**Clase 14:**  Introducción a Node.js y Express.js

Objetivos de la clase

* Comprender qué es Node.js y su función en el desarrollo de aplicaciones web.
* Familiarizarse con Express.js como un framework web para Node.js.
* Configurar un entorno de desarrollo Node.js y crear una aplicación básica con Express.js.

**Preguntas orientadoras para la lectura**

* *¿Cómo se define un endpoint en una aplicación Express.js y qué papel desempeña en el manejo de las solicitudes HTTP?*
* *¿Qué diferencias existen entre los métodos HTTP (GET, POST, PUT, DELETE) al definir endpoints en una API, y cómo influyen en las operaciones que se realizan en el servidor?*
* *¿Cómo se configura una ruta en Express.js para manejar múltiples métodos HTTP y cómo se organizan las funciones de manejo para responder a las solicitudes?*

**Introducción a Node.js**

*Definición*

Node.js es un entorno de ejecución de JavaScript basado en el motor V8 de Google Chrome, diseñado para construir aplicaciones del lado del servidor.

A diferencia de los entornos tradicionales que utilizan lenguajes como PHP o Ruby en el servidor, Node.js permite ejecutar JavaScript en el backend.

Esto facilita el desarrollo de aplicaciones web completas utilizando un solo lenguaje tanto en el cliente como en el servidor.

*Ventajas y casos de uso*

Node.js se destaca por su arquitectura orientada a eventos y su capacidad para manejar operaciones de entrada / salida (I/O) de manera asíncronica, lo que lo hace ideal para aplicaciones en tiempo real y con alta concurrencia, como chats en línea, plataformas de juegos y sistemas de notificaciones.

Su modelo de arquitectura “sin bloqueo” permite a Node.js manejar un gran número de conexiones simultáneas con eficiencia.

*Instalación de Node.js y npm*

Para comenzar a usar Node.js, primero se debe instalar el entorno en la máquina local. La instalación de Node.js también incluye **npm** (Node Package Manager), una herramienta esencial para gestionar las dependencias del proyecto y los paquetes de terceros.

La instalación se puede realizar desde el sitio web oficial de Node.js, donde se proporcionan paquetes para diferentes sistemas operativos.

**Introducción a Express.js**

*Definición*

Express.js es un framework minimalista y flexible para Node.js que simplifica el desarrollo de aplicaciones web y APIs.

Nos proporciona una serie de herramientas y funcionalidades que facilitan el enrutamiento de solicitudes, el manejo de middleware y la gestión de respuestas HTTP, lo que acelera el proceso de desarrollo en comparación con la implementación de estos elementos desde cero.

*Características*

* **Enrutamiento:** permite definir rutas para manejar las solicitudes HTTP en diferentes **endpoints** de la aplicación, facilitando la organización y el manejo de las rutas.
* **Middleware:** son funciones que se ejecutan durante el ciclo de vida de una solicitud HTTP. Permiten realizar tareas como el análisis de cuerpos de solicitud, la autenticación de usuarios y la gestión de errores.
* **Manejo de solicitudes y respuestas:** nos proporciona métodos simplificados para enviar respuestas y manejar solicitudes, permitiendo una integración fluida con distintos tipos de datos y formatos.

|  |
| --- |
| **Importante:**  En el contexto de aplicaciones web y APIs, un **endpoint** es una URL específica a la que se puede acceder para realizar una operación particular en el servidor.  Cada **endpoint** corresponde a una ruta en el servidor que está diseñada para manejar ciertos tipos de solicitudes HTTP.  Los **endpoints** definen cómo las solicitudes del cliente deben ser gestionadas y qué respuestas se deben enviar de vuelta. |

*Instalación de Express.js*

Para usar Express.js, primero se debe crear un proyecto Node.js y luego instalar Express.js como una dependencia mediante **npm**. La instalación se realiza con el comando **npm install express**, que descarga e instala el paquete en el proyecto.

**Configuración del entorno de desarrollo**

*Creación de un proyecto Node.js con express.js*

Para comenzar a trabajar con Node.js y Express.js, se debe iniciar un nuevo proyecto de Node.js. Esto implica crear un directorio para el proyecto, inicializarlo con **npm init** para generar un archivo **package.json**, e instalar Express.js.

La estructura básica del proyecto incluirá archivos como **app.js** o **index.js** para definir el servidor y las rutas.

*Estructura básica de un proyecto Express.js*

Una estructura común para un proyecto Express.js incluye:

* **app.js** o **server.js**: Archivo principal que configura el servidor y las rutas.
* **routes/**: Carpeta para definir las rutas de la aplicación.
* **views/**: Carpeta para las plantillas de vistas (si se utiliza un motor de plantillas).
* **public/**: Carpeta para archivos estáticos como CSS y JavaScript.

*Uso de npm para administrar dependencias y scripts de Inicio*

npm no solo gestiona las dependencias del proyecto, sino que también permite definir scripts para iniciar el servidor, ejecutar pruebas y realizar otras tareas automatizadas. Estos scripts se definen en el archivo **package.json** bajo la sección **scripts**.

**Creación de una aplicación básica con Express.js**

*Definición de rutas y controladores*

En una aplicación Express.js, las rutas definen los endpoints que manejan las solicitudes HTTP.

Los controladores son funciones que responden a estas solicitudes.

En un archivo de rutas, se pueden definir rutas para diferentes métodos HTTP y asociarlas con sus respectivos controladores.

*Creación de rutas para diferentes métodos HTTP*

* GET: Para recuperar datos de un recurso. Ejemplo: **app.get('/api/users', userController.getAllUsers);**
* POST: Para enviar datos al servidor y crear un nuevo recurso. Ejemplo: **app.post('/api/users', userController.createUser);**
* PUT: Para actualizar un recurso existente. Ejemplo: **app.put('/api/users/:id', userController.updateUser);**
* DELETE: Para eliminar un recurso. Ejemplo: **app.delete('/api/users/:id', userController.deleteUser);**

*Pruebas básicas de la aplicación*

Para verificar que la aplicación funciona correctamente, se pueden usar herramientas como **Postman o cURL**.

Estas herramientas permiten enviar solicitudes HTTP a los endpoints definidos y revisar las respuestas para asegurar que la aplicación maneja correctamente las solicitudes y respuestas.