

8-12-2017

Creación de Blog

Gestión de datos para los sistemas interactivos

Mirian Janeth Avalos Viveros
Heber Avalos Viveros

Contenido

Introducción.....	3
Requisitos	4
Instalación de MongoDB.....	4
Instalación NodeJS.....	5
Desarrollo	7
Creación del proyecto con NodeJS	7
Creación de modelos en MongoDB.....	8
Estructura del proyecto.....	10
Desarrollo del servidor.....	10
Controladores con AngularJS.....	12
Diseño Web	14
Conclusiones.....	17

Introducción

En el desarrollo de este documento se mostrará como objetivo principal la implementación de la tecnología para base de datos NoSQL conocida como MongoDB. MongoDB es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto. En vez de guardar los datos en tablas como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Como se dijo la parte de la base de datos será montada por MongoDB, en la parte del backend se usará NodeJS para la creación del servidor y AngularJS para controlar los servicios internos que se mostrarán en el frontend, cabe mencionar que tanto NodeJS como AngularJS trabajan en formato javascript y son de alta compatibilidad con MongoDB por lo cual se eligieron y en la parte de diseño web se trabajó con el framework de HTML5 conocido como Bootstrap.

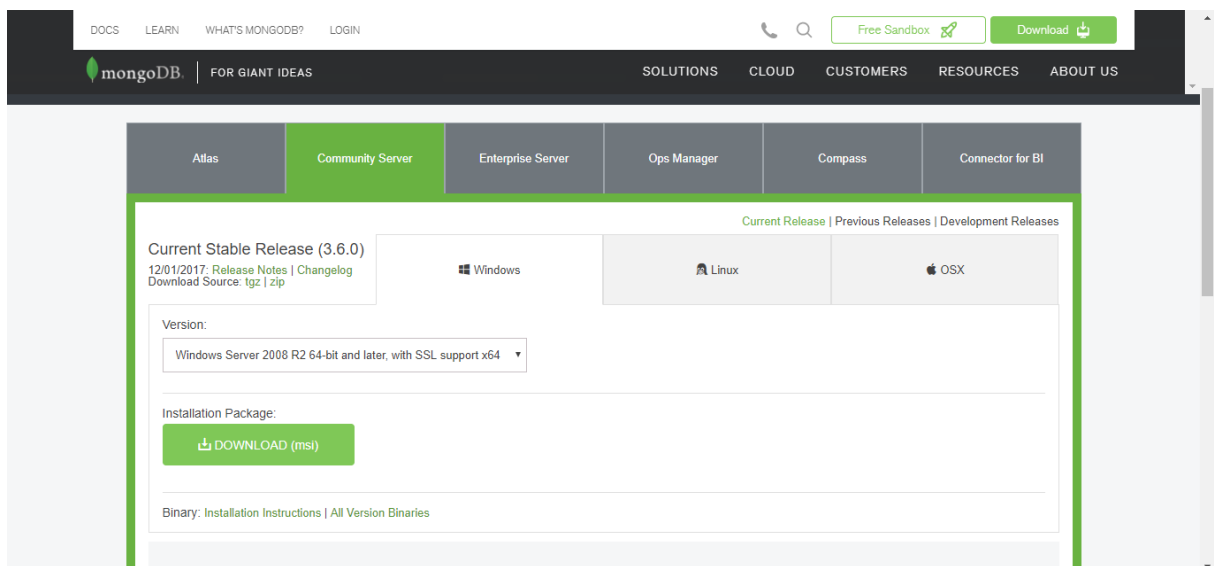
El trabajo se basa en la realización de una práctica tipo blog, en la cual se permita hacer publicaciones y usuarios registrados puedan comentarlas, además de poder editar tanto publicaciones y comentarios realizados.

Requisitos

Para lograr realizar la práctica tenemos que contar con ciertos requisitos, que es el caso de la instalación de MongoDB y NodeJS.

Instalación de MongoDB

Para instalar MongoDB en Windows 10, ingresamos a la página oficial y elegimos la descarga de la versión compatible:

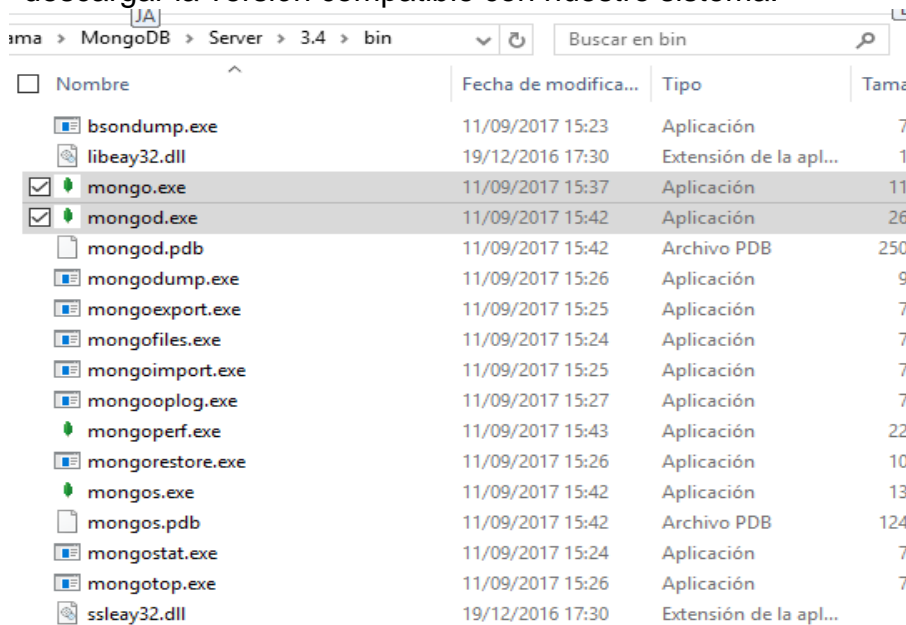


Una vez descargada la versión compatible se instala como cualquier otro programa, cabe mencionar que en ocasiones para poder usar correctamente MongoDB se necesitan realizar unos path's en las variables de entorno. Posterior la forma fácil de correr MongoDB es simplemente ir a la carpeta de instalación de software en nuestro equipo y entrar en la siguiente ruta:
C:\Program Files\MongoDB\Server\3.0\bin

Dentro de dicha ruta encontramos dos archivos importantes mongo y mongod, para correr MongoDB, ejecutamos "mongod" y para interactuar ejecutamos "mongo", así de simple.

Instalación NodeJS

Al igual que con MongoDB lo primero es ingresar a la página oficial de NodeJS y descargar la versión compatible con nuestro sistema:



Nombre	Fecha de modifica...	Tipo	Tama
bsondump.exe	11/09/2017 15:23	Aplicación	7
libeay32.dll	19/12/2016 17:30	Extensión de la apl...	1
<input checked="" type="checkbox"/> mongo.exe	11/09/2017 15:37	Aplicación	11
<input checked="" type="checkbox"/> mongod.exe	11/09/2017 15:42	Aplicación	26
mongod.pdb	11/09/2017 15:42	Archivo PDB	250
mongodump.exe	11/09/2017 15:26	Aplicación	9
mongoexport.exe	11/09/2017 15:25	Aplicación	7
mongofiles.exe	11/09/2017 15:24	Aplicación	7
mongoimport.exe	11/09/2017 15:25	Aplicación	7
mongooplog.exe	11/09/2017 15:27	Aplicación	7
mongoperf.exe	11/09/2017 15:43	Aplicación	22
mongorestore.exe	11/09/2017 15:26	Aplicación	10
mongos.exe	11/09/2017 15:42	Aplicación	13
mongos.pdb	11/09/2017 15:42	Archivo PDB	124
mongostat.exe	11/09/2017 15:24	Aplicación	7
mongotop.exe	11/09/2017 15:26	Aplicación	7
ssleay32.dll	19/12/2016 17:30	Extensión de la apl...	



Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

Important **upcoming security releases**, please review

Descargar para Windows (x64)

8.9.2 LTS
Recomendado para la mayoría

9.2.0 Actual
Últimas características

Otras Descargas | Cambios | Documentación del API | Otras Descargas | Cambios | Documentación del API

Ó revise la Agenda de LTS.

Descargamos el instalador, lo agregamos a las variables de entorno, para posteriormente ingresar el comando node en consola para corroborar que lo tenemos instalado, e incluso consultar la versión con la que contamos.

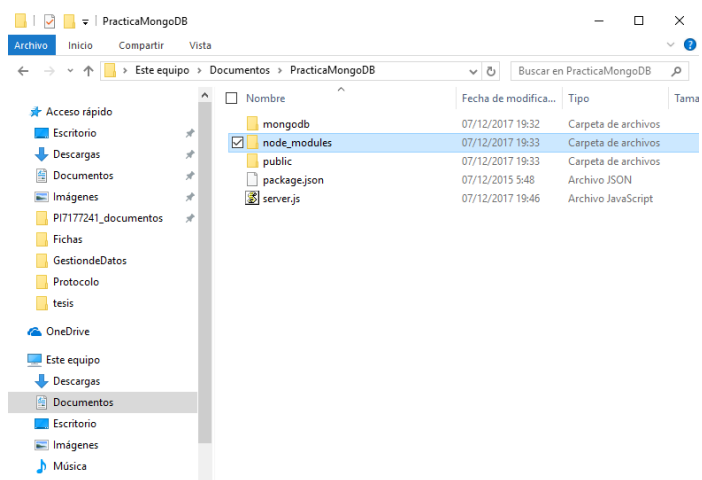
```
C:\Windows\System32\cmd.exe - node
Microsoft Windows [Versión 10.0.10240]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\cetre>node
>
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.10240]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\cetre>node -v
v4.2.1

C:\Users\cetre>
```




Desarrollo

Ahora que tenemos listos los requisitos podemos iniciar con el desarrollo de la práctica, en este proceso lo primero será crear el proyecto en NodeJS, realizar la funcionalidad con Angular y al mismo tiempo trabajar con las interfaces.

Creación del proyecto con NodeJS

NodeJS nos permite crear una estructura de un proyecto de manera automática gracias a la funcionalidad de un archivo base denominado "package.json", el cual está formado por un json que tiene como contenido todos los paquetes (dependencias) necesarios para la realización del proyecto, una vez listo, solo basta con dar un comando en consola y con eso la estructura del proyecto se creará en el directorio donde estemos situados (ahí debe estar el package.json). Mi archivo es el siguiente:



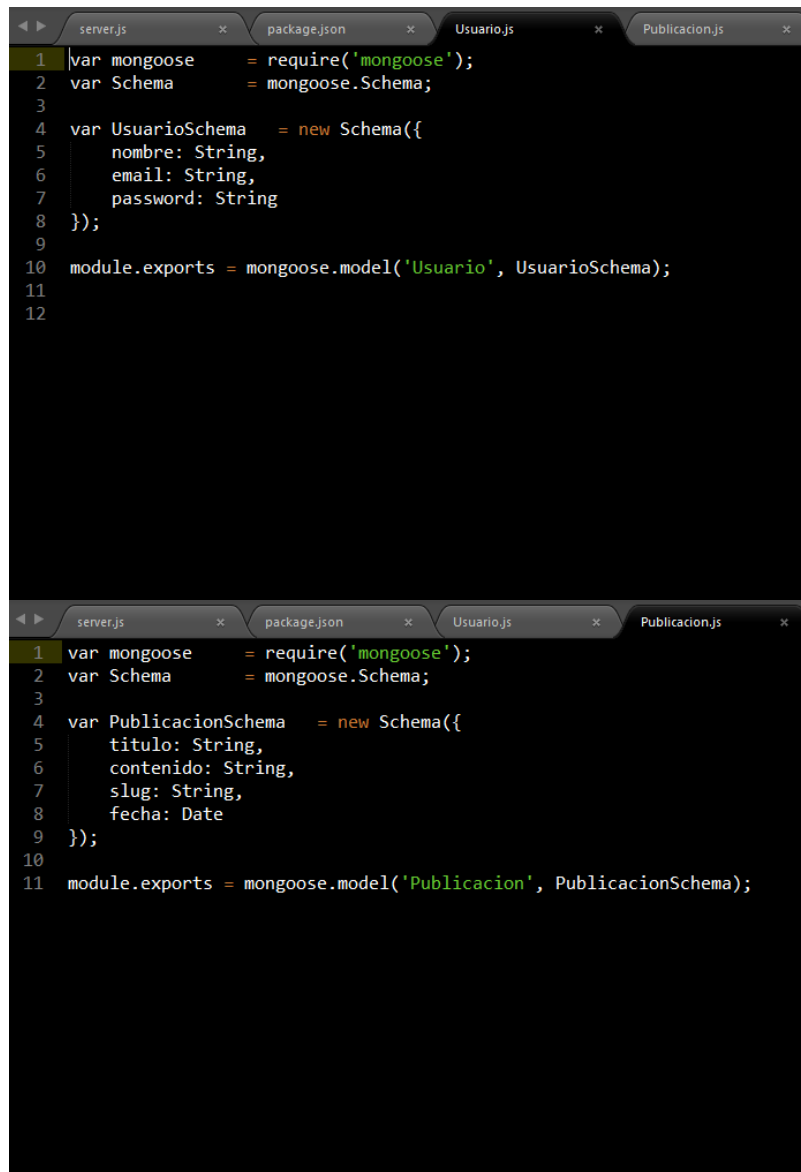
```
1 {
2   "name": "Publicaciones",
3   "version": "0.0.1",
4   "description": "Practica MongoDB",
5   "main": "server.js",
6   "dependencies": {
7     "express": "latest",
8     "mongoose": "~4.2.3",
9     "body-parser": "~1.14.1",
10    "slug": "latest",
11    "morgan": "latest",
12    "jsonwebtoken": "latest"
13  }
14 }
15
16
```

El comando usado para su ejecución es: npm install

Y listo, con esto tenemos listo nuestro proyecto con las dependencias necesarias.

Creación de modelos en MongoDB

La creación de los modelos que se utilizaran en MongoDB se realizaron con la implementación del framework de MongoDB llamado “mongoose”, el cual se puede observar como dependencia en el package.json. Este framework nos permite crear modelos de manera más sencilla y entendible, modelos en formato json que es la forma en la que trabaja MongoDB y además iniciar una base de datos con una simple línea de código. Para la práctica identificamos tres modelos, uno para los usuarios, otro para las publicaciones y uno más para los comentarios, y con base en mongoose se ven así:



```
server.js x package.json x Usuario.js x Publicacion.js x
1 var mongoose = require('mongoose');
2 var Schema = mongoose.Schema;
3
4 var UsuarioSchema = new Schema({
5   nombre: String,
6   email: String,
7   password: String
8 });
9
10 module.exports = mongoose.model('Usuario', UsuarioSchema);
11
12

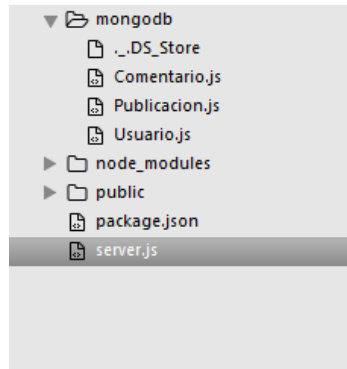
server.js x package.json x Usuario.js x Publicacion.js x
1 var mongoose = require('mongoose');
2 var Schema = mongoose.Schema;
3
4 var PublicacionSchema = new Schema({
5   titulo: String,
6   contenido: String,
7   slug: String,
8   fecha: Date
9 });
10
11 module.exports = mongoose.model('Publicacion', PublicacionSchema);
```



```
server.js x package.json x Usuario.js x Publicacion.js x Comentario.js
1 var mongoose = require('mongoose');
2 var Schema = mongoose.Schema;
3
4 var ComentarioSchema = new Schema({
5   comentarioId: [{
6     type: mongoose.Schema.Types.ObjectId,
7     ref: 'Comentario'
8   }],
9   publicacionId: {type: mongoose.Schema.Types.ObjectId, ref: 'Publicacion'},
10  usuarioId: {type: mongoose.Schema.Types.ObjectId, ref: 'Usuario'},
11  comentario: String,
12  principal: Number,
13  fecha: Date
14 });
15
16 module.exports = mongoose.model('Comentario', ComentarioSchema);
17
```

Estructura del proyecto

Nuestro proyecto cuenta con la siguiente estructura:



En la carpeta `mongodb` se tienen todos los modelos antes mencionados, en `node_modules` como ya se mencionó se encuentran las dependencias y en la carpeta `public` se tienen todos los archivos de las vistas, es decir, los archivos `.html`, `.js`, `.css`, y el archivo `server.js` es el archivo donde se ejecutan los servicios de `nodeJS`.

Desarrollo del servidor

El servidor se encuentra en el archivo `server.js`, para ejecutar este archivo basta con situarnos en la carpeta del proyecto desde consola y arrancar el siguiente comando:
`Node server.js`

Y listo el servidor estará en línea:

```
C:\Users\heber\OneDrive\Documentos\PracticaMongoDB>node server.js
Servidor online
```

Dentro del archivo server.js se crea la base de datos gracias a mongoose con una simple línea de código:

```
mongoose.connect('mongodb://localhost/practicaMongoDB');
```

Se generan las variables que se usarán en las funciones del servidor:

```
3
4 var express    = require('express'),
5 app=express(),
6 server=require('http').createServer(app),
7 bodyParser=require('body-parser'),
8
9 usuario={};
10 //var app      = express();
11
12 var PORT = process.env.PORT || 3000,
13     HOST = process.env.HOST || '192.168.0.16';
14 server.listen(PORT,HOST);
15 //var bodyParser = require('body-parser');
16 var slug      = require('slug');
17 var mongoose  = require('mongoose');
18 var morgan    = require("morgan");
19
```

Podemos ver las variables, como por ejemplo mongoose que nos permitirá el uso de tal dependencia o por ejemplo la variable port la cual define el puerto en donde funcionará el servidor.

Además se definen otros aspectos importantes para la correcta comunicación, como el permitir la implementación de JSON, la creación de las funciones en las diversas vistas, aquí dejo un fragmento de código de una función que actualiza datos de los comentarios dentro del servidor.

```
server.js  x  package.json  x  Usuario.js  x  Publicacion.js  x  Comentario.js  x
319     });
320   });
321   .put(function(req, res){
322     Comentario.findOne({_id: req.params.comentarioId}, function(err, comentario) {
323       if(err)
324         res.send(err);
325       comentario.comentario = req.body.comentario;
326       comentario.save(function(err) {
327         if(err)
328           res.send(err);
329         Comentario.find({publicacionId: req.params.publicacionId, principal: 1})
330           .populate('comentarioId')
331           .populate('usuarioId')
332           .sort('-fecha')
333           .exec(function(err, docs) {
334             if(err) res.send(err);
335             Comentario.populate(docs, {
336               path: 'comentarioId.usuarioId',
337               model: 'Usuario'
338             },
339             function(err, comentarios) {
340               if(err) res.send(err);
341               console.log(comentarios);
342               res.json(comentarios);
343             });
344           });
345         });
346     });
347   });
348
```

Controladores con AngularJS

Tenemos también controladores con AngularJS que permiten las funciones dentro del frontend, a continuación, se muestran algunas.

```
server.js  x  package.json  x  Usuario.js  x  Publicacion.js  x  Comentario.js  x
149
150
151
152 router.route('/registro')
153   .post(function(req, res) {
154     Usuario.findOne({email: req.body.email}, function(err, usuario) {
155       if(err) {
156         res.json({type: false, data: "Ha ocurrido un error!: " + err});
157       }
158       if (usuario) {
159         res.json({type: false, data: "El email ya se encuentra registrado!"});
160       } else {
161         var usuarioModel = new Usuario();
162         usuarioModel.email = req.body.email;
163         usuarioModel.password = req.body.password;
164         usuarioModel.nombre = req.body.nombre;
165         usuarioModel.save(function(err, usuario) {
166           if(err) {
167             res.json({type: false, data: "Ha ocurrido un error!: " + err});
168           }
169           res.json({type: true, data: usuario});
170         })
171       }
172     });
173   });
174
175
```

```
server.js  x  package.json  x  Usuario.js  x  Publicacion.js  x  Comentario.js  x
173
174   });
175
176 router.route('/login')
177   .post(function(req, res) {
178     Usuario.findOne({email: req.body.email, password: req.body.password}, function(err, usuario) {
179       if(err) {
180         res.json({type: false, data: "Ha ocurrido un error!: " + err});
181       }else{
182         if(!usuario){
183           res.json({type: false, data: 'El correo no se encuentra registrado' });
184         }else{
185           res.json({type: true, data: usuario});
186         }
187       }
188     });
189   });
190
191
192
193 router.route('/usuarios')
194   .get(function(req, res) {
195     Usuario.find(function(err, usuarios) {
196       if(err) {
197         res.json({type: false, data: "Ha ocurrido un error!: " + err});
198       }
199       res.json(usuarios);
200     });
201   });
202
```

```

server.js  x  package.json  x  Usuario.js  x  Publicacion.js  x  Comentario.js  x
202     });
203
204     router.route('/comentario')
205     .post(function(req, res) {
206         Comentario.create({
207             comentario : req.body.texto,
208             usuarioId : req.body.usuarioId,
209             publicacionId : req.body.publicacionId,
210             principal : 1,
211             fecha : new Date()
212         }, function(err, comentario) {
213             if (err)
214                 res.send(err);
215             //Obtienes los comentarios de la publicacion y sus comentarios hijos
216             Comentario.find({publicacionId: req.body.publicacionId, principal: 1})
217                 .populate('comentarioId')
218                 .populate('usuarioId')
219                 .sort('-fecha')
220                 .exec(function(err, docs) {
221                     if(err) res.send(err);
222                     Comentario.populate(docs, {
223                         path: 'comentarioId.usuarioId',
224                         model: 'Usuario'
225                     },
226                     function(err, comentarios) {
227                         if(err) res.send(err);
228                         res.json(comentarios);
229                     });
230                 });
231     });

```

```

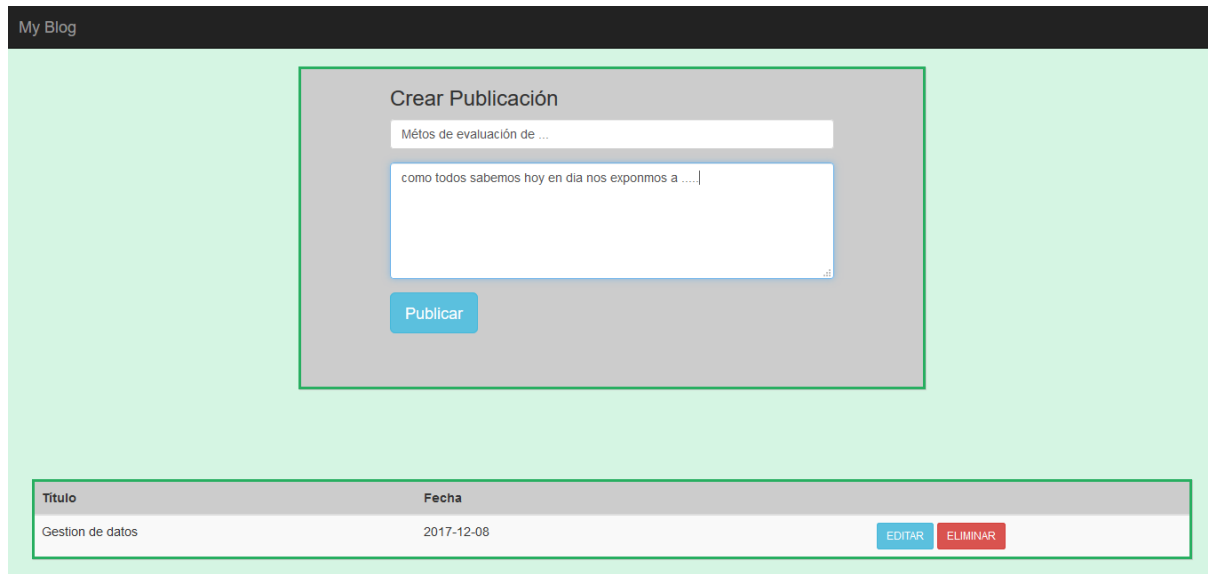
server.js  x  package.json  x  Usuario.js  x  Publicacion.js  x  Comentario.js  x
233
234     router.route('/comentario/hijo')
235     .post(function(req, res) {
236         Comentario.create({
237             comentario : req.body.texto,
238             usuarioId : req.body.usuarioId,
239             publicacionId : req.body.publicacionId,
240             principal : 0,
241             fecha : new Date()
242         }, function(err, comentario) {
243             if (err)
244                 res.send(err);
245
246             Comentario.findOne({_id: req.body.comentarioId}, function(err, com) {
247                 if (err)
248                     res.send(err);
249
250                 com.comentarioId.push(comentario._id);
251                 com.save(function(err){
252                     if (err)
253                         res.send(err);
254
255                 Comentario.find({publicacionId: req.body.publicacionId, principal: 1})
256                     .populate('comentarioId')
257                     .populate('usuarioId')
258                     .sort('-fecha')
259                     .exec(function(err, docs) {
260                         if(err) res.send(err);
261                         Comentario.populate(docs, {
262                             path: 'comentarioId.usuarioId',

```

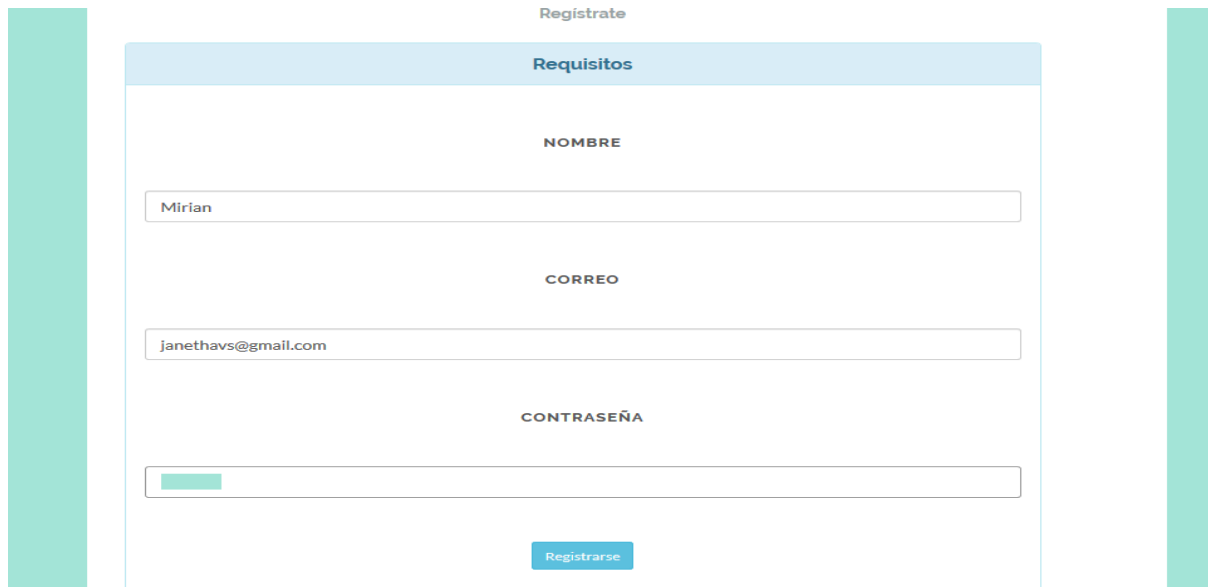
Diseño Web

La parte final es el diseño web, a continuación, se muestran las diferentes interfaces de interacción en la práctica realizada en el administrad

1. Pantalla de inicio para crear y editar o eliminar según sea el caso de una publicación en el blog.



2. pantalla para que un usuario nuevo cree una cuenta de usuario y pueda entrar al blog y hacer comentarios



3. En caso de ya contar con una cuenta de usuario es necesario que inicie sesión para poder hacer comentarios:

¿Ya eres usuario?

Iniciar sesión

Login

4. Pantalla en la que el usuario se dirige a leer el blog y crear comentarios

Blog...

08-12-2017

GESTION DE DATOS

hoy en día existen multiples...

Hacer un comentario

5. Así es como el usuario va a hacer comentarios y va a poder editar comentarios y eliminar si así lo desea.



6. En esta parte se puede observar la forma en que se anidan los comentarios, la estructura del comentario va en un arreglo en el cual se colocan los hijos (los comentarios anidados) y su longitud.

```

C:\Users\Janeth\Desktop\PracticaMongoDB>node server.js
Servidor online
GET /favicon.ico 404 1.745 ms - 24
GET /publicaciones 200 556.963 ms - 395
GET /publicaciones 200 398.113 ms - 395
POST /publicaciones 200 186.370 ms - 535
GET /publicaciones/5a2ad3b7c726ed1c4b6680ce 200 24.331 ms - 139
PUT /publicaciones/5a2ad3b7c726ed1c4b6680ce 200 101.952 ms - 538
GET /publicaciones 200 3.668 ms - 538
DELETE /publicaciones/5a2a3035db1329604e89e613 200 50.508 ms - 359
GET /publicaciones 200 6.922 ms - 359
GET /publicaciones 304 3.954 ms - -
GET /publicacion/gestion 200 33.521 ms - 142
[]
GET /comentarios/5a2ad3b7c726ed1c4b6680ce 200 11.046 ms - 2
POST /login 200 2.267 ms - 60
POST /login 200 2.819 ms - 129
GET /publicaciones 304 3.088 ms - -
GET /publicacion/gestion 304 2.993 ms - -
[]
GET /comentarios/5a2ad3b7c726ed1c4b6680ce 304 8.974 ms - -
POST /comentario 200 244.752 ms - 295
[ { comentarioId: [],
  v: 0,
  fecha: 2017-12-08T10:04:00.302Z,
  principal: 1,
  publicacionId: 5a2ad3b7c726ed1c4b6680ce,
  usuarioId:
    { v: 0,
  }
} ]
    
```

```

/ / / protocol:op_query 10ms
2017-12-08T12:04:06.525-0600 I COMMAND [conn4] command blogbasedatos1000.usuarios command: find { find: "usuarios", filter: { _id: { $in: [ ObjectId('5a2a31c0db1329604e89e614') ] } } } planSummary: IXSCAN { _id: 1 } keysExamined:1 docsExamined:1 cursorExhausted:1 numYields:1 nreturned:1 reslen:217 locks:{ Global: { acquireCount: { r: 4 } }, Database: { acquireCount: { r: 2 } }, Collection: { acquireCount: { r: 2 } } } protocol:op_query 127ms
    
```


Conclusiones

El trabajar con MongoDB es sin duda algo que es importante para la implementación de tecnología web, sobre todo porque permite manipular el formato JSON de una forma muy sencilla, por lo cual manejar API's es muy fácil y la comunicación es rápida, además el trato de la base de datos es más simple, aunque para proyectos muy amplios podría resultar deficiente por cuestiones de escalabilidad, aun así, existen versiones empresariales que permiten manejo de más datos.

MongoDB es sin duda una gran opción para probar el funcionamiento de las bases de datos NoSQL que no son para nada difícil de entender y aplicar. Además, su implementación al lado de NodeJS es muy eficiente ya que son del todo compatibles y se aprenden nuevos lenguajes que tienen un gran auge en la actualidad dentro del mundo del desarrollo web.