# JavaScript 2023

## Práctica 4

### Objetivos

- node.js, npm.

### Ejercicio 1

Siguiendo los pasos de este tutorial construya una aplicación para servir **archivos estáticos** como por ejemplo `html` , `css`, `js` y, `png`.

A medida que sigue el tutorial, analice qué hace cada uno de estos comandos.

- `mkdir express-static-file-tutorial`
- `cd express-static-file-tutorial`
- `npm init -y`
- `touch index.js`
- `npm install express --save`

**Nota:** en Windows `mkdir` es `md` y el comando `touch` de Linux crea un archivo vacío.

1- ¿Qué función cumple el archivo `package.json`?

2- ¿Qué versión de la biblioteca `express` está usando?

3- ¿Por qué se le indica a la app que escuche el puerto 3000?

4- ¿Cómo se le indica a la biblioteca `express` en qué carpeta van a estar los archivos estáticos?

5- ¿Para qué sirve la función `require` que se usa en index.js?

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

6- ¿Qué significa este bloque de código en `index.js`?

```
app.get('/', (req, res) => {
    res.send('Hello World!');
});
```

7- Por qué la URL de la imagen `shark.png` es `http://localhost:3000/shark.png` y no `http://localhost:3000/static/shark.png`?

**Nota:** la documentación de express.

### Ejercicio 2

Implemente un formulario de login como el que sigue.


Agregue a la aplicación servidor que les brindamos desde la catedra, un controller para recibir los datos del formulario e imprimirlos **en la consola del servidor**.

¿Con qué método HTTP recibe los datos?

¿Tuvo que realizar alguna configuración en **express** para recibir los datos de un formulario?

## Ejercicio 3

Modifique formulario del ejercicio 2 para validar **en el cliente** (es decir en el navegador) que los campos ingresados no sean vacíos y que el nombre de usuario sea una dirección de e-mail válida. En caso de que algún dato no sea válido, no se debe enviar el formulario al servidor y se debe mostrar un mensaje al usuario indicando qué datos son inválidos.

¿Cuál es **la mejor manera** de validar el formato de una dirección de e-mail?

## Ejercicio 4

Modifique formulario del ejercicio 3 para que al recibir en el servidor los datos del formulario se verifiquen el usuario y la contraseña comparándolos con 2 constantes predefinidas (por ejemplo `'admin@mail.com'` y `'1234'`).

En caso de ser correctos, se debe **redirigir** a una página que informe el éxito y en caso de no serlo, a la misma página de login.

¿Qué es una redirección en el contexto del protocolo HTTP?

¿Cómo se realiza una redirección utilizando **express**?

## Ejercicio 5

Modifique el ejercicio 4 de manera que en lugar de tener 2 constantes con el login válido, haya un archivo llamado `usuarios.json`. Al recibir los datos del formulario se debe buscar en ese archivo para determinar si el usuario y la contraseña son correctos.

¿En qué carpeta del servidor guardó el archivo `usuarios.json` y por qué?

## Ejercicio 6

Agregue al archivo `usuarios.json` del ejercicio 5 (que ya tiene el e-mail y la password), el nombre completo de la persona. Por ejemplo el contenido podría ser el que sigue.

```
[
    {
        "username": "mimail@email.com",
        "password": "xasadsaflkjaskjd",
        "name": "Jason"
    },
    {
        "username": "otro@email.com",
        "password": "serewrewrw",
        "name": "Gary"
    }
]
```

Luego implemente un controller para consultar el nombre de un usuario a partir de su `username`. Al acceder a la URL `http://localhost:3000/names?username=mimail@email.com` el resultado debe ser

```
{
    "name": "Jason"
}
```

¿Con qué método HTTP se recibe la solicitud? ¿Cómo se devuelve una respuesta en formato JSON con **express**? ¿Qué devuelve si se recibe como parámetro un username que no existe en el archivo? ¿Qué devuelve si no se envía el parámetro `username`? ¿Qué sucede si se accede a esta URL `http://localhost:3000/names?username=mimail@email.com&username=otro@email.com`?

## Ejercicio 7

Basándose en el **ejercicio 13 de la práctica 2**, guarde los datos de las personas en un archivo JSON llamado `people.json` (o una db). Luego implemente las siguientes API REST.

1. `/peoples/overweight`: devuelve en formato JSON un arreglo con los nombres de las personas con un IMC mayor a 25.
2. `/peoples/by_age`: devuelve en formato JSON un arreglo de las edades de las personas indexado por el nombre de cada una. (Por ejemplo algo de la forma `["Bobby": 22, "Mark": 36]`).
3. `/peoples/imc_over_40`: devuelve en formato JSON un arreglo con el IMC de los mayores de 40.
4. `/peoples/average_imc`: devuelve en formato JSON el IMC promedio de todas las personas. Por ejemplo `{"avg": 23}`.
5. `/peoples/youngest`: devuelve en formato JSON la persona más joven.
6. `/peoples/by_height`: devuelve en formato JSON un arreglo de personas ordenadas por estatura.

Por ejemplo, acceder a la URL `http://localhost:3000/peoples/average_imc` devuelve `{"avg": 23}`.

**Consideraciones sobre el ejercicio**

- Elegir 2 APIS de las generadas anteriormente y realizar una vista que la consuma
- Se debe realizar en grupo.

- Se debe subir al repositorio git asignado al grupo en gitlab.
- Será evaluado.
- Cada integrante del grupo debe hacer sus *commits* con su aporte a la solución.