4 - Un middleware es una funcionalidad que se ejecuta previamente a que la request llegue al controlador. Se podría utilizar un middleware para securizar endpoints, por ejemplo que valide un Token JWT (ej: se podría validar la sesión de un usuario) se podrian enviar logs detallado fecha, hora y endpoint al que se realizó la request para un sistema de tracing.

Ejemplo de Middlewares

app.js > ...

```
const express = require("express");
const app = express();
const routes = require("./routes");
const port = 3000;

// MIDDLEWARE APLICADO A UN ENDPOINT ESPECIFICO
app.use("/user/:id", (req, res, next) => {
  console.log(
    `timestamp: ${Date.now()} - Path: ${req.baseUrl} - Data: ${
      req.body
    } - Params: ${req.params}`
  );
  next();
});

// MIDDLEWARE APLICADO A TODOS LAS REQUEST ENTRANTES
app.use((req, res, next) => {
  console.log(`Global Middleware Method: ${req.method}`);
  next();
});

routes(app);

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});

```

5 - tanto THEN como AWAIT se utilizan para handlear promesas, cuando se utiliza la key word AWAIT se le esta diciendo al programa que debe esperar a que finalize la petición realizada (ya sea el parseo de un csv, llamada a base de datos, o una request hecha a otro microservicio) luego es posible leer el resultado de la operacion (si se guardo en memoria la respuesta) o simplemente, en el caso de insertar datos en la DB, sabemos que se inserto el dato y podemos continuar con el flujo del programa.

Por otro lado el THEN, se utiliza cuando luego de que la promesa finaliza, se le pasa a este mismo una funcion la cual ejecuta un comportamiento luego de que la promesa fue resuelta. Por ejemplo, al ejecutar una request, podriamos utilizar tanto AWAIT como THEN, en el caso de THEN, podriamos pasarle una funcion que valide que el status code de la request http haya sido 200 o 201 y que realize alguna otra accion que necesite si o si que la promesa haya sido resuelta corretamente.

6 - Se podría por un lado implementar la función Promise.All() que ejecuta todas las peticiones y se resuelve una vez estén todas completas, y tambien se podria agregar una caché ya sea con Redis o simplemente en memoria, si el usuario se encuentra en cache, se evita realizar una llamada innecesaria.

7 - Las bases de datos SQL son relacionales. Provee una relación predefinida entre sus elementos, hay que construir su esquema previamente a usarla (tablas, columnas, relaciones entre tablas, tipos de datos que se estarán guardando. etc..) Mientras que las DB NoSQL son todo lo opuesto, en lugar de guardar registros, guardan documentos que pueden contener cualquier tipo de dato, y no es necesario construir un esquema. Por otro lado, las bases de datos NoSQL son más rápidas en comparación con las SQL.

Se puede utilizar bases de datos NoSQL cuando las consultas deben ser rápidas, los dispositivos son de bajos recursos, o para analizar grandes cantidades de datos.
En el caso de SQL, cuando los datos deben ser consistentes y el volumen de datos no tiene un gran crecimiento.

8 -

A - **Ventajas**: tenemos seguridad de que nuestro código funciona correctamente y no omitir ningún detalle. Además de validar que no existan vulnerabilidades.

Desventajas: cada vez que se modifique el código, será necesario readaptar nuestros test a las nuevas funcionalidades añadidas, por lo que se debe invertir tiempo en esta tarea que ralentiza el desarrollo de nuevas funcionalidades.

B - Las pruebas unitarias se encargan de testear independientemente cada función declarada y su comportamiento. Mientras que las pruebas funcionales se encargan de probar una feature en si, por ejemplo Comprar un producto, se probaría que se pueda comprar el producto, que se realice el pago correctamente, y al usuario le sea entregado lo que compro.

C - En Total hay 81 combinaciones para realizar pruebas

OS	BROWSER	DATABASE	COUNTRY
Windows	Chome	SQL	US
Windows	Chome	SQL	CH
Windows	Chome	SQL	RU
Windows	Chome	Mongo	US
Windows	Chome	Mongo	CH
Windows	Chome	Mongo	RU
Windows	Chome	MySQL	US
Windows	Chome	MySQL	CH
Windows	Chome	MySQL	RU
Windows	IE	SQL	US
Windows	IE	SQL	CH
Windows	IE	SQL	RU
Windows	IE	Mongo	US
Windows	IE	Mongo	CH
Windows	IE	Mongo	RU
Windows	IE	MySQL	US
Windows	IE	MySQL	CH
Windows	IE	MySQL	RU
Windows	Firefox	SQL	US
Windows	Firefox	SQL	CH
Windows	Firefox	SQL	RU
Windows	Firefox	Mongo	US
Windows	Firefox	Mongo	CH
Windows	Firefox	Mongo	RU
Windows	Firefox	MySQL	US
Windows	Firefox	MySQL	CH
Windows	Firefox	MySQL	RU

OS	BROWSER	DATABASE	COUNTRY
Mac	Chome	SQL	US
Mac	Chome	SQL	CH
Mac	Chome	SQL	RU
Mac	Chome	Mongo	US
Mac	Chome	Mongo	CH
Mac	Chome	Mongo	RU
Mac	Chome	MySQL	US
Mac	Chome	MySQL	CH
Mac	Chome	MySQL	RU
Mac	IE	SQL	US
Mac	IE	SQL	CH
Mac	IE	SQL	RU
Mac	IE	Mongo	US
Mac	IE	Mongo	CH
Mac	IE	Mongo	RU
Mac	IE	MySQL	US
Mac	IE	MySQL	CH
Mac	IE	MySQL	RU
Mac	Firefox	SQL	US
Mac	Firefox	SQL	CH
Mac	Firefox	SQL	RU
Mac	Firefox	Mongo	US
Mac	Firefox	Mongo	CH
Mac	Firefox	Mongo	RU
Mac	Firefox	MySQL	US
Mac	Firefox	MySQL	CH
Mac	Firefox	MySQL	RU

OS	BROWSER	DATABASE	COUNTRY
Linux	Chome	SQL	US
Linux	Chome	SQL	CH
Linux	Chome	SQL	RU
Linux	Chome	Mongo	US
Linux	Chome	Mongo	CH
Linux	Chome	Mongo	RU
Linux	Chome	MySQL	US
Linux	Chome	MySQL	CH
Linux	Chome	MySQL	RU
Linux	IE	SQL	US
Linux	IE	SQL	CH
Linux	IE	SQL	RU
Linux	IE	Mongo	US
Linux	IE	Mongo	CH
Linux	IE	Mongo	RU
Linux	IE	MySQL	US
Linux	IE	MySQL	CH
Linux	IE	MySQL	RU
Linux	Firefox	SQL	US
Linux	Firefox	SQL	CH
Linux	Firefox	SQL	RU
Linux	Firefox	Mongo	US
Linux	Firefox	Mongo	CH
Linux	Firefox	Mongo	RU
Linux	Firefox	MySQL	US
Linux	Firefox	MySQL	CH
Linux	Firefox	MySQL	RU

D 1 -

Suma:

```
assertEquals( 5, suma(3, 2))
assertEquals( 10, suma(4, 6))
assertNotEquals( 10, suma(3, 6) )
assertNotEquals( 5, suma( 2, 2) )
```

Resta:

```
assertEquals( -9, resta(-2, -7))
assertEquals( 8, resta(16, 8))
assertEquals( -8, resta(8, -16))
assertEquals( 0, resta(0, 0) )
assertNotEquals( 5, resta( -3, -2) )
assertNotEquals( 0, resta( -3, 0) )
```

Multiplicación:

```
assertEquals( 14, multiplicacion(-2, -7))
assertEquals( 8, multiplicacion(2, 4))
assertEquals( -18, multiplicacion(2, -9))
assertEquals( 0, multiplicacion(8, 0))
assertEquals( 0, multiplicacion(0, 0) )
assertNotEquals( 15, multiplicación( -3, 3) )
assertNotEquals( 16, multiplicacion( 4, 3) )
```

División:

```
assertEquals( 8, division(16, 2))
assertEquals( 0, division(0, 2) )
assertEquals( -1, division( -3, 3) )
assertNotEquals( -1, division( -3, -3) )
assertNotEquals( 16, division( 4, 4) )
assertThrows( división(0, 0) )
```

D 2 -

se podría verificar que el tipo de dato sea Integer, o similares. En caso que este sea **False** se podría o lanzar algún tipo de error.
Por otro lado, también sería una posibilidad parsear el número no entero a Integer