# Teoria

**Node.js**

## **¿Qué es Node.js?**

Node.js es un entorno de ejecución de JavaScript del lado del servidor. Es decir, permite ejecutar código JavaScript fuera de un navegador web, lo que tradicionalmente estaba reservado para el front-end (la parte visible de una aplicación web). Fue creado por Ryan Dahl en 2009 y está construido sobre el motor V8 de Google Chrome.

**Características principales:**

* **Asincronía y Event-Driven:** Node.js utiliza un modelo de operaciones de entrada/salida (E/S) no bloqueante y orientado a eventos, lo que lo hace eficiente y adecuado para aplicaciones que manejan múltiples conexiones simultáneamente, como servidores web.
* **Un solo lenguaje para todo el stack:** Con Node.js, puedes usar JavaScript tanto en el front-end como en el back-end, lo que simplifica el desarrollo y la comunicación entre equipos.
* **NPM (Node Package Manager):** Node.js viene con NPM, un gestor de paquetes que facilita la instalación y gestión de bibliotecas y módulos de terceros.

**Usos comunes:**

* **Desarrollo de servidores web:** Crear aplicaciones web escalables y rápidas.
* **APIs en tiempo real:** Aplicaciones de chat, notificaciones en tiempo real, etc.
* **Automatización de tareas:** Scripts para automatizar procesos de desarrollo.

## **Express.js**

**¿Qué es Express.js?**

Express.js es un framework minimalista y flexible para Node.js que simplifica el desarrollo de aplicaciones web y APIs. Fue creado por TJ Holowaychuk en 2010 y es uno de los frameworks más populares en el ecosistema de Node.js.

**Características principales:**

* **Simplicidad y Minimalismo:** Proporciona una estructura básica sin imponer demasiadas restricciones, permitiendo a los desarrolladores estructurar sus aplicaciones según sus necesidades.
* **Middleware:** Express utiliza middleware para manejar solicitudes HTTP, permitiendo agregar funcionalidades como manejo de sesiones, autenticación, manejo de errores, entre otros.
* **Rutas:** Facilita la definición de rutas para diferentes endpoints de una aplicación, permitiendo manejar diferentes tipos de solicitudes (GET, POST, PUT, DELETE, etc.).
* **Compatibilidad:** Es altamente compatible con otras herramientas y bibliotecas del ecosistema de Node.js.

**Usos comunes:**

* **APIs RESTful:** Crear APIs que permiten la comunicación entre el front-end y el back-end.
* **Aplicaciones web de servidor completo:** Construir aplicaciones web que manejan tanto el front-end como el back-end.
* **Microservicios:** Desarrollar servicios pequeños y escalables que forman parte de una arquitectura más grande.

## **npm (Node Package Manager)**

**¿Qué es npm?**

npm es el gestor de paquetes predeterminado para Node.js. Permite a los desarrolladores instalar, compartir y gestionar dependencias de sus proyectos de manera sencilla. Además, npm facilita la creación y distribución de paquetes reutilizables.

**Características principales:**

* **Repositorio de Paquetes:** npm alberga millones de paquetes (bibliotecas y herramientas) que los desarrolladores pueden usar en sus proyectos.
* **Gestión de Dependencias:** Facilita la instalación y actualización de las dependencias necesarias para un proyecto, manejando automáticamente las versiones compatibles.
* **Scripts:** Permite definir scripts en el archivo package.json para automatizar tareas comunes como pruebas, compilación, despliegue, etc.
* **Versionado Semántico:** Ayuda a mantener el control de las versiones de los paquetes, asegurando compatibilidad y facilitando actualizaciones.

**Comandos básicos:**

* **Inicializar un proyecto:** npm init crea un archivo package.json donde se almacenan las dependencias y configuraciones del proyecto.
* **Instalar un paquete:** npm install nombre-paquete descarga e instala el paquete en el proyecto.
* **Actualizar paquetes:** npm update actualiza las dependencias a sus versiones más recientes compatibles.
* **Eliminar un paquete:** npm uninstall nombre-paquete elimina el paquete del proyecto.

**Ejemplo de uso:**

Supongamos que quieres usar Express.js en tu proyecto. Primero, inicializas tu proyecto y luego instalas, Express:

* npm init -y
* npm install express

## **¿Qué es una API REST?**

**API REST** es un estilo arquitectónico para diseñar servicios web que permiten la comunicación entre diferentes sistemas a través de la web. Una **API** (Interfaz de Programación de Aplicaciones) RESTful sigue los principios y restricciones definidos por REST para facilitar la interacción entre clientes y servidores.

### Principios Fundamentales de REST

1. **Cliente-Servidor:**
   * **Separación de responsabilidades:** El cliente (por ejemplo, una aplicación web o móvil) se encarga de la interfaz de usuario, mientras que el servidor maneja la lógica de negocio y el almacenamiento de datos.
2. **Sin Estado (Stateless):**
   * **Independencia de las solicitudes:** Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud.
   * **Escalabilidad:** Al no mantener el estado entre solicitudes, es más fácil escalar la aplicación.
3. **Cacheable:**
   * **Mejora del rendimiento:** Las respuestas deben indicar si son cacheables o no, permitiendo que los clientes almacenen en caché las respuestas y reduzcan la carga del servidor.
   * **Eficiencia:** Minimiza la necesidad de repetir solicitudes idénticas.
4. **Interfaz Uniforme:**
   * **Simplificación de la arquitectura:** Se define una interfaz estándar para la comunicación, lo que facilita la interacción entre diferentes componentes.
   * **Elementos clave:**
     + **Identificación de recursos:** Cada recurso (como usuarios, productos, etc.) se identifica mediante una URL única.
     + **Manipulación de recursos a través de representaciones:** Los clientes interactúan con los recursos mediante representaciones, generalmente en formatos JSON o XML.
     + **Mensajes auto-descriptivos:** Cada mensaje contiene toda la información necesaria para procesarlo.
     + **Hipermedia como el motor del estado de la aplicación (HATEOAS):** Las respuestas pueden incluir enlaces a otras acciones o recursos relacionados.
5. **Sistema en Capas:**
   * **Arquitectura escalable:** La arquitectura puede estar compuesta por múltiples capas (por ejemplo, servidores proxy, balanceadores de carga) sin que el cliente lo note.
   * **Seguridad y manejo de tráfico:** Facilita la implementación de mecanismos de seguridad y gestión del tráfico.
6. **Código Bajo Demanda (Opcional):**
   * **Extensibilidad:** El servidor puede proporcionar código ejecutable (como scripts JavaScript) que el cliente puede usar para extender su funcionalidad.

### Componentes Clave de una API REST

1. **Recursos:**
   * **Entidades identificables:** Cada recurso representa una entidad específica y está identificado por una URL única.
   * **Ejemplos:** /usuarios, /productos/123, /ordenes/456.
2. **Métodos HTTP:**
   * **Operaciones sobre los recursos:**
     + **GET:** Recuperar información de un recurso.
     + **POST:** Crear un nuevo recurso.
     + **PUT:** Actualizar completamente un recurso existente.
     + **PATCH:** Actualizar parcialmente un recurso existente.
     + **DELETE:** Eliminar un recurso.
3. **Representaciones:**
   * **Formatos de datos:** Los recursos pueden representarse en diferentes formatos, siendo JSON y XML los más comunes.
   * **Transmisión de datos:** El cliente y el servidor intercambian representaciones de recursos a través de las solicitudes y respuestas HTTP.
4. **Estado de las Respuestas:**
   * **Códigos de estado HTTP:** Indican el resultado de la solicitud.
     + **200 OK:** Solicitud exitosa.
     + **201 Created:** Recurso creado exitosamente.
     + **400 Bad Request:** Solicitud incorrecta.
     + **401 Unauthorized:** Autenticación requerida.
     + **404 Not Found:** Recurso no encontrado.
     + **500 Internal Server Error:** Error en el servidor.

### Ventajas de Usar una API REST

* **Simplicidad:** Utiliza protocolos estándar como HTTP, lo que facilita su implementación y uso.
* **Escalabilidad:** La separación de cliente y servidor y la naturaleza sin estado facilitan la escalabilidad horizontal.
* **Flexibilidad:** Permite que diferentes clientes (web, móviles, IoT) interactúen con el mismo servidor.
* **Interoperabilidad:** Al seguir estándares, es más fácil integrar sistemas diversos.

# Practica

Pasos a seguir para crear un proyecto en express js

1. Tener instalado node js
2. Crear el directorio que contendrá al proyecto
3. Abrir git bash y escribir:
   1. npm install -g express-generator: este codigo va a hacer que podamos usar el generador de express globalmente
   2. express –-no-view [nombre de proyecto]: Crea el proyecto
   3. Creamos carpeta src y ahí guardamos la carpeta routes y app.js