API

DEV.F::
DESARROLLAMOS(PERSONAS);

Comunicación entre distintos back end

DEV.F

¿Qué es un API?

Application Programming Interface, son **mecanismos** que **permiten** a **dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos**.

- Normalmente un API conecta un front end y un back end pero también es usual que conecte dos back end.

Ejemplo: El sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de tu teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en pantalla.

Conceptos

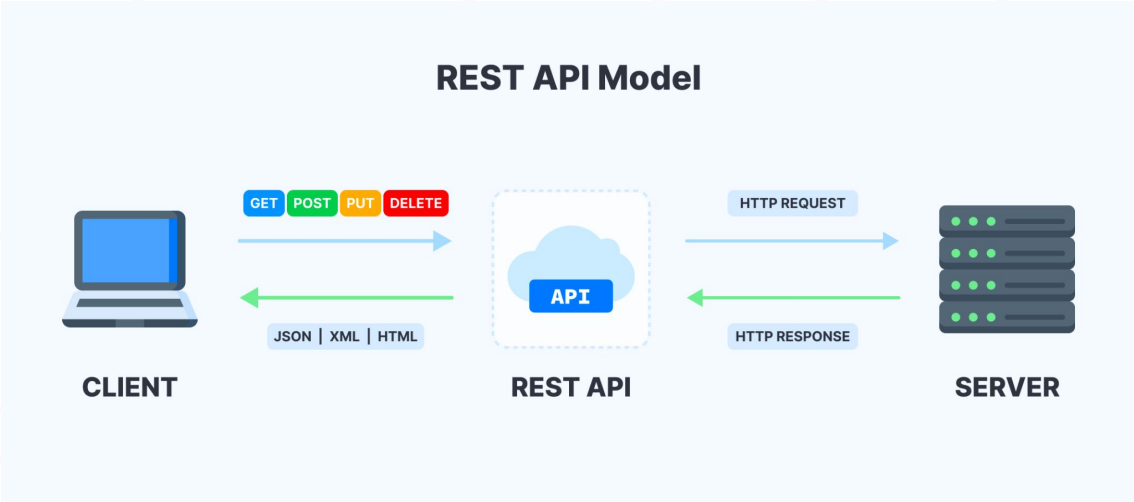
DEV.F:
DESARROLLAMOS(PERSONAS);

Conceptos de un API

- **Client:** Es quien realiza peticiones de información.
- **Request:** Es la solicitud o petición que realiza el cliente y se conforma de un contrato.
- **Contrato:** Son las condiciones establecidas para poder consumir cada endpoint de un API. Se conforma de:
  - **Url:** Dirección de internet.
  - **Payload:** Información que va en el body de la request.
  - **Método o verbo:** GET, POST, PUT, PATCH y DELETE.
- **Recurso:** Es el endpoint expuesto por el API y consumido por el servidor.
- **Response:** Respuesta de la API al consumir el recurso.
- **Server:** Es donde se aloja el API y es quien responde las peticiones del cliente.

API, API Rest y API Restful

Un **API es un interfaz de comunicación** de aplicación, **REST son los principios que idealmente debe seguir un API** y **ful** hace referencia a un API que está netamente construida bajo los principios REST.



Concepto de Url

Uniform Resource Locator **es una dirección que es dada a un recurso único en la Web**. En teoría, cada URL válida apunta a un único recurso. Dichos recursos pueden ser páginas HTML, documentos CSS, imágenes, etc.

Estructura de una url

Protocolo

Un protocolo es un **conjunto de reglas** (estándares y políticas formales) **para el intercambio de información entre dos o más dispositivos a través de una red.**

Listado de protocolos

- **TCP/IP** : Transfer control protocol/Internet Protocol.
- **HTTP:** Se encarga de la comunicación entre un servidor web y un navegador web. HTTP se utiliza para enviar las peticiones de un cliente web (navegador) a un servidor web, volviendo contenido web (páginas web) desde el servidor hacia el cliente.
- **HTTPS:** Lo mismo pero más seguro. La información viaja de manera segura y encriptada mediante un certificado SSL/TLS.

- **FTP**: File transfer protocol, como su nombre lo indica es un protocolo para transferencia de archivos.
- **SMTP**: Simple Mail Transfer Protocol.
- **IMAP** - Internet Message Access Protocol.
- **POP** - Post Office Protocol.
- **SSL** - Secure Sockets Layer.
- **TLS** - Transport Layer Security.

Status Codes

NOTA: Consultar estatus recomendados a aprenderse aquí.

Entendiendo conceptos

- Servicio ≈ endpoint ≈ recurso ≈ servicio web.
- API ≈ API REST ≈ API Restful.
- Request = Petición = Solicitud.
- Response = Respuesta.
- Path ≈ Dirección ≈ Url ≈ URI.
- File = Fichero = Archivo.
- Terminal = Línea de comandos = CLI = Bash.
- Script ≈ Programa ≈ Conjunto de líneas de código o instrucciones.
- Entidad (BD) ≈ Clase (Programación) ≈ Recurso (API).

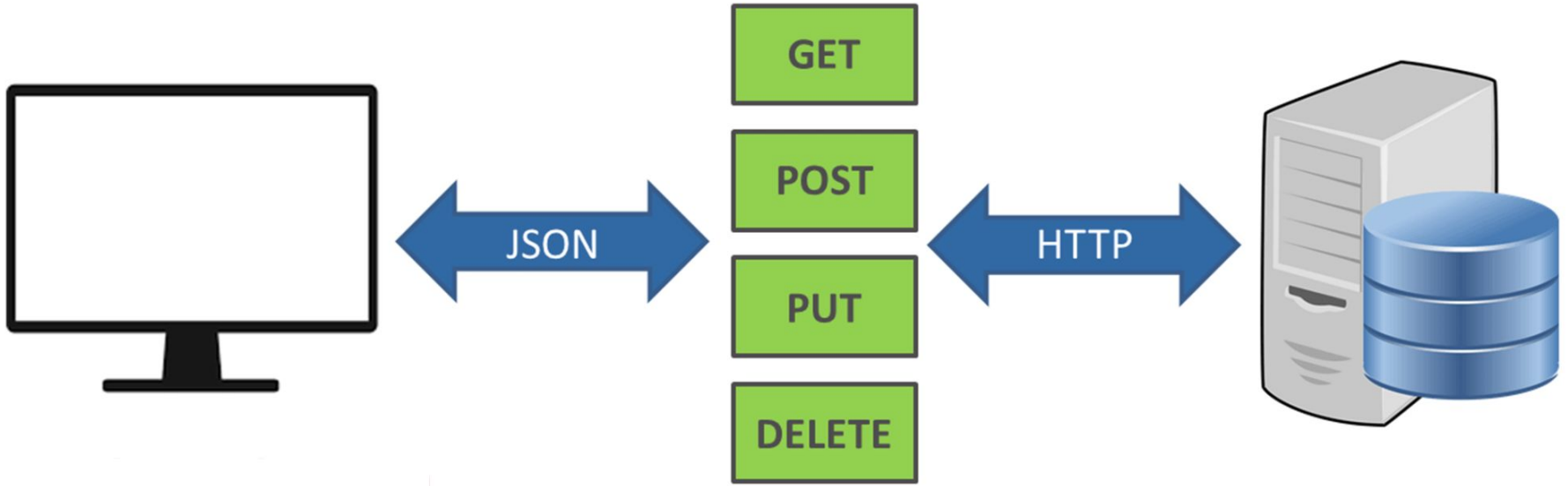
REST

DEV.F::
DESARROLLAMOS(PERSONAS);

¿Qué es REST?

- Es una **serie de principios** (buenas prácticas) que las **API** siguen para “volverse” **=> API REST (API Restful)**.
- Es una “evolución” de SOAP (servicios web).
- REST es una interfaz para conectar varios sistemas. Se basa en el protocolo HTTP y nos sirve para obtener y generar datos, dichos datos son transmitidos en distintos formatos como como XML y JSON.

Arquitectura REST



Principios de REST

- **Client - server:** El cliente y el servidor deben **ser completamente independientes** entre sí. El cliente solo debe conocer el **contrato** del recurso solicitado.
- **Stateless**: Cada solicitud debe incluir toda la información necesaria para procesarla. Las aplicaciones de servidor no pueden almacenar ningún dato relacionado con una solicitud de cliente.
- **Cacheable**: Tanto en el servidor como en el cliente. El objetivo es mejorar el rendimiento en el lado del cliente, al mismo tiempo que aumenta la escalabilidad en el lado del servidor.

Principios de REST

- **Uniform Interface**: Todas las solicitudes de API para el mismo recurso deben ser iguales. Mismas entradas, mismas salidas.
- **Layered system**: En las API REST, las llamadas y respuestas pasan por diferentes capas. Como regla general, no debe suponerse que las aplicaciones de cliente y de servidor se conectan directamente entre sí.
- **Code on demand (optional):** Las API REST envían recursos estáticos, pero en algunos casos, las respuestas también pueden contener un código ejecutable (como applets de Java). En estos casos, el código solo debería ejecutarse bajo demanda.

REST - CRUD

CREATE - READ - UPDATE - DELETE

API Name	HTTP Method	Path	Status Code	Description
GET Employees	GET	`/api/v1/employees`	200 (OK)	All Employee resources are fetched.
POST Employee	POST	`/api/v1/employees`	201 (Created)	A new Employee resource is created.
GET Employee	GET	`/api/v1/employees/{id}`	200 (OK)	One Employee resource is fetched.
PUT Employee	PUT	`/api/v1/employees/{id}`	200 (OK)	Employee resource is updated.
DELETE Employee	DELETE	`/api/v1/employees/{id}`	204 (No Content)	Employee resource is deleted.