«Talento Tech»

# Desarrollo Web 3

Clase 01

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

# Clase N° 1 | Conceptos básicos

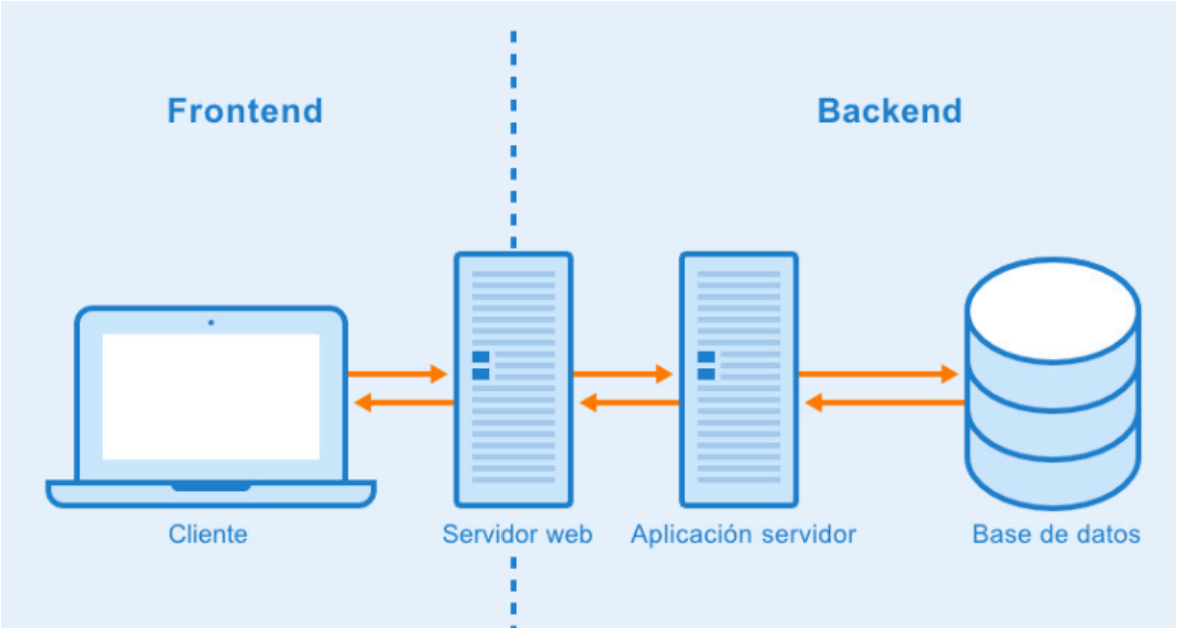
## Temario:

- Introducción al Backend y su rol en DesarrolloWeb
- Repaso de ECMA6 (let, const, arrow function, template, objetos, propiedades, métodos y arrays)
- Async + promesas
- Instalación de Node.js

Buenos Aires aprende
Agencia de Habilidades para el Futuro

# Introducción al BackEnd en Desarrollo Web

Frontend vs Backend




**Frontend:** Desarrollo web en el navegador con HTML, CSS y JavaScript.Desarrolla la interfaz visual.
No es diseño, sino programación de la interfaz de usuario (UI).
Frameworks como Bootstrap, React.js, Angular.js.

**Backend:** Utiliza Node.js para programar en el servidor, manejar la base de datos y conectar con el frontend.Utiliza Node.js y otros lenguajes (Ruby, Python, Java, c++).
Consiste en servidor, aplicación y base de datos.
Almacena datos en el servidor para interacción continua con el usuario.

**Full Stack**: Dominio de ambas áreas para crear páginas web dinámicas.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

# Repaso de ECMA6

**ECMA6: Nueva estandarización del lenguaje de Javascript (?)**

En **Desarrollo web 3**, vimos que con el estándar **ECMA en su versión 6**, había una nueva (y recomendada) forma de crear variables.

Con la palabra reservada **let** creamos variables con un scope (alcance) de bloque (**function, if, else, for, etc**), lo que la hace más privada que su antecesora var (vista en Desarrollo web 2), ya que esta última es una variable de tipo global. La podíamos declarar e inicializar de la siguiente manera:

```
let nombre = "José";
console.log(nombre);
```

O también podíamos declarar sin inicializar su valor en la misma línea, y hacerlo luego.
Manejar espacios para que quede claro la correspondencia entre texto explicativo y código Javascript

```
let nombre;
nombre = "José";
console.log(nombre);
```

También podíamos crear variables constantes, que tienen la particularidad de que su valor **no puede cambiarse** a través de la reasignación. Además, es necesario inicializar la constante, es decir, se debe especificar su valor en la misma sentencia en la que se declara, lo que tiene sentido, dado que no se puede cambiar posteriormente.

```
//Forma incorrecta
const n = 14;
n = 15;
console.log(n);
```

```
//Forma incorrecta
const n;
n = 14;
console.log(n);
```

```
//Forma correcta
const n = 14;
console.log(n);
```

### Ejemplo de Implementación:

- **Arrow Functions:**

```
// Función arrow para saludar
const saludar = nombre => alert("Hola " + nombre);
saludar("Pedro");
```

Descripción: Utiliza una arrow function para crear una función de saludo más concisa.

- **Objetos:**

```
// Definición de un objeto "auto"
let auto = { marca: 'Ford', modelo: 'Fiesta', anio: 2018, tieneAire: true };
console.log(auto.marca); // Ford
```

Descripción: Muestra cómo crear y acceder a propiedades de un objeto.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

- **Arrays - Métodos:**

```
// Manejo de un array de programadores
const programadores = ['Ada', 'Guido', 'Ryan'];
console.log(programadores[1]); // Guido

programadores[3] = 'Grace Hopper';
console.log(programadores[3]); // Grace Hopper
```

Descripción: Ejemplo de manipulación de un array, incluyendo agregar un nuevo elemento.

- **forEach():**

```
// Iteración sobre un array de colores
const colores = ['Rojo', 'Verde', 'Amarillo', 'Azul'];
colores.forEach((color, indice) => console.log(color, indice));
```

Descripción: Utilización del método forEach() para recorrer un array.

- **map():**

```
// Uso de map para convertir a mayúsculas
const mayusculas = colores.map(color => color.toUpperCase());
console.log(mayusculas); // ['ROJO', 'VERDE', 'AMARILLO', 'AZUL']
```

Descripción: Utilización del método map() para transformar los elementos de un array.

- **filter():**

```
// Filtrado de notas desaprobadas
const notas = [1, 2, 6, 8, 10, 5];
const desaprobadas = notas.filter(nota => nota < 6);
console.log(desaprobadas); // [1, 2, 5]
```

Descripción: Uso de filter() para obtener elementos que cumplen con cierta condición.

- **Template Strings:**

```
// Concatenación eficiente con Template Strings
let nombre = 'Juan Pedro';
let apellido = 'Pérez Galarza';
let edad = 30;
console.log(`Tu nombre es ${nombre} ${apellido} y tienes ${edad} años.`);
```

Descripción: Utiliza Template Strings para una concatenación más legible y eficiente.

# Async + promesas

## Promesas en JavaScript

Las promesas son acciones esperadas que pueden cumplirse o no. Se compara con el ejemplo de comprar comida, donde el ticket es una promesa que puede cumplirse (obtenemos la comida) o no (sin comida debido a un error).

**Creación de una Promesa en JavaScript:**
Se crea una promesa usando new Promise con argumentos ‘resolve’ y ‘reject’. En este caso, la promesa se resuelve si la condición del if es verdadera.

**Código:**

```
const promesa = new Promise((resolve, reject) => {
    if (true) {
        resolve('Ha funcionado');
    } else {
        reject('Error');
    }
});
```

## Manejo de Promesas en JavaScript

- **Utilización de .then() y .catch()**

Se utiliza **.then()** para manejar la resolución de la promesa y **.catch()** para manejar el rechazo. En este caso, se imprimirá el mensaje de resolución debido al valor booleano true en la condición del if.

**Código:**

```
    .then((mensaje) => console.log(mensaje))
    .catch((error) => console.error(error));
```

- **Utilización de async y await:**

Se utiliza async y await para trabajar con promesas de manera más sincrónica. La función ejecutarPromesa espera que la promesa se resuelva o sea rechazada antes de continuar

**Código:**

```
async function ejecutarPromesa() {
    try {
        const resultado = await promesa;
        console.log(resultado);
    } catch (error) {
        console.error(error);
    }
}

ejecutarPromesa();
```

## Asincronía, Call Stack, Event Loop y Promesas: Resumen

**Asincronía y Orden de Ejecución**
La asincronía no se trata solo del tiempo de ejecución sino del orden en la lista de órdenes.Incluso con un **setTimeout** de 0 milisegundos, se ejecuta de manera asincrónica debido al **Event Loop**.

**Call Stack**
Una pila donde se apilan las tareas a ejecutar en orden.
**JavaScript** es **single-threaded**, con un único stack para la ejecución.
La ejecución es sincrónica y se resuelve una tarea a la vez.

**Event Loop**
Permite sincronizar tareas asincrónicas con el **call stack.**
Las tareas asincrónicas se resuelven como callbacks en un callback queue. Cuando el call stack está vacío, el event loop incorpora tareas desde el callback queue al call stack.

**SetTimeout y setInterval:**

- **SetTimeout** ejecuta una función después de un tiempo dado.

- **SetInterval** ejecuta una función repetidamente cada cierto intervalo.
- **ClearTimeout y ClearInterval** se usan para detener timeouts e intervals respectivamente.

**Promesas**

Representan un evento futuro que puede completarse y producir un valor.
Tienen tres estados: **pending, fulfilled, y rejected**.

Se resuelven o rechazan llamando a resolve() o reject() respectivamente.

- Los métodos **then()** y **catch()** manejan la resolución o el rechazo de la promesa.

- El método **finally()** se ejecuta siempre al finalizar la secuencia, sea resuelta o rechazada.

➔ **Ejemplo Práctico.** Simular Petición de Datos

Se simula una petición de datos con una promesa y un setTimeout.
Al resolver la promesa, se actualiza un array de productos y se genera una vista.

```
// Ejemplo de setTimeout y Event Loop
console.log("Inicia proceso");

setTimeout(() => {
    console.log("Mitad de proceso");
}, 0);

console.log("Fin proceso");

// Ejemplo de setInterval y clearInterval
let counter = 0;
const interval = setInterval(() => {
    counter++;
    console.log("Counter: ", counter);
    if (counter >= 5) {
        clearInterval(interval);
```

```
        console.log("Se removió el intervalo");
    }
}, 1000);

// Ejemplo de Promesas
const eventoFuturo = (res) => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            res ? resolve('Promesa resuelta') : reject('Promesa rechazada');
        }, 2000);
    });
};

eventoFuturo(true)
    .then((response) => {
        console.log(response); // Promesa resuelta
    })
    .catch((error) => {
        console.log(error);
    })
    .finally(() => {
        console.log("Fin del proceso");
    });

// Ejemplo Aplicado: Simular Petición de Datos
const BD = [
    { id: 1, nombre: 'Producto 1', precio: 1500 },
    { id: 2, nombre: 'Producto 2', precio: 2500 },
    { id: 3, nombre: 'Producto 3', precio: 3500 },
    { id: 4, nombre: 'Producto 4', precio: 3500 },
];

const pedirProductos = () => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve(BD);
        }, 3000);
```

```
    });
};

let productos = [];

const renderProductos = (arr) => {
    // función que genere la vista de los productos
    console.log("Renderizando productos:", arr);
};

pedirProductos()
    .then((res) => {
        productos = res;
        renderProductos(productos);
    });
```

Este código de ejemplo ilustra los conceptos de **asincronía, call stack, event loop**, **setTimeout, setInterval, y promesas** en un contexto práctico de simulación de petición de datos y actualización de una vista.

# Node.js entorno de ejecución para JavaScript

## Introducción a Node.js

Node.js es un entorno de ejecución para JavaScript del lado del servidor, construido sobre el motor V8 de Google Chrome. A diferencia de la mayoría de los entornos de ejecución de JavaScript, que se **enfocan en el desarrollo del lado del cliente**, Node.js está diseñado para ejecutarse en el servidor y facilitar la construcción de aplicaciones web escalables y de alto rendimiento.

A continuación, se presentan algunas características clave de Node.js:

Buenos Aires aprende
Agencia de Habilidades para el Futuro

**JavaScript en el Servidor:**

Node.js permite a los desarrolladores utilizar JavaScript tanto en el lado del cliente como en el servidor, lo que simplifica la coherencia en el desarrollo de aplicaciones web.

**Event-Driven y Asíncrono:**

La arquitectura de **Node.js** es **event-driven** y asíncrona, lo que significa que está diseñada para manejar muchas conexiones simultáneas sin bloquear el hilo principal de ejecución. Esto es ideal para aplicaciones en tiempo real, como chats y juegos en línea.

**Módulos y NPM:**

Node.js utiliza un sistema de módulos que permite organizar el código en archivos separados y reutilizables. Además, NPM (Node Package Manager) facilita la gestión de dependencias y la incorporación de bibliotecas externas.

**Velocidad y Eficiencia:**

El motor V8 de Google Chrome, sobre el cual se construye Node.js, es conocido por su velocidad de ejecución. Node.js está optimizado para realizar operaciones de entrada/salida de manera eficiente, lo que lo hace adecuado para aplicaciones escalables y de alto rendimiento.

**Ecosistema y Comunidad Activa:**

Node.js cuenta con un amplio ecosistema de bibliotecas y herramientas gracias a su activa comunidad de desarrolladores. Esto facilita la construcción de diversas aplicaciones y servicios.

**Aplicaciones Web y APIs:**

Node.js es especialmente efectivo para construir aplicaciones web y APIs (interfaces de programación de aplicaciones) debido a su capacidad para manejar muchas conexiones simultáneas y gestionar eventos de manera eficiente.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

En resumen, Node.js ha revolucionado el desarrollo del lado del servidor al proporcionar un entorno de ejecución eficiente y escalable para JavaScript. Su enfoque en la asincronía y la capacidad de manejar muchas conexiones concurrentes lo convierten en una opción popular para aplicaciones web modernas.

## Instalación de Node.js

Habiendo finalizado el repaso, vamos a terminar esta clase con la instalación de Node en nuestras máquinas. Lo primero que debemos hacer es ingresar a su página web y descargar el instalador según el sistema operativo que usemos . Se sugiere instalar el que dice “LTS”, recomendado para la mayoría):

**Link de Descarga:** https://nodejs.org/en

Una vez que finalizamos, podremos comprobar si efectivamente Node se instaló de manera correcta en nuestro sistema. Para esto, hacemos click en el Inicio de Windows, y escribimos cmd . Esto nos abrirá la consola de comandos en Windows. Una vez abierta, escribimos:

```
node -v
```

Al presionar enter, nos debe aparecer la versión (-v) instalada desde la web de Node.


Buenos Aires
aprende
Buenos
Aires
Ciudad