Prácticas de Sistemas Distribuidos

Creando un API RESTful Básico con MEAN

WS, RESTful, CRUD, MEAN, (NodeJS, express, MongoDB, AngularJS), JSON

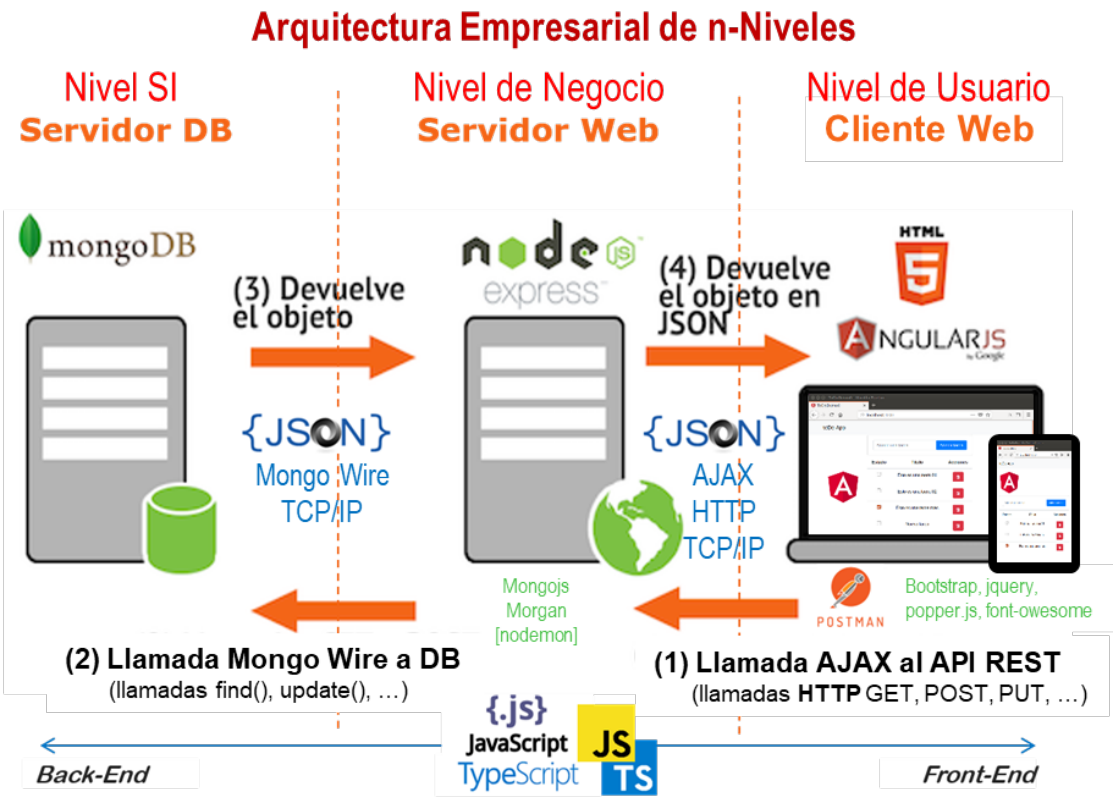
Introducción

Objetivo de la práctica

Explorar un conjunto de tecnologías que facilitan el diseño de aplicaciones distribuidas basadas en servicios o microservicios distribuidos por Internet. Todas estas tecnologías se apilan bajo las siglas **MEAN**.

En concreto, el objetivo es crear un API RESTful sobre HTTP que se comporte como un **CRUD** (*Create, Read, Update and Delete*). Para ello se emplearán tecnologías **MEAN**: **Node** + **Express** como servidor HTTP, el gestor de DB **Mongo** (junto con la biblioteca **mongojs** para escribir menos código).

A partir de este servicio **CRUD**, realizaremos dos aplicaciones (en función del tiempo que tengamos): un *To-Do* y *la gestión de una ficha de empleados*. Para ellos, nos apoyaremos nuevamente en la **Pila MEAN**, más concretamente, se hará uso de **Angular** para el *fornt-end*, apoyado con la biblioteca **Bootstrap** para facilitar la creación de controles *responsive*.



En la siguiente figura se puede observar cómo se relacionan y organizan las diferentes herramientas y tecnologías que forman parte de la Pila MEAN, cuáles son los principales elementos que intervienen en la misma y cómo encaja dicha tecnología y elementos dentro de un estilo arquitectural basado en una **arquitectura empresarial de n-Niveles**.

Trabajo que realizar

La estudiante debe seguir las guías que se proporciona en este documento, realizando las siguientes tareas:

- Investigar acerca de los conceptos relacionados: CRUD, MEAN, otros.
- Investigar acerca de las arquitecturas para sistemas distribuidos (estilos, modelos, patrones) y patrones de intercambio de mensajes (MEP).
- Investigar acerca de la especificación de las interfaces entre servicios: WS, API, REST, RESTFUL, GraphQL, SOAP, SPA (Single-Page Application), otros.
- Crear el *back-end* consistente en un servicio Web RESTful y una base de datos MongoDB.
- Crear el *front-end*, Implementando una aplicación SPA (Single-Page Application) tipo *To-Do* basada en el Servicio Web (y la pila MEAN).
- Crear un nuevo *front-end*, implementando una aplicación de gestión de empleados basada en el mismo servicio Web.
- Memoria de la práctica.

Requisitos

Para realizar las prácticas, se debe contar con los siguientes elementos:

- Equipo con Windows, Linux u OS X con conexión a Internet. Preferiblemente una máquina virtual con la última versión estable de Linux.
- Tener instaladas las siguientes aplicaciones:
    - Visual Studio Code + Plugins para Node, JS y TS.
    - Postman.
    - NodeJS (node y npm).
    - Git (y una cuenta con un repositorio en GitHub o Bitbacket, por ejemplo).
    - MongoDB.
- Conocimientos básicos de administración de SO, JS, HTML y CSS

Entregas

Se realizará una entrega a través de la aplicación **UACloud** en las fechas y formas que indicará el profesor.

Dicha entrega debe contener, como mínimo, la documentación relacionada en el apartado anterior junto con toda la documentación que se considere adecuada:

- Crear una memoria que será similar a la presente guía, pero:
    - documentando cada apartado de las guías con capturas de pantalla cuando sea conveniente (por ejemplo, la interacción con **Postman**).
    - corrigiendo (si fuera necesario) el guion (bien por erratas o, sobre todo, por cambio de las versiones, directorios, etc.),
    - indicando y actualizando las aplicaciones, librerías y sus respectivas versiones por las que hayan optado (si es el caso),
    - creando un glosario de términos en los que se explique brevemente cada término, acrónimo, etc. (por ejemplo: API, REST, RESTful, express, node, CRUD, Mongo, …).

    - Instalar y configurar las aplicaciones y librerías indicadas (documentando las decisiones tomadas)
- Crear un **PDF** con toda la documentación
- Crear un archivo **Zip** con toda la estructura creada en la carpeta "`node`" en la que se programará el servidor que implementará el **API RESTful** y las aplicaciones web. Es importante recordad que siempre se debe eliminar la carpeta "node_modules" de cada proyecto debido a que se puede y se debe reconstruir cada vez que movamos el código y a que ocupa mucho espacio de almacenamiento.

Guía 1. Instalación del Back-End

Esta guía se centra en la instalación, configuración y utilización básicas de las aplicaciones, *frameworks* y utilidades que necesitaremos para realizar la práctica.

Aunque la práctica se puede desarrollar prácticamente igual en Windows, Linux u OS X, recomendamos crear una máquina virtual con LINUX para tener un seguimiento y un mantenimiento futuros más sencillos.

La máquina virtual recomendable para realizar toda la práctica tendrá un mínimo de 2GHz de procesador, 2GB de RAM y 25GB de HD. También nos aseguraremos de instalar la última versión estable de 64 bits de Linux (por el momento la 20.04 LTS).

Para verificar estos parámetros, abrimos una terminal <Ctrl+Alt+T>

```
$ lsb_release -a
$ uname -m
$ df -h
```

Comenzamos instalando y probando NodeJS

Descargar versión actual (PKG) de https://nodejs.org y seguir las instrucciones de instalación. O bien se puede hacer desde los gestores de paquetes (**Ubuntu Snap Store** ya incluye **node**), siempre en función del sistema operativo en el que estemos trabajando:

Ubuntu 20.04

Abrir una terminal <Ctrl+Alt+T>

```
$ sudo apt update
$ sudo apt install nodejs
$ sudo apt install npm
```

Comprobamos versiones

```
$ node --version
V10.19.0
$ npm -v
6.14.4
```

Nota: para eliminarlo definitivamente…

```
$ sudo apt purge nodejs npm
```

Probamos la consola **node** que no deja de ser un intérprete de JS

```
$ node
> 3+4
7
> console.log("Hola a todos!");
Hola a todos!
undefined
> [^C para salir]
```

Instalamos el gestor de repositorios

Supondremos en esta sección que seguimos trabajando bajo Linux, aunque la forma de operar es básicamente la misma en todos los entornos.

Instalamos y configuramos **git** con nuestros datos de acceso

```
$ sudo apt install git
$ git config --global user.name pmacia
$ git config --global user.email pmacia@dtic.ua.es
$ git config --list
user.name=pmacia
user.email=pmacia@dtic.ua.es
```

Nota 1: si deseamos configurar el usuario y correo individualmente para cada proyecto, utilizaremos los mismos comandos antes expuestos sin la opción **`--global`** ejecutándolos desde la carpeta de cada proyecto.

Nota 2: si no queremos que nos solicite siempre la contraseña podemos utilizar la opción

```
$ git config --global credential.helper store
```

Nota 3: si nos preocupa dejar siempre la contraseña, podemos decirle también que la guarde, por ejemplo, una hora después de la última vez que ejecutemos algún comando git que acceda al repositorio den el servidor remoto:

```
$ git config --global credential.helper 'cache --timeout=36000'
```

Creamos el primer proyecto

Para crear un nuevo proyecto tenemos básicamente dos opciones: que ya tengamos un código previo (guardado en un repositorio) sobre el que queremos trabajar, o bien que queramos comenzar desde cero. Veamos a continuación cómo proceder en cada caso.

En ambos casos, mediante una terminal de comandos (<Ctrl+Alt+T>), desde nuestro home (**`cd $HOME`**), crearemos una carpeta de trabajo para todos nuestros proyectos en **`node`**:

```
$ cd
$ mkdir node
$ cd node
```

Si partimos de un código previo, lo recuperaremos del repositorio que tengamos o que nos indiquen. Por ejemplo:

```
$ git clone https://usuario@bitbucket.org/usuario/practica2.git api-rest
$ cd api-rest
```

En caso de comenzar desde cero, a partir de la carpeta **`node`** antes creada…

```
$ mkdir api-rest
$ cd api-rest
```

Creamos el repositorio local…

```
$ git init
$ echo "#Backend CRUD API REST" > README.md
```

Nota: en este punto, debemos tener creado un repositorio (mejor vacío) en cualquier servidor. Por ejemplo, crearemos uno en Bitbucket accesible a través de: https://pmacia@bitbucket.org/pmacia/api-rest.git.

Conectamos el repositorio local con el repositorio remoto (lo llamaremos **origin**)…

```
$ git remote -v
$ git remote add origin https://pmacia@bitbucket.org/pmacia/api-rest.git
```

Nota: podemos manipular el repositorio mediante las opciones `add/remove/rename/-v/show` de `git remote`.

Finalmente, sincronizamos el repositorio local con el remoto…

```
$ git status
```

Si había algo en el repositorio, primero debemos traerlo al local (**origin**)…

```
$ git fetch origin
$ git remote show origin
$ git status
$ git pull origin master
```

Si el repositorio remoto no tenía nada (como es este caso) o ya estaba sincronizado, subiremos los cambios locales al remoto por primera vez…

```
$ git add .
$ git commit -m "Primer envío: README.md"
$ git push -u origin master
```

Ahora sí, empezamos un proyecto nuevo utilizando **npm**:

```
$ npm init
```

Y completamos las cuestiones que se solicitan (se pueden dejar por defecto)

```
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
See `npm help json` for definitive documentation on these fields
and exactly what they do.
Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.
Press ^C at any time to quit.
package name: (api-rest)
version: (1.0.0)
description: Proyecto de API RESTful con Node.js y Express para SD
entry point: (index.js)
test command:
git repository:
keywords: CRUD, REST, node, express, mongo
author: Paco Maciá
license: (ISC)
About to write to $HOME/node/RESTFulServer/package.json:
{
        "name": "api-rest",
        "version": "1.0.0",
        "description": "Proyecto de API RESTful con Node.js y Express para SD",
        "main": "index.js",
        "scripts": {
                "test": "echo \"Error: no test specified\" && exit 1"
        },
        "keywords": [
                "CRUD",
                "REST",
                "node",
```

```
                "express",
                "mongo"
        ],
        "author": "Paco Maciá",
        "license": "ISC"
}
Is this ok? (yes) y
```

Nota: recordemos que debíamos tener instalado algún editor adecuado para programación (por ejemplo, **VSCode)**, siguiendo las instrucciones de cada fabricante, para cada sistema operativo. **VSCode**, puede ser instalado desde el *Store de Ubuntu* directamente.

Abrimos nuestro editor de código indicándole que queremos trabajar en la carpeta actual…

```
$ code .
```

Nota: iniciando nuestro editor con un punto provoca que se abra en la carpeta actual, lo que facilitará trabajar con múltiples archivos y subcarpetas.

Podremos comprobar que se ha creado un archivo **package.json** en el directorio de trabajo (**`$HOME/node/api-rest`**) con la información antes introducida

```
{
        "name": "api-rest",
        "version": "1.0.0",
        "description": "Proyecto de API RESTful con Node.js y Express para SD",
        "main": "index.js",
        "scripts": {
                "test": "echo \"Error: no test specified\" && exit 1"
        },
        "keywords": [
                "CRUD",
                "REST",
                "node",
                "express",
                "mongo"
        ],
        "author": "Paco Maciá",
        "license": "ISC"
}
```

Creamos una primera aplicación para comprobar el funcionamiento de **NodeJS** y lo sencillo que es poner en marcha un servidor Web bajo esta tecnología. Para ello creamos con nuestro editor de código el archivo **index.js** y escribimos en él el siguiente código:

```
var http = require('http');

http.createServer( (request, response) => {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hola a todas y a todos!\n');
}).listen(8080);

console.log('Servidor ejecutándose en puerto 8080...');
```

Para probarlo, desde la terminal ejecutamos el servidor **NodeJS** con nuestro código **`index.js`**.

```
$ node index.js
Servidor ejecutándose en 8080...
[^C para salir]
```

Nota: recordemos ir subiendo una copia al repositorio cada vez que tengamos una versión nueva. Por ejemplo, mediante los comandos:

```
$ git add .
```

```
$ git commit -m "Primer servidor HTTP con NodeJS"
$ git push
```

Ahora podemos conectarnos desde nuestro navegador web a través del puerto que hemos establecido (http://localhost:8080/) para ver el resultado.

Nota: si modificamos la línea **`response.write`** para que se envíe el recurso solicitado, con este sencillo código ya tendríamos implementado un servidor HTTP básico.

Una versión alternativa un poco más refactorizada de este código podría ser la siguiente:

```
var http = require('http');
var server = http.createServer();

function HTTP_Response(request, response) {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.write('Hola a todas y a todos!\n');
    response.end();
}

server.on('request', HTTP_Response);
server.listen(8080);

console.log('Servidor ejecutándose en puerto 8080...');
```

Nota: nuevamente, subiremos una copia al repositorio:

```
$ git add .
$ git commit -m "Primer servidor HTTP "
$ git push
```

Instalación de Express

A continuación, instalamos y probamos la biblioteca **express**. Esta biblioteca proporciona una capa adicional sobre **NodeJS** que facilita enormemente la gestión de métodos y recursos HTTP.

```
$ npm i -S express
```

Nota: es la forma simplificada del comando **`npm install express --save`**.

Esto creará una carpeta **`node_modules`**, **dentro de la carpeta de nuestro proyecto**.
La opción `-S` (o `--save`) fuerza a que se registre una entrada en el archivo **`package.json`, lo que permitirá posteriormente realizar una fácil instalación de todos los módulos que requiere un proyecto**.

```
"dependencies": {
        "express": "^4.16.3"
}
```

Creamos nuestra primera aplicación **node+express** en el archivo **`index.js`** (si lo deseamos, podemos guardar en el repositorio la versión anterior) para probar desde nuestro editor de texto que estará ubicado en el mismo directorio.

```
'use strict'

const express = require('express');
const app = express();
```

```
app.get('/hola', (request, response) => {
	response.send('Hola a Todas y a Todos desde Express!')
});

app.listen(8080, () => {
	console.log('API REST ejecutándose en http://localhost:8080/hola');
});
```

Desde la terminal ejecutamos la aplicación

```
$ node index.js
API REST ejecutándose en http://localhost:8080
^C para cerrar la aplicación
```

Ahora podemos probarlo con nuestro navegador conectándonos a la URL http://localhost:8080/hola y el resultado debe ser:

```
Hola a Todas y a Todos desde Express!
```

Nota: subimos una copia al repositorio, pero antes de hacerlo debemos eliminar del proceso los archivos de la carpeta **node_modules** en la que se encuentran todas las bibliotecas de código que vayamos instalando. Esta carpeta ocupa mucho espacio y se regenera fácilmente mediante el comando **npm init**, por lo que no tiene sentido subirla al repositorio:

```
$ git status
$ echo "node_modules/" > .gitignore
$ git status
$ git add .
$ git commit -m "Primer API REST con Express"
$ git push
```

Ampliamos la aplicación anterior modificando **index.js** para que soporte el método **GET** sobre múltiples recursos, en este caso, que se encuentre bajo la ruta `/hola/:unNombre`.

```
'use strict'

const port = process.env.PORT || 8888;

const express = require('express');
const app = express();

app.get('/hola/:unNombre', (req, res) => {
	res.status(200).send({ mensaje: `Hola ${req.params.unNombre} desde SD!` });
});

app.listen(port, () => {
	console.log(`API REST ejecutándose en http://localhost:${port}/hola/:unNombre`);
});
```

Nota 1: al utilizar dos puntos (":") en la especificación de la ruta `/hola/:unNombre`, le estamos indicando que "`unNombre`" no es un literal, sino un parámetro variable.

Nota 2: cuando utilicemos *expansión de parámetros* en nuestro código mediante la expresión `${parameter}`, hay que modificar las comillas ' por las comillas ` para que se realice dicha *expansión*.

Nuevamente tendremos que detener desde la terminal el anterior servidor, y volver a lanzar la nueva versión:

```
^C para cerrar la aplicación, si estaba en marcha…
```

```
$ node index.js
API REST ejecutándose en http://localhost:8888/hola/:unNombre
```

Lo probamos con nuestro navegador conectándonos a la URL: http://localhost:8888/hola/Estudiantes) y el resultado debe ser:

```
{"mensaje":"Hola Estudiantes desde SD!"}
```

Nota: si modificamos la función `app.get()` para que responda al método GET enviando al cliente el recurso solicitado, con este sencillo código tendríamos implementado un servidor HTTP básico similar al que hemos realizado en la práctica 2, pero con muchísimo más potencial… y apenas una decena de líneas de código.

Nota: antes de terminar, subimos una copia al repositorio:

```
$ git add .
$ git commit -m "Primer API REST con Express parametrizable"
$ git push
$ git status
```