



**M E A N**

**WEB FULL STACK DEVELOPER**

*Germán Caballero Rodríguez*  
*germanux@gmail.com*



# Cómo construir una API Rest

express

+

node JS

+



mongoDB



# INDICE

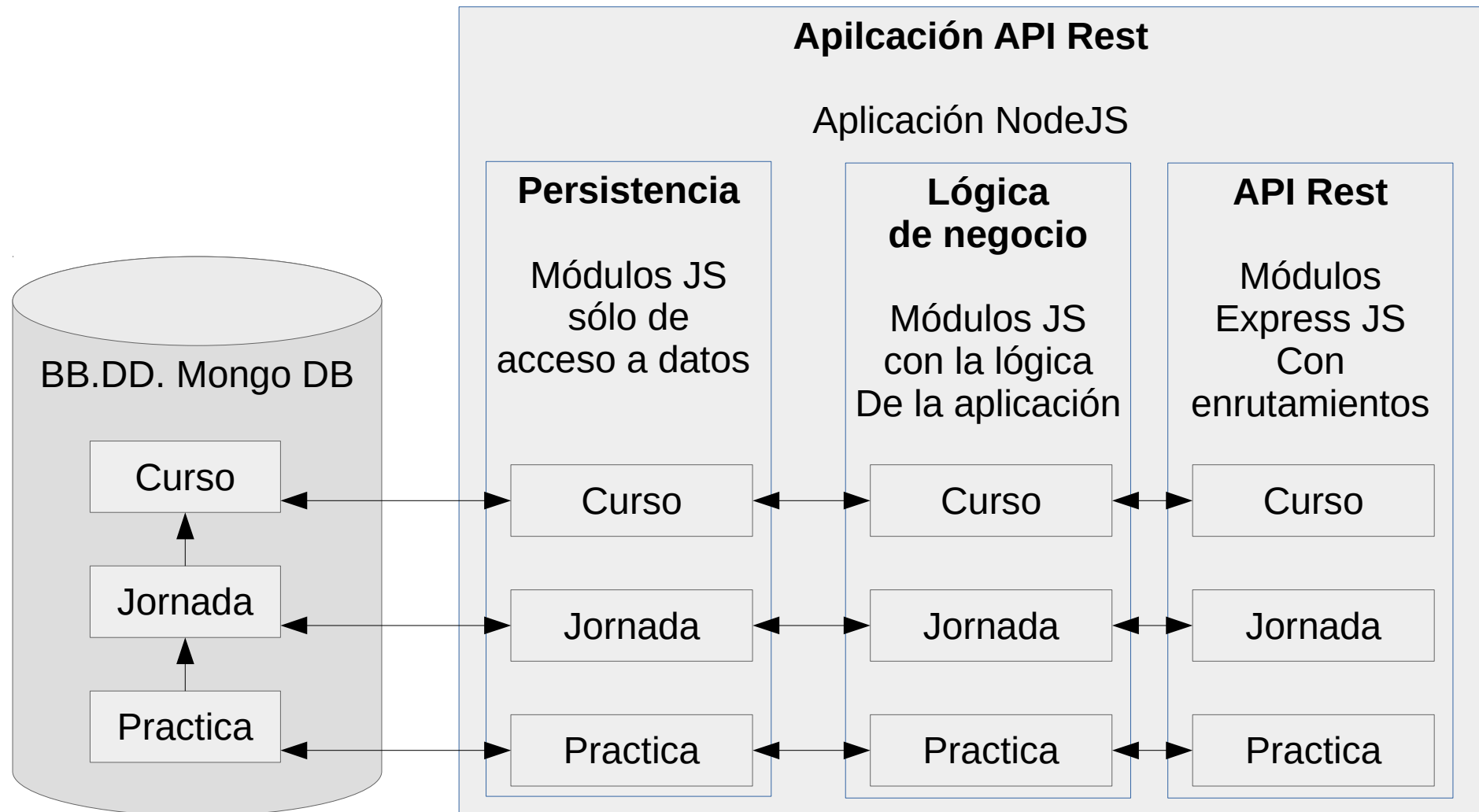
- 1) Requisitos
- 2) Diseño
- 3) Generación de la Aplicación
- 4) Intro Jade
- 5) Estructura API Restful
- 6) Estructura del proyecto
- 7) Assert
- 8) Conexión con MongoDB
- 9) API Rest contra MongoDB

# Requisitos

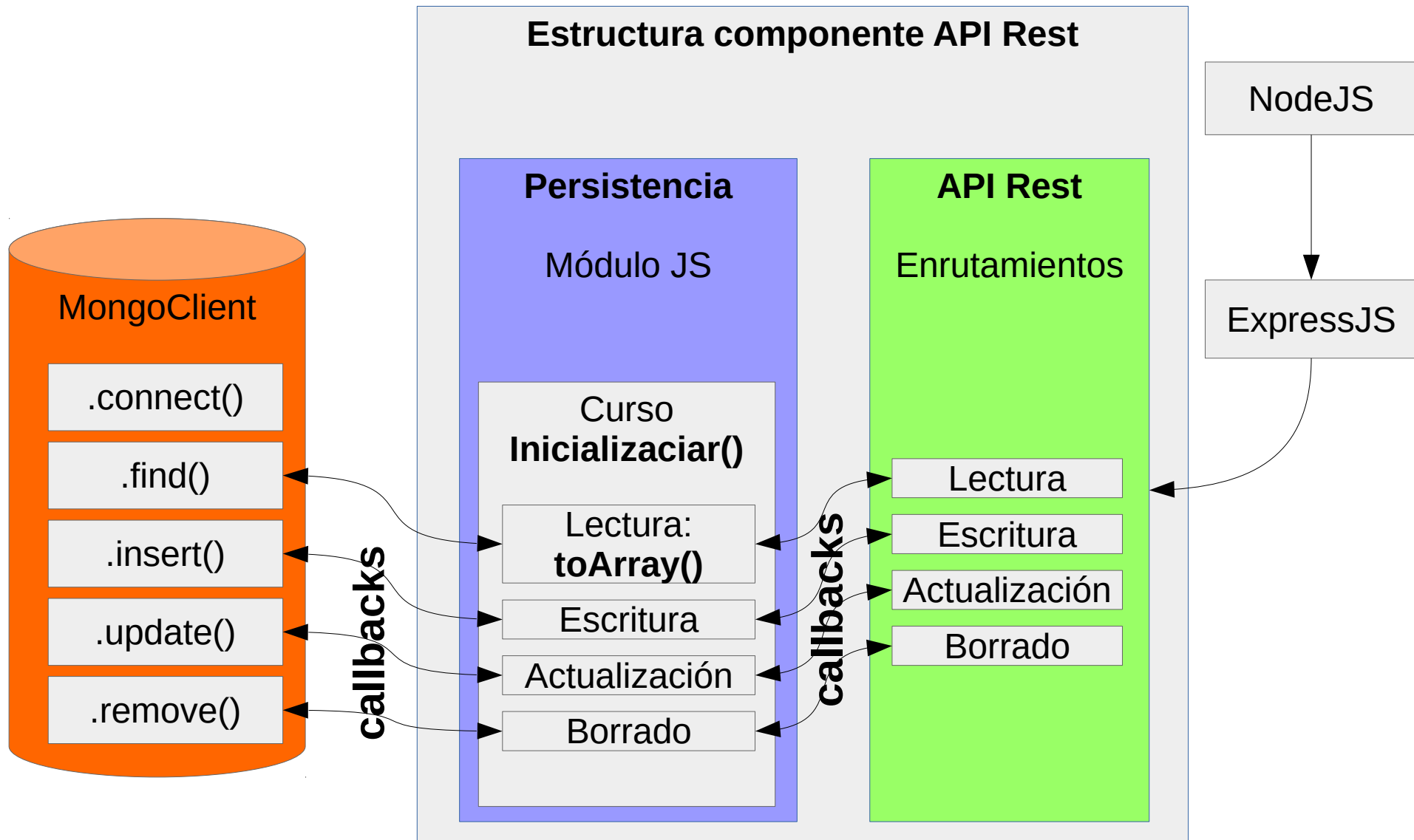
- API Rest para gestionar un curso:
  - Curso
    - Temario: Listado de tecnologías
    - Fecha inicio y fecha fin
  - Diario:
    - Listado de prácticas
      - Enunciado
      - Fichero/Directorio
      - Ruta
      - Fichero
      - Es ejercicio
    - Listado de teorías

# Diseño

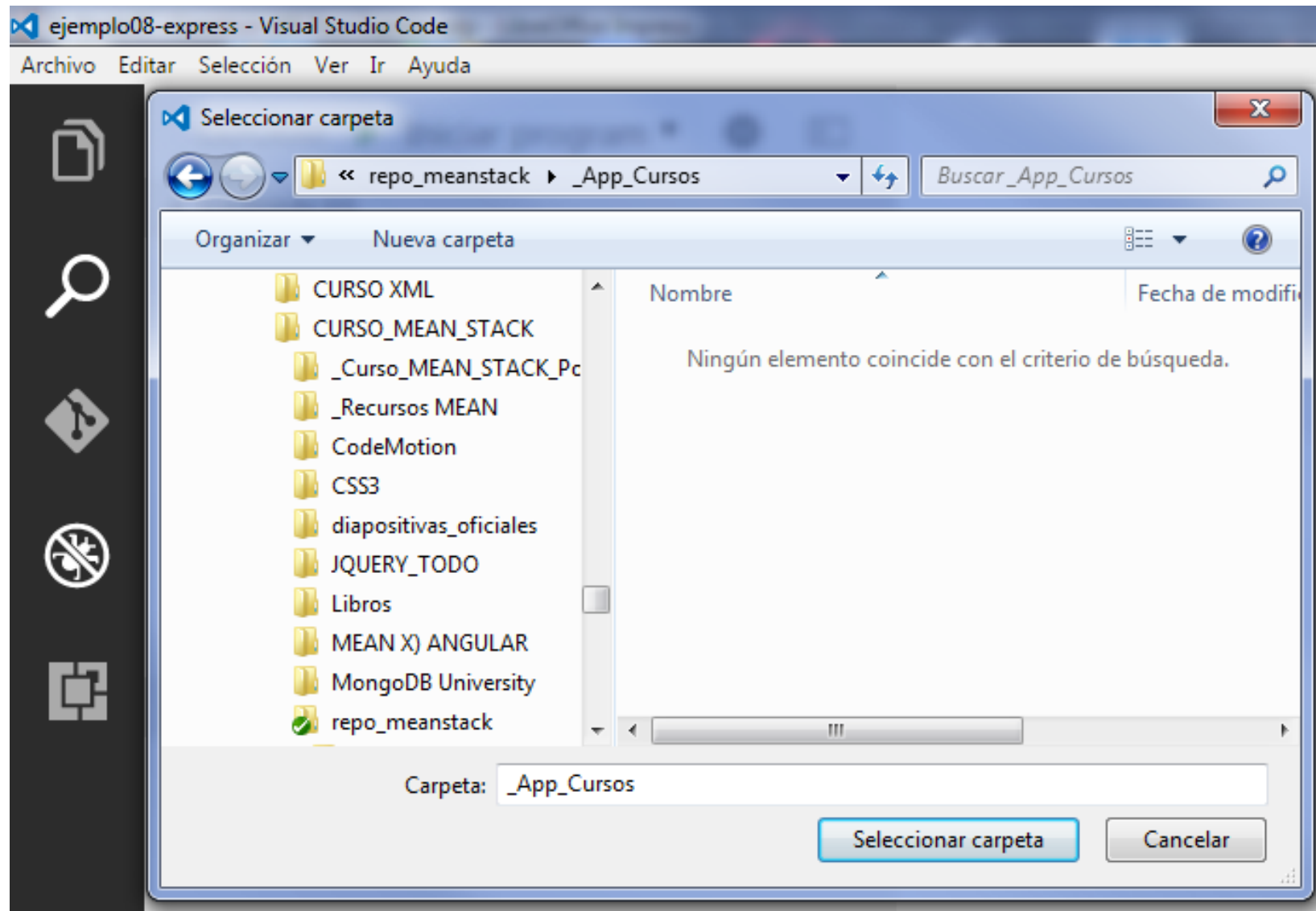
- Con Libre Office



# Diseño



# Generación de la app ExpressJS



# Generación de la app ExpressJS

- Generador de aplicaciones Express:
  - Utilice la herramienta de generador de aplicaciones, express, para crear rápidamente un esqueleto de aplicación.
  - Instale express con el siguiente mandato:
    - `D:\_App_Cursos>npm install express-generator -g`
  - Muestre las opciones de mandato con la opción -h:

```
D:\_App_Cursos>express -h
```

```
Usage: express [options] [dir]
```

```
Options:
```

-h, --help	output usage information
--version	output the version number
-e, --ejs	add ejs engine support
--pug	add pug engine support
--hbs	add handlebars engine support
-H, --hogan	add hogan.js engine support
-v, --view <engine>	add view <engine> support (ejs hbs hjs jade pug twig vash) (defaults to jade)
-c, --css <engine>	add stylesheet <engine> support (less stylus compass sass) (defaults to plain css)
--git	add .gitignore
-f, --force	force on non-empty directory



# Generación de la app ExpressJS

- El código siguiente crea una aplicación Express denominada **appcursos** en el directorio de trabajo actual:

```
D:\_App_Cursos>express --view=jade appcursos
```

```
create : appcursos
create : appcursos/package.json
create : appcursos/app.js
create : appcursos/public/javascripts
create : appcursos/routes
create : appcursos/routes/index.js
create : appcursos/routes/users.js
create : appcursos/public/images
create : appcursos/public
create : appcursos/views
create : appcursos/views/index.jade
create : appcursos/views/layout.jade
create : appcursos/views/error.jade
create : appcursos/public/stylesheets
create : appcursos/public/stylesheets/style.css
create : appcursos/bin
create : appcursos/bin/www
```

```
install dependencies:
```

```
> cd appcursos && npm install
```

```
run the app:
```

```
> SET DEBUG=appcursos:* & npm start
```

# Generación de la app ExpressJS

A continuación, instale las dependencias:

```
$ cd myapp  
$ npm install
```

En MacOS o Linux, ejecute la aplicación con este mandato:

```
$ DEBUG=myapp:* npm start
```

En Windows, utilice este mandato:

```
D:\_App_Cursos\appcursos>set DEBUG=appcursos:* & npm start
```

```
> appcursos@0.0.0 start D:\_App_Cursos\appcursos
```

```
> node ./bin/www
```

```
appcursos:server Listening on port 3000 +0ms
```

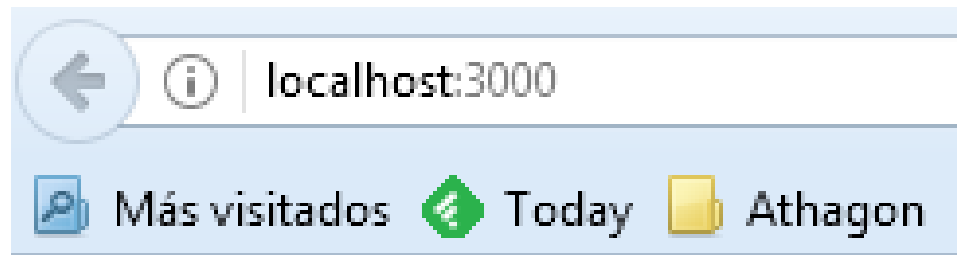
# Generación de la app ExpressJS

- La estructura de la aplicación creada por el generador es sólo una de las muchas formas de estructurar las aplicaciones Express.
- Podemos utilizar esta estructura o modificarla según nuestras necesidades.

```
.  
├── app.js  
├── bin  
│   └── www  
├── package.json  
├── public  
│   ├── images  
│   ├── javascripts  
│   └── stylesheets  
│       └── style.css  
├── routes  
│   ├── index.js  
│   └── users.js  
└── views  
    ├── error.jade  
    ├── index.jade  
    └── layout.jade
```

# Generación de la app ExpressJS

- A continuación, cargue `http://localhost:3000/` en el navegador para acceder a la aplicación.



## Express

Welcome to Express

# Generación de la app ExpressJS

- Mongoskin es un driver para conectar con MongoDB de una manera rápida y sencilla.
- Instalamos mongoskin:

```
D:\_App_Cursos\appcursos>npm install mongoskin --save
appcursos@0.0.0 D:\_App_Cursos\appcursos
+-- UNMET PEER DEPENDENCY mongodb@^2.0
`-- mongoskin@2.1.0
```

# Generación de la app ExpressJS

- Jade es un lenguaje de plantillas desarrollado por el creador de Express para simplificar la sintaxis de HTML y acelerar el proceso de desarrollo.
- Este lenguaje intercambia tener que cerrar las etiquetas HTML por la indentación, es decir, todo bloque de texto que esté hacia la derecha de la etiqueta que abre, significa que va dentro.
- También elimina los símbolos "<" y ">", y los parámetros de cada etiquetas se pasan entre paréntesis como si fuera una función.

# Generación de la app ExpressJS

- Sintaxis Básica de Jade:
  - Como podemos ver, se elimina el markup para darle al código un look mucho más limpio.
  - Si queremos agregar clases o id, solo tenemos que colocarlo al lado:
  - Podemos agregar todos los que queramos y combinarlos.

```
h1 Hola, mi nombre es
```

```
h2 y me gusta programar
```

```
ul
```

```
  li Javascript
```

---

```
h1.titulo Hola, mi nomb
```

```
h2.subtitulo y me gusta
```

```
ul#lenguajes
```

```
  li Javascript
```

```
h1.titulo Hola, mi nombre es
```

```
h2.subtitulo.cursiva y me gus
```

```
ul#lenguajes.lista
```

```
  li Javascript
```

# Estructura Restful API

Listar colección	GET	http://localhost/api:coleccion
Obtener un documento de la colección	GET	http://localhost/api:coleccion/id
Insertar un documento en la colección	POST	http://localhost/api:coleccion
Actualizar un documento	PUT	http://localhost/api:coleccion/id
Eliminar un documento	DELETE	http://localhost/api:coleccion/id

- La parte api permanecerá fija para indicar que estamos refiriéndonos a nuestra Restful API
- Las palabras colección e id se modificarán según nuestras llamadas.



# Estructura proyecto

- Descargamos u

# Assert

- Assert
  - El módulo assert proporciona un conjunto simple de pruebas de aserción que pueden usarse para probar invariantes.
  - El módulo está diseñado para su uso interno por Node.js, pero puede utilizarse en el código de aplicación a través de require ('assert').
  - Sin embargo, no es un marco de pruebas, y no está destinado a ser utilizado como una biblioteca de aserción de propósito general.
  - La API para el módulo de aserción está bloqueada.
  - Esto significa que no habrá adiciones o cambios en ninguno de los métodos implementados y expuestos por el módulo.

# Assert

## `assert(value[, message])`

Added in: v0.5.9

An alias of `assert.ok()`.

```
const assert = require('assert');

assert(true);
// OK

assert(1);
// OK

assert(false);
// throws "AssertionError: false == true"

assert(0);
// throws "AssertionError: 0 == true"

assert(false, 'it\'s false');
// throws "AssertionError: it's false"
```

## `assert.equal(actual, expected[, message])`

Prueba la igualdad superficial y coercitiva entre los parámetros real y esperado usando el operador de comparación igual (`==`).

```
const assert = require('assert');

assert.equal(1, 1);
// OK, 1 == 1

assert.equal(1, '1');
// OK, 1 == '1'

assert.equal(1, 2);
// AssertionError: 1 == 2

assert.equal({a: {b: 1}}, {a: {b: 1}});
//AssertionError: { a: { b: 1 } } == { a: { b: 1 } }
```

# Conexión con MongoDB

- Descargamos u

```
D:\_App_Cursos\appcursos>npm install mongodb --save
appcursos@0.0.0 D:\_App_Cursos\appcursos
`-- mongodb@2.2.23
   |
   |_ 2017-02-14T10:44:50.049+0100 I CONTROL [initandlisten] targetMinO
```

```
D:\_App_Cursos>mongod --dbpath bd_mongo_cursos
2017-02-14T10:44:50.049+0100 I CONTROL [initandlisten] MongoDB st
920 port=27017 dbpath=bd_mongo_cursos 64-bit host=PC-ATHAGON-GER
2017-02-14T10:44:50.050+0100 I CONTROL [initandlisten] targetMinO
indows Server 2008 R2
```

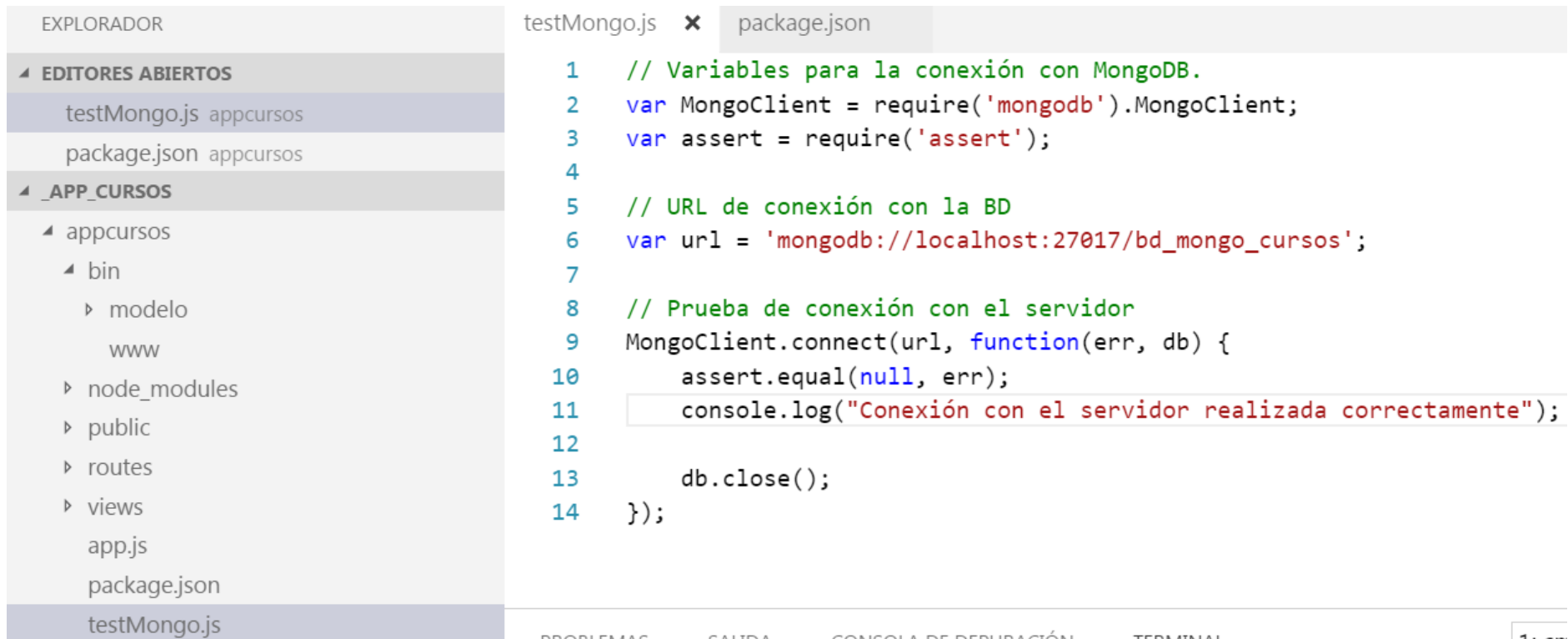
```
sos>mongo
v3.4.2
b://127.0.0.1:27017
```

# Conexión con MongoDB

- En primer lugar vamos a crear el fichero testMongo.js en el directorio raíz del proyecto.
- Este fichero servirá para comprobar que las dependencias se han instalado correctamente y que la aplicación se puede conectar con el servidor.

# Conexión con MongoDB

- El contenido de este fichero es el siguiente:



The screenshot shows a code editor interface. On the left is the 'EXPLORADOR' (Explorer) panel. It has two sections: 'EDITORES ABIERTOS' (Open Editors) and '\_APP\_CURSOS'. Under 'EDITORES ABIERTOS', 'testMongo.js appcursos' and 'package.json appcursos' are listed. Under '\_APP\_CURSOS', a tree structure is shown: 'appcursos' contains 'bin' (which contains 'modelo' and 'www'), 'node\_modules', 'public', 'routes', 'views', 'app.js', 'package.json', and 'testMongo.js'. The 'testMongo.js' file is selected. The main editor area shows the content of 'testMongo.js' with line numbers 1 through 14. The code connects to MongoDB and logs a success message. At the bottom, there are tabs for 'PROBLEMAS', 'SALIDA', 'CONSOLA DE DEPURACIÓN', and 'TERMINAL'. The 'TERMINAL' tab is active, showing '1 of 2'.

```
testMongo.js x package.json
1 // Variables para la conexión con MongoDB.
2 var MongoClient = require('mongodb').MongoClient;
3 var assert = require('assert');
4
5 // URL de conexión con la BD
6 var url = 'mongodb://localhost:27017/bd_mongo_cursos';
7
8 // Prueba de conexión con el servidor
9 MongoClient.connect(url, function(err, db) {
10   assert.equal(null, err);
11   console.log("Conexión con el servidor realizada correctamente");
12
13   db.close();
14 });
```

# Conexión con MongoDB

- Las primeras líneas sirven para cargar las dependencias de MongoDB y assert. Concretamente la primera sirve para cargar en la variable MongoClient el módulo encargado de la conexión con el servidor.
- La segunda línea carga en la variable assert el módulo encargado de las pruebas unitarias en Node.js.
- La siguiente línea sirve para definir la variable url que contendrá la cadena de conexión al servidor de Base de Datos de MongoDB en local.

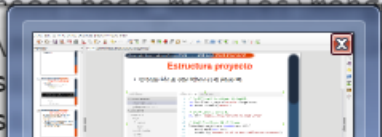
# Conexión con MongoDB

- Si no va bien:

```
D:\_App_Cursos\appcursos>node testMongo.js
```

```
D:\_App_Cursos\appcursos\node_modules\mongodb\lib\mongo_client.js:332
    throw err
    ^
```

```
AssertionError: null == { MongoError: failed to connect to server [localhost:27017] on first connect
    at Pool.<anonymous> (D:\_App_Cursos\appcursos\n
    at D:\_App_Cursos\appcursos\testMongo.js:10:12
    at connectCallback (D:\_App_Cursos\appcursos\node_modules\mongodb\lib\mongo_client.js:422:5)
    at D:\_App_Cursos\appcursos\node_modules\
    at _combinedTickCallback (internal/process
    at process._tickCallback (internal/process
```





# Conexión con MongoDB

- Añadiendo datos
- Por el momento la base de datos está vacía por lo que empezaremos introduciendo datos.
- En un directorio bin/modelo, creamos un JS que será el módulo de acceso a los cursos

# Conexión con MongoDB

- Ejemplo de inserción de datos

```
// Prueba para la inserción de datos.
var insertarPelículas = function(db, callback) {
  // Carga de la colección de películas.
  var collection = db.collection('películas');

  // Inserción de algunas películas
  collection.insert([
    {titulo : 'Akira', director : 'Katsuhiro Otomo'},
    {titulo : 'La guerra de las galaxias', director : 'George Lucas'},
    {titulo : 'Blade runner', director : 'Ridley Scott'}
  ], function(err, docs) {
    // Tests unitarios
    assert.equal(err, null);
    assert.equal(3, docs.result.n);
    assert.equal(3, docs.ops.length);

    // Log de consola
    console.log("Insertadas películas en la colección de películas.");

    callback(docs);
  });
}
```

# Conexión con MongoDB

- Conexión y llamada a la función de inserción

```
// Conexión con el servidor para la consulta de datos
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Conexión al servidor para las operaciones CRUD");

  // Invocación encadenada de las operaciones.
  insertarPeliculas(db, function() {
    db.close();
  });
});
```

# Conexión con MongoDB

- Consulta de datos

```
// Pruebas para la lectura de datos insertados
var leerPelículas = function(db, callback) {
  // Carga de la colección de películas.
  var collection = db.collection('películas');

  // Consulta de los documentos (películas) de la colección
  collection.find({}).toArray(function(err, res) {
    assert.equal(err, null);
    assert.equal(3, res.length);
    console.log("Se encontraron las siguientes películas");
    console.dir(res)
    callback(res);
  });
}
```

# Conexión con MongoDB

- Consulta de datos

```
// Pruebas para la lectura de datos insertados
var leerPelículas = function(db, callback) {
  // Carga de la colección de películas.
  var collection = db.collection('películas');

  // Consulta de los documentos (películas) de la colección
  collection.find({}).toArray(function(err, res) {
    assert.equal(err, null);
    assert.equal(3, res.length);
    console.log("Seencontrado las siguientes películas");
    console.dir(res)
    callback(res);
  });
}
```

método `find({})`. Este método hace la búsqueda en la colección con los parámetros de búsqueda dados dentro de las llaves.

Además se invoca el método `toArray` para formatear el resultado y poder pasarle la función encargada de hacer los tests unitarios y sacar por pantalla los resultados.

# Conexión con MongoDB

- Invocar a la función de lectura de datos

```
// Conexión con el servidor para la consulta de datos
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Conexión al servidor para las operaciones CRUD");

  // Invocación encadenada de las operaciones.
  leerPelículas(db, function() {
    db.close();
  });
});
```

Si queremos devolver los datos, en la función callback pasada a leerPelículas tendremos que añadir un parámetro que recibirá los resultados (ver anterior

# Conexión con MongoDB

- borrado de los datos.

```
// Pruebas para el borrado de películas
var borrarPelículas = function(db, callback) {
  // Carga de la colección de películas.
  var collection = db.collection('películas');

  // Borrar la película con el título Akira
  collection.remove({ titulo : 'Akira' }, function(err, res) {
    assert.equal(err, null);
    assert.equal(1, res.result.n);
    console.log("Se eliminado la película con el título Akira");
    callback(res);
  });
}
```

Se invoca el método update con 2 parámetros:

- 1) El primer parámetro es un filtro que indica qué documentos se desean eliminar. En este caso se empieza eliminando la película cuyo título es 'Akira'.
- 2) El segundo parámetro es la postfunción que se va a ejecutar tras el borrado de documentos. En este caso la función hace unas pruebas unitarias y muestra un mensaje por consola.
- 3)

# Conexión con MongoDB

- Implementación del módulo modelo.js

```
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');
var url = 'mongodb://localhost:27017/bd_mongo_cursos';

var modelo = {
  acceder: function(accion, miFuckingCallback) {
    // Conectar e insertar
    MongoClient.connect(url, function(err, db) {
      assert.equal(null, err);
      console.log("Conexión con el servidor realizada correctamente");

      modelo.cursos[accion](db, function(res) {

        console.log("Cerrando base de datos");
        db.close();
        if (typeof miFuckingCallback == "function")
          miFuckingCallback(res);
      })
    });
  },
  cursos: {
    insertarVarios: function(db, callback) {
```



# Conexión con MongoDB

- Implementación del módulo modelo.js

```
...  
},  
cursos: {  
  insertarVarios: function(db, callback) {  
    // Carga de la colección de cursos.  
    var collection = db.collection('cursos');  
  
    collection.insert([  
      { titulo: 'MEAN Full Stack Developer', maestro: 'Germán Caballero' },  
      { titulo: 'Desarrollo con Apache Cordova', maestro: 'Alfonso El Sabio' },  
      { titulo: 'Curso de patrones de diseño', maestro: 'Sergio el programador' },  
      { titulo: 'Curso de desarrollo web', maestro: 'Daniel Algo Más' },  
      { titulo: 'Curso de programación aleatoria', maestro: 'Francisco el Vietmanita' }  
    ], function(err, docs) {  
      assert.equal(err, null);  
      assert.equal(5, docs.result.n);  
      assert.equal(5, docs.ops.length);  
  
      console.log("Insertados cursos en las colección de cursos.");  
      callback(docs);  
    });  
  },  
}
```

# Conexión con MongoDB

- Implementación del módulo modelo.js

```
        console.log("Insertados cursos en las colección de cursos.");  
        callback(docs);  
    });  
},  
leerCursos: function(db, callback) {  
    var collection = db.collection('cursos');  
    var arrayCursos = collection.find({}).toArray(function(err, res) {  
        assert.equal(err, null);  
        assert.notEqual(0, res.length);  
        console.log("Se han encontrado los siguientes cursos");  
        console.dir(res)  
        callback(res);  
    });  
    return arrayCursos;  
}
```

```
    // LLamamos a la función de inicializar  
    modelo.acceder("insertarVarios");  
    module.exports = modelo;
```

# API Rest contra MongoDB

- En el fichero app.js

```
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
/* IMPORTART EL MODELO*/
var modelo = require("../bin/modelo/modelo");

var index = require('./routes/index');
var users = require('./routes/users');
/* IMPORTAR EL MODULO DE RUTAS DE CURSOS */
var cursos = require('./routes/cursos');
```

# API Rest contra MongoDB

- En el fichero app.js

```
app.use(express.static(path.join(__dirname, 'public')));
```

```
app.use('/', index);
```

```
app.use('/users', users);
```

```
/* CREAR LA RUTA DE CURSOS */
```

```
app.use('/cursos', cursos);
```

```
// catch 404 and forward to error handler
```

# API Rest contra MongoDB

- En el fichero cursos.js

```
var express = require('express');
var modelo = require('../bin/modelo/modelo');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  modelo.acceder("leerCursos", (datos) => {
    res.send(datos);
  });
});

module.exports = router;
```