

## Presentación y objetivos del curso

# Introducción

- Elementos de una aplicación NodeJS
- Construyendo una red social con NodeJS
- E-commerce con NodeJS

# Elementos de una aplicación con NodeJS

- Introducción e Instalación
- Modules
- Callbacks
- Eventos
- Streams
- Buffers
- Conexión a bd
- WebSocket
- APIs RESTful...

# Construyendo una red social con NodeJS

- Construcción del Usuario: Login y Registro
- Construcción del sistema de seguimiento: *follows*
- Las publicaciones
- Mensajería Privada

# Construyendo un e-commerce con NodeJS

- Gestión de la sesión
- Gestión de productos
- Gestión de categorías
- Gestión del carrito

# Introducción

# Qué es NodeJS

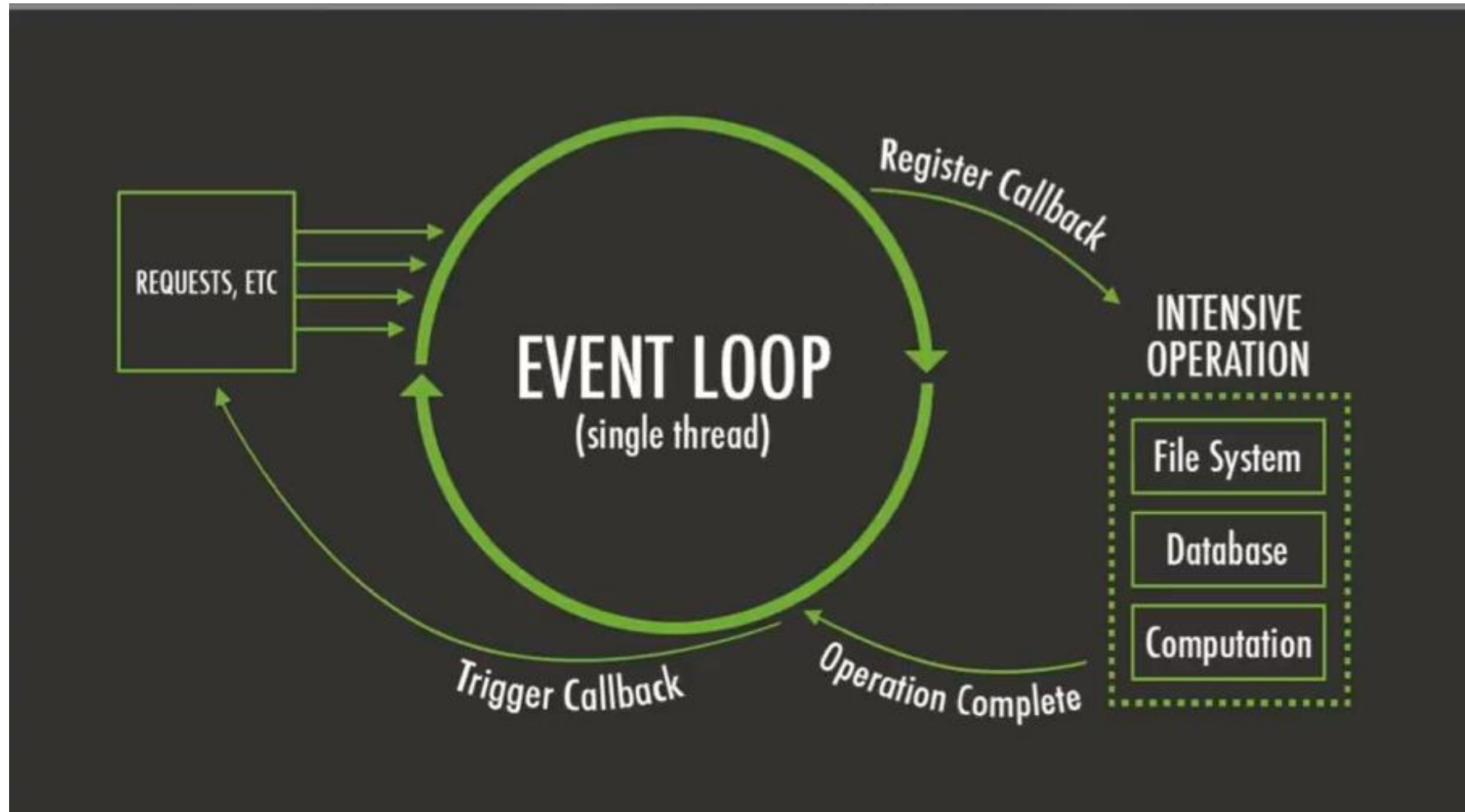
- Plataforma construida en tiempo de ejecución de JavaScript de Chrome para construir fácilmente aplicaciones de red rápidas y escalables
- Utiliza un modelo orientado a eventos, sin bloqueo de E/S que hace que sea ligero y eficiente, ideal para aplicaciones en tiempo real de datos intensivos que se ejecutan a través de dispositivos distribuidos
- Proporciona una amplia biblioteca de varios módulos de JavaScript que simplifica el desarrollo de aplicaciones web usando Node.js

# Introducción: NodeJS

- Single Thread
- Concurrente
- No paralelismo
- Ejecución de callbacks (varias acciones)



# NodeJS



# Características

- **Asíncrono y controlado por eventos:** todas las API de la biblioteca Node.js son asíncronas, es decir, sin bloqueo. Significa esencialmente que un servidor basado en Node.js nunca espera a que una API devuelva datos. El servidor se mueve a la siguiente API después de llamarla y un mecanismo de notificaciones de Eventos de Node.js ayuda al servidor a obtener una respuesta de la API llamada anteriormente.
- **Muy rápido:** Construido sobre el motor de JavaScript V8 de Google Chrome, la biblioteca Node.js es muy rápida en la ejecución de código.
- **Único subproceso pero altamente escalable:** Node.js utiliza un único modelo de roscado con el evento de bucle.
- **Sin búfer:** Las aplicaciones simplemente dan salida a los datos en fragmentos, no los almacenan.

# Casos de uso

- <https://www.netguru.co/blog/top-companies-used-nodejs-production>

# Instalación

# Instalación

- Conectarse a la página oficial: <https://nodejs.org/es/>
- LTS versión de Node.js con Long Term Support (LTS). Esta versión puede no tener disponibles las últimas tecnologías que todavía no se consideran estables.
- Current: esta es la versión más reciente de Node.js e incluye todas las funcionalidades, incluso aquellas más novedosas y que no se consideran estables.
- Se recomienda seleccionar la versión LTS

# Instalación

- Utilizar el archivo .msi y seguir las instrucciones para instalar Node.js.
- De forma predeterminada, el instalador utiliza la distribución Node.js en C:\Archivos de programa\nodejs.
- El instalador establece el directorio C:\Archivos de programa\nodejs\bin en la variable de entorno PATH de la ventana.


# Verificar la instalación: Ejecución de un archivo

- Una vez instalado, para probar que todo funciona correctamente, abrir una consola de comandos y ejecutar el comando “node”. Si todo ha ido bien, estaremos dentro de la consola de Node.js y se puede ejecutar la siguiente línea de código:

```
console.log("hola mundo");
```

- Crear un archivo main.js con el código anterior y ejecutar con node.

# Instalación de IDE: VisualStudioCode

- Descargamos e instalamos de la página oficial:  
<https://code.visualstudio.com>
- Trabajar con node en VScode:  
<https://code.visualstudio.com/docs/nodejs/nodejs-tutorial>
- Extensiones interesantes de VSCode para NodeJS: 
  - Node.js Extension Pack
  - <https://www.sitepoint.com/vs-code-extensions-javascript-developers/>



# Primera App

- Crearemos una aplicación web con "Hello, World!" con los siguientes Componentes:
  - **Import de los módulos requeridos** – Mediante la directiva **require** que carga módulos Node.js.
  - **Create server** – Genera un servidor que escucha peticiones http
  - **Escucha peticiones y devuelve respuestas** – El servidor recibe una petición http de un cliente, navegador o consola y devuelve una respuesta

```
const http = require("http");

const handler = (request, response) => {
  console.log('Recibimos petición');
  response.end("<p>Hola Mundo web</p>");
}

const server = http.createServer(handler);

server.listen(8080);

console.log('Server running at
http://127.0.0.1:8080/');
```

# NodeJS: NPM

- Node Package Manager (NPM) tiene dos funciones principales:
  - Repositorio Online de paquetes/módulos de node.js accesibles desde [search.npmjs.org](https://search.npmjs.org)
  - Herramienta de línea de comando para instalar paquetes u gestionar sus dependencias y versiones
- En la actualidad ronda los 800.000 paquetes

# NodeJS: NPM

- Para instalar dependencias:

```
$ npm install <Module Name>
```

- Ej. Para instalar express: 

```
$ npm install express
```

- Después ya se podría utilizar en el código:

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

# NodeJS: NPM

- Por defecto NPM instala las dependencias en modo local. Es decir, crea un directorio **node\_modules** dentro del directorio donde estamos ejecutando el comando.
  - Se puede usar **npm ls** para listar los módulos instalados en local.
- En la instalación global los paquetes se almacenan en el directorio system. Estas dependencias se pueden utilizar con un CLI (Command Line Interface) pero no se pueden importar en la aplicación directamente con `require()`

```
$ npm install express -g
```

# NodeJS: NPM

- Para desinstalar un package: `$ npm uninstall express`
- Para actualizar un package: `$ npm update express`
- Para buscar un package: `$ npm search express`

# NodeJS: Package.json

- Se encuentra en el directorio raíz de cualquier aplicación/modulo de node.
- Se utiliza para definir las propiedades de una aplicación o Package
- Para generarlo en nuestra aplicación utilizaremos:

```
$ npm init
```

# NodeJS: Package.json

- Propiedades:
- **name**
  - Campo obligatorio (*junto a version*). Es un string que representa el nombre del proyecto actual y forma un identificador único entre este campo y version en caso de que sea publicado al registro de npm.
  - no puede contener mayúsculas ni empezar con un punto o guión bajo.
  - Largo máximo 214 caracteres
- **version**
  - string con la versión actual del proyecto.
  - siguen las convenciones definidas en Semantic Versioning (semver), la cual define la forma de versionamiento: **MAJOR.MINOR.PATCHname**
- **description**
  - string que describa lo que hace este proyecto.
- **keywords**
  - array de strings que incluye términos que puedan ser utilizados para una eventual búsqueda.
- **homepage**
  - Es un string con la URL del proyecto.

# NodeJS: Package.json

- Propiedades:
- **bugs**
  - string con una URL válida para reportar problemas con el proyecto. Usualmente el link de los issues del repositorio.
- **license**
  - string que especifica que tipo de licencia definimos para el uso de este proyecto, ya sea de forma personal, comercial, abierta y/o privada.
- **author**
  - string o un objeto con la información del creador del proyecto. Si es un objeto, incluye las siguientes propiedades:
    - **name**
    - **email**
    - **url**
  - Si es un string, es en formato: "Nombre <email> (url)"
- **contributors**
  - Igual a author, array de colaboradores del proyecto.



# NodeJS: Package.json

- Propiedades:
- **files**
  - array de strings o patrones (ejemplo: \*.js) que serán incluidos en caso de publicar el proyecto en el registro de npm. Si no se incluye esta sección, todos los archivos serán publicados, a excepción de los excluidos (por ejemplo los definidos dentro del .gitignore).
  - Como alternativa a esta sección, se puede incluir un .npmignore que funciona de manera similar a un .gitignore dentro de un proyecto.
- **main**
  - string que define la ruta del archivo principal o punto de entrada al proyecto.
- **bin**
  - string (si es uno solo) o un objeto (si son múltiples) con la definición de scripts que queremos instalar como ejecutables en el PATH.
- **man**
  - string, o un array de strings, que especifica uno (o muchos archivos) que se relacionarán a este proyecto si se corre el comando man en la máquina donde se haya instalado.

# NodeJS: Package.json

- Propiedades:
- **directories**
  - objeto que especifica las rutas para la estructura de directorios del proyecto. Ejemplo:

```
{  "bin": "./bin",  "doc": "./doc",  "lib": "./lib"}
```
- **repository**
  - objeto que especifica el tipo y URL del repositorio donde está el código del proyecto. Se usa el siguiente formato:
  - Ejemplo:

```
{  "type": "git",  "url": "https://github.com/mi-usuario/mi-proyecto"}
```
- **scripts**
  - objeto que indica comandos que se pueden correr dentro del proyecto, asociándolos a una palabra clave para que npm (o yarn) los reconozca cuando queramos ejecutarlos.
  - Hay algunos scripts que vienen predefinidos en todos los proyectos al momento de utilizar npm, como son: start, install, preinstall, pretest, test y posttest entre otros (para una lista completa, pueden revisar este enlace).
- **config**
  - objeto al que se pueden pasar valores y usarlos como variables de ambiente dentro del proyecto.

# NodeJS: Package.json

- Propiedades:
- **dependencies**
  - objeto que guarda los nombres y versiones de cada dependencia instalada dentro del proyecto.
  - cada vez que se ejecute el comando npm install, se instalarán todas las dependencias que aquí estén definidas
  - se definen de la siguiente forma:  
"nombre-de-la-dependencia": "(^|~|version)|url"
  - Pueden llevar como valor la versión que están utilizando, o de una URL desde donde obtenerla (incluso una ruta local en la misma máquina), la cual por lo general apunta a una versión específica.
    - Si la versión tiene un ^: Se buscará una versión compatible con la que está definida ahí.
    - Si la versión tiene un ~: Se buscará una versión lo más cercana posible a la definida.
    - Si no tiene ningún símbolo: Se instalará la misma versión.
- **devDependencies**
  - Mismo formato que las dependencias, Incluye todas las librerías que no son necesarias para que el proyecto se ejecute en producción, o cuando sea requerido e instalado dentro de otro.
- **peerDependencies**
  - Mismo formato anterior, Dependencias que son necesarias para el uso del proyecto

# NodeJS: Package.json

- Propiedades:
- **engines**
  - objeto en el cual podemos definir la versión mínima de node y npm necesarias para ejecutar el proyecto. La definimos de la forma:  

```
"engines": {  
  "node": "≥ 6.0.0",  
  "npm": "≥ 3.0.0"  
}
```
  - Cuando se instala el proyecto o package, se verifican las versiones instaladas, si no se cumple el requisito, no se continuará el proceso de instalación.
  - Al igual que con las dependencias, podemos usar ~ y ^ junto al número de versión.

# Ejercicio 1

- Crea un proyecto **ejercicio1** con un fichero main.js y el código hola mundo web.
- Añade un fichero package.json y crea un script “start” para ejecutar el proyecto
- Añade los package jade, express y mongoose.