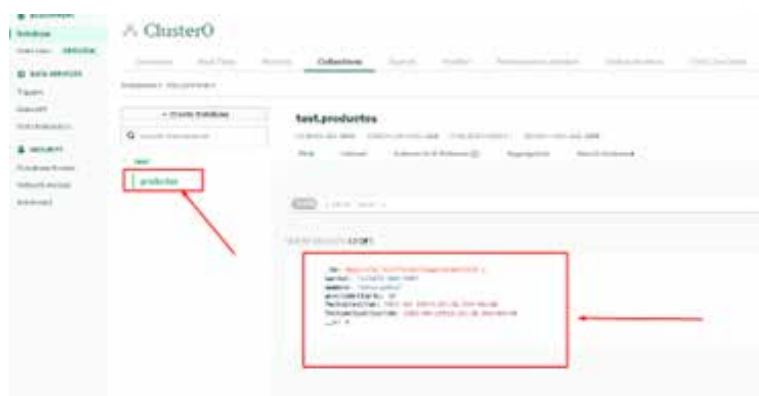
Cuando se da clic en el botón “Send”, se obtiene la siguiente respuesta.

```
{
  "serial": "123456-000-000",
  "nombre": "detergente",
  "precioUnitario": 10,
  "fechaCreacion": "2022-09-18T22:02:10.992Z",
  "fechaActualizacion": "2022-09-18T22:02:10.992Z",
  "_id": "63279af4178eed2e5d357925",
  "__v": 0
}
```

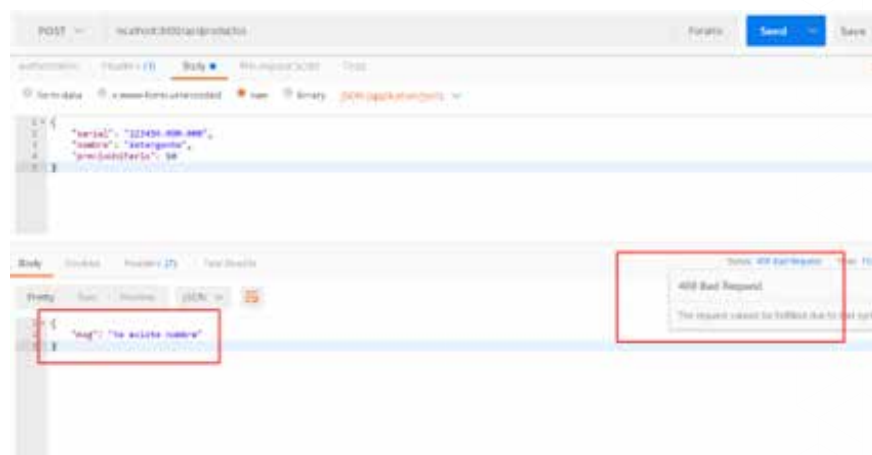


Si verifica en Atlas, podrás ver el registro guardado.

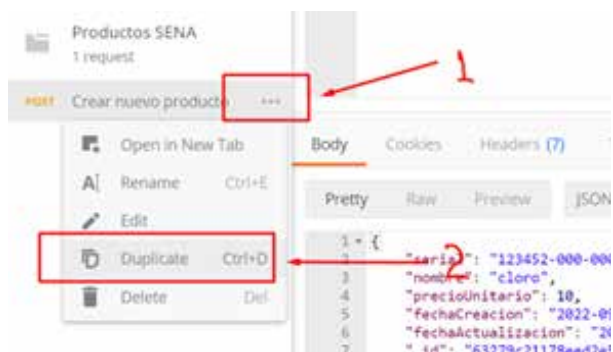


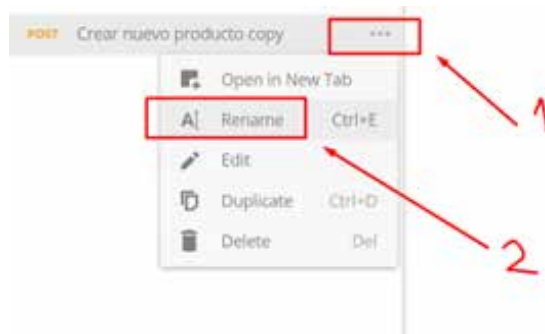


Podrá visualizar la colección “productos” y el registro guardado, si desde el postman intenta guardar el producto con el mismo nombre, verá que obtienes un error, puesto que se puso la condición de poder guardar un producto con el mismo nombre:



Nota: Pruebe guardando más productos de aseo. En el código, también debería poder controlar que no se puedan guardar productos con el mismo serial, ¡este reto es para usted!  
Ahora cree el request para consultar todos los productos.



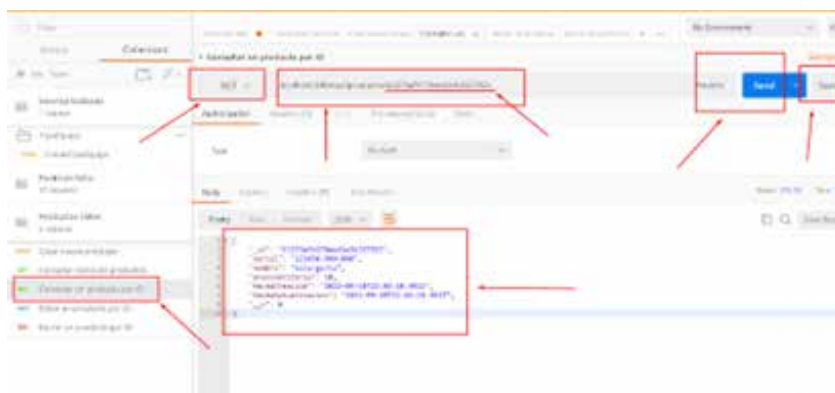


Puede colocarle el nombre de: “Consultar todos los productos” y asignar el verbo y demás.



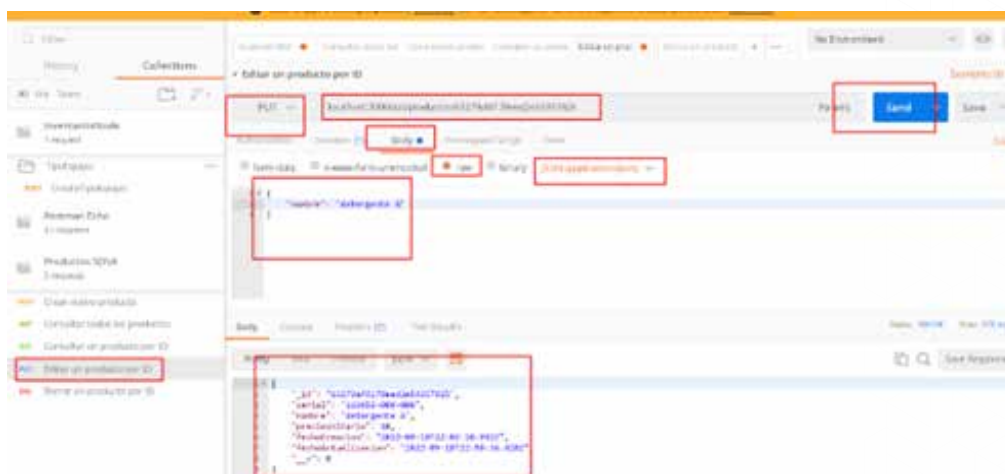
En 1, está el request con el nombre que le asignó; en 2, selecciona GET; en 3, deja la misma URL, porque es la misma que se dejó en la API REST creada, la diferencia con la creación es que esta es con GET y la de creación con POST; en 4, da clic para realizar la petición, y puede visualizar la respuesta en 5; por último, en 6, guarda el request en la colección del postman.

De forma similar va a crear los request para consultar un producto por su Id, puede tomar en Id, de alguno de los productos creados, simplemente realice la consulta anterior de todos los productos y cualquier valor de la clave “\_id”.



De la imagen anterior se indican los cambios destacados para entender la petición

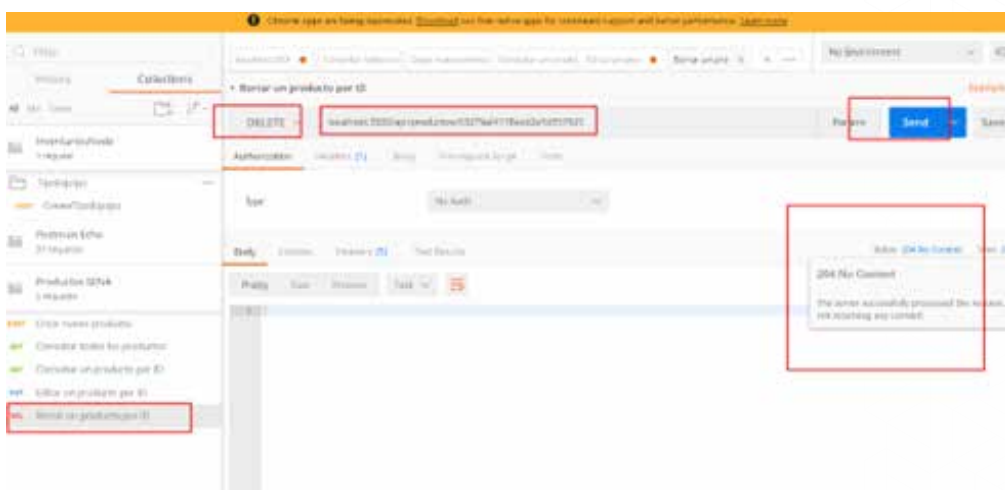
Revise ahora la petición para editar un producto por su ID. Vamos a editar el nombre únicamente, para este caso. Puede probar cambiando más información.



Si revisa en Atlas, encontrará el nombre del producto modificado y su fecha de actualización



Por último, realice la petición para el borrado de un registro por su ID.



Si revisa en Atlas, no debería existir el producto. Si nota, el código de respuesta en un 204 No content, como le indicé al endpoint en el método para el borrado en el controlador.

Nota, la consulta de un producto, la actualización y borrado, se están haciendo por su ID, pruebe realizando por su serial u otro atributo. ¡Este es otro reto para usted!