



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO



FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

Ingeniería en Computación

SISTEMAS DE INFORMACIÓN

Profesor: AARON VELASCO AGUSTIN

Grupo: 2859

Proyecto Final

Nombre:

Jiménez Prado Hassel

25 de mayo 2021



Para la creación de mi proyecto decidí implementar 3 tipos de tecnologías, entre las cuales están Express, Node js y MongoDB.

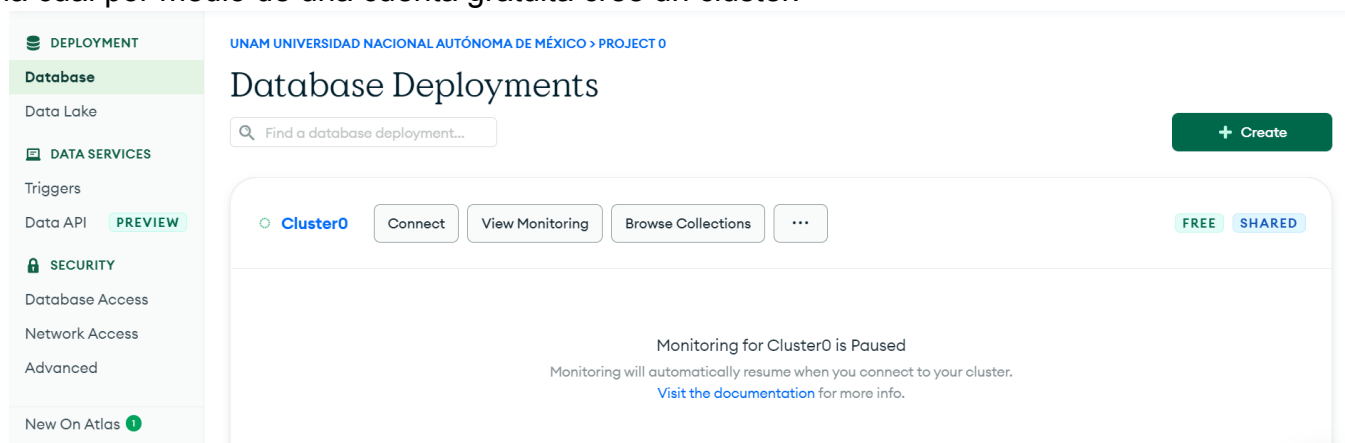
Decidí implementar la herramienta de Express para crear un proyecto de forma instantánea con los ficheros necesarios e información para realizar el crud de proyecto.

Para iniciar con el proceso primero utilice una ventana de comandos en este caso git bash para crear mis archivos con el comando: **express**

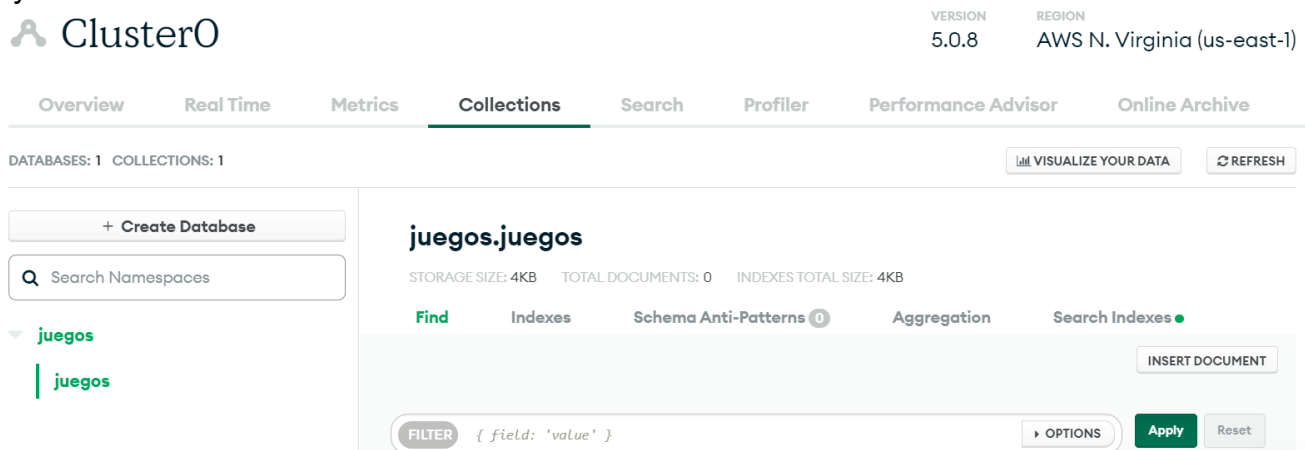
```
Nombre
bin
modelos
node_modules
public
routes
views
app.js
package.json
package-lock.json
```

Después de ese paso ocupe el comando: **install npm** para que se descargue la carpeta node_modules que es necesaria para el funcionamiento de mi proyecto. Una vez realizado ese proceso ahora tuve que usar el comando **npm install --save mongoose** para agregar a las dependencias del proyecto en el archivo package.json la versión de mongoose que se va a emplear.

Ahora para poder almacenar mis datos en una aplicación en línea tuve que usar MongoDB en la cuál por medio de una cuenta gratuita creé un cluster.



En él se almacenarán los datos que vaya subiendo con ayuda de la aplicación de POSTMAN, para eso debemos de crear un acceso a base de datos y también crear la base de datos que voy a utilizar.












Se crea la base de datos nombrada como **juegos** en la colección con el mismo nombre. Después se crea el usuario junto con el password de acceso y los permisos.

Database Access

Database Users

Custom Roles

+ ADD NEW DATABASE USER

User Name	Authentication Method	MongoDB Roles	Resources	Actions
 usrautos	SCRAM	readWriteAnyDatabase@admin	All Resources	 EDIT  DELETE
 usrfes	SCRAM	readWriteAnyDatabase@admin	All Resources	 EDIT  DELETE
 usrjuegos	SCRAM	readWriteAnyDatabase@admin	All Resources	 EDIT  DELETE

El usuario que generé es el **usrjuegos** que nos identifica como creador de contenido en la BD mediante el password que viene en los archivos que subiré a git al final.

Ahora una vez que tenemos este usuario y la base entonces solo falta la conectividad a MongoDB.

Network Access

IP Access List

Peering

Private Endpoint

+ ADD IP ADDRESS

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
189.189.185.48/32 (includes your current IP address)	Juegos	<div><div></div>Active</div>	<div><div> EDIT</div><div> DELETE</div></div>

Agrego la ip de conectividad y género una actual para evitar que no funcione.

Una vez que realicé este proceso entonces vamos a generar los respectivos archivos de subida de datos.

```
EXPLORER
└─ PROYECTO
  └─ bin
  └─ modelos
    └─ JS juegos.js
  └─ node_modules
  └─ public
  └─ routes
    └─ JS index.js
    └─ JS juegos.js
    └─ JS users.js
  └─ views
  └─ JS app.js
  └─ {} package-lock.json
  └─ {} package.json

JS juegos.js
1 var mongoose = require('mongoose');
2 var Schema = mongoose.Schema;
3
4 var JuegosSchema= Schema({
5   id:Number,
6   nombre: String,
7   anio: String,
8   compania: String,
9   consola: String
10 });
11
12 module.exports=mongoose.model('juegos', JuegosSchema);
```

Primero creé una carpeta llamada modelos en la cuál se contendrá el constructor de mis datos y se hará la exportación del modelo de mongoose en la cual mandé la base de datos y el constructor de mis datos.

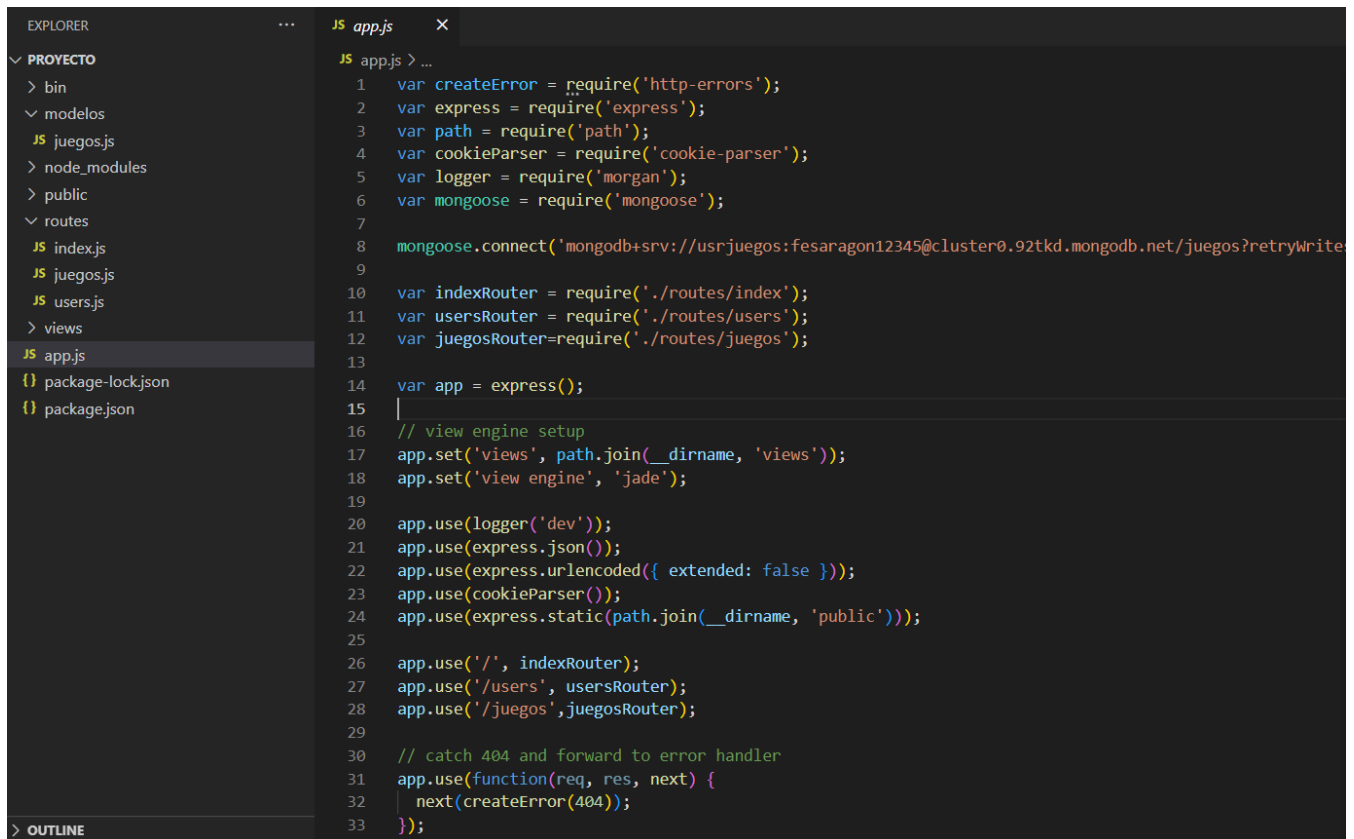
Una vez creado el modelo entonces procedo a crear una copia del archivo users de rutas para renombrarlo como **juegos.js**

```
EXPLOSER
PROYECTO
  bin
  modelos
  JS juegos.js
  node_modules
  public
  routes
    JS index.js
    JS juegos.js
    JS users.js
  views
  JS app.js
  {} package-lock.json
  {} package.json
  > OUTLINE

JS juegos.js
routes > JS juegos.js > router.get('/') callback
1  var express = require('express');
2  var router = express.Router();
3  var mongoose= require('mongoose');
4  var Juego = require('../modelos/juegos');
5
6  /* GET users listing. */
7  router.get('/', function(req, res, next) {
8    //res.render('index', { title: 'Express' });
9    Juego.find({}, (err, datos)=>{
10     if(err){
11       res.json({'Error':'No existe'})
12     }else{
13       res.status(200).json(datos);
14     }
15   });
16 });
17
18 router.post('/', (req, res, next)=>{
19   var game = Juego({
20     id:req.body.id,
21     nombre: req.body.nombre,
22     anio: req.body.anio,
23     compania: req.body.compania,
24     consola: req.body.consola
25   });
26   game.save((err,data)=>{
27     if(err){
28       res.json({'error':"Error al insertar"});
29     }else{
30       res.status(200).json(data);
31     }
32   });
33 });
```

Aquí se crea la petición al constructor de los datos de juegos y con ello se procede a realizar la subida y creación de archivos por medio de los métodos **GET**, para obtención de datos, **POST**, para creación de datos, **DELET**, para borrar datos.

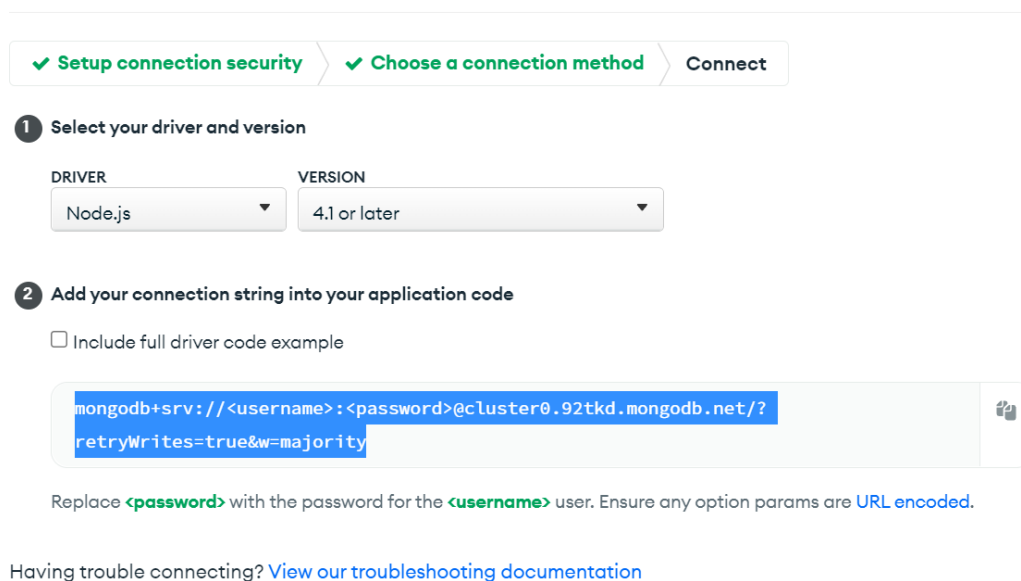
Pero no he terminado aún puesto que no pueden funcionar sin que sean agregados al archivo de app.js.



```
JS app.js > ...
1  var createError = require('http-errors');
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6  var mongoose = require('mongoose');
7
8  mongoose.connect('mongodb+srv://usrjuegos:fesaragon12345@cluster0.92tkd.mongodb.net/juegos?retryWrite
9
10 var indexRouter = require('./routes/index');
11 var usersRouter = require('./routes/users');
12 var juegosRouter=require('./routes/juegos');
13
14 var app = express();
15 |
16 // view engine setup
17 app.set('views', path.join(__dirname, 'views'));
18 app.set('view engine', 'jade');
19
20 app.use(logger('dev'));
21 app.use(express.json());
22 app.use(express.urlencoded({ extended: false }));
23 app.use(cookieParser());
24 app.use(express.static(path.join(__dirname, 'public')));
25
26 app.use('/', indexRouter);
27 app.use('/users', usersRouter);
28 app.use('/juegos', juegosRouter);
29
30 // catch 404 and forward to error handler
31 app.use(function(req, res, next) {
32 |   next(createError(404));
33 });
```

En esta parte se crea el llamado a mongoose, la clave de conectividad que nos genera MongoDB al darle en conectar con la aplicación:

Connect to Cluster0



✓ Setup connection security > ✓ Choose a connection method > Connect

1 Select your driver and version

DRIVER: Node.js | VERSION: 4.1 or later

2 Add your connection string into your application code

☐ Include full driver code example

`mongodb+srv://<username>:<password>@cluster0.92tkd.mongodb.net/?retryWrites=true&w=majority`

Replace **<password>** with the password for the **<username>** user. Ensure any option params are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

Copié esa clave y la puse en **mongoose.conect()**; modificando el usuario y la contraseña y agregando la base de datos a la que vamos a subir los juegos.

Una vez realizado este proceso entonces solo me falta correr la aplicación desde una consola en la cuál colocaremos el comando: **DEBUG=proyecto:* & npm start**

Nos muestra una consola en la que existe la conectividad y por medio de esta se nos confirma que podemos usar POSTMAN para subir los archivos:

localhost:3000/juegos

POST localhost:3000/juegos Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

none form-data **x-www-form-urlencoded** raw binary GraphQL

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> id	1			
<input checked="" type="checkbox"/> nombre	Crash Bandicoot			
<input checked="" type="checkbox"/> anio	1996			
<input checked="" type="checkbox"/> compania	Naughty Dog			
<input checked="" type="checkbox"/> consola	PS1			
Kev	Value	Description		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 103 ms Size: 369 B Save Response

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "_id": "628fb078148018993f8f5963",
4     "id": 1,
5     "nombre": "Crash Bandicoot",
6     "anio": "1996",
7     "compania": "Naughty Dog",
8     "consola": "PS1",
9     "__v": 0
10  }
11 ]

```

Para poder usarla primero debemos de colocar la ip de localhost, el puerto desde donde se envían los datos y la ruta de donde se guardan los datos, por último se crea una petición con ayuda de **Body** y se manda con el boton de send.

Como podemos ver se ha creado el objeto 1, del juego Crash Bandicoot, pero como sabemos que es confiable y que se almacenó en MongoDB, pues solo es cuestión de revisar en Mongo y en la parte de nuestra BD:

DATABASES: 1 COLLECTIONS: 1 VISUALIZE YOUR DATA REFRESH

+ Create Database

Q Search Namespaces

juegos

juegos

juegos.juegos

STORAGE SIZE: 36KB TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 34KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

INSERT DOCUMENT

FILTER { field: 'value' } OPTIONS Apply Reset

```

_id: ObjectId("628fb078148018993f8f5963")
id: 1
nombre: "Crash Bandicoot"
anio: "1996"
compania: "Naughty Dog"
consola: "PS1"
__v: 0

```

```

_id: ObjectId("628fb14a148018993f8f5966")
id: 2
nombre: "Halo"
anio: "2001"
compania: "Bungie"
consola: "Xbox"
__v: 0

```

Aquí se puede ver el resultado de nuestra API REST finalizada y jalando perfectamente.