UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

DSIC

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Integración de aplicaciones

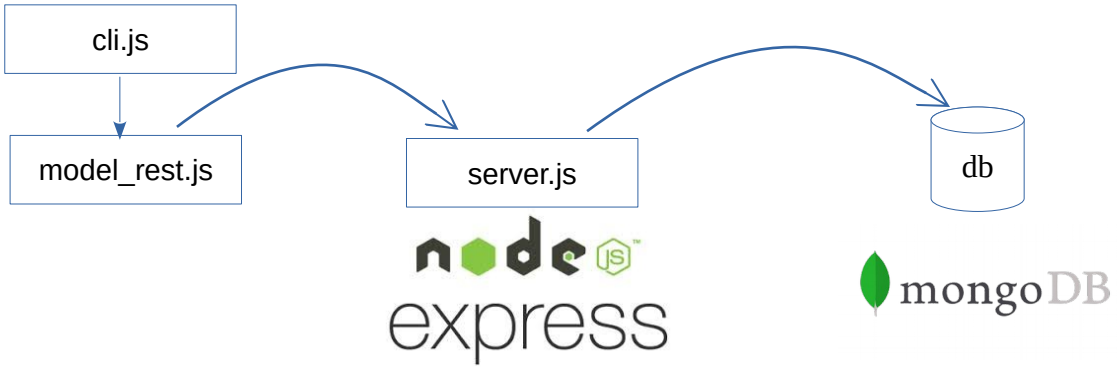
Laboratorio 2

Contenido

1. Introducción

En este laboratorio desarrollaremos un servicio RESTful que publique todas las operaciones de nuestra aplicación Twitter Lite. Implementaremos el servicio utilizando Node.js y Express. El servicio delegará el almacenamiento de toda la información a MongoDB.

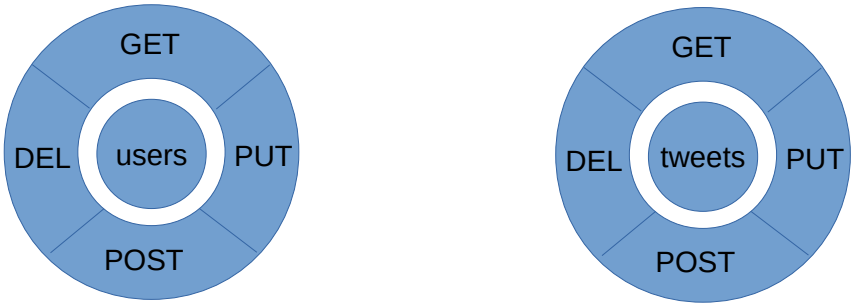
Para ello, por un lado diseñaremos el servicio en el fichero server.js que se ejecutará en el servidor y que atenderá peticiones HTTP de entrada. Por otro lado, diseñaremos una nueva librería de operaciones *model_rest.js*, que consumirá el servicio RESTful. Esta librería se inyectará en el cliente de nuestra aplicación, y reemplazará la librería desarrollada en el laboratorio anterior *model_mongo.js.* A continuación se presenta la arquitectura de la nueva aplicación a desarrollar.



En las siguientes secciones, primero diseñaremos la API del nuevo servicio RESTful y después la implementaremos utilizando Express. Por último, diseñaremos la librería *model_rest.js* que nos permitirá consumir dicho servicio RESTful desde el ciente *cli.js*.

2. La API del servicio RESTful

El primer paso para construir un servicio consiste en diseñar su API (contrato). En nuestro caso vamos a diseñar un servicio RESTful, y por tanto todas las operaciones deben girar alrededor de una colección de recursos. Los recursos principales de nuestra aplicación son claramente *users* y *tweets*.



Por tanto, a priori podríamos definir dos puntos de entrada principales en nuestro servicio:

- */twitter/users*
- */twitter/tweets*

Existen algunas subcolecciones que es interesante identificar:

- */twitter/users/:id/following*
- */twitter/users/:id/followers*
- */twitter/tweets/:id/retweets*
- */twitter/tweets/:id/likes*
- */twitter/tweets/:id/dislikes*

Además, es necesario habilitar un mecanismo de autenticación. Para ello, definiremos un nuevo recurso “artificial” *sessions.*

/twitter/sessions

El usuario deberá primero crear una nueva sesión, efectuando una petición HTTP POST y suministrando su e-mail y password, en formato JSON. En caso de éxito, el servicio responderá con un token y la información sobre el usuario autenticado, en formato JSON. Una vez se dispone del token será posible solicitar cualquier operación que requiera la autenticación previa del usuario.

A continuación resumimos la API de nuestro servicio RESTful en la siguiente tabla:

Método	**URL**	**Comentarios**
`POST`	`/twitter/sessions`	Abre una nueva sesión. Datos (JSON): {email, password} Resultado (JSON): {token, user}
Recurso users		
`GET`	`/twitter/users`	Recupera todos los usuarios del sistema Parámetros: token, opts (JSON) {query, ini, count, sort} Resultado (JSON): [user]
`GET`	`/twitter/users/XXX`	Recupera el usuario con id XXX Parámetros: token Resultado (JSON): user
`POST`	`/twitter/users`	Crea un nuevo usuario Datos (JSON): user Resultado (JSON): user
`PUT`	`/twitter/users/XXX`	Actualiza el usuario con id XXX Parámetros: token Datos (JSON): user Resultado (JSON): user
`DELETE`	`/twitter/users/XXX`	Elimina el usuario con id XXX Parámetros: token
Recurso following/followers		
`GET`	`/twitter/users/XXX/following`	Recupera todos los usuarios a quien sigue el usuario con id XXX Parámetros: token, opts (JSON) {query, ini, count, sort} Resultado (JSON): [user]
`POST`	`/twitter/users/XXX/following`	Añade un nuevo usuario a la lista de usuarios seguidos Parámetros: token Datos (JSON): {id}
`DELETE`	`/twitter/users/XXX/following/YYY`	Elimina al usuario YYY de la lista de usuarios seguidos del usuario XXX Parámetros: token
`GET`	`/twitter/users/XXX/followers`	Recupera todos los usuarios que siguen al usuario XXX Parámetros: token, opts (JSON) {query, ini, count, sort}

		Resultado (JSON): [user]
Recurso tweets		
`GET`	`/twitter/tweets`	Recupera todos los tweets del sistema Parámetros: token, opts (JSON) {query, ini, count, sort} Resultado (JSON): [tweets]
`GET`	`/twitter/tweets/XXX`	Recupera el tweet con id XXX Parámetros: token Resultado (JSON): tweet
`POST`	`/twitter/tweets`	Crea un nuevo tweet Datos (JSON): tweet Resultado (JSON): tweet
`POST`	`/twitter/tweets/XXX/retweets`	Crea un nuevo retweet a partir del tweet con id XXX Parámetros: token Resultado (JSON): tweet
Recurso likes/dislikes		
`POST`	`/twitter/tweets/XXX/likes`	Añade un like al tweet con id XXX Parámetros: token
`POST`	`/twitter/tweets/XXX/dislikes`	Añade un dislike al tweet con id XXX Parámetros: token

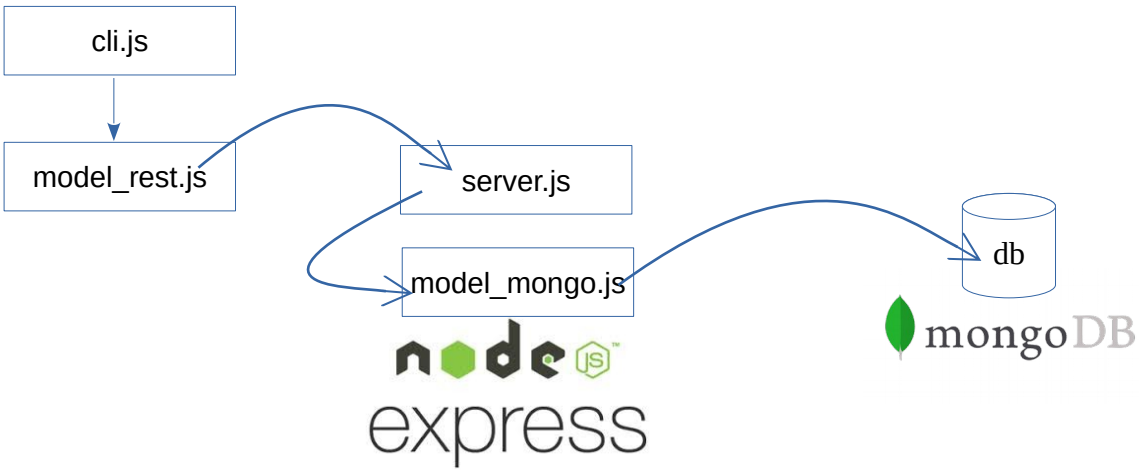
3. Implementación del servicio RESTful

Diseñaremos el servicio RESTful utilizando Node.js y el framework Express. Para ello, lo primero será instalar el paquete. Nos ubicaremos en el directorio raíz del proyecto y ejecutaremos el comando:

```
> npm install express
```

Vamos a diseñar el servicio de la manera más sencilla posible. Para ello, crearemos la aplicación Express e iremos registrando las rutas que sirven todas las operaciones descritas en la API. Cada ruta recuperará toda la información de la petición HTTP y accederá a MongoDB para modificar/recuperar el modelo de datos. Cuando la operación finalice se devolverá la respuesta HTTP.

Ya disponemos de una librería que implementa todas las operaciones necesarias sobre MongoDB. Se trata de la librería *model_mongo.js* implementada en el laboratorio 1. Esta librería se utilizaba directamente en el módulo cliente *cli.js*. En este laboratorio, el acceso ya no es directo, sino que el módulo *cli.js* consume un servicio RESTful que a su vez accede a MongoDB. Por lo tanto, con modificaciones mínimas podemos reaprovechar la librería *model_mongo.js* en la implementación del servicio RESTful, tal y como se ilustra en la siguiente figura:



Incluiremos la implementación de nuestro servicio RESTful en el mismo proyecto. Crearemos por tanto un nuevo módulo *server.js* en el directorio raíz del proyecto.

A continuación importaremos las operaciones de *model_mongo.js* desde *server.js*. Además, importaremos el paquete *express* y el middleware *body-parser* que nos permitirá parsear el contenido JSON.

```
var model = require('./model_mongo');
var express = require('express');
var bodyParser = require('body-parser');
var app = express();

app.use(bodyParser.json());
```

El siguiente paso consiste en idear una estrategia que permita autorizar las distintas operaciones en el servicio RESTful. Si recordamos, cuando el usuario se autentica recibe un token. Gracias a ese token podrá solicitar el resto de operaciones. Existen no obstante dos operaciones que no requieren un token, la de crear usuario y la de autenticarse. Para ello, definiremos un *middleware* que se activará en todas las peticiones, y que comprobará contienen el parámetro *token*, a excepción de las operaciones comentadas. En caso de no hacerlo, se generará un error.

```
app.use(function (req, res, next) {
    console.log('authorize ' + req.method + ' ' + req.originalUrl);
    /* Authorization */
    if ((req.path == '/twitter/sessions' && req.method == 'POST') ||
        (req.path == '/twitter/users' && req.method == 'POST')) {
        next();
    } else if (!req.query.token) res.status(401).send('Token not found');
    else next();
});
```

A continuación registraremos la ruta que permite autenticar al usuario. Esta ruta comprobará que los datos de entrada han sido especificados y delegará en la librería *model_mongo.js*.

```
/** Login **/
app.post('/twitter/sessions', function (req, res) {
  console.log('login ' + JSON.stringify(req.body));
  if (!req.body.email || !req.body.password)
    res.status(400).send('Parameters missing');
  else {
    model.login(req.body.email, req.body.password, (err, token, user) => {
      if (err) {
        console.log(err.stack);
        res.status(400).send(err);
```

```
      } else {
        res.send({ token: token, user: user });
      }
    });
  }
});
```

A continuación sería necesario implementar una a una todas las operaciones definidas en la API. Se muestran a modo de ejemplo las siguientes:

- Creación de usuarios (*addUser*)
- Listado de los usuarios seguidos por uno dado (*listFollowing*)
- Listado de tweets (*listTweets*)

```
app.post('/twitter/users', function (req, res) {
  console.log('add user  ' + JSON.stringify(req.body));
  model.addUser(req.body, (err, user) => {
    if (err) {
      console.log(err.stack);
      res.status(400).send(err);
    } else res.send(user);
  });
});

app.get('/twitter/users/:me/following', function (req, res) {
  console.log('list following  ' + JSON.stringify(req.query));
  if (req.query.token != req.params.me)
    res.status(400).send('Forbidden operation');
  else {
    let opts = {};
    if (req.query.opts) opts = JSON.parse(req.query.opts);
    model.listFollowing(req.query.token, opts, (err, users) => {
      if (err) {
        console.log(err.stack);
        res.status(400).send(err);
      } else {
        res.send(users);
      }
    });
  }
});


app.get('/twitter/tweets', function (req, res) {
  console.log('list tweets  ' + JSON.stringify(req.query));
  let opts = {};
  if (req.query.opts) opts = JSON.parse(req.query.opts);
  model.listTweets(req.query.token, opts, (err, tweets) => {
    if (err) {
      console.log(err.stack);
      res.status(400).send(err);
    } else {
      res.send(tweets);
    }
  });
});
```

Existen diversas herramientas que nos permiten comprobar que las operaciones funcionan. La más sencilla consiste en abrir el navegador y solicitar una URL. En este caso estamos limitados a efectuar únicamente peticiones HTTP GET. Si deseamos comprobar el resto de operaciones POST/PUT/DELETE tendremos que usar una herramienta más especializada, por ejemplo Postman.

https://www.postman.com/

Se trata de una herramienta que funciona en cualquier plataforma y que nos permite construir cualquier tipo de petición HTTP.

4. Consumo del servicio RESTful

El último paso en el laboratorio consiste en diseñar una nueva librería *model_rest.js* que implemente todas las operaciones del modelo, y que se encargue de traducirlas a peticiones HTTP sobre el servicio RESTful implementado en la sección anterior. Esta librería será importada por el módulo *cli.js*.

Para comunicarnos con el servicio RESTful utilizaremos el paquete *axios.*

```
> npm install axios
```

En *model_rest.js* importamos el paquete y definimos la URL raíz del servicio.

```
const axios = require('axios');
const url = 'http://localhost:8080/twitter';
```

Comenzamos a codificar todas las operaciones de la librería. Se muestran a modo de ejemplo las operaciones: *login()*, *addUser(), listFollowing()* y *listTweets().*

```
function login(email, password, cb) {
    axios.post(url + '/sessions',
        { email: email, password: password })
        .then(res => {
            cb(null, res.data.token, res.data.user)
        })
        .catch(err => {
            cb(err);
        });
}

function addUser(user, cb) {
    if (!user.name) cb(new Error('Property name missing'));
    else if (!user.surname) cb(new Error('Property surname missing'));
    else if (!user.email) cb(new Error('Property email missing'));
    else if (!user.nick) cb(new Error('Property nick missing'));
    else if (!user.password) cb(new Error('Property password missing'));
    else {
        axios.post(url + '/users', user)
            .then(res => {
                cb(null, res.data)
            })
            .catch(err => {
                cb(err);
            });
    }
}

}
function listFollowing(token, opts, cb) {
    axios.get(url + '/users/' + token + '/following',
        {
            params: { token: token, opts: JSON.stringify(opts) }
        })
        .then(res => {
            cb(null, res.data)
```

```
        })
        .catch(err => {
            cb(err);
        });
}

function listTweets(token, opts, cb) {
    axios.get(url + '/tweets', {
        params: { token: token, opts: JSON.stringify(opts) }
    })
        .then(res => {
            cb(null, res.data)
        })
        .catch(err => {
            cb(err);
        });
}
```