

Trabajo Práctico 1: Introducción a Node.js y Metodología Scrum en la Educación

Índice

1. Metodología Scrum aplicada a la educación

- 1.1 Introducción a Scrum
- 1.2 Ceremonias clave de Scrum
- 1.3 Roles en Scrum
- 1.4 Uso de Trello para la gestión ágil de proyectos
- 1.5 Ejemplo de implementación de Scrum en un proyecto educativo

2. Introducción a Node.js

- 2.1 ¿Qué es Node.js?
 - 2.2 Arquitectura de Node.js
 - 2.3 Instalación de Node.js y configuración del entorno de desarrollo
 - 2.4 Casos de uso de Node.js
 - 2.5 Ventajas y desventajas de Node.js
 - 2.6 Comparativa con otras tecnologías
 - 2.7 Ejemplos prácticos y detallados
-

1. Metodología Scrum aplicada a la educación

1.1 Introducción a Scrum

Scrum es una metodología ágil diseñada para organizar y gestionar el desarrollo de proyectos complejos. Su enfoque permite a los equipos trabajar de forma colaborativa y eficiente para alcanzar objetivos comunes a través de ciclos cortos de trabajo llamados **sprints**.

Documentación respaldatoria:

Scrum es un marco de trabajo ágil que se enfoca en la mejora continua y en la entrega de productos funcionales a través de iteraciones regulares. Está estructurado en ceremonias, roles y artefactos que aseguran un flujo de trabajo eficiente y colaborativo.

1.2 Las ceremonias clave en Scrum

Scrum se estructura alrededor de cuatro ceremonias fundamentales que aseguran el progreso continuo y el ajuste del equipo en cada sprint:

1. Sprint Planning (Planificación del Sprint)

- **Cuándo:** Al inicio de cada sprint, generalmente los lunes.
- **Qué es:** Una reunión donde se decide el trabajo que se realizará durante las próximas dos semanas.
- **Objetivo:** Definir un plan claro para todo el equipo sobre qué tareas se realizarán y quién será responsable de cada una.

2. Daily Stand-up (Reunión diaria)

- **Cuándo:** Todos los días, con reuniones oficiales con el docente los martes y jueves.
- **Duración:** 5-10 minutos.
- **Qué es:** Cada miembro del equipo responde tres preguntas clave:
 1. ¿Qué hice ayer?
 2. ¿Qué haré hoy?
 3. ¿Hay algo que me está bloqueando?
- **Objetivo:** Mantener a todos alineados y resolver obstáculos rápidamente.

3. Sprint Review (Revisión del Sprint)

- **Cuándo:** El último viernes de cada sprint.
- **Qué es:** El equipo presenta el trabajo completado, recibe retroalimentación del docente y de sus compañeros.
- **Objetivo:** Validar el trabajo realizado y generar aprendizaje a partir de la retroalimentación recibida.

4. Sprint Retrospective (Retrospectiva)

- **Cuándo:** Despues de la Sprint Review.
- **Qué es:** Reflexión interna del equipo sobre el trabajo realizado. ¿Qué salió bien? ¿Qué se puede mejorar?
- **Objetivo:** Identificar oportunidades de mejora para el próximo sprint.

1.3 Roles en Scrum

En este módulo, los equipos de trabajo estarán organizados en grupos de 4 personas, cada una con roles definidos:

- **Capitán del equipo:** Responsable de liderar la planificación y organización del equipo.
- **Presentador:** Encargado de presentar los resultados al final de cada sprint.
- **Desarrolladores:** Trabajan en las tareas asignadas, participando activamente en las reuniones diarias y en la ejecución del plan. Todos en el equipo seremos desarrolladores.

1.4 Uso de Trello para la gestión ágil de proyectos

Para organizar las tareas de cada equipo, utilizaremos **Trello**, una herramienta de gestión de proyectos visual y fácil de usar. Cada equipo contará con un tablero dividido en las siguientes columnas:

1. **Backlog del Sprint:** Tareas pendientes de realizar durante el sprint.
 2. **En progreso:** Tareas en las que el equipo está trabajando actualmente.
 3. **Bloqueado/En espera:** Tareas que no pueden completarse por alguna razón y necesitan atención.
 4. **Completado:** Tareas terminadas.
-

1.5 Ejemplo de implementación de Scrum en un proyecto educativo

Imagina que tienes que organizar una **cena de fin de año** familiar. Lo primero es crear un plan, asignar roles (¿quién prepara la comida? ¿quién se encarga de las bebidas?) y revisar el progreso de las tareas asignadas. Scrum funciona de manera similar en un equipo de desarrollo: se planifica, se trabaja en equipo y se ajusta el plan según sea necesario.

En este curso, aplicarás Scrum para gestionar proyectos en Node.js, utilizando Trello para mantener el flujo de trabajo organizado y obtener resultados exitosos al final de cada sprint.

2. Introducción a Node.js

2.1 ¿Qué es Node.js?

Node.js es un entorno de ejecución para JavaScript construido sobre el motor V8 de Google Chrome. Permite ejecutar código JavaScript en el lado del servidor, expandiendo las capacidades de JavaScript más allá del navegador.

Según *Node.js for Beginners* (2024), Node.js combina el motor V8 con la biblioteca **libUV**, que proporciona un modelo de I/O no bloqueante y basado en eventos, lo que permite manejar múltiples conexiones simultáneas de manera eficiente.

2.2 Arquitectura de Node.js

Node.js se basa en un modelo de un solo hilo (single-threaded) y un bucle de eventos (event loop). Aunque es de un solo hilo, puede manejar operaciones de I/O de manera asíncrona gracias a su arquitectura basada en eventos y callbacks, lo que evita el bloqueo del hilo principal.

Componentes clave:

- **Motor V8:** Compila código JavaScript a código máquina nativo, proporcionando alta velocidad de ejecución.

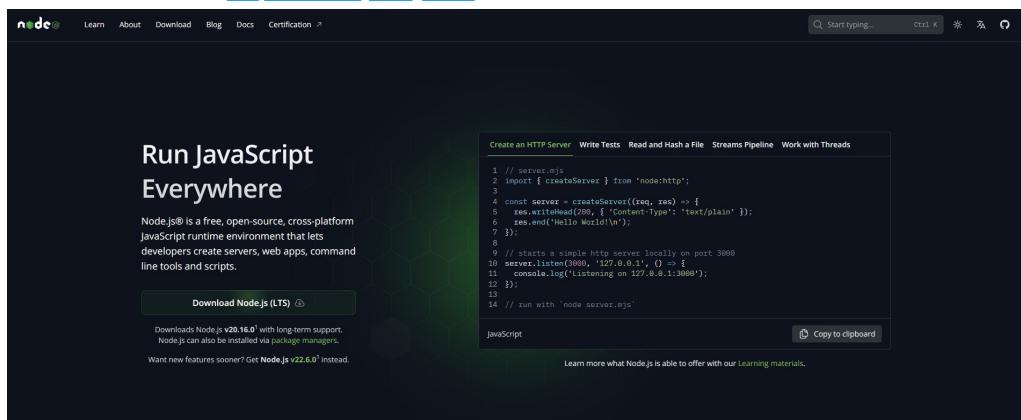
- **libUV**: Biblioteca que proporciona I/O asíncrono y no bloqueante, manejando el sistema de archivos, redes y otros.
- **Event Loop**: Mecanismo que gestiona las operaciones asíncronas, permitiendo que Node.js maneje múltiples operaciones concurrentes sin múltiples hilos.

2.3 Instalación de Node.js y entorno de desarrollo

2.3.1 Instalación de Node.JS

Como instalar Node.js ?

1. Ingresar a la pagina <https://nodejs.org/en/>



2. Click en Download Node.js(LTS) que significa LTS preguntar ?

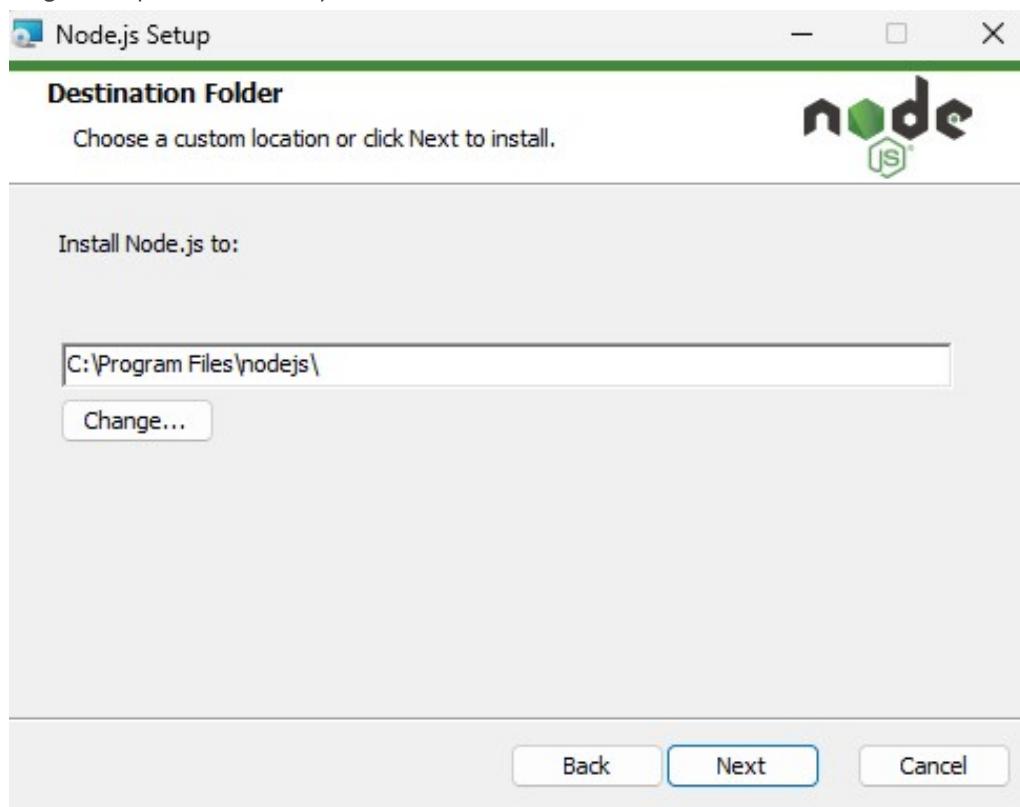
3. Correr el instalador



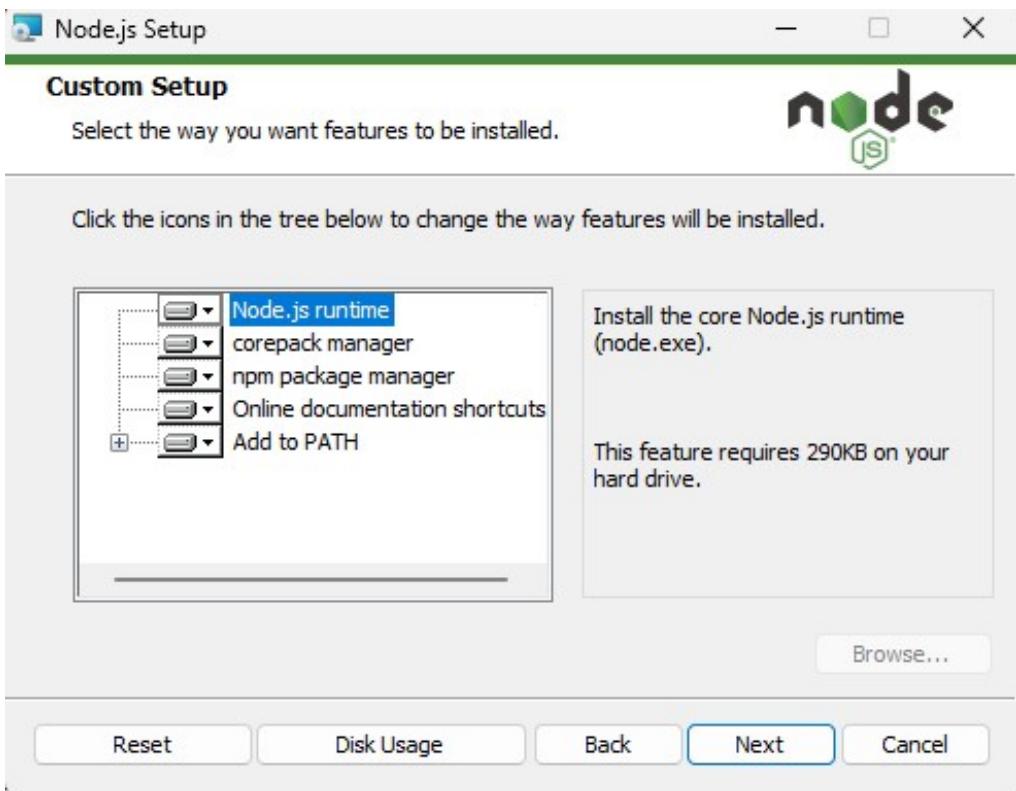
4. Aceptar el acuerdo de licencia y continuar



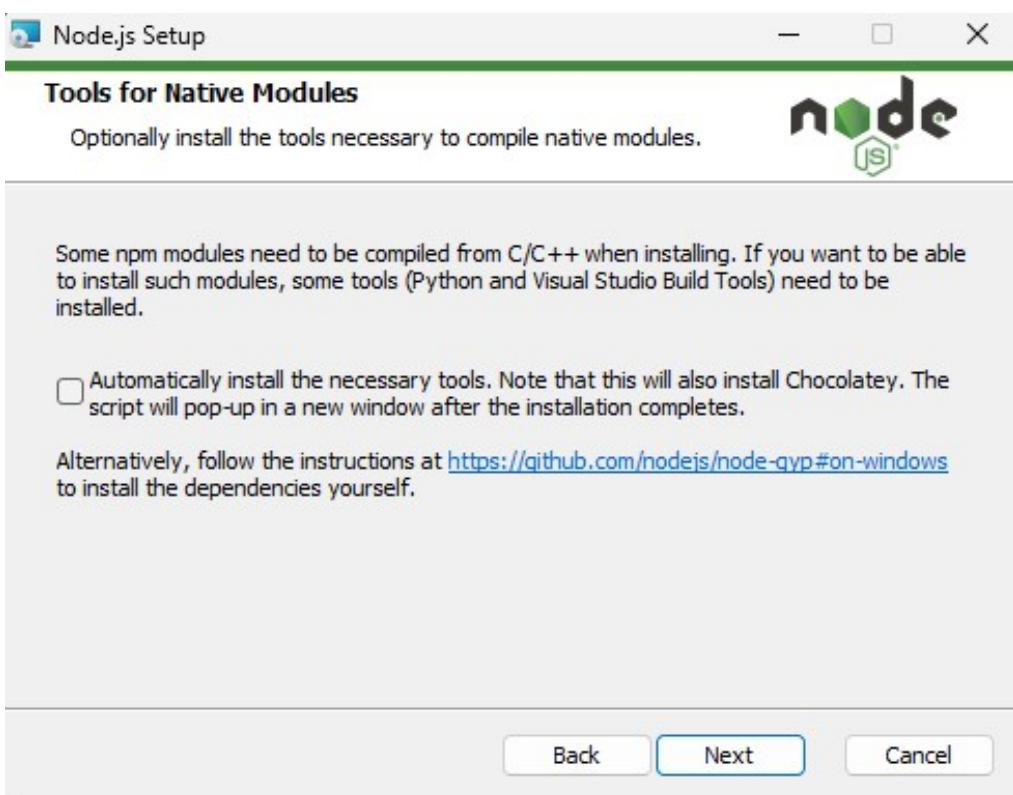
5. Elegir la carpeta de destino y continuar



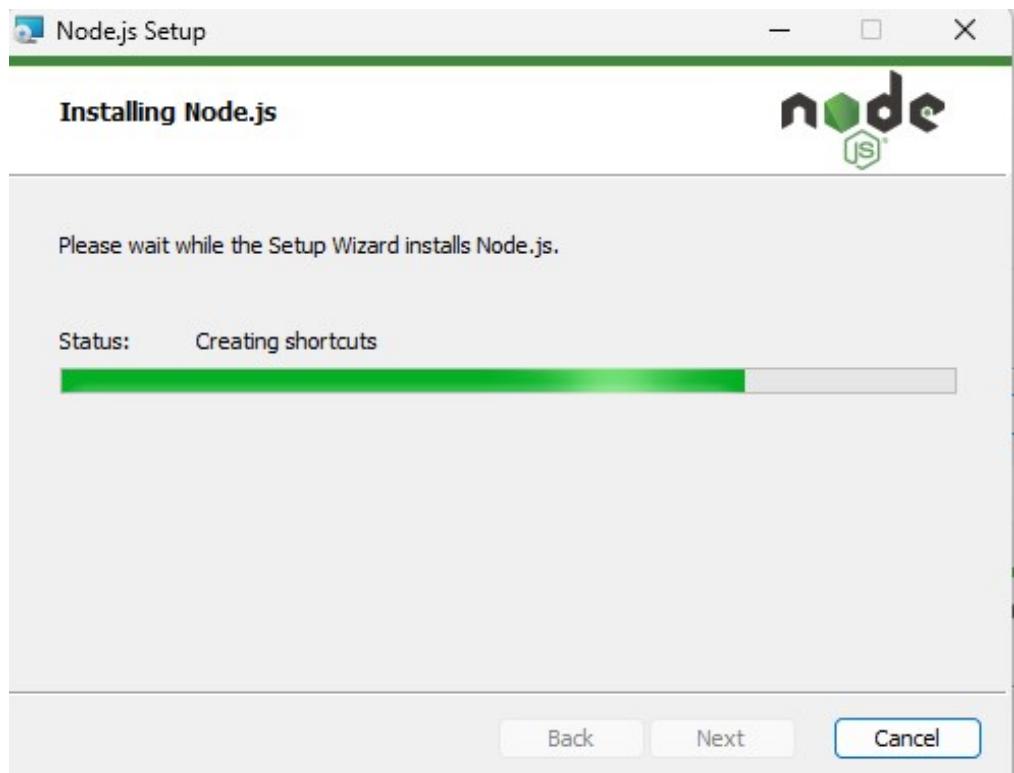
6. Aceptar la configuracion por defecto



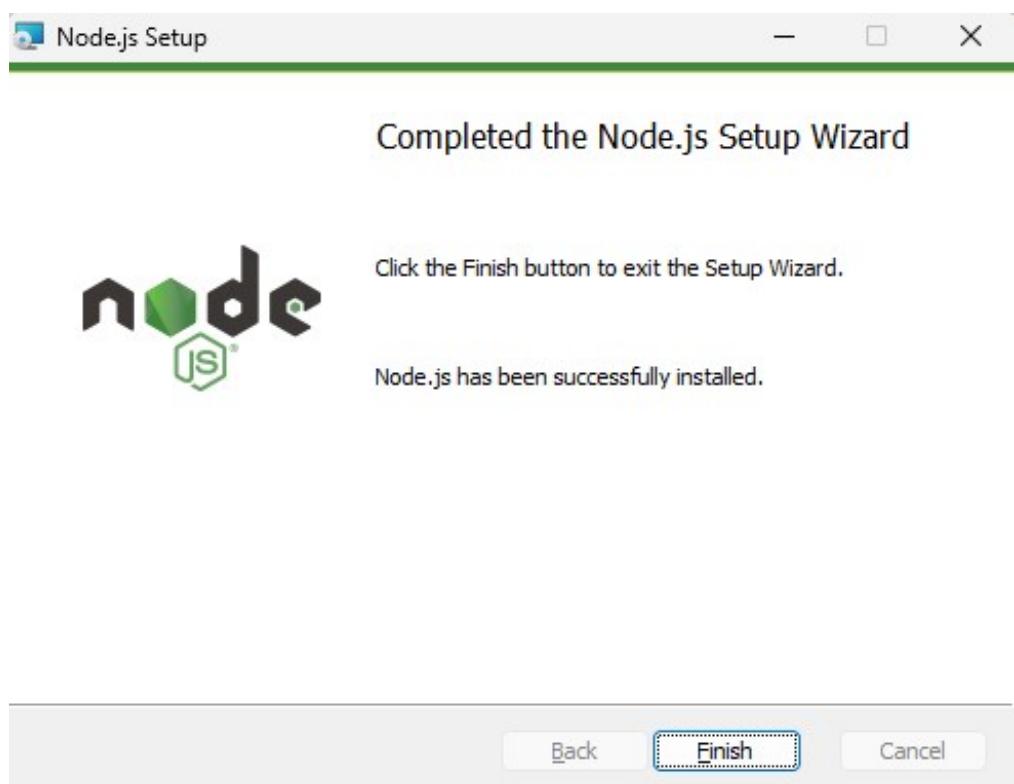
7. No hace falta instalar herramientas adicionales



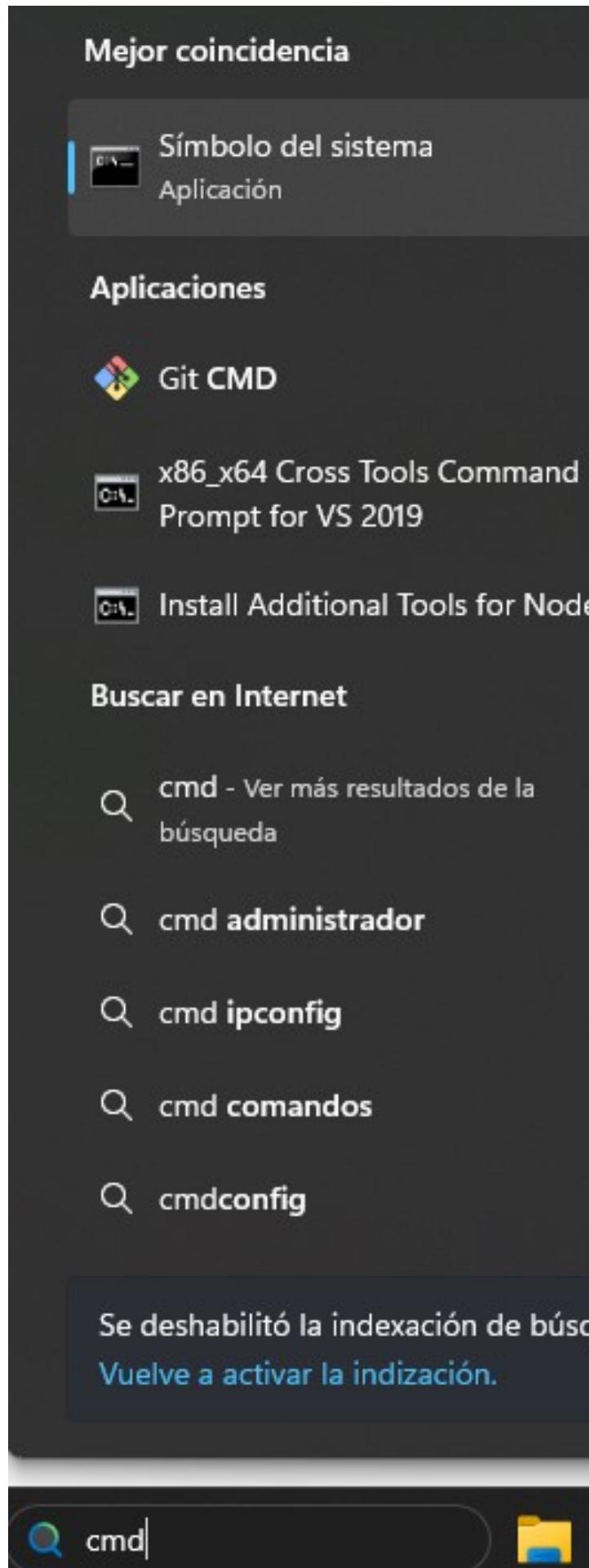
8. Esperamos q termine el instalador



9. Listo termino

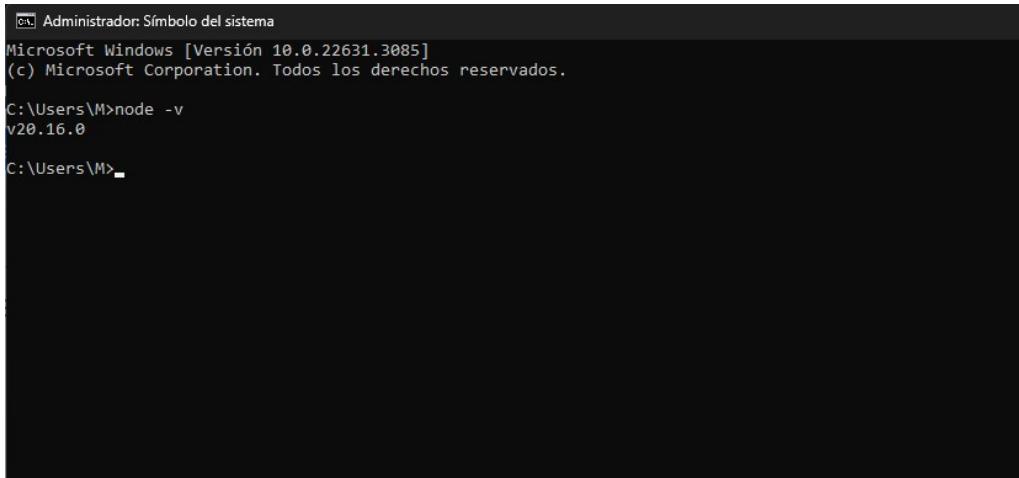


10. Para comprobar la instalacion, en la consola simbolo de sistema en windows (en menu de inicio ponemos cmd)



11. y corremos el comando:

```
node -v
```



```
Administrator: Símbolo del sistema
Microsoft Windows [Versión 10.0.22631.3085]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\M>node -v
v20.16.0

C:\Users\M>-
```

Al figurarnos la versión que tenemos, ya estamos seguros que se instaló correctamente !!

2.4 Casos de uso de Node.js

- **Aplicaciones en tiempo real:** Chats, juegos multijugador, colaboración en tiempo real.
- **APIs RESTful y servicios web:** Backend para aplicaciones móviles y web.
- **Aplicaciones de streaming:** Plataformas de video y audio en directo.
- **Automatización y scripts:** Tareas de automatización, CLI tools.
- **Microservicios:** Arquitecturas basadas en microservicios para escalabilidad.

Ejemplo práctico:

- **Chat en tiempo real:** Node.js, combinado con **WebSocket**, permite establecer conexiones bidireccionales entre el cliente y el servidor, esencial para aplicaciones de chat.

2.5 Ventajas y desventajas de Node.js

Ventajas:

- **I/O no bloqueante:** Manejo eficiente de operaciones de entrada/salida, ideal para aplicaciones de alto rendimiento.
- **Un solo lenguaje:** Uso de JavaScript en frontend y backend, facilitando el desarrollo full-stack.
- **Gran ecosistema:** Amplia variedad de módulos y paquetes disponibles en npm.
- **Escalabilidad:** Fácil de escalar horizontalmente añadiendo más nodos.

Desventajas:

- **Operaciones intensivas en CPU:** No es ideal para tareas que requieren mucho procesamiento, como cálculos matemáticos complejos.
- **Madurez del ecosistema:** Aunque ha crecido, algunas herramientas y prácticas aún están en desarrollo comparado con lenguajes más antiguos.
- **Callback Hell:** Código difícil de leer debido a múltiples callbacks anidados, aunque mitigado con promesas y async/await.

2.6 Comparativa con otras tecnologías

Node.js vs. Python (Django, Flask):

- **Rendimiento:** Node.js suele ser más rápido en operaciones I/O gracias a su naturaleza no bloqueante.
- **Lenguaje:** JavaScript vs. Python; elección depende de las habilidades del equipo y requisitos del proyecto.
- **Comunidades activas:** Ambos tienen grandes comunidades y soporte.

Node.js vs. Java (Spring Boot):

- **Concurrencia:** Node.js maneja concurrencia mediante un solo hilo y event loop, mientras que Java utiliza múltiples hilos.
- **Ecosistema:** Java tiene un ecosistema más maduro para aplicaciones empresariales; Node.js es más ligero y rápido de desarrollar.

2.7 Ejemplos prácticos y detallados

2.7.1 Ejemplo teórico: Hola Mundo en Node.js

Objetivo: Crear un servidor HTTP básico que responda “Hola Mundo”.

Código (server.mjs):

```
// Importamos el módulo 'http' de Node.js
import http from 'http';

// Definimos el hostname y el puerto
const hostname = '127.0.0.1';
const port = 3000;

// Creamos el servidor
const server = http.createServer((req, res) => {
    // Establecemos el código de estado y los headers
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');

    // Enviamos la respuesta al cliente
    res.end('Hola Mundo\n');
});

// El servidor comienza a escuchar en el puerto y hostname especificados
server.listen(port, hostname, () => {
    console.log(`El servidor se está ejecutando en http://${hostname}:${port}/`);
});
```

Explicación detallada:

1. Importación del módulo `http`:

```
import http from 'http';
```

El módulo `http` es parte de los módulos core de Node.js y permite crear servidores web.

2. Definición del hostname y puerto:

```
const hostname = '127.0.0.1';
const port = 3000;
```

- **hostname**: Dirección IP local donde el servidor escuchará.
- **port**: Puerto en el que el servidor estará activo.

3. Creación del servidor:

```
const server = http.createServer((req, res) => {
  // ...
});
```

- **http.createServer**: Método que crea un nuevo servidor HTTP.
- **(req, res) => { ... }**: Función callback que maneja cada solicitud entrante.

4. Configuración de la respuesta:

```
res.statusCode = 200;
res.setHeader('Content-Type', 'text/plain');
res.end('Hola Mundo\n');
```

- **res.statusCode = 200**: Establece el código de estado HTTP a 200 (OK).
- **res.setHeader('Content-Type', 'text/plain')**: Indica que el contenido es texto plano.
- **res.end('Hola Mundo\n')**: Envía la respuesta y finaliza la conexión.

5. Inicio del servidor:

```
server.listen(port, hostname, () => {
  console.log(`El servidor se está ejecutando en http://${hostname}:${port}/`);
});
```

- **server.listen**: Inicia el servidor y lo hace escuchar en el puerto y hostname especificados.
- **Callback**: Función que se ejecuta cuando el servidor comienza a escuchar.

2.7.2 Ejemplo práctico: Servidor HTTP que gestiona diferentes rutas

Objetivo: Crear un servidor que maneje diferentes rutas y métodos HTTP.

Código (advanced_server.mjs):

```
import http from 'http';
import url from 'url';

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  // Parseamos la URL de la solicitud
  const parsedUrl = url.parse(req.url, true);
  const pathname = parsedUrl.pathname;
```

```

const method = req.method;

// Rutas y métodos
if (pathName === '/' && method === 'GET') {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Página de inicio\n');
} else if (pathName === '/about' && method === 'GET') {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Acerca de nosotros\n');
} else if (pathName === '/data' && method === 'POST') {
  let body = '';

  // Recolegemos los datos enviados en el cuerpo de la solicitud
  req.on('data', chunk => {
    body += chunk.toString();
  });

  // Una vez recibidos todos los datos, procesamos la información
  req.on('end', () => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'application/json');
    res.end(`Datos recibidos: ${body}\n`);
  });
} else {
  res.statusCode = 404;
  res.end('Ruta no encontrada\n');
}

server.listen(port, hostname, () => {
  console.log(`Servidor corriendo en http://:${hostname}:${port}/`);
});

```

Explicación detallada:

1. Importación de módulos:

```

import http from 'http';
import url from 'url';

```

- `url`: Módulo para parsear URLs y manejar rutas.

2. Extracción de la ruta y método HTTP:

```

const parsedUrl = url.parse(req.url, true);
const pathName = parsedUrl.pathname;
const method = req.method;

```

- `req.url`: Contiene la URL de la solicitud.
- `req.method`: Indica el método HTTP utilizado (GET, POST, etc.).

3. Manejo de rutas y métodos:

- Ruta / con método GET:

```
if (pathName === '/') && method === 'GET') {  
    // ...  
}
```

Responde con “Página de inicio”.

- Ruta /about con método GET:

```
else if (pathName === '/about' && method === 'GET') {  
    // ...  
}
```

Responde con “Acerca de nosotros”.

- Ruta /data con método POST:

```
else if (pathName === '/data' && method === 'POST') {  
    // ...  
}
```

Recolecta los datos enviados en el cuerpo de la solicitud y los devuelve en la respuesta.

4. Recolección y manejo de datos en una solicitud POST:

```
let body = '';  
  
req.on('data', chunk => {  
    body += chunk.toString();  
});  
  
req.on('end', () => {  
    // Procesamos los datos recibidos  
});
```

- `req.on('data', callback)`: Evento que se dispara cuando se reciben datos.
- `req.on('end', callback)`: Evento que se dispara cuando se han recibido todos los datos.

5. Respuesta para rutas no encontradas:

```
else {  
    res.statusCode = 404;  
    res.end('Ruta no encontrada\n');  
}
```

2.7.3 Ejemplo práctico: Interacción con el sistema de archivos

Objetivo: Leer y escribir en un archivo utilizando el módulo `fs`.

Código (`file_operations.mjs`):

```
import fs from 'fs';

// Escribir en un archivo
const data = 'Este es un texto de ejemplo.\n';

fs.writeFile('ejemplo.txt', data, (err) => {
  if (err) throw err;
  console.log('El archivo ha sido guardado.');

  // Leer el archivo recién creado
  fs.readFile('ejemplo.txt', 'utf8', (err, content) => {
    if (err) throw err;
    console.log('Contenido del archivo:');
    console.log(content);
  });
});
```

Explicación detallada:

1. Importación del módulo `fs`:

```
import fs from 'fs';
```

2. Escribir en un archivo:

```
fs.writeFile('ejemplo.txt', data, (err) => {
  if (err) throw err;
  // ...
});
```

- `fs.writeFile`: Método para escribir datos en un archivo.
- Si el archivo no existe, se crea; si existe, se sobrescribe.

3. Leer el archivo:

```
fs.readFile('ejemplo.txt', 'utf8', (err, content) => {
  if (err) throw err;
  // ...
});
```

- `fs.readFile`: Método para leer el contenido de un archivo.
- La codificación '`'utf8'`' es importante para interpretar correctamente el contenido.

4. Manejo de errores:

- En ambos casos, si ocurre un error, se lanza una excepción.
- En aplicaciones reales, es recomendable manejar los errores de manera más elegante.

Nota final: A medida que avanzamos en los siguientes trabajos prácticos, reutilizaremos y ampliaremos el código y los conceptos aprendidos, fomentando una comprensión profunda y la capacidad de integrar diferentes módulos y técnicas en proyectos más complejos.

Conclusión

En este trabajo práctico, hemos profundizado en la introducción a Node.js, explorando su arquitectura, instalación y casos de uso. También hemos implementado ejemplos prácticos que ilustran cómo crear servidores, manejar rutas y interactuar con el sistema de archivos. Además, hemos aprendido sobre la metodología Scrum y cómo aplicarla en entornos educativos para mejorar el aprendizaje colaborativo y la gestión de proyectos.

Referencias:

- *Node.js for Beginners* (2024)
 - Documentación oficial de Node.js: <https://nodejs.org/es/docs/>
 - Documentación de Visual Studio Code: <https://code.visualstudio.com/docs>
-

Próximos pasos:

En los siguientes trabajos prácticos, exploraremos módulos core de Node.js como `path`, `os`, `fs`, `events` y `http`, y aplicaremos la arquitectura MVC para desarrollar aplicaciones más estructuradas y escalables.