NODE.JS y MONGODB
CLASE 5 : MONGOOSE - SOCKET.IO

MONGOOSE

Mongoose proporciona una solución directa basada en esquemas para modelar los datos de su aplicación. Incluye casting de tipo incorporado, validación, creación de consultas, hooks de lógica de negocios y más, listo para usar.

La librería surge como consecuencia de la falta de validaciones dentro de un esquema de base de datos lo cual nos aseguraría una manera de perseverar el tipo de dato de un campo determinado de un registro.

Instalación

Podemos instalar Mongoose usando NPM de manera sencilla :

```
npm install --save mongoose
```

Uso

Lo primero que debemos hacer es incluir mongoose en nuestro proyecto y abrir una conexión a la base de datos de prueba en nuestra instancia local de MongoDB :

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');
```

Tenemos una conexión pendiente con la base de datos de prueba que se ejecuta en localhost. Ahora debemos recibir una notificación si nos conectamos correctamente o si ocurre un error de conexión:

```
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
  // we're connected!
});
```

Una vez que se abra nuestra conexión, se llamará a nuestra devolución de llamada. Para abreviar, supongamos que todo el código siguiente está dentro de esta devolución de llamada.

Con Mongoose, todo se deriva de un esquema.

Schema

Todo en Mongoose comienza con un Esquema. Cada esquema se asigna a una colección MongoDB y define la forma de los documentos dentro de esa colección.

```
var mongoose = require ('mangosta');
   var Schema = mongoose.Schema;

   var blogSchema = new Schema ({
     título: String,
     autor: String,
     cuerpo: String,
     comentarios: [{body: String, date: Date}],
     fecha: {type: Date, default: Date.now},
     oculto: Boolean,
     meta: {
       votos: Number,
       favs: Number
     }
   })
```

Cada clave en nuestro código blogSchema define una propiedad en nuestros documentos que se lanzará a su SchemaType asociado. Por ejemplo, hemos

definido un título de propiedad que se convertirá en String SchemaType y la fecha de la propiedad que se convertirá en un Date SchemaType. A las claves también se les pueden asignar objetos anidados que contengan más definiciones de clave / tipo, como la propiedad meta anterior.

Los SchemaTypes permitidos son:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array
- Decimal128
- Map

Los esquemas no solo definen la estructura de su documento y la conversión de propiedades, sino que también definen métodos de instancia de documento, métodos de Modelo estáticos, índices compuestos y ganchos de ciclo de vida de documento denominados middleware.

Para usar nuestra definición de esquema, necesitamos convertir su Esquema de blog en un Modelo con el que podamos trabajar. Para hacerlo, lo pasamos a mongoose.model(modelo, nombre, esquema).

Model

Los modelos son constructores sofisticados compilados a partir de definiciones de esquema. Una instancia de un modelo se llama documento. Los modelos son responsables de crear y leer documentos de la base de datos MongoDB subyacente.

```
var schema = new mongoose.Schema ({nombre: 'string', apellido:
'string'});
var Persona = mongoose.model('Persona, schema);
```

El primer argumento es el nombre singular de la colección para la que es su modelo. Mongoose busca automáticamente la versión plural del nombre de su

modelo. Por lo tanto, para el ejemplo anterior, el modelo Persona es para la colección de personas en la base de datos. La función .model () hace una copia del esquema.

Una instancia de un modelo se llama documento. Crearlos y guardarlos en la base de datos es fácil.

```
var juan = new Persona({ nombre: 'Juan' });
juan.save(function (err) {
  if (err) return handleError(err);
  // Guardado!
});

// or

Persona.create({ nombre : 'Juan' }, function (err, small) {
  if (err) return handleError(err);
  // Guardado!
});
```

Consultando

Encontrar documentos es fácil con Mongoose, que es compatible con la rica sintaxis de las consultas de MongoDB. Los documentos se pueden recuperar usando cada modelo find, findById, findOne, o donde estén los métodos estáticos.

```
Persona.find({ nombre : 'Juan' }).Where('createdDate').Gt
(unMesAtras).exec(callback);
```

Eliminando

Los modelos tienen funciones estáticas deleteOne () y deleteMany () para eliminar todos los documentos que coinciden con el filtro dado.

```
Persona.deleteOne({ nombre : 'Juan' }, function (err) {
  if (err) devuelve handleError (err);
```

```
  // eliminado como máximo un documento de persona
});
```

Actualizando

Cada modelo tiene su propio método de actualización para modificar documentos en la base de datos sin devolverlos a su aplicación. Vea los documentos de API para más detalles.

```
Persona.updateOne({ nombre : 'Juan' }, {nombre: 'Jose'}, function
(err, res) {
  // Actualizado como máximo un documento, `res.modifiedCount` contiene
el número
  // de documentos que MongoDB actualizó
});
```

Si desea actualizar un solo documento en el archivo db y devolverlo a su aplicación, use findOneAndUpdate en su lugar.

Dado que ahora controlamos nuestros datos a través de un Schema definido por Mongoose , no podremos actualizar ni insertar ningún registro o parte de él que no haya sido exactamente definido en nuestro esquema, permitiéndonos tener una capa de validación intermedia.

SOCKET.IO

El protocolo HTTP fue concebido desde sus orígenes para ofrecer comunicaciones en un sólo sentido, desde el servidor hacia el cliente. Sin embargo las aplicaciones web de hoy en día demandan más que eso para poder ofrecer una experiencia de usuario más rica, necesitan flujo de información en ambos sentidos en el mismo instante en el que ocurren los eventos.

Para mitigar esa necesidad han aparecido varias estrategias, entre ellas long polling y Websocket. En long polling el cliente se mantiene haciendo preguntas al servidor sobre un determinado evento mientras que con Websocket tenemos a nuestra disposición un nuevo protocolo que permite la interacción entre el cliente y el servidor, facilitando la transmisión de datos en tiempo real en ambas direcciones. Es aquí donde entra Socket.io.

Socket.io es una librería en JavaScript para Node.js que permite una comunicación bidireccional en tiempo real entre cliente y servidor. Para ello se basa principalmente en *Websocket* pero también puede usar otras alternativas como *sockets* de Adobe Flash, JSONP polling o long polling en AJAX, seleccionando la mejor alternativa para el cliente justo en tiempo de ejecución.

Instalación

Instalarlo es tan sencillo como ejecutar un comando de *npm*:

```
npm install socket.io
```

SOCKET.IO Y EXPRESS

Escribir una aplicación de chat con pilas de aplicaciones web populares como LAMP (PHP) ha sido tradicionalmente muy difícil. Implica interrogar al servidor por los cambios, mantener un registro de las marcas de tiempo, y es mucho más lento de lo que debería ser.

Para integrar un servidor de WebSockets con Express primero debemos levantar un servidor web Express :

```
var app = require('express')();
var http = require('http').Server(app);

app.get('/', function(req, res){
  res.send('<h1>Hola Mundo</h1>');
});

http.listen(3000);
```

Podemos inclusive servir contenido estático HTML utilizando métodos nativos de Node.js o Handlebars :

```
app.get('/', function(req, res){
     res.sendFile(__dirname + '/index.html');
});

//ó

app.get('/', function(req, res){
     res.render("template");
});
```

Ahora simplemente traemos el módulo de Socket.IO , el cual usa un servidor HTTP de Node.js :

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.get('/', function(req, res){
  res.sendFile(__dirname + '/index.html');
});
```

```
io.on('connection', function(socket){
  console.log('Un usuario se conectó');
});

http.listen(3000);
```

Observe que inicializo una nueva instancia de socket.io pasando el objeto http (el servidor HTTP). Luego escucho sobre el evento de conexión para los sockets entrantes, y lo registro en la consola.

Ahora en index.html agrego el siguiente fragmento antes del </ body>:

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();
</script>
```

Eso es todo lo que se necesita para cargar el socket.io-client, que expone io global, y luego se conecta.

Emitiendo eventos

La idea principal detrás de Socket.IO es que puedes enviar y recibir cualquier evento que desees, con cualquier información que desees. Cualquier objeto que pueda codificarse como JSON funcionará, y también se admitirán datos binarios.

Hagamos que cuando el usuario ingrese un mensaje, el servidor lo reciba como un evento de mensaje de chat. La sección de scripts en index.html debería verse ahora de la siguiente manera:

```
<script src="/socket.io/socket.io.js"></script>
<script src="https://code.jquery.com/jquery-1.11.1.js"></script>
<script>
  $(function () {
    var socket = io();
    $('form').submit(function(){
      socket.emit('chat message', $('#m').val());
      $('#m').val('');
      return false;
    });
  });
</script>
```

Y en index.js imprimimos el evento del mensaje de chat:

```
io.on('connection', function(socket){
  socket.on('mensaje de chat', function(msg){
    console.log('message: ' + msg);
  });
});
```

Broadcast

El próximo objetivo es que emitamos el evento desde el servidor al resto de los usuarios. Para enviar un evento a todos, Socket.IO nos da el io.emit:

```
io.emit ('algún evento', {para: 'todos'});
```

Si desea enviar un mensaje a todos excepto a un determinado socket, tenemos el indicador de transmisión:

```
io.on ('connection', función (socket) {
```

```
  socket.broadcast.emit ('hi');
});
```

En este caso, en aras de la simplicidad, enviaremos el mensaje a todos, incluido el remitente.

```
io.on ('connection', función (socket) {
  socket.on ('mensaje de chat', función (msg) {
    io.emit ('mensaje de chat', msg);
  });
});
```

Y en el lado del cliente cuando capturamos un evento de mensaje de chat lo incluiremos en la página. El código total de JavaScript del lado del cliente ahora equivale a:

```
<script>
  $ (función () {
    var socket = io ();
    $ ('form'). submit (función () {
      socket.emit ('mensaje de chat', $ ('# m'). val ());
      $ ('# m'). val ('');
      falso retorno;
    });
    socket.on ('mensaje de chat', función (msg) {
      $ ('# messages'). append ($ ('<li>'). text (msg));
    });
  });
</ script>
```


1. http://mongoosejs.com/
2. http://mongoosejs.com/docs/guide.html
3. http://mongoosejs.com/docs/schematypes.html
4. http://mongoosejs.com/docs/models.html
5. https://socket.io/
6. https://hipertextual.com/archivo/2014/08/socketio-javascript/

7. https://socket.io/get-started/chat/