Práctica 3

Se pide desarrollar un API REST utilizando Node.js, Express y MongoDB. La temática será libre e implementará al menos los siguientes endpoints:

A. Obtener todos los documentos de la colección.
B. Obtener un documento dado su id.
C. Obtener todos los documentos que cumplan dos condiciones.
D. Insertar uno o varios documentos.
E. Actualizar un documento dada una condición.
F. Eliminar un documento dado su id

Comentario del alumno

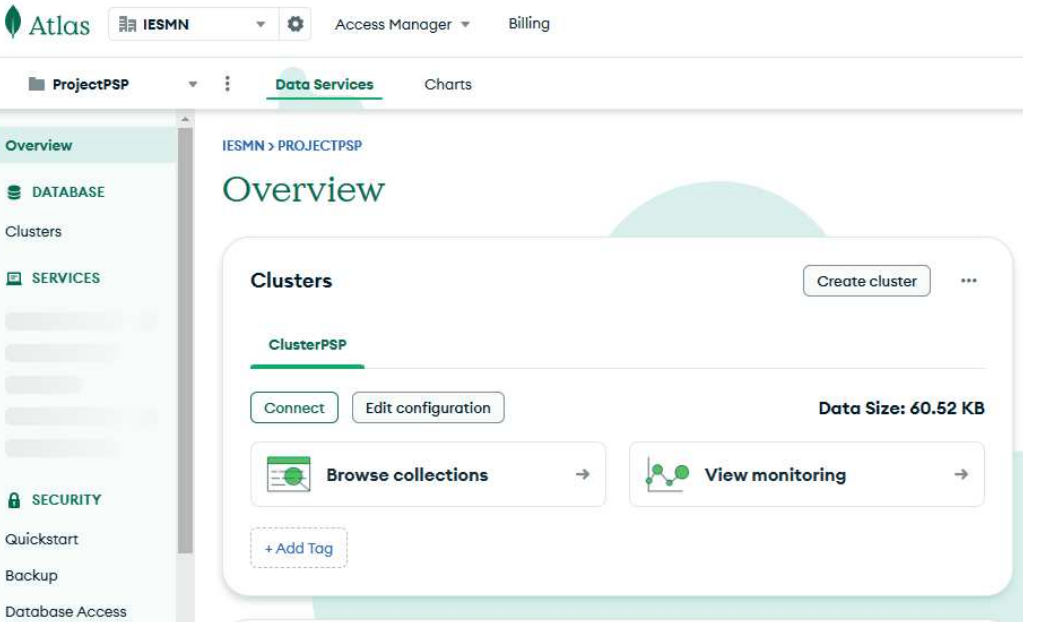
Este documento muestra los pasos que he seguido para crear y gestionar una base de datos MongoDB.
Me hubiera gustado, en un documento adjunto, explicar cómo se crean y consumen los endpoints de una API creado con el entorno 'EXPRESS', pero no es el caso.
Ya me parece bastante útil aprender a manejar las bases de datos 'MongoDB', la aplicación de escritorio 'MongoDBCompass' y el framework de 'Mongoose'. Este último sirve para gestionar las bases de datos desde una aplicación node.js.

Primeros pasos

La práctica consiste en gestionar 3 entornos diferentes, que finalmente realizan la tarea conjunta de administrar una base de datos en MongoDB. Ya ha quedado explicado qué es MongoDB. Una base de datos no relacional, formada a base de colecciones de documentos con formato JSON y con un esquema definido por el administrador, que finalmente contiene las llaves para manejar toda la información.

Personalmente, pienso que las bases de datos no relacionales presentan facilidades evidentes frente a las relacionales. Además de esto, el analista inteligente que conoce perfectamente su base de datos, es capaz de extraer ventajas superiores al manejo de tablas con identificadores unicos.

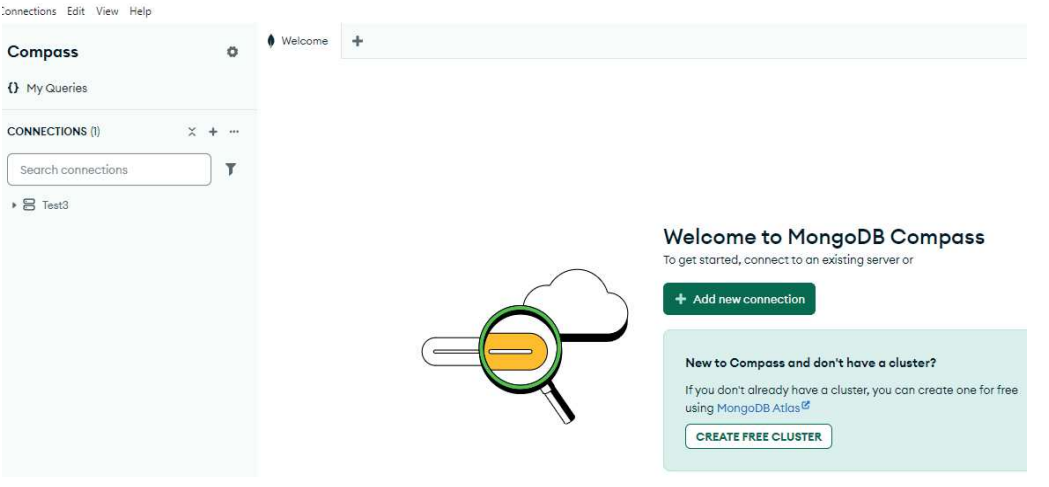
En primer lugar MongoDB requiere de un usuario registrado, que suscriba su cuenta gratuita o de pago. Esto se hace a través de su página WEB.



MongoDB version WEB

En segundo lugar, aunque no es obligatorio, me ha resultado muy útil contar con la aplicación de escritorio MongoDB Compass, para visualizar las colecciones de manera interactiva gracias a su interfaz gráfica.

Estas aplicaciones son fundamentales en la transición desde un entorno gráfico, hacia la linea de comandos o el propio código de mongoose.



Finalmente, utilizamos NPM para añadir dependencias a nuestro proyecto node.js. Podemos observar en el margen el contenido del fichero 'package.json' que hemos generado con el comando 'npm init' y 'npm install –save mongoose'.

Ya estamos preparados para crear nuestra primera base de datos en Mongo.

```
// fichero package.json
{
  "name": "practica3",
  "version": "1.0.0",
  "main": "practica3_version1.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "mongoose": "^8.11.0"
  }
}
```

Creación de la base de datos

'Mongo ATLAS', creamos un cluster que sirve de almacenamiento de nuestras bases de datos. El motivo del cluster es doble, por un lado define la capacidad máxima de la que disponemos, y por otro lado supone nuestro plan de suscripción a la plataforma online. Mejores planes implican mejores condiciones de cluster.

Ahora ya podemos conectar con MongoATLAS desde nuestro programa node.js. Simplemente necesitamos una linea de código:

```
const mo ngoose = require('mongoose');
mongoose.connect('mongodb+srv://yveelilop:****@clus-
terpsp.wy5yj.mongodb.net/');
```

Si no se especifica una base de datos, seremos automáticamente conectados a una base de datos 'test'.

Para conectarnos a nuestra base de datos dentro del cluster, añadimos la ruta en la linea de conexión:

```
var databasePath = 'databaseTest1';
mongoose.connect(`mongodb+srv://yveelilop:****@clus-
terpsp.wy5yj.mongodb.net/${databasePath}`);
```

Añadiendo documentos a nuestra base de datos

El funcionamiento para añadir documentos a nuestra base de datos es increíblemente sencillo.

Primero creamos el esquema que da formato a un documento. Es importante darse cuenta que podemos tener tantos documentos como deseemos y asi, tantos esquemas como necesitemos. Nuestra base de datos se libera del esquema rígido del modelo relacional

A continuación creamos el documento propiamente dicho. Debemos darle parametros de nombre de base de datos y esquema del modelo.

Se crea un nuevo registro, atendiendo al esquema.

Finalmente se guarda el registro en la colección de la base de datos.

```
let deportesSchema = new mongoose.Schema({
    deporte: String,
    nombre_equipo: String
});

let Deporte = mongoose.model('collectionTest1', deportesSchema,
'collectionTest1');

let deporte1 = new Deporte({
    deporte: "Baloncesto",
    nombre_equipo: "Lucentum HLA"
});

deporte1.save().then(
    result => {
        console.log("Deporte añadido:", result);
    }
).catch(
    error => {
        console.log("Error añadiendo deporte, error);
    }
);
```

Cómo buscar documentos dentro de nuestra base de datos

Es muy sencillo. Primero definimos el modelo del documento que queremos encontrar. Entonces guardamos en una variable el modelo de respuesta. Para terminar ejecutamos el comando ‘find’ con el patrón de búsqueda ‘{}’ que implica buscar cualquier documento.

```
let deportesSchema = new mongoose.Schema({
    deporte: String,
    nombre_equipo: String
});

let Deporte = mongoose.model('collectionTest2', deportesSchema,
'collectionTest2');

Deporte.find({}).then(
    result => {
        console.log("Documentos encontrados:", result);
    }
).catch(
    error => {
        console.log("Error: ", error);
    }
);
```

Realizando tareas de mantenimiento de nuestra coleccion

A continuación vamos a realizar tareas sencillas con nuestra coleccion de documentos.

Podemos empezar buscando documentos según diferentes parametros. En este caso, estamos buscando un documento por el _id que usa de manera interna MongoDB

```
let id = '67c331f8b1e61630d8865c85'
Deporte.find({ _id: new mongoose.Types.ObjectId(id) })
    .then(result => console.log("Documentos encontrados:", result))
    .catch(error => console.log("Error: ", error));
```

Tambien podemos encontrar un documento y actualizarlo con nuevos parametros

```
Deporte.findOneAndUpdate(
    { nombre_equipo: 'Lucentum HLA' },
    { nombre_equipo: 'HLA Alicante' },
    { new: true }
).then(doc => {
    if (doc) {
        console.log('Documento actualizado:', doc);
    } else {
        console.log('No se encontró el documento para actualizar.');
    }
}).catch(error => console.log('Error al actualizar:', error));
```

Y por último, también podemos encontrar por el id y luego borrar el documento.

```
Deporte.findByIdAndDelete(id)
    .then(doc => {
        if (doc) {
            console.log('Documento eliminado:', doc);
        } else {
            console.log('No se encontró el documento con ese _id.');
        }
    })
    .catch(error => console.log('Error al eliminar:', error));
```