Anexo Ionic

En **Visual code**, al abrir la consola si nos muestra un error, deberemos otorgar permisos de la siguiente forma:

1. Abrimos como administrador **powershell.**
2. Ejecutamos **Set-executionpolicy unrestricted.**

Para mostrar una **barra de búsquedas** en la página donde mostramos los datos, podemos emplear varias estrategias, siendo una de las más sencillas emplear Angular:
https://www.positronx.io/how-to-build-list-filtering-and-searching-in-ionic/

Para mostrar una **splash screen** debemos seguir los siguientes pasos:
https://ionicframework.com/docs/native/splash-screen

Si queremos mostrar en nuestra App la **barra de estado** de nuestro dispositivo móvil:
https://ionicframework.com/docs/native/status-bar

https://www.positronx.io/customize-ionic-status-bar-with-ionic-native-cordova/

Para poder **desplegar** nuestra App y generar el paquete a distribuir a un dispositivo móvil:
https://ionicframework.com/docs/cli/commands/capacitor-build

Y terminar de generar apk en Android studio.

Si queremos generar una Progressive Web App:
https://ionicframework.com/docs/angular/pwa

Para desplegar una PWA, podemos emplear:
https://www.npmjs.com/package/http-server

Si necesitamos generar una aplicación de escritorio podemos emplear **electron**:
https://ionicframework.com/docs/deployment/desktop-app

Para gestionar un sistema de login podemos emplear varios mecanismos, entre ellos:
https://devdactic.com/ionic-5-navigation-with-login

Anexo OpenApi

Para solucionar el problema de **CORS** con la App de Ionic con un Stub generado mediante **swagger**, debemos de modificar el fichero express.app.config.js , ubicado dentro del proyecto generado con swagger en: node_modules\oas3-tools\dist\middleware , permitiendo en este el uso de CORS, como por ejemplo se indica aquí:

https://expressjs.com/en/resources/middleware/cors.html

Anexo Servicios Web Rest

Ejercicio 1. Nodejs Hola Mundo

Para la creación de la primera aplicación “Hola Mundo” vamos a realizar los siguientes pasos:

El primer paso será comprobar que tenemos instalado nodejs. Para ello abrimos un terminal de CMD y escribimos “node –v” para así comprobar que disponemos de nodejs y de la última versión, en caso de no disponer de nodejs o no tener la última versión, accederemos a: https://nodejs.org/es/ para descargar e instalar nodejs.

A continuación, mediante un visual code escribimos el siguiente código y lo guardamos como **Ejercicio1.js**:

```
var express = require('express')
const app = express()
const puerto = 8080

//http://localhost:8080/HolaMundo/{"nombre":"Hugo"}
app.get('/HolaMundo/:nombre', (req, resp) => {
        var entrada=JSON.parse(req.params.nombre);
        resp.writeHead(200, {'content-type': 'text/plain'});
        resp.write('!Hola, Mundo '+entrada.nombre+' !');
        resp.end();
})

app.listen(puerto, () => {
  console.log('Aplicación escuchando en: http://localhost:'+puerto)

})
```

En la primera línea, importaremos los módulos necesarios, en nuestro caso *express*, establecemos el puerto de escucha en el 8080, posteriormente definimos que hará la petición get / de nuestra API (devolver *Hola Mundo* y el estado 200) y por último ponemos nuestra app en escucha en el puerto indicado anteriormente.

Para comprobar el funcionamiento:

```
node Ejercicio.js
```

Y probamos a acceder desde el navegador al recurso expuesto:

```
http://localhost:8080/HolaMundo/{“nombre”:”Hugo”}
```

Ejercicio 2. Diseño API Rest

En este ejercicio vamos a diseñar un API Rest con varias operaciones típicas a emplear en una aplicación real.

Las operaciones a diseñar serán:

Operación	Función	Petición	Url	Parameters		
				URL	Query	Body
Read	GetAll	GET	./items			
	GetFiltered	GET	./items?filter=ABC		Filter	
	GetById	GET	./id	Id		
Create	Add	POST	./items			Item
Update	UpdateById	PUT	./items			Id, Item
Delete	DeleteById	DELETE	./items/id	Id		

Estas serán las típicas operaciones CRUD realizadas sobre una base de datos.

Crearemos una carpeta que llamaremos, ApiRestDISM. Ahora instalamos el módulo **Express.js**. Para ello, desde la consola de comandos, nos desplazamos al directorio que hemos creado previamente y escribimos:

```
npm install express
```

Una vez finalizado la instalación de Express, creamos un nuevo fichero que llamaremos, Ejercicio2.js. Dentro de este fichero copiamos el siguiente contenido:

```
// Cargar modulos y crear nueva aplicacion
var express = require("express");
var app = express();
var bodyParser = require('body-parser');
app.use(bodyParser.json()); // soporte para bodies codificados en jsonsupport
app.use(bodyParser.urlencoded({ extended: true })); // soporte para bodies codificados

// Ejercicio: GET http://localhost:8080/items
app.get('/items', function(req, res, next) {
  if(req.query.filter) {
   next();
   return;
  }
  res.send('Get all');
});

// Ejercicio: GET http://localhost:8080/items?filter=ABC
app.get('/items', function(req, res) {
  var filter = req.query.filter;
  res.send('Get filter ' + filter);
});

// Ejercicio: GET http://localhost:8080/items/10
app.get('/items/:id', function(req, res, next) {
  var itemId = req.params.id;
  res.send('Get ' + req.params.id);
});

// Ejercicio: POST http://localhost:8080/items
app.post('/items', function(req, res) {
   var data = req.body.data;
   res.send('Add ' + data);
});

// Ejercicio: PUT http://localhost:8080/items
app.put('/items', function(req, res) {
   var itemId = req.body.id;
   var data = req.body.data;
   res.send('Update ' + itemId + ' with ' + data);
});

// Ejercicio: DELETE http://localhost:8080/items
app.delete('/items/:id', function(req, res) {
   var itemId = req.params.id;
   res.send('Delete ' + itemId);
});

var server = app.listen(8080, function () {
    console.log('Servidor iniciado en puerto 8080...');
});
```

En primer lugar, hemos cargado los módulos necesarios, y creado una aplicación de Express.

En el resto del código, estamos definiendo distintas rutas para nuestra aplicación, según la definición que hemos realizado en el apartado anterior.

En estas funciones, empleamos el objeto ‘request’ para obtener los parámetros que hemos pasado al API en la petición. En función del tipo de request, usamos la propiedad oportuna (params, query o body).

Por otro lado, empleamos el objeto ‘response’ para devolver los resultados al cliente.

En este ejercicio, simplemente estamos devolviendo un texto informando de que la petición correspondiente se ha recibido correctamente. En una aplicación real se realizarían consultas a una base de datos, operaciones con los datos, validaciones, etc...

Finalmente, en el último bloque de código iniciamos la aplicación poniéndola a escuchar peticiones en localhost, puerto 8080.

Para comprobar el funcionamiento:

```
node Ejercicio2.js
```

Y probamos a acceder desde el navegador a cualquier recurso de los expuestos:

http://localhost:8080/items
http://localhost:8080/items?filter=ABC
http://localhost:8080/items/10

Ejercicio 3. Acceso a Base de Datos desde API Rest

A continuación, vamos a crear una API Rest, que accede a una base de datos MySQL. Lo primero que debemos de realizar es crear la base de datos, para ello emplearemos el script de BD adjunto, en el cual creamos una BD llamada DISM que contiene una tabla llamada Usuarios.

Ahora debemos de instalar el paquete *Express*, *Mysql* y *Cors* para nodejs mediante npm:

```
npm install mysql
```
https://www.npmjs.com/package/mysql
```
npm install cors
```
https://www.npmjs.com/package/cors
```
npm install express
```
https://www.npmjs.com/package/express

Y ahora procedemos a escribir el ejercicio y lo guardamos como **Ejercicio3.js**:

```
var express = require("express");
var mysql      = require('mysql');
var app = express();
var bp = require('body-parser');
const cors = require('cors');
app.use(cors());
app.options('*', cors());

app.use(bp.json());

var connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : 'root',
  database : 'dism'
});

// Ejercicio: GET http://localhost:8080/usuarios
app.get('/usuarios', function(req, resp) {
        connection.query('select * from usuarios', function(err, rows) {
                if (err) {
                        console.log('Error en /usuarios '+err);
                        resp.status(500);
                        resp.send({message: "Error al obtener usuarios"});
                }
                else {
                        console.log('/usuarios');
                        resp.status(200);
                        resp.send(rows);
                }
        })
});

var server = app.listen(8080, function () {
    console.log('Servidor iniciado en puerto 8080…');
});
```

Para comprobar el funcionamiento:

```
node Ejercicio3.js
```

Y probamos a acceder desde el navegador al recurso expuesto:

http://localhost:8080/usuarios

NOTA: Para MySQL 8.0, se debe de crear un usuario nuevo, en nuestro caso crearemos uno llamado *usuario* con contraseña *clave* y le daremos permisos:

```
CREATE USER 'usuario'@'localhost' IDENTIFIED WITH mysql_native_password BY 'clave';
GRANT ALL ON dism.* TO 'usuario'@'localhost';
```

Eso lo realizaremos desde MySQL WorkBench.