Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

# Práctica 5
# Express

## Sistemas y Tecnologías Web

## Grado en Ingeniería Informática

Curso 2015-2016

Francisco Javier Fabra Caro

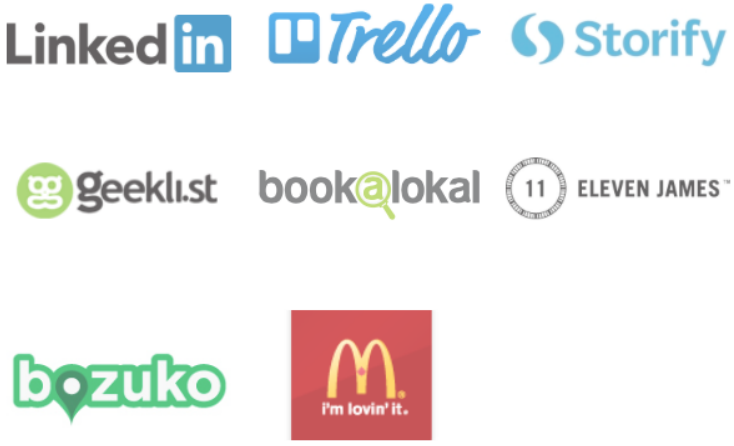
jfabra@unizar.es

# mongoose

http://mongoosejs.com

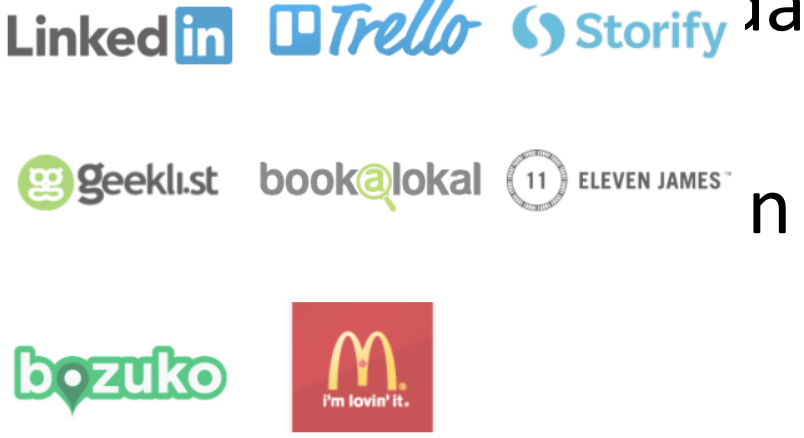
http://mongoosejs.com/docs/guide.html

http://mongoosejs.com/docs/api.html

# mongoose

- Complemento de MongoDB que evita los *problemas* de la validación, conversión y lógica de negocio detrás de Mongo
  - Realmente es un wrapper sobre el driver nativo de MongoDB
  - El sitio oficial de MongoDB también recomienda este módulo
- Mongoose proporcio la en schemas para mo
- Incluye conversión, v n de queries etc



# Guía rápida

- Requiere NodeJS y MongoDB instalados

1. Instalación
2. Creación de un schema
3. Añadir un método de documento personalizado
4. Almacenar documentos con el schema
5. Interrogar la BdD

http://mongoosejs.com/docs/index.html

# Instalación

```
$ npm install mongoose
```

# Conexión con la BdD

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
      // Nos hemos conectado
      // A partir de aquí, añadiremos el código que queramos ejecutar dentro del callback
});
```

# Definición y manejo de *schemas*

```
// Schema con una única propiedad, “name”
var kittySchema = mongoose.Schema({
    name: String
});

// Ahora compilamos el schema para generar un modelo
var Kitten = mongoose.model('Kitten', kittySchema);

// Un documento será un kitten con las propiedades definidas en el schema
var silence = new Kitten({ name: 'Silence' });
console.log(silence.name); // 'Silence’
```

# Adición de funcionalidades a los documentos

```
// ATENCIÓN: los métodos se deben añadir al schema antes de compilarlo en un modelo

kittySchema.methods.speak = function () {
  var greeting = this.name
    ? "Meow name is " + this.name
    : "I don't have a name";
  console.log(greeting);
}

var Kitten = mongoose.model('Kitten', kittySchema);

var fluffy = new Kitten({ name: 'fluffy' });
fluffy.speak(); // "Meow name is fluffy"
```

# Guardar/buscar documentos en MongoDB

```
// Guardar un documento
fluffy.save(function (err, fluffy) {
  if (err) return console.error(err);
  fluffy.speak();
});
```

```
// Buscar un documento
Kitten.find(function (err, kittens) {
  if (err) return console.error(err);
  console.log(kittens);
})
```

```
// Utilizar filtros en la búsqueda
Kitten.find({ name: /^Fluff/ }, callback);
// Buscaremos todos los documentos que tengan una propiedad “name” que comience con
// “Fluff” y devolveremos el resultado como un array de kittens al callback
```

# Express.js

**http://expressjs.com**

**http://expressjs.com/en/guide/routing.html**

**http://expressjs.com/en/4x/api.html**

# Creación de un servicio sencillo

```
$ npm init

// Vamos respondiendo a las preguntas o usamos este JSON en el fichero que crea al final
package.json
{
    "name": "nodejs-api-sample",
    "version": "1.0.0",
    "description": "An example of making a Node.js API server",
    "author": "Nic Raboy",
    "license": "MIT"
}


// Instalaremos express en el proyecto
npm install express --save

// Y el paquete body-parser, que nos permitirá procesar peticiones POST
npm install body-parser --save
```

# Creación de un servicio sencillo

**SimpleAPI**
**—-> package.json**
**—-> app.js**
**—-> routes**
**——-> routes.js**

**app.js**

```
var express = require("express");
var bodyParser = require("body-parser");
var app = express();

// Aceptaremos JSON y valores codificados en la propia URL
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// Todos los endpoint del API los colocaremos en este fichero
var routes = require("./routes/routes.js")(app);

var server = app.listen(3000, function () {
    console.log("Listening on port %s...", server.address().port);
});
```

# Creación de un servicio sencillo

**routes/routes.js**

```
var appRouter = function(app) {

     app.get("/", function(req, res) {
         res.send("Hello World");
     });

     app.get("/account", function(req, res) {
         var accountMock = {
             "username": "nraboy",
             "password": "1234",
             "twitter": "@nraboy"
         }
         if(!req.query.username) {
             return res.send({"status": "error", "message": "missing username"});
         } else if(req.query.username != accountMock.username) {
             return res.send({"status": "error", "message": "wrong username"});
         } else {
             return res.send(accountMock);
         }
     });
}

module.exports = appRouter;
```

# Creación de un servicio sencillo

```
app.post("/account", function(req, res) {
    if(!req.body.username || !req.body.password || !req.body.twitter) {
        return res.send({"status": "error", "message": "missing a parameter"});
    } else {
        return res.send(req.body);
    }
});

app.put("/account", function(req, res) {
        console.log("PUT API request..");
        return res.send("{status: ok}");
});

app.delete("/account", function(req, res) {
        console.log("DELETE API request..");
        return res.send("{status: ok}");
});

// Instalaremos las dependencias NPM
$ npm install

// Y ya podemos ejecutar la aplicación
$ node app.js
```

# Aplicación Web de gestión de usuarios

**package.json**

```
{
  "name": "gestionUsuarios",
  "version": "1.0.0",
  "description": "Aplicación de gestión de usuarios",
  "main": "main.js",
  "dependencies": {
    "body-parser": "^1.13.3",
    "express": "^4.13.3",
    "mongoose": "^4.1.2"
  }
}
```

**$ npm install**

# Aplicación Web de gestión de usuarios

**server.js**

```
var express     =   require("express");
var app         =   express();
var bodyParser  =   require("body-parser");
var router      =   express.Router();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({"extended" : false}));

router.get("/",function(req,res){
    res.json({"error" : false,"message" : "Hello World"});
});

app.use('/',router);

app.listen(3000);
console.log("Listening to PORT 3000");
```

**$ npm start**

# Aplicación Web de gestión de usuarios

**$ mkdir models**

**models/mongo.js**

```
var mongoose    =   require("mongoose");
mongoose.connect('mongodb://localhost:27017/demoDb');
// create instance of Schema
var mongoSchema =   mongoose.Schema;
// create schema
var userSchema  = {
    "userEmail" : String,
    "userPassword" : String
};
// create model if not exists.
module.exports = mongoose.model('userLogin',userSchema);
```

**server.js**

```
...
// Añadiremos una línea
var mongoOp     =   require("./models/mongo");
```

# API RESTful

- Nuestro recurso serán los **usuarios**

- **GET /users** – Devuelve todos los usuarios desde MongoDB
- **POST /users** – Añade un nuevo usuario en MongoDB
- **GET /users/:id** – Devuelve un usuario con ID
- **PUT /users/:id** – Actualiza la información del usuario
- **DELETE /users/:id** – Eliminar un usuario con ID

# GET /users

**server.js**

```
var express     =   require("express");
var app         =   express();
var bodyParser  =   require("body-parser");
var mongoOp     =   require("./models/mongo");
var router      =   express.Router();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({"extended" : false}));

router.get("/",function(req,res){
    res.json({"error" : false,"message" : "Hello World"});
});

router.route("/users")
    .get(function(req,res){
        var response = {};
        mongoOp.find({},function(err,data){
        // Mongo command to fetch all data from collection.
            if(err) {
                response = {"error" : true,"message" : "Error fetching data"};
            } else {
                response = {"error" : false,"message" : data};
            }
            res.json(response);
        });
    });

app.use('/',router);

app.listen(3000);
console.log("Listening to PORT 3000");
```

# POST /users

```
// Añadiremos un nuevo endpoint, esta vez del tipo POST

router.route("/users")
 .get(function(req,res){
        ------------------------------
    })
  .post(function(req,res){
        var db = new mongoOp();
        var response = {};
        // fetch email and password from REST request.
        // Add strict validation when you use this in Production.
        db.userEmail = req.body.email;
        // Hash the password using SHA1 algorithm.
        db.userPassword =  require('crypto')
                           .createHash('sha1')
                           .update(req.body.password)
                           .digest('base64');
        db.save(function(err){
        // save() will run insert() command of MongoDB.
        // it will add new data in collection.
            if(err) {
                response = {"error" : true,"message" : "Error adding data"};
            } else {
                response = {"error" : false,"message" : "Data added"};
            }
            res.json(response);
        });
    });
```

# GET /users/:id

```
// Añadiremos un nuevo endpoint

router.route("/users")
    .get(function(req,res){
        ------------------------------
    })
    .post(function(req,res){
        ------------------------------
    });

router.route("/users/:id")
    .get(function(req,res){
        var response = {};
        mongoOp.findById(req.params.id,function(err,data){
        // This will run Mongo Query to fetch data based on ID.
            if(err) {
                response = {"error" : true,"message" : "Error fetching data"};
            } else {
                response = {"error" : false,"message" : data};
            }
            res.json(response);
        });
    })
```

# PUT /users/:id

```
router.route("/users/:id")
    .get(function(req,res){
        -------------------------------
    })
    .put(function(req,res){
        var response = {};
        // first find out record exists or not
        // if it does then update the record
        mongoOp.findById(req.params.id,function(err,data){
            if(err) {
                response = {"error" : true,"message" : "Error fetching data"};
            } else {
            // we got data from Mongo.
            // change it accordingly.
                if(req.body.userEmail !== undefined) {
                    // case where email needs to be updated.
                    data.userEmail = req.body.userEmail;
                }
                if(req.body.userPassword !== undefined) {
                    // case where password needs to be updated
                    data.userPassword = req.body.userPassword;
                }
                // save the data
                data.save(function(err){
                    if(err) {
                        response = {"error" : true,"message" : "Error updating data"};
                    } else {
                        response = {"error" : false,"message" : "Data is updated for "+req.params.id};
                    }
                    res.json(response);
                })
            }
        });
    })
```

# DELETE /users/:id

```
// Añadiremos un nuevo endpoint

router.route("/users/:id")
    .get(function(req,res){
        ----------------------------------------
    })
    .put(function(req,res){
        ----------------------------------------
    })
    .delete(function(req,res){
        var response = {};
        // find the data
        mongoOp.findById(req.params.id,function(err,data){
            if(err) {
                response = {"error" : true,"message" : "Error fetching data"};
            } else {
                // data exists, remove it.
                mongoOp.remove({_id : req.params.id},function(err){
                    if(err) {
                        response = {"error" : true,"message" : "Error deleting data"};
                    } else {
                        response = {"error" : true,"message" : "Data associated with "+req.params.id+"is deleted"};
                    }
                    res.json(response);
                });
            }
        });
    })
```

# Tutoriales interesantes

- Express → *Getting started*
  - http://expressjs.com
- Create a Web App and RESTful API Server Using the MEAN Stack
  - https://devcenter.heroku.com/articles/mean-apps-restful-api
- Develop a RESTful API Using Node.js With Express and Mongoose
  - https://pixelhandler.com/posts/develop-a-restful-api-using-nodejs-with-express-and-mongoose

# Tarea

Modificar el desarrollo de la tarea de la Práctica 4 para que utilice Express.js para publicar un API RESTful

**Elabora una breve memoria en la que especifiques los recursos que has identificado y las operaciones sobre dichos recursos**

Entrega: hasta el 21 de Abril a las 23:55h