# **Introducción al maravilloso mundo de los frameworks basados en JavaScript**

Bienvenidos al **maravilloso mundo de los frameworks de JavaScript**, un lugar donde nuestro viejo conocido —JavaScript— dejó de ser ese lenguaje tímido que solo hacía “alert(‘Hola’)” en el navegador, y se convirtió en un auténtico **todoterreno** que construye desde páginas web hasta aplicaciones móviles, de escritorio y hasta servidores completos.



En este universo encontramos frameworks y librerías que nos hacen la vida más fácil:

* **Node.js** nos permitió sacar a JavaScript del navegador y usarlo en el backend.
* **Express** llegó para que levantar un servidor no sea un dolor de cabeza.
* En el frontend, héroes como **React**, **Vue** y **Svelte** nos dieron superpoderes para crear interfaces dinámicas y reactivas sin morir en el intento.
* Y si lo que queremos es orden y estilo, motores de plantillas como **Pug** nos ayudan a mantener el HTML limpio y dinámico.

Cada framework tiene su carácter: algunos son minimalistas y dejan que hagamos lo que queramos (Express), otros son más estructurados y casi nos dicen cómo organizar la casa (Angular). El punto es que ya no estamos solos picando líneas infinitas de JavaScript puro: los frameworks son **aliados** que ahorran tiempo, simplifican tareas repetitivas y permiten enfocarnos en la lógica y la creatividad.

En pocas palabras: los frameworks son como una buena banda de amigos. Algunos te acompañan en las fiestas pequeñas, otros en los conciertos gigantes, pero todos están ahí para que JavaScript brille.

## **Línea del tiempo (muy resumida)**

### **1995 – JavaScript nace.** Un script pequeñito para darle vida al navegador. Apenas podía mostrar alertas, pero ya apuntaba manera

### **2009 – Node.js aparece.** De pronto, JavaScript escapa del navegador y empieza a correr en servidores. **2010 – Express**

Primer gran framework para Node. Minimalista, rápido y flexible, se convierte en el estándar para levantar servidores.  
  
**2010 – Backbone.js.** Uno de los pioneros en traer orden al caos del frontend. MVC puro y duro.

### **2012 – AngularJS (1.x).** Google lanza AngularJS, con data binding mágico y un montón de directivas. Fue amado y odiado a partes iguales. **2013 – React.** Facebook presenta React, el maestro del Virtual DOM. Revoluciona la forma de pensar interfaces. **2014 – Vue.** Un ex de Google crea Vue, con la sencillez de AngularJS y la potencia de React, pero con menos drama. **2016 – Angular (2+).** Google reinventa Angular desde cero. Nuevo, más ordenado y con TypeScript. **2016 – Svelte.** Un framework que compila el código y entrega componentes ligerísimos. Sin Virtual DOM, sin tanto rollo. **Hoy** El ecosistema está lleno de frameworks y librerías, cada uno con su estilo y su “tribu”. Pero todos comparten un mismo origen: JavaScript

Veamos algunos conceptos básicos:  
**Node.js.** Node.js es un entorno de ejecución que permite usar JavaScript fuera del navegador, directamente en el servidor. Su magia está en que es rápido, eficiente y basado en eventos, lo que lo convierte en la base perfecta para crear aplicaciones web modernas, desde simples scripts hasta APIs escalables.

### **NPM.** NPM (Node Package Manager) es el gestor de paquetes que acompaña a Node.js. Piensa en él como una enorme caja de herramientas donde encontramos librerías listas para instalar con un solo comando. Gracias a NPM no reinventamos la rueda: simplemente buscamos, instalamos y usamos.

### **Express.** Express es un framework minimalista para Node.js que hace muy sencillo crear servidores web y APIs. Nos da rutas, middlewares y utilidades que convierten al servidor en un buen ciudadano de la web. Es rápido de aprender, flexible y prácticamente un estándar de facto en el ecosistema Node.

### **Pug.** Pug es un motor de plantillas que transforma un código limpio y basado en indentación en HTML completo. Con él evitamos etiquetas repetitivas, escribimos menos y mantenemos vistas mucho más legibles. Además, permite usar variables y lógica ligera, lo que lo convierte en un gran compañero de Express.

### **JavaScript.** JavaScript es el lenguaje que empezó como el rey indiscutible del navegador y ahora gobierna también en el servidor gracias a Node.js. Su flexibilidad, enorme ecosistema y comunidad lo han convertido en uno de los lenguajes más usados del mundo, ideal para conectar frontend y backend en un mismo idioma.

**Breve tutorial**

## **1️. Instalar Node.js**

1. Vamos a https://nodejs.org/ y descargamos la **versión LTS** (la estable).
2. Instalamos siguiendo las instrucciones del instalador.
3. Verificamos que se instaló correctamente abriendo la terminal o CMD y escribiendo:

node -v

npm -v

Deberías ver dos números de versión, algo como:

v20.5.1

9.6.0

* node → ejecuta JavaScript en tu máquina.
* npm → gestor de paquetes para instalar librerías como Express y Pug.

## **2️. Crear un proyecto**

1. Creamos una carpeta para nuestro proyecto, por ejemplo:

mkdir mi-proyecto-pug

cd mi-proyecto-pug

1. Inicializamos npm para crear package.json:

npm init -y

Esto crea un archivo package.json con configuración básica.

## **3️. Instalar Express y Pug**

npm install express pug

* express → el servidor web.
* pug → motor de plantillas HTML.

## **4️. Crear la estructura de carpetas**

Dentro de la carpeta del proyecto, crea:



## **5️. Escribir el servidor (index.js)**

const express = require('express');

const app = express();

const port = 3000;

*// Configuramos Pug*

app.set('view engine', 'pug');

app.set('views', './views');

*// Ruta principal*

app.get('/', (req, res) => {

res.render('index', { titulo: 'Hola Mundo', mensaje: 'Bienvenido a Express + Pug' });

});

*// Arrancamos servidor*

app.listen(port, () => {

console.log(`Servidor corriendo en http://localhost:${port}`);

});

## **6️. Crear la plantilla (views/index.pug)**

doctype html

html

head

title= titulo

body

h1= mensaje

p Esto es Pug, HTML limpio y dinámico.

* = indica que vamos a mostrar el valor de la variable que pasamos desde Express (titulo, mensaje).
* La indentación define la jerarquía de las etiquetas, no hace falta cerrar nada.

## **7️. Ejecutar el servidor**

En la terminal, desde la carpeta del proyecto:

node index.js

* Deberías ver: Servidor corriendo en http://localhost:3000
* Abrimos el navegador en esa URL y veremos la página renderizada.

**Los métodos de Express**

Los métodos que estamos usando dentro de **Express**, tienen cada uno un rol muy claro. Vamos a desentrañarlos:

## **app.get(ruta, callback)**

Sirve para **definir una ruta** que responde a peticiones **HTTP GET** (o sea, cuando un navegador pide una página con la URL).

Ejemplo:

app.get('/', (req, res) => {

res.send('Hola mundo');

});

Cuando alguien entra a http://localhost:3000/, Express ejecuta ese callback y responde “Hola mundo”.

* req → trae datos de la petición (parámetros, cabeceras, etc.)
* res → es lo que enviamos al navegador (texto, JSON, vistas…).

## **app.set(clave, valor)**

Configura opciones internas de Express.  
 Se usa sobre todo para decirle **qué motor de vistas usar** y **dónde están las plantillas**.

Ejemplo:

app.set('view engine', 'pug');

app.set('views', './views');

Que traducido sería: “vas a usar Pug y las plantillas estarán en la carpeta views”.

## **res.render(vista, datos)**

No es de app sino de res (respuesta). Sirve para **renderizar una plantilla** con el motor que configuramos (pug, ejs, etc.).

Ejemplo:

app.get('/', (req, res) => {

res.render('index', { titulo: 'Hola', mensaje: 'Bienvenido a Pug + Express' });

});

Busca views/index.pug, mete las variables (titulo, mensaje) y devuelve el HTML generado al navegador.

## **app.listen(puerto, callback)**

Levanta el servidor para que empiece a escuchar peticiones en un puerto específico.

Ejemplo:

app.listen(port, () => {

console.log(`Servidor corriendo en http://localhost:${port}`);

});

A partir de ahí, todo lo que definiste con app.get() y otras funcionalidades empieza a funcionar.