«Talento Tech»

Back-End

Node JS

Clase 08

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 8- Servidores Web y patrones de Arquitectura

Temario:

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

Objetivos de la Clase

El objetivo de esta clase es que los estudiantes comprendan qué es un servidor y cómo funciona, distinguiendo entre un servidor de hardware y uno de software, y entendiendo la interacción entre ambos para ofrecer servicios en la red. También se busca que los alumnos identifiquen y analicen los diferentes tipos de servidores, como los servidores web, de bases de datos y de archivos, explorando sus aplicaciones prácticas en el ecosistema tecnológico. Además, se profundizará en el concepto de servidores web, explicando cómo procesan solicitudes HTTP para entregar contenido a los clientes, destacando la importancia del alojamiento y los servicios de hosting para garantizar el acceso global a sitios y aplicaciones. Finalmente, se abordarán patrones de arquitectura como MVC y API Rest, contrastando sus características y usos en el diseño y desarrollo de sistemas modernos.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

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.

En consecuencia aparecen dos definiciones de servidor:

Servidor de Hardware.

Un servidor basado en hardware es una máquina física integrada en una red informática en la que, además del sistema operativo, funcionan uno o varios servidores basados en software. Una denominación alternativa para un servidor basado en hardware es "host" (término inglés para "anfitrión"). En principio, todo ordenador puede usarse como "host" con el correspondiente software para servidores.

Servidor de Software.

Un servidor basado en software es un programa que ofrece un servicio especial que otros programas denominados clientes (clients) pueden usar a nivel local o a través de una red. El tipo de servicio depende del tipo de software del servidor. La base de la comunicación es el modelo cliente-servidor y, en lo que concierne al intercambio de datos, entran en acción los protocolos de transmisión específicos del servicio.

¿Cómo funciona un servidor?

La puesta a disposición de los servicios del servidor a través de una red informática se basa en el modelo cliente-servidor, concepto que hace posible distribuir las tareas entre los diferentes ordenadores y hacerlas accesibles para más de un usuario final de manera independiente. Cada servicio disponible a través de una red será ofrecido por un servidor (software) que está permanentemente en espera. Este es el único modo de asegurar que los clientes como el navegador web o los clientes de correo electrónico siempre tengan la posibilidad de acceder al servidor activamente y de usar el servicio en función de sus necesidades.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Tipos de Servidores

Servidor Web

Un servidor web almacena, organiza y entrega páginas web a los navegadores de los usuarios mediante los protocolos HTTP o HTTPS. Su principal función es enviar documentos HTML junto con elementos como imágenes, hojas de estilo o scripts. Ejemplos populares de servidores web incluyen Apache, Nginx y Microsoft IIS.

Servidor de archivos

El servidor de archivos almacena datos para que múltiples usuarios puedan acceder a ellos a través de una red. Este tipo de servidor facilita el trabajo en equipo, evita conflictos entre versiones de archivos y permite la creación de copias de seguridad centralizadas. Para la transferencia de archivos se usan protocolos como FTP, SFTP, SMB y NFS.

Servidor de correo electrónico

Un servidor de correo electrónico gestiona el envío, recepción y reenvío de correos. Funciona principalmente con el protocolo SMTP y permite a los usuarios acceder a los mensajes mediante IMAP o POP a través de un cliente de correo electrónico.

Servidor de base de datos

Este tipo de servidor permite a otros programas acceder a sistemas de bases de datos a través de una red. Es esencial para almacenar y recuperar datos, especialmente en aplicaciones web. Ejemplos comunes incluyen MySQL, PostgreSQL, Oracle y Microsoft SQL Server.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Servidor de juegos

Un servidor de juegos está diseñado para gestionar datos en juegos multijugador online, permitiendo la interacción en tiempo real entre los jugadores. Puede estar alojado en centros de datos especializados o en redes domésticas.

Servidor proxy

El servidor proxy actúa como intermediario entre un cliente y otros servidores. Filtra la comunicación, controla el uso del ancho de banda, almacena datos en caché y mejora la privacidad ocultando la dirección IP del cliente.

Servidor DNS

El servidor DNS traduce nombres de dominio (como www.example.com) en direcciones IP, facilitando la navegación en internet. Es fundamental para el funcionamiento de la web.

*Nota: Un solo servidor de **hardware** puede alojar varios tipos de servidores de **software**, pero suelen estar distribuidos en equipos separados para optimizar el rendimiento y evitar interferencias entre servicios.

Servidores Web

Apache

Es uno de los servidores web más populares y utilizados en el mundo, aunque recientemente ha perdido algo de terreno frente a alternativas como Nginx y Microsoft IIS.

Su principal fortaleza es que es un software de código abierto, gratuito y compatible con múltiples plataformas, lo que lo convierte en una opción muy accesible. Sin embargo, una de sus principales desventajas es su rendimiento, ya que puede disminuir significativamente cuando se enfrenta a miles de solicitudes simultáneas, tanto en el procesamiento de contenido dinámico como en la entrega de archivos estáticos.

Nginx

Nginx es conocido por ser una de las mejores alternativas a Apache gracias a su enfoque en el alto rendimiento y la eficiencia.

Este servidor, también de código abierto y gratuito, es ligero, rápido y muy seguro. Se destaca especialmente en el manejo de tráfico elevado y en la capacidad de gestionar un gran número de conexiones simultáneas. Además, es altamente compatible con tecnologías y lenguajes de programación modernos. Como punto en contra, Nginx no soporta los archivos `.htaccess` que son comunes en Apache, aunque tiene su propio sistema de reescrituras.

LiteSpeed

LiteSpeed es un servidor web diseñado para ofrecer alto rendimiento, con una versión gratuita de código abierto y otra comercial con diferentes opciones de licencia. Es especialmente eficiente en el manejo de conexiones simultáneas, incluso cuando se trata de aplicaciones que consumen muchos recursos, como las que utilizan PHP. A nivel de rendimiento con archivos estáticos, está a la par con Nginx. Esto lo hace una excelente opción para sitios que necesitan velocidad y estabilidad sin un consumo excesivo de recursos.

Microsoft IIS

Internet Information Services (IIS) es el servidor web desarrollado por Microsoft y se ha destacado por su integración nativa con el sistema operativo Windows y con herramientas como Visual Studio. Esto lo convierte en la opción preferida para empresas que utilizan la plataforma Azure, el servicio de Cloud Hosting de Microsoft. IIS es especialmente popular en el mundo empresarial gracias a su facilidad de uso en entornos corporativos y su soporte especializado.

Otros servidores web populares.

Existen otros servidores web menos conocidos pero relevantes en ciertos contextos, como Lighttpd, Caddy, Cherokee, **Node JS**, Sun Java System Web Server o Google Web Server (GWS). Cada uno está diseñado para escenarios específicos y puede ser más adecuado dependiendo de las necesidades del proyecto.

Alojamiento y Hostings.

Mientras que a las grandes empresas les resulta más rentable adquirir hardware para montar sus propios servidores, los autónomos y los particulares que quieren desarrollar proyectos en un servidor propio recurren normalmente al alquiler.

Los proveedores especializados ofrecen diferentes modelos de servidores de alquiler en los que los usuarios no tienen que preocuparse por el funcionamiento de la máquina física. La gama de productos abarca desde servidores dedicados cuyos componentes de hardware se ponen a disposición de los usuarios de manera exclusiva, hasta servicios de hosting compartido para alojar a varios clientes virtuales en una base de hardware común.

Tipos de Alojamientos o Hostings Web

1. Hosting Compartido

Es como vivir en un dormitorio compartido: económico, pero con recursos y espacios comunes.

El hosting compartido es la opción más básica y económica. En este modelo, múltiples sitios web comparten los recursos de un único servidor físico.

Características:

- Recursos compartidos entre varios usuarios (CPU, RAM, almacenamiento).
- Limitado en personalización y rendimiento.
- Administración sencilla, ideal para principiantes.

Ventajas:

- Muy económico y fácil de configurar.
- Requiere pocos conocimientos técnicos, ya que el proveedor se encarga de la configuración.

Desventajas:

- Problemas de rendimiento si otros sitios en el servidor tienen mucho tráfico.
- Personalización y control limitados.
- Menor seguridad debido al entorno compartido.

Ideal para: Pequeñas empresas, blogs y sitios con bajo tráfico.

2. Hosting VPS (Servidor Virtual Privado)

Es como tener un departamento en un edificio: tenés tu espacio privado, pero compartís la infraestructura básica con otros residentes.

Un VPS utiliza tecnología de virtualización para dividir un servidor físico en múltiples servidores virtuales. Cada partición funciona como un servidor independiente con recursos dedicados.

Características:

- Recursos asignados exclusivamente a tu sitio web.
- Acceso completo al sistema operativo (control raíz).
- Alto grado de personalización y escalabilidad.

Ventajas:

- Mayor rendimiento y fiabilidad que el hosting compartido.
- Personalización completa del entorno de servidor.
- Escalabilidad para manejar el crecimiento del sitio.

Desventajas:

- Más caro que el hosting compartido.
- Requiere conocimientos técnicos para la administración.

Ideal para: Negocios en crecimiento, sitios de comercio electrónico y aplicaciones web con tráfico moderado a alto.

3. Hosting Dedicado

Es como alquilar una casa: control total, pero con mayores responsabilidades y costos.

Con este servicio, tienes acceso exclusivo a un servidor físico completo. Es la opción más potente y personalizable.

Características:

- Control total sobre hardware y software.
- Sin compartir recursos con otros usuarios.
- Requiere habilidades técnicas avanzadas para configurar y mantener.

Ventajas:

- Máximo rendimiento y velocidad.
- Altísima seguridad.
- Personalización sin límites.

Desventajas:

- Es la opción más costosa.
- Gestión compleja si no se tiene experiencia.

Ideal para: Grandes empresas, sitios con alto tráfico y proyectos críticos que demandan recursos constantes.

4. Cloud Hosting (AWS, Google Cloud, Microsoft Azure)

Es como alquilar varias casas interconectadas: siempre hay una disponible para tus necesidades, con la máxima flexibilidad.

El hosting en la nube utiliza un clúster de servidores interconectados para alojar sitios web. Si un servidor falla o está saturado, otro toma el relevo automáticamente.

Características:

- Alta disponibilidad y escalabilidad.
- Los recursos se distribuyen entre múltiples servidores.
- Pagas por los recursos que consumes (modelo "pay-as-you-go").

Ventajas:

- Tiempo de inactividad mínimo.
- Ideal para manejar picos de tráfico.
- Escalable en tiempo real según la demanda.

Desventajas:

- Costos variables según el uso.
- Puede ser complicado de administrar sin experiencia en entornos en la nube.

Ideal para: Proyectos que requieren alta disponibilidad, como aplicaciones SaaS (software as a service) y sitios con tráfico variable como tiendas en línea en épocas festivas o de promociones.

5. Hosting Especializado (WordPress, Netlify, Vercel)

Este tipo de hosting está diseñado para necesidades específicas, como WordPress o sitios estáticos.

Características:

- Configuraciones optimizadas para un tipo de sitio web o aplicación.
- En el caso de WordPress, incluye características como instalación en un clic y plugins preinstalados.
- Servicios como Netlify y Vercel están enfocados en sitios estáticos rápidos y fáciles de implementar.

Ventajas:

- Simplifica la gestión para casos específicos.
- Optimizado para un mejor rendimiento y tiempo de carga.
- Integraciones útiles para desarrolladores.

Desventajas:

- Menos flexible para otros usos fuera de su propósito.
- Opciones limitadas si necesitas personalización avanzada.

Ideal para: Blogs en WordPress, sitios estáticos, y proyectos que no necesitan backend complejo.

Patrones de Arquitectura.

Un **patrón de arquitectura** es una guía probada y estructurada que resuelve problemas comunes de diseño en el desarrollo de software. Estos patrones no son recetas específicas de implementación, sino más bien enfoques generales que ayudan a diseñar sistemas escalables, mantenibles y robustos.

SOFTWARE ARCHITECTURE

Al utilizar patrones de arquitectura, los desarrolladores pueden evitar errores comunes y garantizar una base sólida para sus proyectos. Existen diferentes tipos de patrones, cada uno diseñado para abordar necesidades específicas, y su elección depende de las características y objetivos del sistema a desarrollar.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Tipos de Patrones

Monolito

La arquitectura monolítica sigue un enfoque tradicional, donde las funcionalidades de una aplicación se desarrollan y despliegan como una única unidad. Esto es, que **todos los componentes de la aplicación, como la interfaz de usuario, la lógica de negocio o el acceso a los datos, están integrados en un solo programa o código base**.

Capas

Una alternativa de la arquitectura monolítica donde se dividen el software en capas, cada una con una responsabilidad específica. Ejemplos de patrones de capas son: **MVC** (Modelo-Vista-Controlador) y **MVP** (Modelo-Vista-Presentador).

Basados en eventos.

Se centran en el intercambio de mensajes o eventos entre componentes. Ejemplos de patrones de eventos son: **Publicar-Suscribir**, **Observer** y **Reactor**.

Basados en servicios:

Se enfocan en la creación de servicios reutilizables. Por ejemplo: Arquitectura **SOA** (Orientada a Servicios) y **REST** (Representational State Transfer).

Basados en microservicios.

La arquitectura de microservicios es un estilo arquitectónico en el que **una aplicación, por lo general compleja, se divide en un conjunto de servicios pequeños**. Estos son independientes y autónomos. Por lo que, cada microservicio es responsable de una funcionalidad específica y puede ser desarrollado, desplegado y escalado de manera independiente al resto.

Por ejemplo una aplicación donde contemos con un servidor que guarde la lógica de autenticación, otro servidor que maneje el acceso a los productos de una tienda y otro servidor donde se guarde el historial de pedidos, ventas y carritos, todos estos, consumidos directamente por un cliente frontend donde se encuentre la parte visual de la aplicación.

Arquitectura hexagonal

La arquitectura hexagonal **aísla las entradas y salidas de aplicación de la lógica interna de la aplicación**. Gracias a este aislamiento, se generan partes independientes que no dependen de los cambios externos. Por tanto, estas podrían modificarse de forma individual, sin afectar al resto.

MVC vs API Rest

Para entender mejor los patrones de arquitectura, comparemos dos (2) de los más distribuidos y utilizados en el mercado actual. Ellos son el patrón MVC y el patrón Rest.

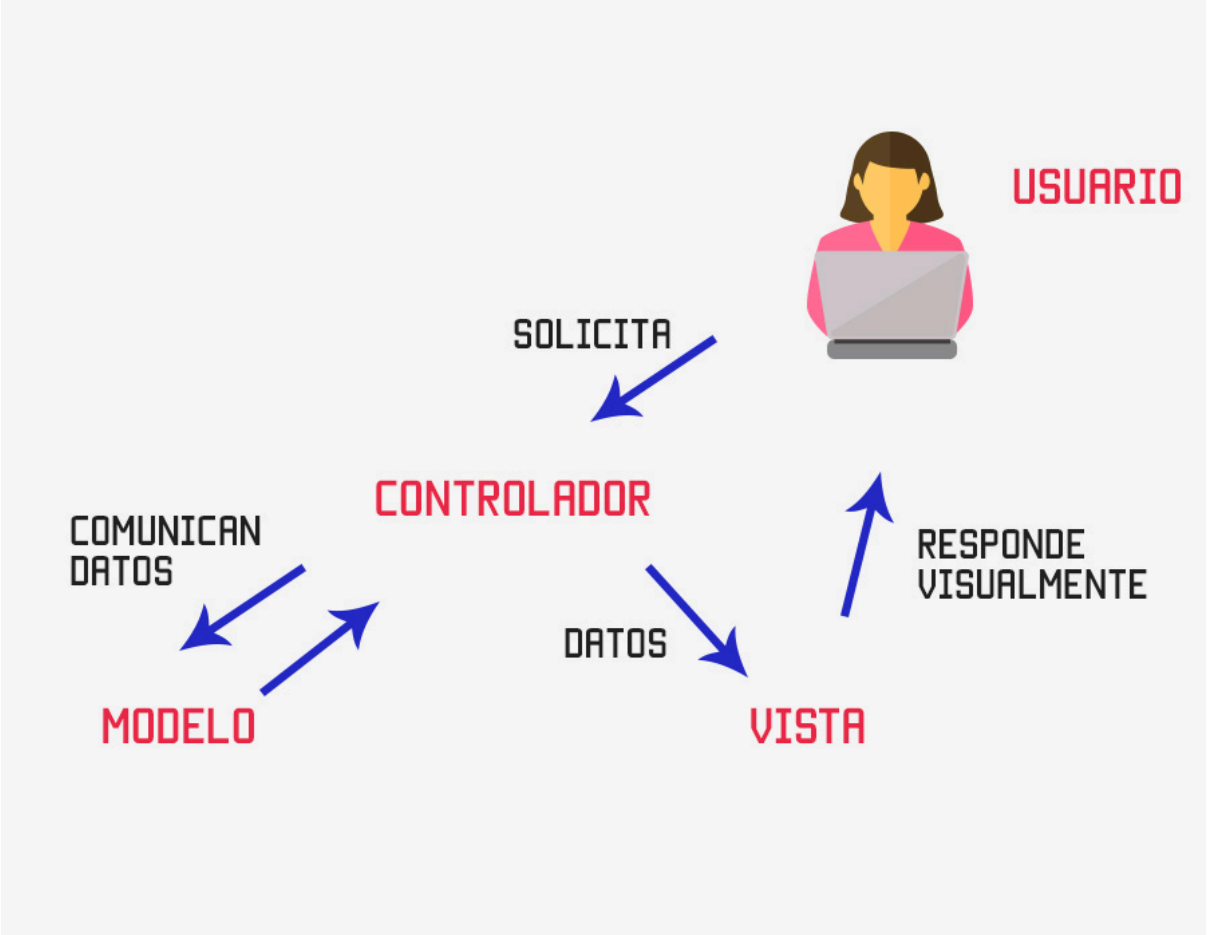
En el caso de **MVC**, el HTML y CSS se generan directamente en el servidor, construyéndose dinámicamente en respuesta a cada solicitud del usuario. En contraste, una **API REST** se enfoca exclusivamente en proporcionar datos, generalmente en formato JSON, sin incluir elementos de presentación. Esto plantea la pregunta: ¿qué ocurre con la capa visual en una arquitectura basada en REST? La respuesta es que se trata de un servicio independiente, separado de la lógica del servidor. Esta capa visual, a menudo implementada como una aplicación frontend, consume los datos de la API REST mediante solicitudes HTTP dirigidas a rutas específicas, definidas durante la creación o desarrollo del servidor. Este enfoque fomenta un mayor desacoplamiento entre las capas y permite una mayor flexibilidad en el desarrollo de aplicaciones modernas, además, permite ser consumido por más de un programa frontend que disponga estos datos mediante distintos diseños según la necesidad.

Un ejemplo de esto último podría ser una API Rest que contenga la información con los productos publicados en una plataforma de marketplace y que cada usuario autenticado pueda consultar sus productos publicados para reutilizarlos en una tienda en línea personalizada, evitando así crear un servicio backend nuevo, que necesite además estar sincronizado y dejando únicamente la necesidad de contar un con programa frontend que obtenga y muestre esa información para generar un flujo de compra.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

MVC

El patrón **Modelo-Vista-Controlador** (MVC) es una de las arquitecturas más ampliamente adoptadas en el desarrollo de software, especialmente en aplicaciones web. Su principal fortaleza radica en la separación de responsabilidades, lo que facilita el mantenimiento, la escalabilidad y la organización del código. Este patrón divide la aplicación en tres componentes fundamentales: el modelo, la vista y el controlador, cada uno con un propósito específico.



El **modelo** es la capa que gestiona los datos y la lógica de negocio de la aplicación. Es responsable de interactuar con la base de datos, procesar las reglas del negocio y garantizar la integridad de los datos. En términos simples, el modelo encapsula toda la información y las operaciones necesarias para manejar los datos de la aplicación.

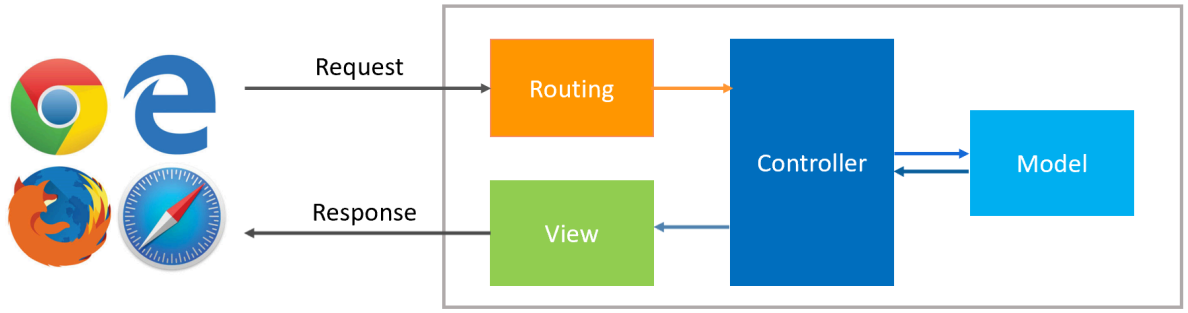
La **vista** representa la interfaz de usuario, es decir, lo que los usuarios ven y con lo que interactúan. Se encarga de presentar los datos proporcionados por el modelo en un formato entendible y atractivo, utilizando tecnologías como HTML, CSS y, en algunos casos, JavaScript. Es importante destacar que la vista no contiene lógica de negocio, lo que garantiza una clara separación entre la presentación y los procesos subyacentes.

El **controlador** actúa como un intermediario entre el modelo y la vista. Es responsable de recibir las solicitudes del usuario, procesarlas, interactuar con el modelo para obtener los datos necesarios y devolverlos a la vista para su presentación. Además, gestiona la lógica de navegación y coordina las interacciones entre los otros dos componentes.

Esta estructura ofrece múltiples ventajas. Por un lado, permite a los desarrolladores trabajar de forma simultánea en diferentes componentes de la aplicación sin interferir en el trabajo de los demás, lo que mejora la eficiencia del desarrollo. Por otro lado, facilita la reutilización de componentes y reduce el riesgo de errores al mantener las responsabilidades bien definidas.

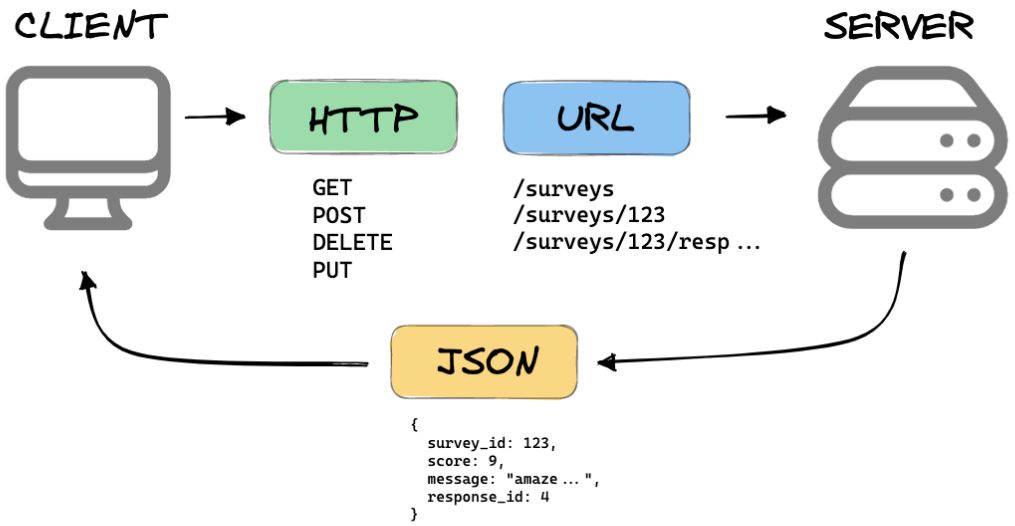
Sin embargo, el patrón MVC también tiene sus desafíos. En aplicaciones muy grandes, puede llegar a ser complejo gestionar la comunicación entre el modelo, la vista y el controlador, especialmente si no se siguen buenas prácticas de desarrollo. A pesar de ello, su capacidad para organizar el código de manera clara y escalable lo convierte en una elección común para el desarrollo de aplicaciones modernas.

Si bien este modelo implementa las 3 capas anteriormente mencionadas, en muchas ocasiones se le acopla una 4 capa llamada router que funciona como receptor de las peticiones, estableciendo la organización sobre qué controlador será responsable de responder a cada solicitud:



API Rest

Una **API REST (Representational State Transfer)** es un estilo de arquitectura ampliamente utilizado en el desarrollo de aplicaciones web y servicios. Este enfoque permite a los sistemas interactuar entre sí de manera coherente y eficiente, asegurando que las solicitudes y respuestas sean entendidas independientemente de las plataformas involucradas. Más que un protocolo o estándar, REST establece principios arquitectónicos que guían el diseño de servicios web.



El núcleo de REST radica en la exposición de recursos a través de URLs específicas y en el uso de los verbos HTTP estándar (GET, POST, PUT, DELETE) para interactuar con esos recursos. Por ejemplo, un recurso puede ser un usuario, un producto o cualquier entidad manejada por la aplicación, y cada operación (como recuperar, crear o eliminar) está claramente definida mediante los métodos HTTP.

Los datos en una API REST se transmiten comúnmente en formatos como JSON o XML, aunque otros como texto plano, HTML o incluso archivos en otros lenguajes pueden ser utilizados. JSON se destaca por su simplicidad y eficiencia, siendo el formato más popular en los servicios REST debido a su capacidad para ser fácilmente procesado por sistemas modernos y navegadores web.

REST también introduce un conjunto de restricciones arquitectónicas que fomentan la creación de sistemas flexibles, escalables y fáciles de mantener. Entre estas restricciones, destaca la interfaz uniforme, que asegura que todos los recursos sean manipulados de manera consistente. Además, REST promueve la ausencia de estado en las interacciones (stateless), lo que significa que cada solicitud es independiente y no depende de solicitudes previas. Esto mejora la escalabilidad del sistema y facilita su implementación en aplicaciones distribuidas.

Otras características clave de REST incluyen la capacidad de cachear respuestas para optimizar el rendimiento, la separación entre el cliente y el servidor para permitir una mayor independencia de desarrollo y la capacidad de construir sistemas que sean fácilmente escalables y visibles en términos de interacción y monitoreo.

Gracias a su simplicidad y versatilidad, las API REST se han convertido en un estándar de facto en la construcción de servicios web modernos, siendo utilizadas en aplicaciones móviles, servicios en la nube y plataformas de integración entre diferentes sistemas.

Entonces, ¿Qué necesito para crear una API Rest?:

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Definir los recursos:

Identificar los recursos que se van a exponer en la API RESTful, como entidades de negocio o funciones específicas.

Utilizar los códigos de estado HTTP

Utilizar los códigos de estado HTTP para comunicar el resultado de la operación, como 200 para una solicitud exitosa o 404 para un recurso no encontrado.

Definir la estructura de la URL

Utilizar URLs descriptivas para cada recurso, evitando utilizar verbos o adjetivos en la URL y separando los elementos con barras diagonales.

Utilizar el formato de datos correcto

Utilizar el formato de datos adecuado para cada operación, como XML o JSON, y especificar el tipo de contenido en la cabecera HTTP.

Utilizar los verbos HTTP

GET, POST, PUT, DELETE, etc. En base a la intención para manipular los recursos. Cómo usar GET para obtener un recurso y POST para crear uno nuevo.

Utilizar la documentación

Documentar la API RESTful para que los consumidores puedan entender cómo utilizarla y qué recursos están disponibles.

Utilizar la autenticación y la autorización

Proteger la API RESTful mediante la autenticación y la autorización de los usuarios que acceden a los recursos.

Para concluir, veamos una tabla comparativa entre las diferencias de ambos patrones de arquitectura:

Característica	MVC	REST
Enfoque	Separación de preocupaciones y organización en tres componentes claramente definidos: **Modelo**, **Vista** y **Controlador**.	Exposición de recursos y utilización de verbos **HTTP** para manipularlos.
Tipo de aplicación	Principalmente aplicaciones web, aunque también puede utilizarse en otros tipos de aplicaciones.	
Manejo de solicitudes	A través del Controlador, que actúa como intermediario entre la Vista y el Modelo.	A través de los verbos HTTP: GET, POST, PUT y DELETE.
Representación de la información	Utiliza un conjunto de estructuras de datos para representar los datos en la aplicación.	Utiliza formatos como JSON o XML para representar los datos.
Escalabilidad	Escalabilidad limitada debido a la organización de la aplicación en tres componentes.	Altamente escalable debido a la utilización de los verbos HTTP y la exposición de recursos.
Reutilización de código	Permite la reutilización de código a través de la separación clara de las responsabilidades.	Permite la reutilización de código a través de la exposición de recursos y la utilización de los verbos HTTP.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Consignas Pre-Entrega de Proyecto Final

Matías y Sabrina te observan con una mezcla de entusiasmo y expectación. “Hemos llegado al momento clave”, dice Sabrina. “Es hora de demostrar si estás preparado para dar el siguiente paso y unirte a nuestro equipo en TechLab”.

Matías, con una sonrisa, añade: “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”.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

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 id del producto creado como resultado en la consola.

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

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 DELETE products/7`

Buenos Aires aprende
Agencia de Habilidades para el Futuro

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? 🌟

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.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Preguntas para Reflexionar:

- ¿Qué diferencias fundamentales existen entre un servidor de hardware y un servidor de software, y cómo se complementan?
- ¿Qué criterios considerarías para elegir entre un hosting compartido, VPS o dedicado para un proyecto?
- ¿Qué ventajas ofrece la arquitectura de microservicios frente a una monolítica?
- ¿Cómo afecta la decisión de usar patrones como MVC o API REST en el diseño de un sistema?

Próximos Pasos:

Creando un Servidor Web: Daremos nuestros primeros pasos en la creación de servidores web con Node JS.
Modelando una API Rest: Nos iniciaremos en la metodología para crear nuestra primera API Rest.
Request y Response: llegó el momento de aplicar de forma práctica todo nuestro conocimiento sobre la comunicación web.

Buenos Aires Ciudad