

GUÍA DE NODE.JS Y EXPRESS

1. ¿Qué es Node.js?

Node.js es un entorno de ejecución de JavaScript del lado del servidor que utiliza el motor V8 de Google Chrome para ejecutar código JavaScript fuera del navegador. Permite a los desarrolladores utilizar JavaScript para escribir herramientas de línea de comandos y scripts del lado del servidor, además de aplicaciones web tradicionales.

2. ¿En qué lenguaje está escrito Node.js?

Node.js está principalmente escrito en C++ para las partes principales de su funcionalidad. Utiliza JavaScript para la escritura de módulos y aplicaciones, pero el entorno de ejecución y la mayoría de sus bibliotecas fundamentales están implementadas en C++ para optimizar el rendimiento y la interacción con el sistema operativo.

3. Menciona características principales de Node.js.

- **Modelo de E/S sin bloqueo y orientado a eventos:** Node.js utiliza un modelo de E/S no bloqueante que lo hace eficiente y ligero, ideal para aplicaciones con muchas operaciones de E/S.
- **Escalabilidad:** Permite manejar un gran número de conexiones simultáneas con una eficiencia superior a los servidores tradicionales.
- **Ideal para aplicaciones en tiempo real:** Su capacidad para manejar eventos y conexiones en tiempo real lo hace perfecto para aplicaciones como chats, juegos en línea y streaming.
- **Ecosistema robusto con npm:** Node.js tiene acceso a npm, el gestor de paquetes más grande del mundo, que facilita la reutilización de código y la integración de bibliotecas externas.

4. ¿Qué comando se utiliza para verificar la versión de Node.js instalada?

El comando **node -v** se utiliza para verificar la versión de Node.js instalada en el sistema.

5. ¿Qué archivo es fundamental en un proyecto Node.js para gestionar dependencias?

En un proyecto Node.js, el archivo fundamental para gestionar dependencias es **package.json**. Este archivo no solo lista las dependencias del proyecto, sino que también define scripts de npm, información del proyecto como nombre, versión, autor y más metadatos relevantes.

6. ¿Qué es Express?

Express es un **framework** web minimalista y flexible para Node.js. Proporciona una serie de características para facilitar el desarrollo de aplicaciones web y APIs, como el enrutamiento de URLs, manejo de solicitudes y respuestas HTTP, soporte para motores de plantillas, entre otros.

7. ¿Cuál es el propósito principal de Express?

El propósito principal de Express es simplificar el desarrollo de aplicaciones web en Node.js proporcionando una capa de abstracción sobre el manejo de solicitudes y respuestas HTTP. Facilita la creación de rutas, manejo de middlewares y configuración de vistas, permitiendo a los desarrolladores construir aplicaciones web de manera rápida y eficiente.

8. ¿Cómo se instala Express en un proyecto Node.js?

Express se instala en un proyecto Node.js utilizando el comando **npm install express**. Esto descarga e instala el paquete Express desde el registro de npm y lo añade como una dependencia en el archivo package.json del proyecto.

9. ¿Qué método se usa en Express para definir una ruta que maneja solicitudes GET?

En Express, se utiliza el método **app.get()** para definir una ruta que maneja solicitudes GET. Por ejemplo:

```
app.get('/', (req, res) => {  
    res.send('Hola Mundo');  
});
```

Este código define una ruta en la raíz del servidor (`/`) que maneja solicitudes GET y responde con el texto "Hola Mundo".

```
const express = require('express');  
const app = express();  
  
app.get('/', (req, res) => {  
    res.send('Hola Mundo');  
});  
  
app.listen(3000, () => {  
    console.log('Servidor corriendo  
en http://localhost:3000');  
});
```

10. ¿Cuál es la sinaxis correcta para crear una aplicación Express básica?

- **require('express'):** Importa el módulo Express.
- **const app = express();** Crea una instancia de la aplicación Express.
- **app.get('/', ...):** Define una ruta para manejar solicitudes GET en la raíz (`/`) del servidor.
- **app.listen(3000, ...):** Hace que el servidor Express escuche en el puerto 3000 y muestra un mensaje en la consola cuando se inicia.

11. Describa cómo funciona un servidor Node.js comparado con uno Apache.

- **Node.js:** Utiliza un modelo de ejecución de un solo hilo y no bloqueante basado en eventos. Maneja conexiones de manera asíncrona, lo que permite manejar muchas conexiones simultáneas eficientemente.
- **Apache:** Utiliza un modelo de subprocesos o procesos pesados. Asigna un hilo de ejecución por cada solicitud, lo que puede consumir más recursos y ser menos eficiente para aplicaciones con muchas conexiones concurrentes.

12. ¿Cuál es una ventaja de usar Node.js sobre Apache para aplicaciones en tiempo real?

Una ventaja clave de Node.js sobre Apache para aplicaciones en tiempo real es su capacidad para manejar conexiones simultáneas de manera eficiente gracias a su modelo de ejecución no bloqueante. Node.js puede escalar fácilmente para manejar miles de conexiones concurrentes en aplicaciones como chats, juegos en línea o sistemas de notificaciones en tiempo real.

13. ¿Qué módulo de Node.js se utiliza para crear un servidor web básico?

Para crear un servidor web básico en Node.js, se utiliza el módulo `http`. Este módulo proporciona las funcionalidades necesarias para crear un servidor HTTP y manejar solicitudes y respuestas.

14. ¿Cuál es el método correcto para crear un servidor HTTP en Node.js?

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

const server = http.createServer((req, res) => {
  // manejo de solicitudes y respuestas aquí
});

server.listen(3000);
```

- **require('http'):** Importa el módulo HTTP de Node.js.
- **http.createServer():** Crea un servidor HTTP que maneja las solicitudes entrantes y las respuestas correspondientes.
- **server.listen(3000):** Hace que el servidor escuche en el puerto 3000 para las solicitudes entrantes.

15. ¿Qué función se usa para hacer que el servidor escuche en un puerto específico?

Para hacer que el servidor escuche en un puerto específico, se utiliza el método **server.listen(3000)** (donde `3000` es el número de puerto deseado). Esto permite que el servidor esté disponible para recibir solicitudes entrantes en ese puerto.

16. ¿Cuál es el propósito del parámetro de callback en `http.createServer()`?

El parámetro de **callback** en `http.createServer()` es una función que se ejecuta cada vez que el servidor recibe una solicitud HTTP. Esta función maneja la solicitud (`req`) y genera la respuesta correspondiente (`res`) que se enviará de vuelta al cliente.

17. ¿Qué es nodemon?

Nodemon es una herramienta de desarrollo para Node.js que ayuda a los desarrolladores reiniciar automáticamente la aplicación Node.js cuando se detectan cambios en el código fuente. Esto elimina la necesidad de detener y reiniciar manualmente la aplicación después de cada modificación.

18. ¿Cómo se instala nodemon globalmente?

Nodemon se instala globalmente utilizando el comando **npm install -g nodemon**.

El flag `-g` indica que nodemon se instalará de manera global en el sistema, permitiendo su uso desde cualquier directorio.

19. ¿Cómo se usa nodemon para ejecutar una aplicación Node.js?

En lugar de ejecutar `node app.js` para iniciar una aplicación Node.js, se utiliza **nodemon app.js**. Nodemon monitorea los cambios en los archivos del proyecto y reinicia automáticamente la aplicación cada vez que se guarda un archivo, facilitando el desarrollo continuo y la depuración.

20. ¿Qué ventaja ofrece nodemon al desarrollar aplicaciones Node.js?

Nodemon mejora significativamente la productividad del desarrollo al eliminar la necesidad de reiniciar manualmente la aplicación después de cada cambio en el código fuente. Automáticamente detecta y reinicia la aplicación, ahorrando tiempo y esfuerzo durante el ciclo de desarrollo.

21. ¿Qué comando se usa para inicializar un proyecto Node.js con npm?

Para inicializar un proyecto Node.js con npm, se utiliza el comando **npm init**. Este comando guía al usuario a través de la creación de un archivo `package.json` interactivo, donde se definen detalles como el nombre del proyecto, la versión, la descripción, y más.

22. ¿Qué información se incluye típicamente en el archivo package.json?

- **name:** Nombre del proyecto.
- **version:** Número de versión del proyecto.
- **description:** Descripción breve del proyecto.
- **main:** Punto de entrada principal del proyecto.
- **scripts:** Conjunto de comandos de npm para tareas como ejecutar pruebas, iniciar la aplicación, etc.
- **dependencies:** Lista de dependencias del proyecto.
- **devDependencies:** Lista de dependencias de desarrollo.
- **author, license,** entre otros campos opcionales.

23. ¿Qué es el patrón MVC?

MVC (Modelo-Vista-Controlador) es un patrón de arquitectura de software que separa una aplicación en tres componentes: Modelo, Vista y Controlador, para mejorar la modularidad y mantenibilidad del código.

24. ¿Qué componentes forman el patrón MVC?

- **Modelo (Model):** Gestiona la lógica de negocio y los datos de la aplicación.
- **Vista (View):** Se encarga de la presentación de los datos al usuario.
- **Controlador (Controller):** Mediador entre el modelo y la vista, maneja las solicitudes del usuario y actualiza el modelo.

25. ¿Cuál es la función del componente "Model" en MVC?

El componente "Model" en MVC se encarga de gestionar la lógica de negocio y la manipulación de datos de la aplicación. Esto incluye implementar reglas de negocio, realizar operaciones CRUD en la base de datos, responder a consultas de datos y mantener el estado de la aplicación durante su ejecución. Es fundamental para asegurar que la aplicación funcione de manera eficiente y coherente en cuanto a la gestión y manipulación de datos.

26. ¿Cuál es la función del componente "View" en MVC?

En el patrón MVC, el componente "View" se encarga exclusivamente de mostrar los datos del modelo al usuario de manera visualmente atractiva y comprensible. Su función es gestionar la interfaz de usuario y asegurar que la información se presente de forma clara, sin involucrarse en la lógica de negocio o la funcionalidad de la aplicación.

27. ¿Qué método de Express se usa para servir archivos estáticos?

Correcto, en Express el método utilizado para servir archivos estáticos es `express.static()`. Este método se utiliza como middleware para especificar un directorio que contiene archivos estáticos (como imágenes, archivos CSS, JavaScript, etc.) que se enviarán directamente al cliente sin procesamiento adicional por parte del servidor.

```
app.use(express.static('public'));
```

28. ¿Cuál es la forma correcta de servir archivos estáticos desde el directorio "static" en Express?

```
app.use(express.static('static'));
```

app.use() se utiliza para montar el middleware **express.static()**.

express.static('static') especifica que el directorio llamado "static" contiene los archivos estáticos que se desean servir.

29. ¿Dónde se colocan típicamente los archivos estáticos en un proyecto Express?

En un directorio llamado 'public' o 'static' dentro del proyecto.

30. ¿Qué motor de plantillas es comúnmente usado con Express para renderizar HTML?

Correcto, el motor de plantillas comúnmente usado con Express para renderizar HTML es **ejs** (Embedded JavaScript). Este motor de plantillas permite incrustar código JavaScript dentro de archivos HTML, lo que facilita la generación dinámica de contenido HTML basado en datos proporcionados por la aplicación.

Para utilizar **ejs** en una aplicación Express, es necesario configurar Express para que reconozca este motor de plantillas. Esto se hace típicamente usando el método **app.set()** de la siguiente manera:

31. ¿Cómo se configura el motor de plantillas en una aplicación Express?

```
app.set('view engine', 'ejs');
```

32. ¿Qué método se usa para renderizar una vista en Express?

```
res.render('nombre_vista', { datos: datos });
```

33. ¿Cómo se pasan datos a una vista en Express?

Mediante un objeto en `res.render()`.

34. ¿Dónde se almacenan típicamente las vistas en una aplicación Express?

En un directorio llamado 'views'.

35. ¿Qué es HTTP?

Protocolo de comunicación para transferir datos en la web.

36. ¿Cuál es el propósito del método HTTP GET?

Solicitar datos de un recurso específico.

37. ¿Qué método HTTP se utiliza comúnmente para enviar datos al servidor para crear un nuevo recurso?

POST.

38. ¿Cuál es el método HTTP utilizado para eliminar un recurso en el servidor?

DELETE.

39. ¿Qué es un código de estado HTTP 404?

Recurso no encontrado.

40. ¿Cuál es el propósito del encabezado HTTP "Content-Type"?

Especificar el tipo de medio del cuerpo del mensaje HTTP.

41. ¿Qué método de Express se usa para manejar solicitudes PUT?

app.put().

42. ¿Qué método HTTP se usa para solicitar únicamente los encabezados de una respuesta, sin el cuerpo del mensaje?

HEAD.