

Agencia de Habilidades  
para el Futuro

<Talento  
Tech/>

# Node.JS

---

Clase 8: Servidores Web y patrones de Arquitectura

# ¡Les damos la bienvenida!

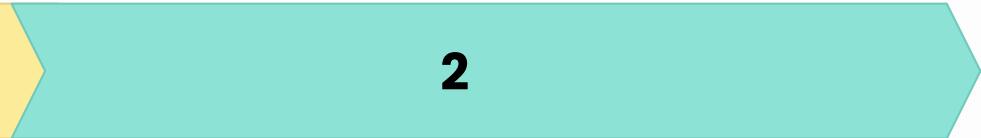
 Vamos a comenzar a grabar la clase.

# Índice

---



1



2

## Servidores Web y patrones de Arquitectura

- Servidores
  - ¿Qué es un servidor?
  - ¿Cómo funciona un servidor?
  - Tipos de Servidores
  - Servidores Web
  - Alojamiento y Hostings
- Patrones de Arquitectura
- MVC vs API Rest

## Creando un Servidor Web

- Servidor Web Node Nativo
- Express JS
- Middlewares

# Objetivos de la Clase

---

- 1** Definir y diferenciar entre un servidor de hardware y un servidor de software. Entender cómo interactúan el hardware y el software para ofrecer servicios en la red. Identificar y analizar los diferentes tipos de servidores
  
- 2** Explicar cómo los servidores web procesan solicitudes HTTP. Destacar la importancia del alojamiento y los servicios de hosting para el acceso global a sitios y aplicaciones. Introducir patrones de arquitectura en sistemas modernos
  
- 3** Abordar el concepto de MVC (Modelo-Vista-Controlador). Explicar API Rest y sus características. Contrastar los usos de MVC y API Rest en el diseño y desarrollo de sistemas modernos.

# **Servidores**

---

# ¿Qué es un servidor?

---

El término servidor tiene dos significados en el ámbito informático.

El primero hace referencia al ordenador que pone recursos a disposición a través de una red, y el segundo se refiere al programa que funciona en dicho ordenador

## Servidor de Hardware

Máquina física integrada en una red informática que ejecuta software de servidor.

## Servidor de Software

Programa que ofrece servicios especiales a otros programas llamados clientes.



# Funcionamiento de un servidor

---

## 1 Espera de solicitudes

El servidor está en espera permanente de solicitudes de clientes.

## 2 Recepción de solicitud

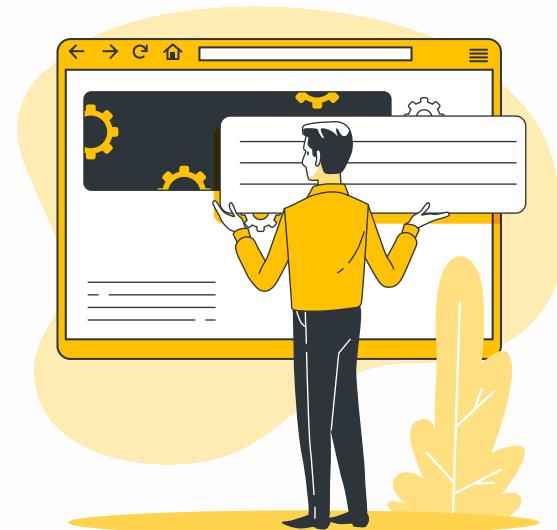
Un cliente envía una solicitud al servidor.

## 3 Procesamiento

El servidor procesa la solicitud y prepara una respuesta.

## 4 Envío de respuesta

El servidor envía la respuesta al cliente.



# **Tipos de Servidores**

---

# Tipos de Servidores

---



## Web

Almacena, organiza y entrega páginas web a los navegadores.



## Archivos

Almacena datos para acceso múltiple a través de una red.



## Correo Electronico

Maneja el envío, recepción y reenvío de correos electrónicos.



## Bases de datos

Apache, Nginx y Microsoft IIS.



## Juegos

Gestiona datos en juegos multijugador online.



## Proxy

Actúa como intermediario entre cliente y otros servidores.



## DNS

Traduce nombres de dominio en direcciones IP.

# Servidores Web Populares: Apache

---



## Ventajas

Software de código abierto, gratuito y compatible con múltiples plataformas.

## Desventajas

Rendimiento puede disminuir con miles de solicitudes simultáneas.

## Uso

Ampliamente utilizado, aunque ha perdido terreno frente a alternativas.



# Servidores Web Populares: Nginx

---

## 1 Características

Ligero, rápido y muy seguro. Enfocado en alto rendimiento.

## 2 Ventajas

Excelente manejo de tráfico elevado y conexiones simultáneas.

## 3 Compatibilidad

Altamente compatible con tecnologías y lenguajes modernos.

# Servidores Web Populares: LiteSpeed



## Rendimiento

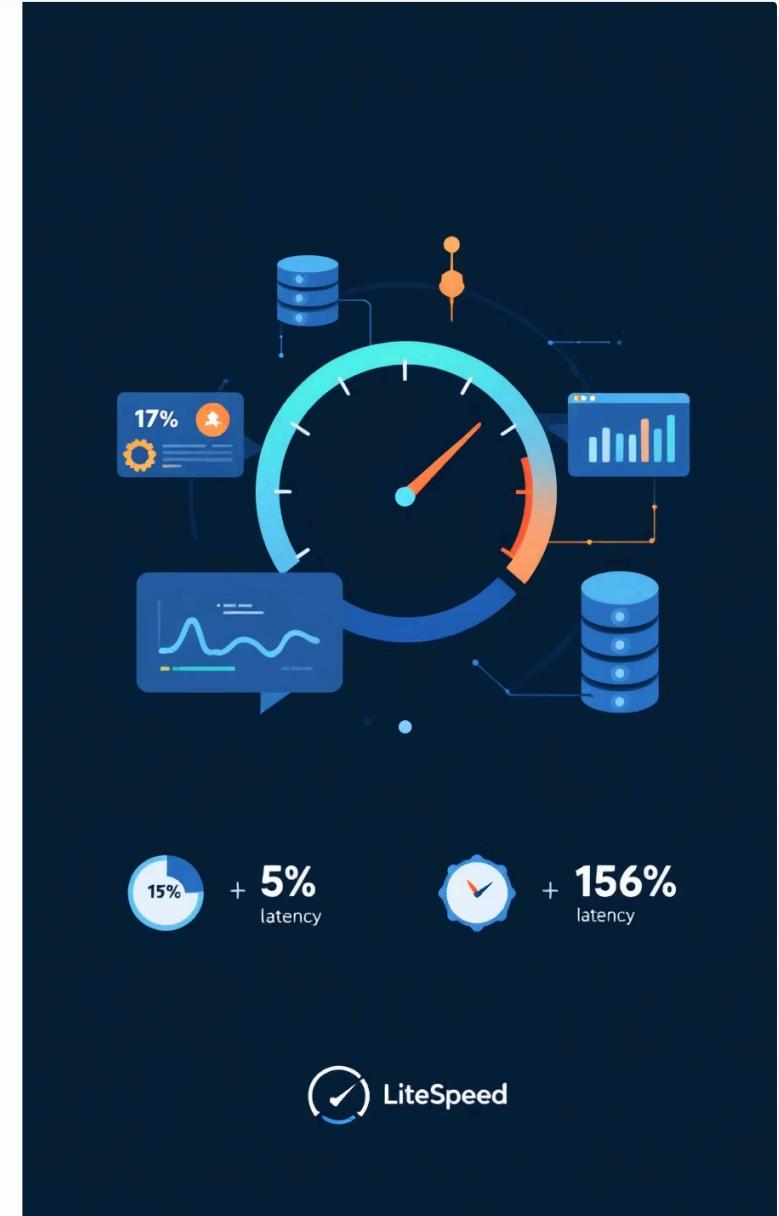
Diseñado para ofrecer alto rendimiento, especialmente con PHP.

## Versiones

Ofrece versión gratuita de código abierto y versión comercial.

## Eficiencia

Equilibra velocidad y estabilidad sin consumo excesivo de recursos.





# Servidores Web Populares: Microsoft IIS

## Integración

Nativa con sistema operativo Windows y herramientas Microsoft.

## Popularidad

Preferido en entornos empresariales que usan tecnologías Microsoft.

## Ventaja

Facilidad de uso en entornos corporativos y soporte especializado.

# **Alojamiento y Hostings**

---

# Alojamiento y Hostings

---



El hosting, web hosting o el alojamiento web es un servicio de almacenamiento que hace que se pueda acceder a tu sitio web en Internet

Existen varios tipos de servicio de almacenamiento:

## 1 Servidor Dedicado

Máximo control y recursos exclusivos.

## 2 Servidor Virtual Privado (VPS)

Balance entre control y costo.

## 3 Hosting Compartido

Económico, recursos compartidos.

# **Patrones de Arquitectura**

---

# Patrones de Arquitectura

---

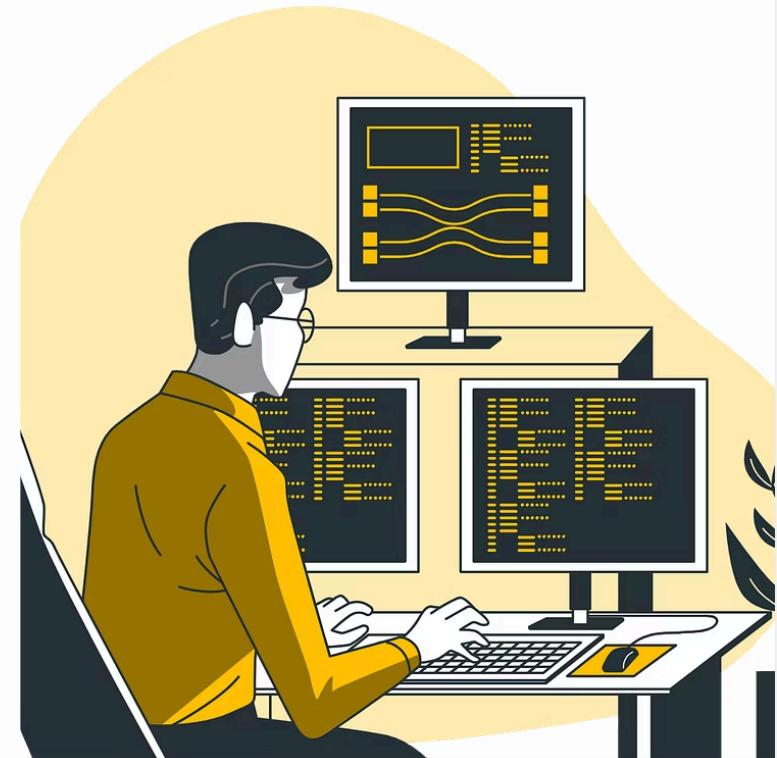
Guías probadas para resolver problemas comunes de diseño en software.

## Propósito

Ayudan a diseñar sistemas escalables, mantenibles y robustos.

## Flexibilidad

No son recetas específicas, sino enfoques generales adaptables.



# Tipos de Patrones de Arquitectura

---

## 1 Monolito

Todas las funcionalidades en una única unidad.

## 2 Capas

División en capas con responsabilidades específicas.

## 3 Basados en eventos

Centrados en intercambio de mensajes entre componentes.

## 4 Microservicios

Aplicación dividida en servicios pequeños e independientes.

# MVC vs API Rest

---

# MVC vs API Rest

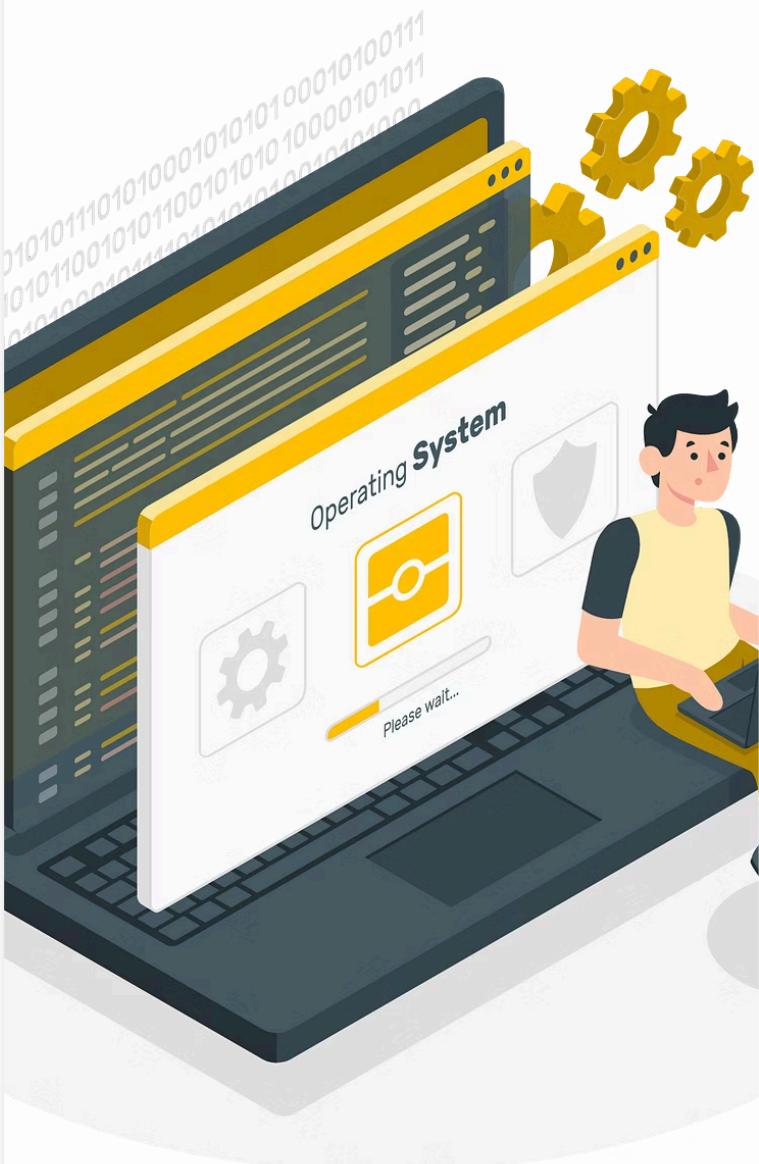
---

## MVC (Modelo-Vista-Controlador)

Genera HTML y CSS en el servidor. Divide la aplicación en Modelo, Vista y Controlador.

## API Rest (Representational State Transfer)

Proporciona datos en formato JSON. Separa completamente frontend y backend.



# Patrón MVC

## Modelo

Gestiona datos y lógica de negocio.

## Vista

Representa la interfaz de usuario.

## Controlador

Intermediario entre Modelo y Vista.

# API Rest

## 1 Recursos

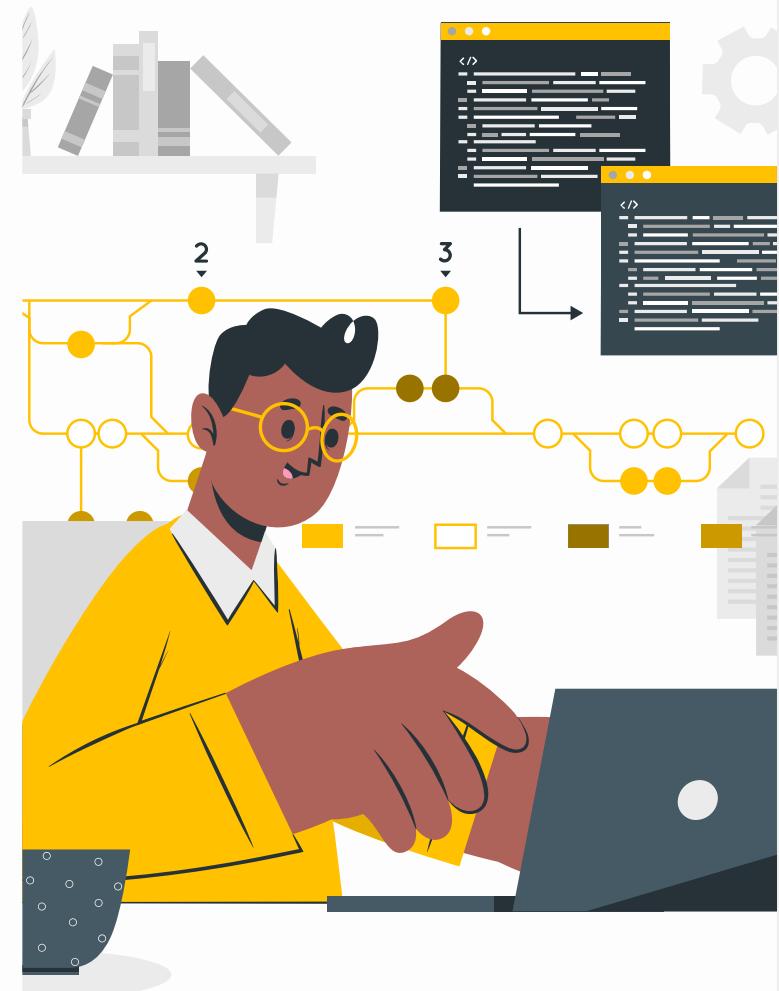
Expone recursos a través de URLs específicas.

## 2 Métodos HTTP

Utiliza GET, POST, PUT, DELETE para interactuar con recursos.

## 3 Formato de datos

Comúnmente usa JSON para transmitir información.





# Conclusión

---

## Servidores

Fundamentales para el funcionamiento de aplicaciones y servicios web.

## Alojamiento

Diversas opciones según necesidades y recursos disponibles.

## Patrones

Guían el diseño de sistemas escalables y mantenibles.

# Materiales y Recursos Adicionales:

---

[Introducción a las API REST](#) para profundizar en los principios arquitectónicos de REST.

[Documentación sobre patrones de arquitectura](#) en el sitio web de Martin Fowler.

Entornos para probar API REST, como [Postman](#) y [Insomnia](#).



# Próximos Pasos:

---

## 1 Creando un Servidor Web

Daremos nuestros primeros pasos en la creación de servidores web con Node JS.

## 2 Modelando una API Rest

Nos iniciaremos en la metodología para crear nuestra primera API Rest.

## 3 Request y Response:

Llegó el momento de aplicar de forma práctica todo nuestro conocimiento sobre la comunicación web.



# Pre-Entrega de Proyecto

---

# Pre-Entrega de Proyecto

Obligatorio



"Hemos llegado al momento clave. Es hora de demostrar si estás preparado para dar el siguiente paso y unirte a nuestro equipo en TechLab".

"Tu desafío es integrar todo lo aprendido en un único programa. Queremos ver cómo manejas estructuras, APIs y lógica dinámica. El objetivo es construir una herramienta funcional para manejar productos de una tienda en línea desde la terminal. ¿Estás listo para el reto?"



## Requerimientos del Proyecto

### Requerimiento #1: Configuración Inicial

- Crea un directorio donde alojarás tu proyecto e incluye un archivo index.js como punto de entrada.

- Inicia Node.js y configura npm usando el comando `npm init -y`.
- Agrega la propiedad "type": "module" en el archivo `package.json` para habilitar ESModules.

Configura un script llamado `start` para ejecutar el programa con el comando `npm run start`.



Sabrina señala: "Este será el corazón de tu proyecto. Queremos un entorno limpio y profesional, como si estuvieras trabajando en un proyecto real".

## Requerimiento #2: Lógica de Gestión de Productos

Con la base del proyecto lista, ahora necesitamos implementar las funcionalidades principales usando la API `FakeStore`. El sistema debe ser capaz de interpretar comandos ingresados en la terminal y ejecutar las siguientes acciones:

### **Consultar Todos los Productos:**

Si ejecutas `npm run start GET products`, el programa debe realizar una petición asíncrona a la API y devolver la lista completa de productos en la consola.

Ejemplo: `npm run start GET products`

### **Consultar un Producto Específico:**

Si ejecutas `npm run start GET products/<productId>`, el programa debe obtener y mostrar el producto correspondiente al productId indicado.

Ejemplo: `npm run start GET products/15`

### **Crear un Producto Nuevo:**

Si ejecutas `npm run start POST products <title> <price> <category>`, el programa debe enviar una petición POST a la API para agregar un nuevo producto con los datos proporcionados (`title`, `price`, `category`) y devolver el resultado en la consola.

### **Eliminar un Producto:**

Si ejecutas `npm run start DELETE products/<productId>`, el programa debe enviar una petición DELETE para eliminar el producto correspondiente al `productId` y devolver la respuesta en la consola.

Ejemplo: `npm run start POST products T-Shirt-Rex`  
300 remeras

Ejemplo: `npm run start DELETE products/7`

### Tips de Desarrollo

- Usa `process.argv` para capturar y procesar los comandos ingresados.
- Implementa `fetch` para interactuar con la API de FakeStore (consulta su [documentación](#) para más detalles).
- Aprovecha el uso de destructuring y spread para manipular los datos.
- Utiliza métodos de arrays y strings para separar cadenas de texto y conjuntos de información y aprovechar solo lo que necesites.

## Conclusión

Matías finaliza: "Este desafío no solo mide tus habilidades técnicas, sino también tu capacidad para organizarte, resolver problemas y crear soluciones escalables. Si logras superar este reto, estaremos más que seguros de que estás listo para unirte a TechLab"

¿Aceptas el desafío? 





# ¡Nuevo cuestionario en Campus!

- ⓘ No olvides que los cuestionarios son de carácter obligatorio para poder avanzar con la cursada.