

## Ejercicios Tema 4

### Ejercicio 1

En primer lugar vamos a ver como instalar MongoDB.

#### \*En MacOS

La forma más cómoda de obtener MongoDB es emplear el gestor de paquetes **brew**, para instalarlo, escribimos en un terminal -> `ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

A continuación, tecleamos en el terminal

```
$ brew install mongodb
```

Una vez descargado para lanzar el servidor de mongo empleamos el comando **mongod**.

El comando fallará si no existe el directorio `/data/db` que emplea por defecto o si el directorio que especificamos con el flag `- - dbpath` no existe. También fallará si no tenemos permisos sobre el directorio.

```
$ mkdir -p /data/db
```

Si todo ha ido bien la terminal debería reconocer el comando `mongod` y lanzar el servidor.

```
waiting for connections on port 27017
```

#### \*En Windows

Desde la página de MongoDB hacemos click en el link de descarga:

<http://www.mongodb.org/downloads>

Escogemos la versión apropiada para la arquitectura de nuestro SO.

Hacemos doble click en el fichero `.msi` y seguimos el proceso de instalación.

Para lanzar el servidor de MongoDB escribiremos en la consola o en un PowerShell el siguiente comando:

```
C:\mongodb\bin\mongod.exe
```

Suponiendo que hemos escogido `C:\mongodb` como el directorio de instalación

El comando fallará si no existe el directorio `/data/db` que emplea por defecto o si el directorio que especificamos con el flag `- - dbpath` no existe.

Crear el directorio:

```
md \data\db
```

Especificar el path de otro directorio:

```
C:\mongodb\bin\mongod.exe --dbpath d:\test\mongodb\data
```

Finalmente la consola debería reconocer el comando mongod.exe y lanzar el servidor que quedará esperando en el puerto 27017.

## Interactuar con MongoDB

Para lanzar la versión de consola de MongoDB tecleamos el comando **mongo**. En windows **mongo.exe** dentro de la carpeta bin del directorio de instalación.

```
$ mongo
```

```
MongoDB shell version: 2.6.6
connecting to: test
>
```

Aquí ya podremos teclear operaciones para interactuar con las colecciones y sus documentos. Empezaremos insertando varios documentos dentro de la colección personas. la colección se creará automáticamente cuando intentemos introducir su primer documento.

```
db.tareas.insert({
  title      : 'establecer requisitos',
  description: 'requisitos del proyecto definidos y consensuados con
cliente',
  status     : 'done',
  tags       : ['desarrollo', 'diseño', 'marketing', 'finanzas']
});

db.tareas.insert({
  title      : 'presupuestar',
  description: 'presupuesto acordado y cerrado',
  status     : 'done',
  tags       : ['desarrollo', 'diseño', 'marketing', 'finanzas']
});

db.tareas.insert({
  title      : 'comprar dominio',
  description: 'estudio de dominio y compra',
  status     : 'done',
  tags       : ['desarrollo', 'finanzas']
});

db.tareas.insert({
```

```

    title      :   'desarrollo front',
    description:   'html5/css3/angular',
    status     :   'review',
    tags       :   ['desarrollo', 'diseño', 'marketing']
  });
  db.tareas.insert({
    title      :   'desarrollo back',
    description:   'desarrollo en node',
    status     :   'doing',
    tags       :   ['desarrollo']
  });
  db.tareas.insert({
    title      :   'implementación',
    description:   'juntar todas las piezas',
    status     :   'doing',
    tags       :   ['desarrollo', 'diseño']
  });
  db.tareas.insert({
    title      :   'test',
    description:   'funciona?',
    status     :   'doing',
    tags       :   ['desarrollo', 'diseño', 'marketing']
  });
  db.tareas.insert({
    title      :   'revision SEO',
    description:   'Sin SEO y viralización no somos nadie',
    status     :   'todo',
    tags       :   ['desarrollo', 'diseño', 'marketing']
  });
  db.tareas.insert({
    title      :   'vacaciones pre lanzamiento',
    description:   'si llegamos a la fecha, vacaciones',
    status     :   'todo',
    tags       :   ['desarrollo']
  });
  db.tareas.insert({
    title      :   'lanzamiento',
    description:   'party hard.',
    status     :   'todo',
    tags       :   ['desarrollo', 'diseño', 'marketing', 'finanzas']
  });
  db.tareas.insert({
    title      :   'mantenimiento',
    description:   'no caer y crecer',
    status     :   'todo',
    tags       :   ['desarrollo', 'diseño', 'marketing', 'finanzas']
  });

```

Con el comando `show collections` podemos ver las colecciones de nuestra db. Para consultar los documentos dentro de una colección empleamos el comando `find`:

```
> db.personas.find()
{ "_id" : ObjectId("54c66c87e1b68b34397f4ad3"), "nombre" : "Claudio",
  "apellido" : "Rivero", "cursos" : [ "Ruby", "Symfony" ] }
{ "_id" : ObjectId("54c66e2722f7e1c1b18132d6"), "nombre" : "Roberto",
  "apellido" : "Losa", "cursos" : [ "CSS" ] }
{ "_id" : ObjectId("54c66e2722f7e1c1b18132d7"), "nombre" : "Agustín",
  "apellido" : "Roldán", "cursos" : [ "Android", "IOS" ] }
{ "_id" : ObjectId("54c66e2722f7e1c1b18132d8"), "nombre" : "Jorge",
  "apellido" : "Silva", "cursos" : [ "HTML", "Silverlight" ] }
{ "_id" : ObjectId("54c66e2722f7e1c1b18132d9"), "nombre" : "Mario",
  "apellido" : "Girón", "cursos" : [ "Django", "Node.js" ] }
```

Podemos añadir todas las condiciones que queramos a las consultas:

```
> db.tareas.find({'title':/vacaciones/})
{ "_id" : ObjectId("564332a41f6decfc2870c8cf"), "title" : "vacaciones pre
lanzamiento", "description" : "si llegamos a la fecha, vacaciones",
  "status" : "todo", "tags" : [ "desarrollo" ] }

> db.tareas.find({tags:'finanzas',status:'todo'})
{ "_id" : ObjectId("564332a41f6decfc2870c8cf"), "title" : "vacaciones pre
lanzamiento", "description" : "si llegamos a la fecha, vacaciones",
  "status" : "todo", "tags" : [ "desarrollo" ] }

> db.tareas.find({tags:'finanzas',status:'todo'})
{ "_id" : ObjectId("5643304b1f6decfc2870c8cd"), "title" : "lanzamiento",
  "description" : "party hard.", "status" : "todo", "tags" :
  [ "desarrollo", "diseño", "marketing", "finanzas" ] }
{ "_id" : ObjectId("564330561f6decfc2870c8ce"), "title" :
  "mantenimiento", "description" : "no caer y crecer", "status" : "todo",
  "tags" : [ "desarrollo", "diseño", "marketing", "finanzas" ] }
```

Para borrar un documento utilizamos el método `remove`:

```
> db.tareas.remove({'title':/vacaciones/})
WriteResult({ "nRemoved" : 1 })
```

## Ampliación

Robomongo es una herramienta de gestión de MongoDB con interfaz gráfica que facilita la comprensión y hace menos tedioso trabajar con la BD.

Instala la herramienta y realiza más cambios en la BD observando los resultados a través del explorador de colecciones.

<http://robomongo.org/>

## Ejercicio 2

En este ejercicio vamos a migrar los datos del array original a mongodb añadiendo funcionalidades CRUD. Confirmamos que funciona por el momento:

```
$ DEBUG=imagina_todo:* ./bin/www
```

El siguiente paso es añadir mongoose a nuestro proyecto y crear el modelo del recurso.

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

Creamos el fichero **Tarea.js** dentro del directorio **models**.

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

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

/**
 * Getters
 */
var getTags = function (tags) {
  return tags.join(', ');
};

/**
 * Setters
 */
var setTags = function (tags) {
  return tags.split(', ');
};

var TareaSchema = new Schema({
```

```

    title: String,
    description: {type : String, default : '', trim : true},
    status:{type: String, enum: ['todo', 'doing', 'review', 'done']},
    tags: {type: [], get: getTags, set: setTags},
    createdAt : {type : Date, default : Date.now}
  });

module.exports = mongoose.model('Tarea', TareaSchema);

```

Posteriormente creamos un controlador llamado **TareaController.js** en el directorio **controllers**. En este fichero especificaremos las distintas rutas para manipular los recursos.

```

var tareas = require('express').Router(),
Tarea = require('../models/Tarea.js');

tareas.get('/', function(req, res) {
  //TODO Obtener todos los documentos
  res.send("Funciona");
});

tareas.post('/', function(req, res) {
  //TODO Nuevo documento
});

tareas.get('/:id', function(req, res) {
  //TODO Obtener documento por id
});

module.exports = tareas;

```

En app.js cargamos las nuevas rutas que hemos definido:

```
var tareas = require('./controllers/TareaController');

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

Verificamos que express detecta nuestro controlador correctamente lanzando una petición al servidor:

```
$ npm start
```

<http://localhost:3000/tareas>

Nos debe devolver el texto *'Funciona'* en el navegador además de un código 200.

```
GET /tareas 200
```

Una vez comprobado el funcionamiento de las rutas vamos a agregar el comportamiento deseado a cada una de ellas en **TareaController.js**.

```
//Obtener todos los documentos
tareas.get('/', function(req, res) {
  Tarea.find(function(err, list){
    if(req.accepts('json')) {
      if(err) {
        return res.json(500, {
          message: 'Error getting tareas.'
        });
      }
      return res.json(list);
    } else {
      if(err) {
        return res.send('500: Internal Server Error', 500);
      }
      return res.render('tareas/index', {tareas: tareas});
    }
  });
});
```

```
//Nuevo documento
tareas.post('/', function(req, res) {

  var tarea = new Tarea({
    title: req.body['title'],
    description: req.body['description'],
    status: req.body['status'],
    tags: req.body['tags']
```

```

});

tarea.save(function(err, tareas){
  if(req.accepts('json')) {
    if(err) {
      return res.json(500, {
        message: 'Error saving item.',
        error: err
      });
    }
    return res.json({
      message: 'saved',
      _id: tarea._id
    });
  } else {
    if(err) {
      return res.send('500: Internal Server Error', 500);
    }
    return res.render('tarea/edit', {tarea: tarea});
  }
});

});

//Obtener documento por id
tareas.get('/:id', function(req, res) {

  var id = req.params.id;
  console.log('find id:'+id);
  Tarea.findOne({_id: id}, function(err, tarea){
    if(req.accepts('json')) {
      if(err) {
        return res.json(500, {
          message: 'Error getting tarea.'
        });
      }
      if(!tarea) {
        return res.json(404, {
          message: 'No such tarea.'
        });
      }
      return res.json(tarea);
    } else {
      if(err) {
        return res.send('500: Internal Server Error', 500);
      }
      if(!tarea) {
        return res.end('No such tarea');
      }
    }
  });
});

```



```

    return res.render('tareas/edit', {tarea: tarea, flash:
'Created.'});
  }
});

});

module.exports = tareas;

```

Ahora ya podremos manipular nuestra colección de tareas con las distintas URL que hemos preparado.

Las peticiones tipo **GET** podemos probarlas con nuestro navegador sin embargo para probar el resto de peticiones tendremos que usar alguna herramienta adicional como **curl** o **postman** o comentar partes de nuestro código y usar nuestro navegador para hacer gets.

The screenshot shows the Postman web client interface. At the top, the method is set to 'POST' and the URL is 'http://localhost:3000/tareas/'. There are buttons for 'Params', 'Send', and a save icon. Below the URL bar, there are two tables for 'Path variable key' and 'URL Parameter Key', each with a 'Value' column and an edit icon. The 'Body' tab is selected, showing a table with four rows: 'title' (new work!), 'description' (nobody expects the...), 'status' (todo), and 'tags' (spanish inquisition). Each row has a blue checkmark in the first column. Below the table, there are radio buttons for 'form-data' (selected), 'x-www-form-urlencoded', 'raw', and 'binary'. At the bottom, there is a 'Key' and 'Value' table with an edit icon.

Path variable key	Value
URL Parameter Key	Value

Authorization	Headers (0)	Body	Pre-request script	Tests
<input type="radio"/> form-data <input checked="" type="radio"/> x-www-form-urlencoded <input type="radio"/> raw <input type="radio"/> binary				
✓	title	new work!		
✓	description	nobody expects the...		
✓	status	todo		
✓	tags	spanish inquisition		
	Key	Value		

## Ejercicio 2

Elabora views que permitan listar las tareas (GET /), crear nuevas (GET /new que muestre un formulario y se envíe a POST /) y que permitan editar (GET /edit/:id que envíe a POST /edit/:id) a **partir de los html que tienes en los recursos** (o genera tus propias vistas). Llegados a este punto puedes elegir si utilizar JADE para las vistas ( este enlace puede serte útil <http://html2jade.org> ), SWIG/TWIG o Handlebarsjs. Para ello emplea el método render del objeto res especificando la view que se debe renderizar.

Por cierto, no te preocupes si no entiendes todo el html y css que hay en los html, se ha utilizado [bootstrap v4 \(alpha\)](#) para generarlas.

Es importante que te sientas cómodo con el sistema de plantillas utilizado, ya que te van a acompañar.

