José Montoya Guzmán

- Tech Lead Front End en BBVA.
- Git Evangelist.
- Experiencia en Java y SQL.
- Sensei por hobby.

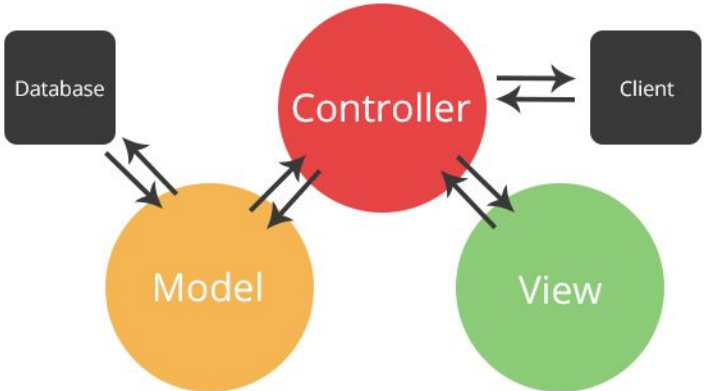
https://github.com/montoyaguzman

KATA
JavaScript Avanzado

DEV.F::
DESARROLLAMOS(PERSONAS);

¿Qué veremos en está kata?

- Node.
- Npm y paquetes.
- Node para front y back end.
- Arquitectura de software.
- Stacks de desarrollo web.
- Asincronía y Event Loop.
- Comparativa entre Node vs JavaScript.
- Conceptos de una API.
- Qué es REST y sus principios.
- API Rest.
- ExpressJS.
- Deploys.



Semana 1

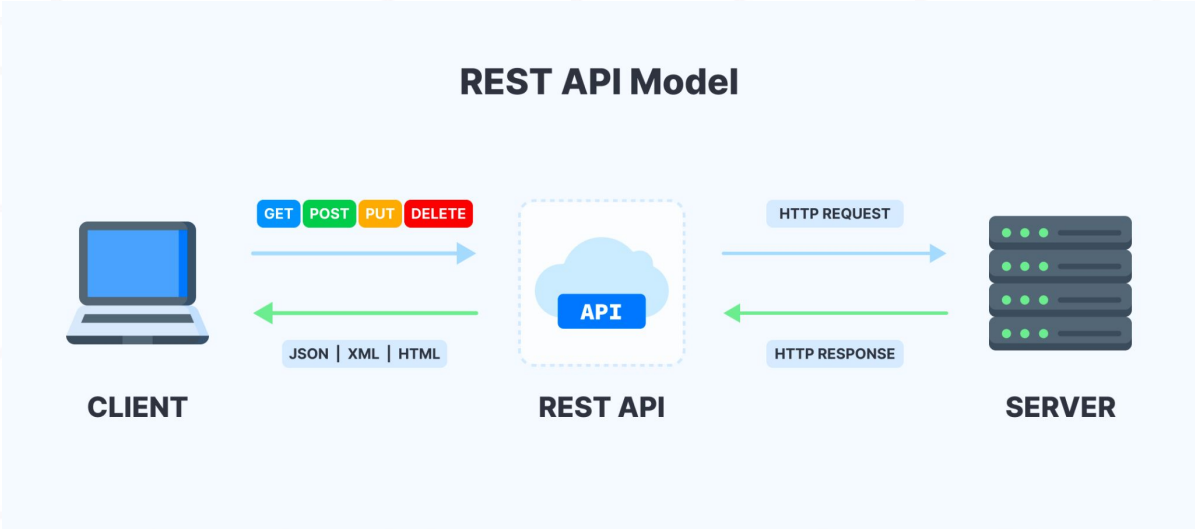
- Node.
- Npm y paquetes.
- Node para front y back end.
- Arquitectura de software.
- Introducción a los stacks de desarrollo web.
- Stacks de desarrollo web.

Semana 2

- Asincronía y Event Loop.
- Comparativa entre Node vs JavaScript.

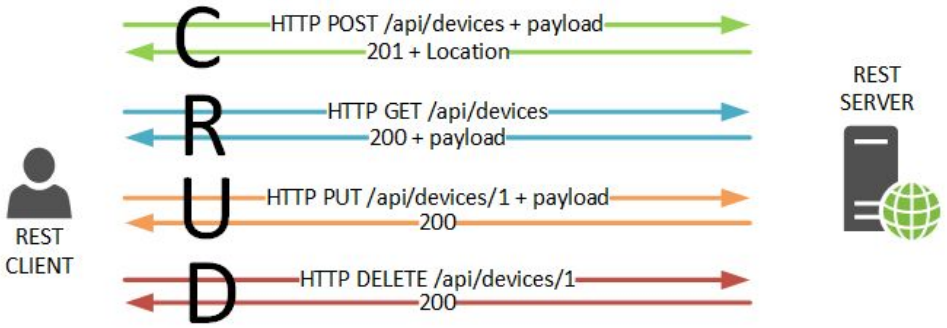
Semana 3

- Conceptos de una API.
- Qué es REST y sus principios.
- API Rest.
- ExpressJS.
- Deploys.



Semana 4

- Construir una API Rest que se conecte con un fake back end, y realice las 4 operaciones básicas de un CRUD.



Node

DEV.F::
DESARROLLAMOS(PERSONAS);

Node

- Mayo 2009 (Ryan Lenhiart).
- Entorno en tiempo de ejecución **multiplataforma** para la **capa del servidor (no se limita a ello).**
- Basado en el motor V8 de google.
- Escrito en C++.
- Basado en módulos.
- Es **asíncrono** y trabaja con base en un **bucle de eventos**.

¿Qué puedo hacer con Node?

- **Realizar API Rest**.
- Acceder a bases de datos relacionales y no relacionales.
- **Generar páginas dinámicas en un servidor web. => server side render**
- Crear, leer y escribir archivos.
- Procesar y almacenar archivos enviados desde una página web.
- Recuperar datos de formularios HTML.

Práctica 1

- Instalación de node y npm.
- Validación en la CLI de Node.
  - node -v
  - npm -v
- Uso de la documentación oficial.

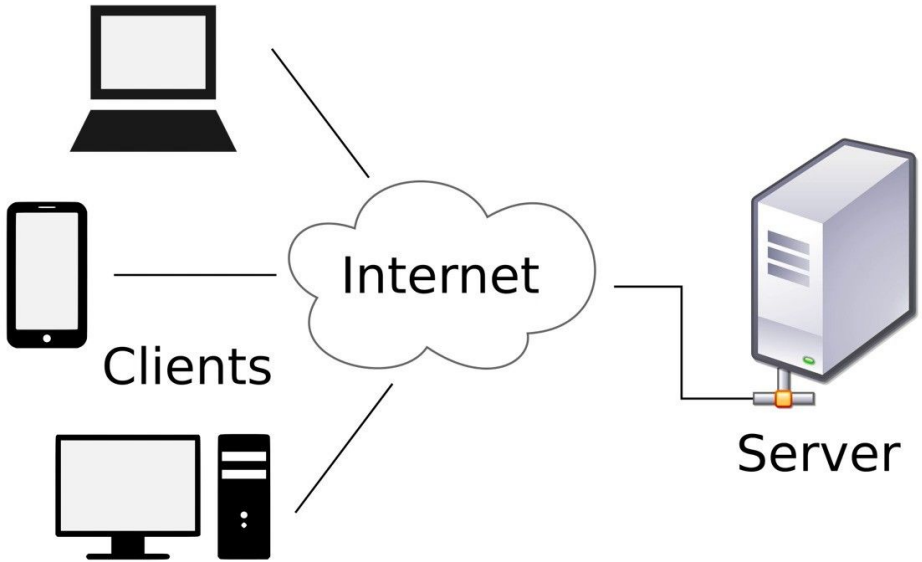
Práctica 2

- Ejecución de código JS en Node.
- Console.
- setTimeout, setInterval.

Práctica 3

- Objeto Global (this).
- Módulos (os, fs, http, url, path).

Intro al modelo cliente servidor



Intro al concepto de URL

Uniform resource identifier, sirve para acceder a un recurso físico o abstracto por Internet. A continuación se muestran los elementos mínimos de una URL: **protocolo://dominio/path?queryParamsEjemplo:**

https://www.tutorialesprogramacionya.com/javascriptya/nodejsya/detalleconcepto.php?punto=1&codigo=1&inicio=0

Mime Types

Los mime types indican la naturaleza y formato de los archivos que son transmitidos en la respuesta de una solicitud web.

- Listado.

Localhost

Es la dirección de mi propio computador, también se le conoce como dirección loopback.

Localhost: http://localhost:8080

Loopback: http://127.0.0.1:8080

Práctica 4

- Crear un archivo de texto con node.

Práctica 5

- Servidor web que devuelve una página HTML.

Módulos

DEV.F:
DESARROLLAMOS(PERSONAS);

Módulos

Permiten aislar parte de nuestro código en diferentes archivos y mandarlos llamar sólo cuándo los necesitamos. Existen dos formas de utilizar módulos en node:

- Common JS.
- ES6 Imports (.mjs, --experimental-modules y “type”: “module” en package.json).

Práctica 6

- Calculadora hecha con módulos, utilizando importaciones de common JS e importaciones ES6.

Práctica 7

- API Rest de páginas web con node.

Npm

DEV.F::
DESARROLLAMOS(PERSONAS);

Npm

Node Package Manager o manejador de paquetes de node, es la herramienta más popular de JavaScript para compartir e instalar paquetes. Se compone de 2 partes:

- **Un repositorio online para publicar paquetes** de software libre para ser utilizados en proyectos Node.js
- **Una herramienta para la terminal (CLI)** para interactuar con dicho repositorio que ayuda a la instalación de utilidades, manejo de dependencias y la publicación de paquetes.

NOTA: Se puede considerar un gestor de dependencias de proyectos de tipo npm.

Comandos de Npm

Inicialización y arranque

- **npm init:** Inicializa una carpeta como un proyecto de npm.
- **npm start:** Es el único comando por defecto para iniciar un proyecto de node.

Comandos de Npm

Instalar/desinstalar dependencias

Dependencia: Son los recursos o librerías externas que utilizan los proyectos para funcionar.

- npm i
- npm install -g <pakage-name>
- npm install <package-name>
- npm install –save <pakage-name>
- npm install -D <package-name>
- npm uninstall <package-name>
- npm uninstall -g <package-name>

NOTA: install = i

Comandos de Npm

Gestión de dependencias

- npm search <package-name>
- npm ls
- npm update -save
- npm list
- npm list -g –depth 0
- npm outdated

Paquetes

Son módulos distribuidos en forma de librerías que resuelven alguna necesidad de desarrollo. A continuación se listan los más populares al 2022:

- npm.
- create-react-app.
- vue-cli.
- grunt-cli.
- mocha.
- react-native-cli.
- gatsby-cli.
- forever.

Scripts

Son comandos propios que se pueden agregar al package.json para poderlos ejecutar con **npm run <my-comand>.**

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node index.js",
  "dev": "nodemon index.js"
},
```

Estructura de proyecto npm

- **node_modules:** Carpeta donde se instalan las dependencias de un proyecto npm, normalmente está carpeta se agrega al .gitignore.
- **package.json:** Guardan las dependencias y los comandos de node.
- **package-lock.json:** Guarda un snapshot de las dependencias que se instalaron en un determinado momento.

Detalle del package.json

Este archivo guarda las dependencias y los comandos de node.

- name.
- version.
- description.
- license.
- scripts.
- **devDependencies:** Son dependencias que sólo se instalan en el entorno local.
- **dependencies:** Son dependencias que se instalan en cualquier entorno (local, test, qa, prepro y pro).

Detalle del package-lock.json

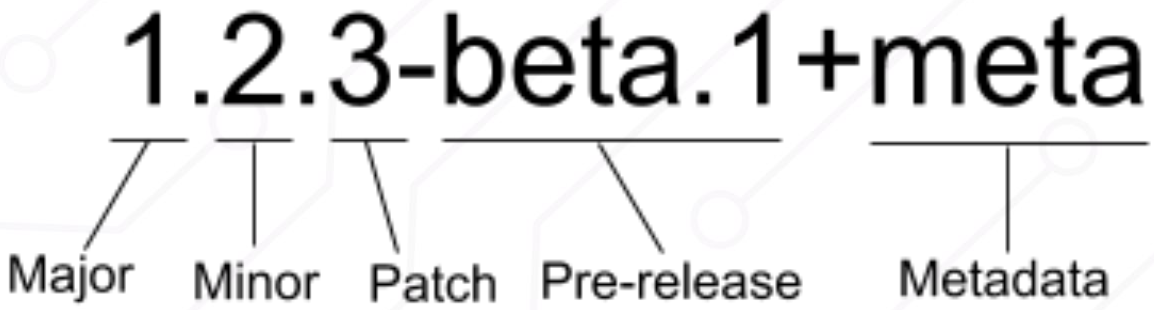
- Este archivo tiene las versiones exactas de las dependencias utilizadas por un proyecto npm.
- No está pensado para ser leído línea por línea por los desarrolladores.
- Es usualmente generado por el comando **npm install.**

Semantic Versioning

DEV.F::
DESARROLLAMOS(PERSONAS);

Semantic Versión

Es un conjunto simple de reglas y requerimientos que dictan cómo asignar e incrementar los números de la versión de un software. Evitan la pérdida de versiones y mejoran la gestión de dependencias.



Funcionamiento de semantic versión

Dado un número de versión **MAYOR.MENOR.PARCHE**, se incrementa:

- La versión **MAYOR** cuando realizas un cambio incompatible en el proyecto.
- La versión **MENOR** cuando añades funcionalidad que compatible con versiones anteriores.
- La versión **PARCHE** cuando reparas errores compatibles con versiones anteriores.

MAYOR.MENOR.PARCHE = MAJOR.MINOR.PATCH.

Ejemplo: 1.2.1

Práctica 8

- Mejora al API Rest de páginas web con node.

Práctica 9

- Definir los siguientes conceptos:

Entorno de ejecución, manejador de paquetes, CLI, comandos, dependencia, gestor de dependencias, y script.

- Explicar en una oración o diagrama como funciona la comunicación en internet.

Práctica 10

- Servidor simple de api rest en node.

Práctica 11

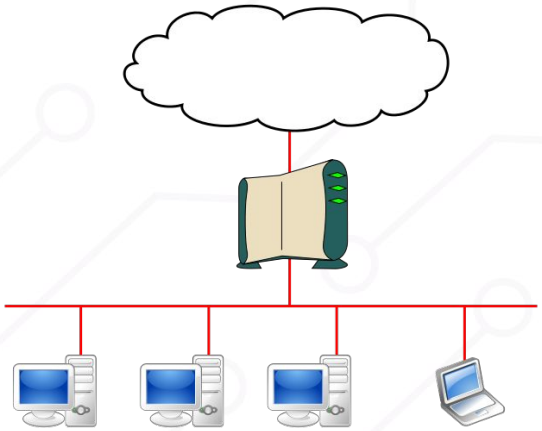
- Servidor de archivos multimedia.

Arquitecturas de software

DEV.F::
DESARROLLAMOS(PERSONAS);

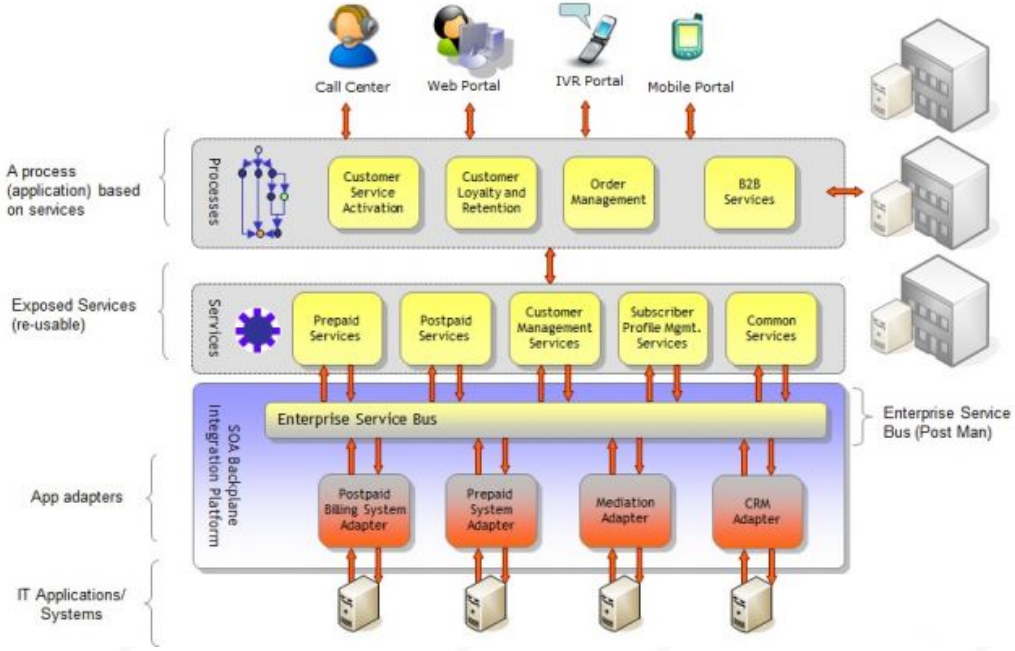
¿Qué es una arquitectura en software?

- Es la estructura y forma en que los componentes de software o hardware se distribuyen y relacionan en el stack.
- La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema.
- Definir las herramientas, patrones y lineamientos con los que una aplicación va a trabajar.



¿Por qué hablar de arquitectura?

A semejanza de los planos de un edificio o construcción, estas indican la estructura, funcionamiento e interacción entre las partes del software.

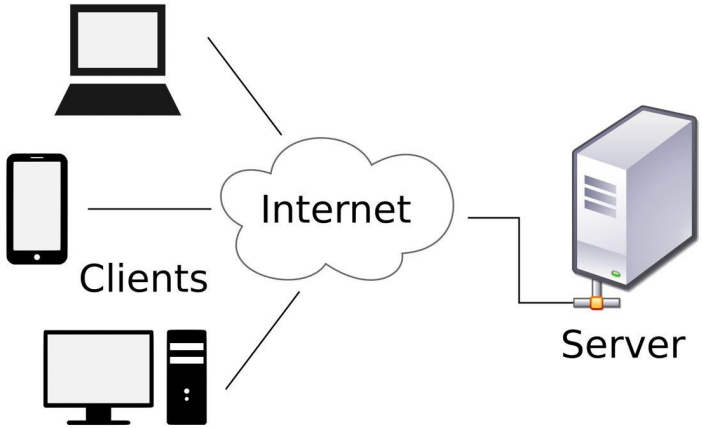


Arquitecturas más comunes

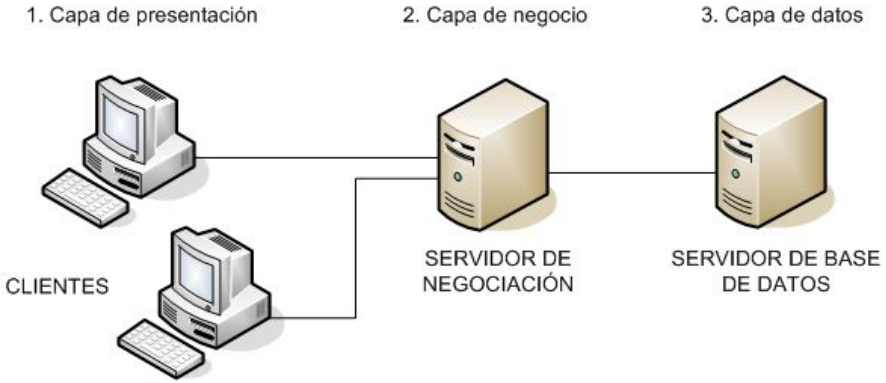
Generalmente, no es necesario inventar una nueva arquitectura de software para cada sistema de información.

Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto.

Cliente Servidor



Arquitectura de Tres Niveles



Arquitectura MVC

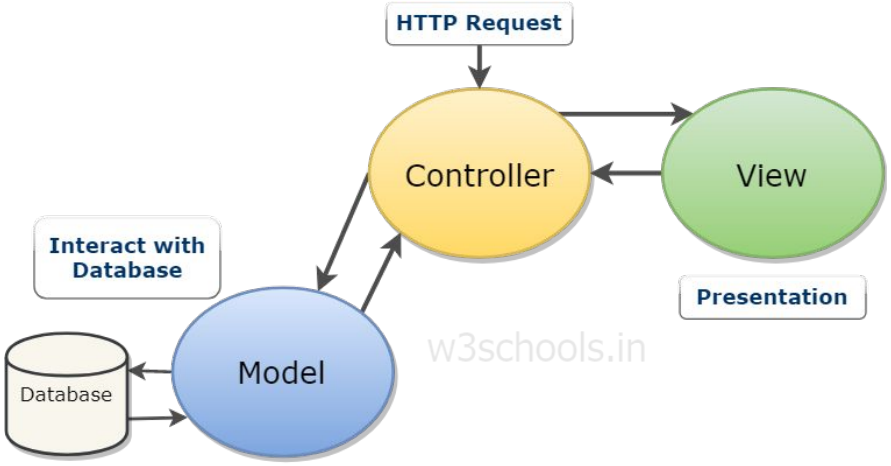


Fig: MVC Architecture

Arquitectura dirigida por eventos

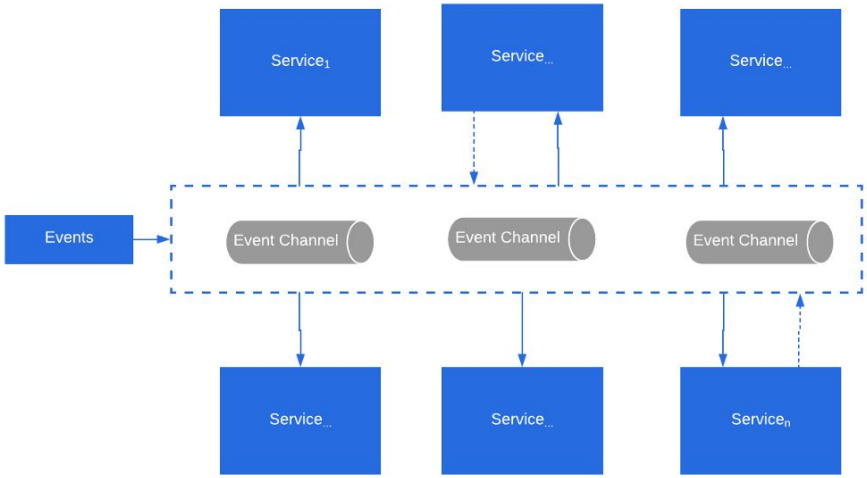
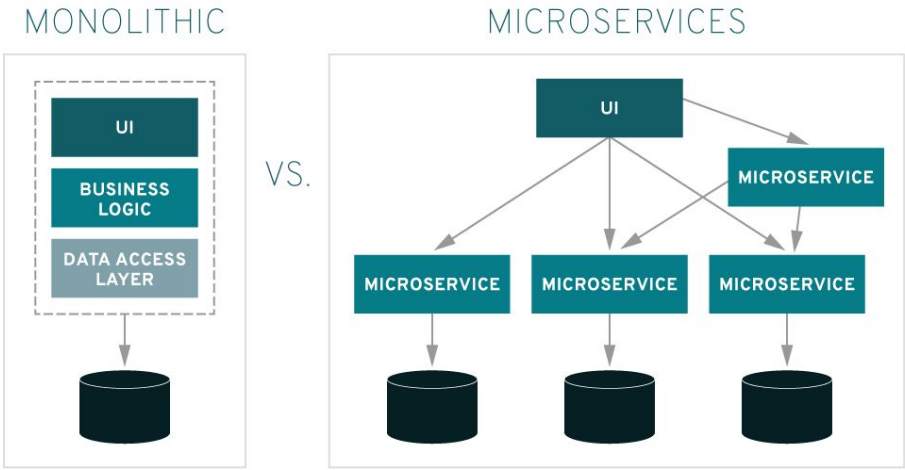


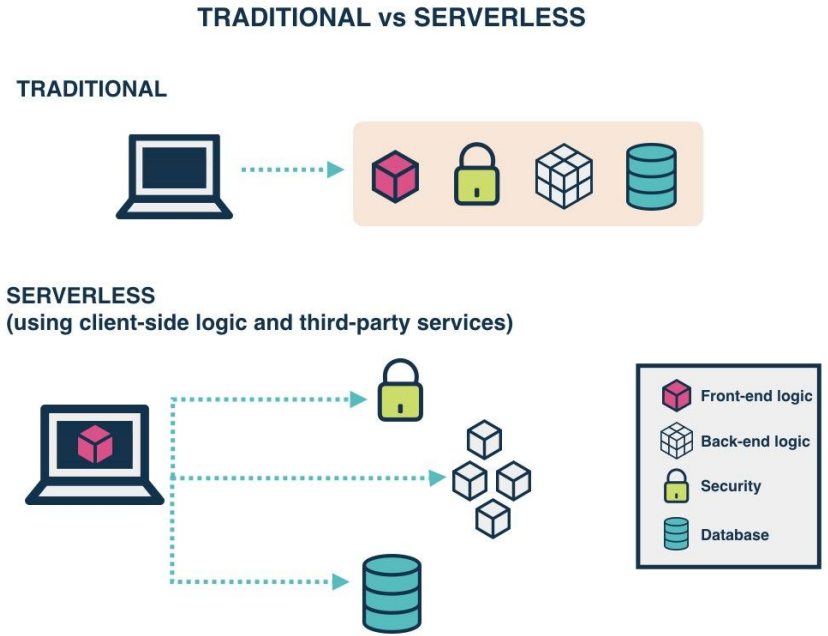
Figure 2 - Broker Topology

Arquitectura de Micro Servicios



DEV.F::

Serverless


TRADITIONAL vs SERVERLESS
TRADITIONAL
SERVERLESS
(using client-side logic and third-party services)
Front-end logic
Back-end logic
Security
Database

Stacks de Desarrollo web

DEV.F:: DESARROLLAMOS(PERSONAS);

¿Qué es un stack tecnológico?

Es un **conjunto de servicios de software que se utilizan para el desarrollo de aplicaciones**. Normalmente, un Stack se conforma por lenguajes de programación, frameworks, bibliotecas, herramientas de desarrollo y enfoques de programación.

Hace referencia al método de apilamiento de los componentes de este conjunto de herramientas, uno encima del otro.

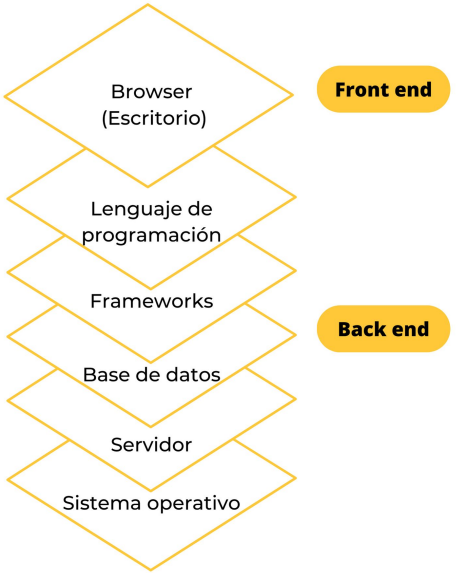
¿Qué es un stack tecnológico?

Un stack de **desarrollo** web se compone de:

- Las tecnologías front end, la API, el backend y la distribución de los mismos.

pero también de:

- Un sistema operativo.
- Un servidor web.
- Una base de datos.
- Un intérprete de lenguaje de programación.

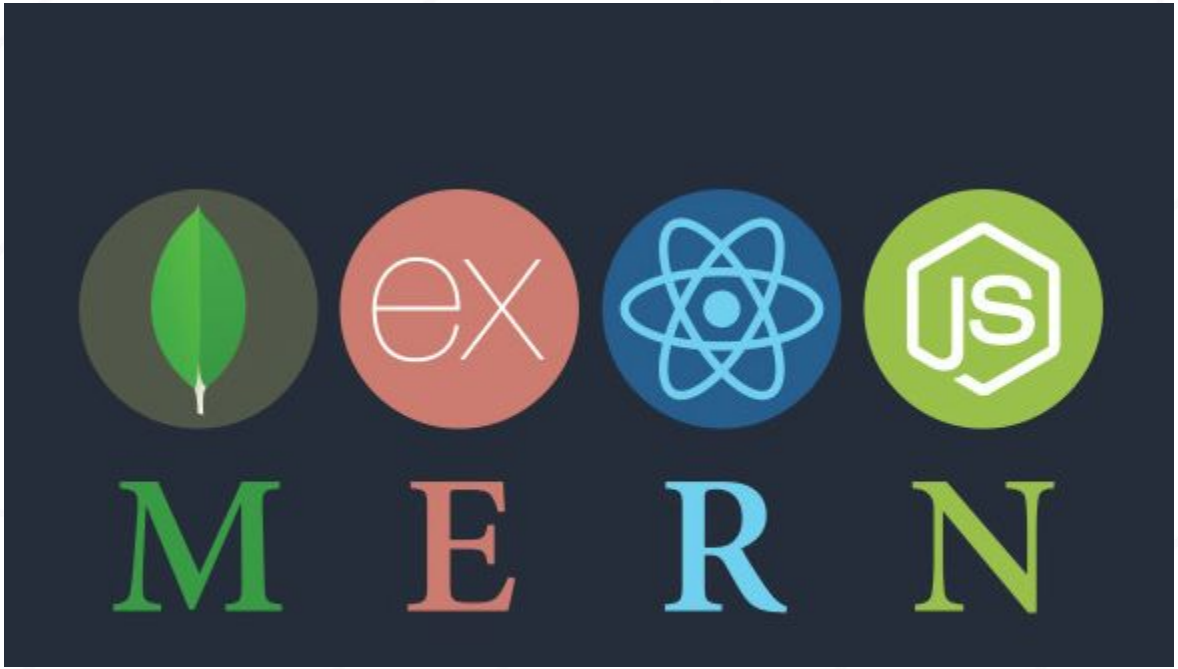


LAMP

ASP.NET

MEAN

MERN




MEVN
MONGODB - EXPRESS - VUEJS - NODEJS

Event Loop y Asincronía

DEV.F:
DESARROLLAMOS(PERSONAS);

Código síncrono

Un código síncrono es aquel código donde cada instrucción espera a la anterior para ejecutarse.

```
1   console.log('Primero');
2   console.log('Segundo');
3   console.log('Tercero');
```

Código asíncrono

Un código asíncrono no espera a las instrucciones diferidas y continúa con su ejecución. Entiéndase por instrucción diferida cualquier cosa que implique un retraso. Esto evita que haya código bloqueante, el código bloqueante se forma en la cola.

```
1   console.log('Primero');
2   setTimeout(_ => {
3     console.log('Segundo');
4   }, 1);
5   console.log('Tercero');
```

¿Dónde aparece el código asíncrono?

- Por naturaleza de JavaScript (Lenguaje no bloqueante).
- Llamado de APIs.
- Escritura de archivos.
- Operaciones con base de datos.
- Al realizar una compra en la plataforma X, se válida primero la tarjeta del cliente y hasta que se comprueba que es válida se ejecuta la compra.
- Filtrar una tabla de datos y generar un reporte de la información.
- Envío de un correo de una sanción a un crédito.

Ventajas de la asincronía

- Permite tener una mejor respuesta en las aplicaciones y reduce el tiempo de espera del cliente.

Métodos de manejo de la asincronía

Para un desarrollador JS es fundamental aprender a buscar mecanismos para dejar claro cuándo ciertas tareas tienen que procesarse de forma síncrona (quedarse a la espera) y cuándo deben ejecutarse de forma asíncrona.

- **Callbacks:** Consiste en pasar una función como parámetro y ejecutarla en el momento que lo necesitemos.
- **Promesas:** Se basan en 3 estados. Cuándo se lanza la petición (pending) y sus posibles respuestas (resolve y reject).
- **Async await (ES8):** Función con la sugar syntaxis de es6 y simula que el código es asíncrono (por detrás sigue siendo una promesa).

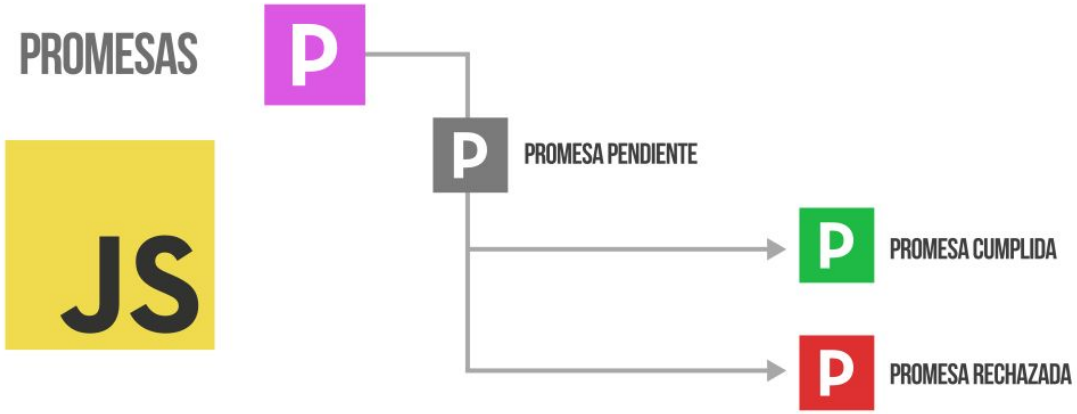
Callbacks

- Consiste en pasar una función como parámetro a otra, para que la segunda función la ejecuté cuando lo requiera. Dicho de otra manera, es la programación la ejecución del código dentro del callback.
- Solución sencilla pero estéticamente compleja de leer y caótica.

```
firstTask(data, function(err, result) {
  secondTask(data, function(err, result) {
    thirdTask(data, function(err, result) {
      fourthTask(data, function(err, result) {
        fifthTask(data, function(err, result) {
          // Code
        });
      });
    });
  });
});
```

Promises

- Consiste en crear un bloque de código y consumirlo. Al consumirlo, primero se tiene una expectativa de su respuesta, después una respuesta de éxito o error.



Async await

- Son sugar syntaxis introducido en es8 (2017) para mejorar la legibilidad de las promesas. Se abandona el modelo de encademiento then.
- Async vuelve a las funciones una promesa y permite usar el keyword await.
- Await espera a que se resuelvan las promesas.
- Para gestionar errores hay que agregar un bloque try - catch.

Comparativa entre Node vs JavaScript

DEV.F:
DESARROLLAMOS(PERSONAS);

JS vs Node

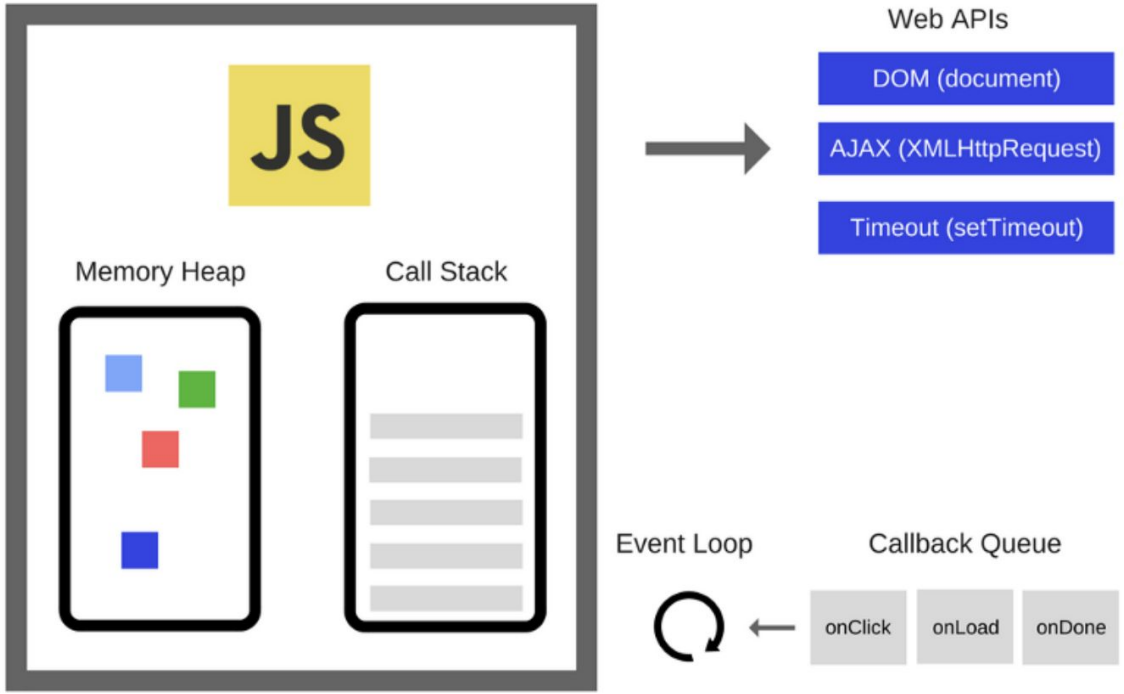
- Lenguaje de scripting vs Entorno de ejecución.
- Acceso al browser vs Acceso al servidor
- Motor del navegador vs V8.
- Js interactúa directamente con el DOM.
- Navegadores usan Libevent.
- Node usa Libuv.
- Ambos se basan en 0 retraso.
- No hay solicitudes Ajax (consumo de API’s).
- No hay temporizador (setTimeout y setInterval).

Conceptos event loop

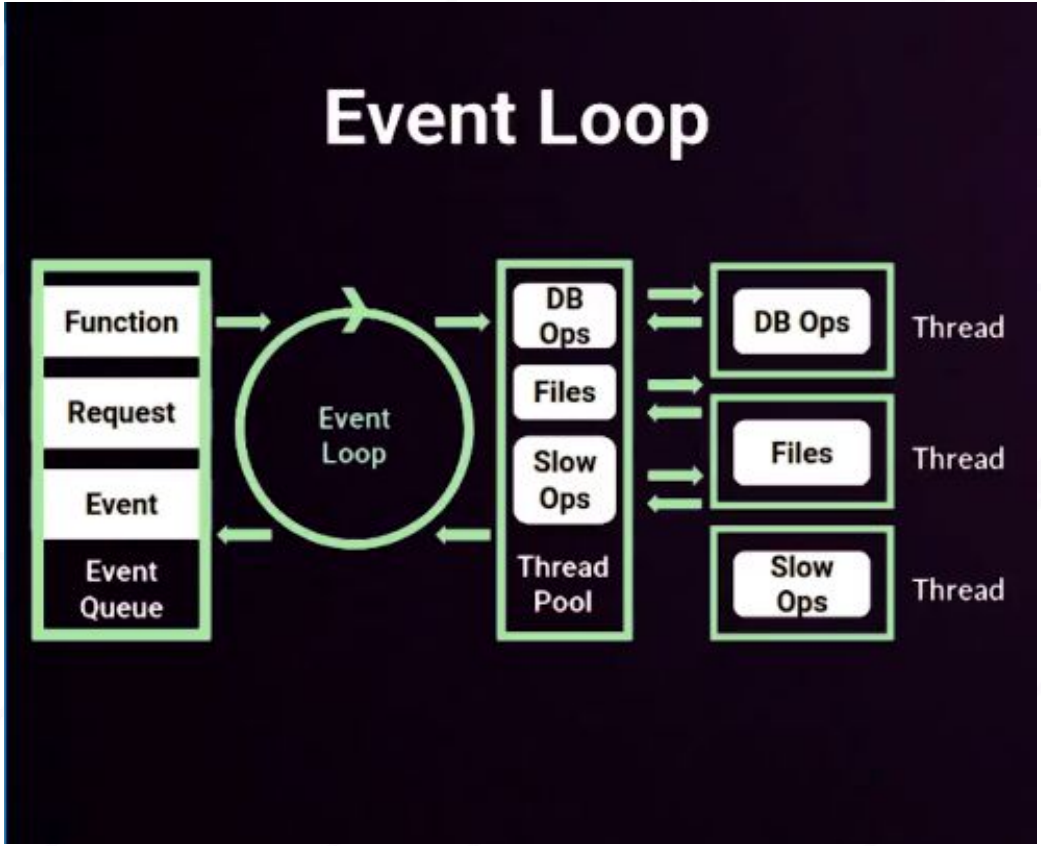
Modelo de concurrencia y ejecución de eventos. Es la base del funcionamiento de JS (y por lo tanto de node).

- Heap.
- Call stack.
- Event loop.
- API Web.
- Callback Queue.

Event Loop JavaScript (libevent)



Event Loop Node (libuv)



Ejemplo de restaurante para asincronismo

Práctica 12

- Hacer un ejemplo práctico de callbacks, promesas y async await.

API

DEV.F::
DESARROLLAMOS(PERSONAS);

Comunicación entre distintos back end

¿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**.

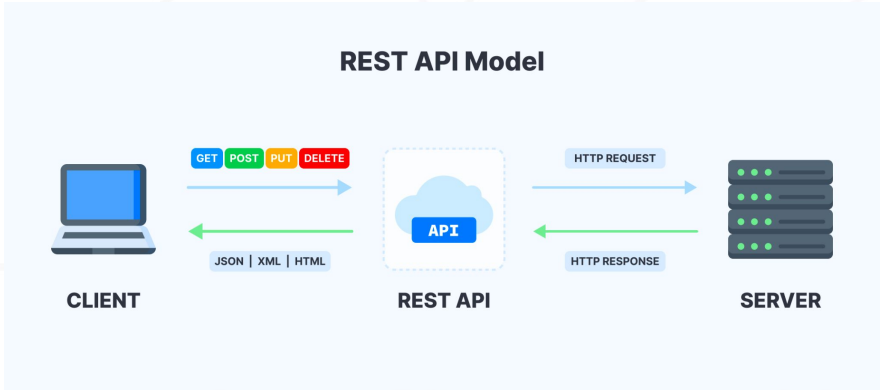
Ejemplo: El sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de tú teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en pantalla.

Conceptos

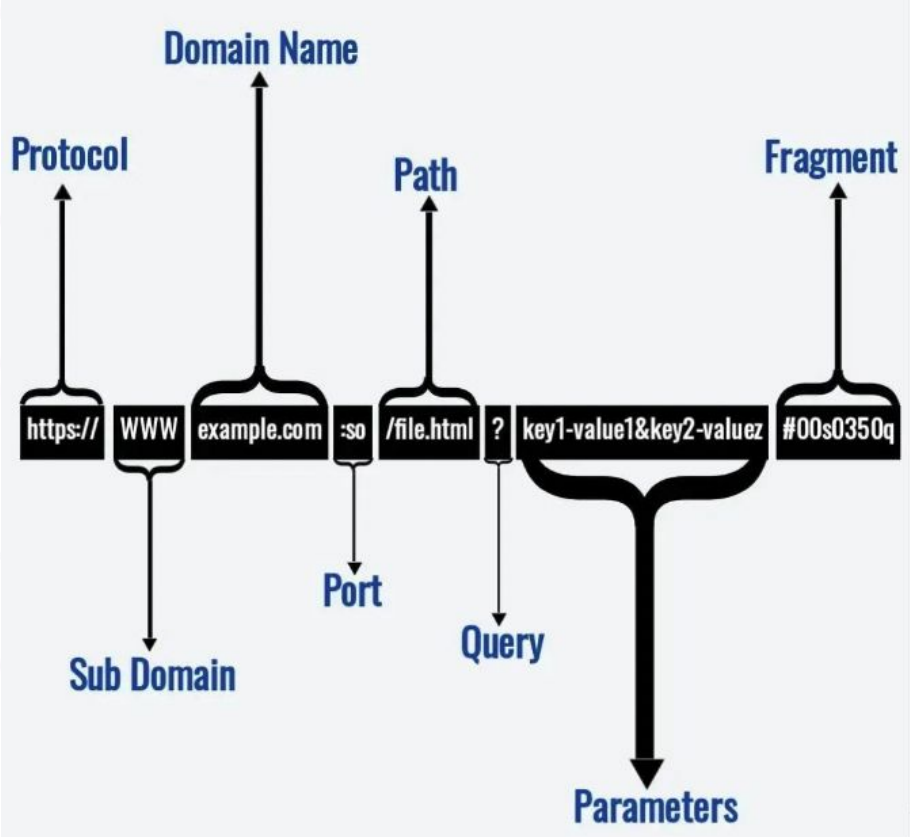
DEV.F:
DESARROLLAMOS(PERSONAS);

Conceptos de un API

- **Client:** Es quien realiza la petición.
- **Request:** Petición http conformada por url, params, payload (body).
- **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 al que llegamos en la API.
- **Response:** Respuesta de la API.
- **Server:** Es quien responde la petición.



Estructura de una url



Concepto de Url

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

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.

Verbos HTTP

- **GET**: Verbo exclusivo para obtener recursos del servidor.
- **POST**: Verbo exclusivo para crear nuevos recursos en el servidor.
- **PUT**: Verbo reemplaza un recurso por completo en el servidor.
- **PATCH**: Verbo que modifica parcialmente el recurso en el servidor.
- **DELETE**: Verbo que elimina física o lógicamente el recurso en el servidor.

Status Codes

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).

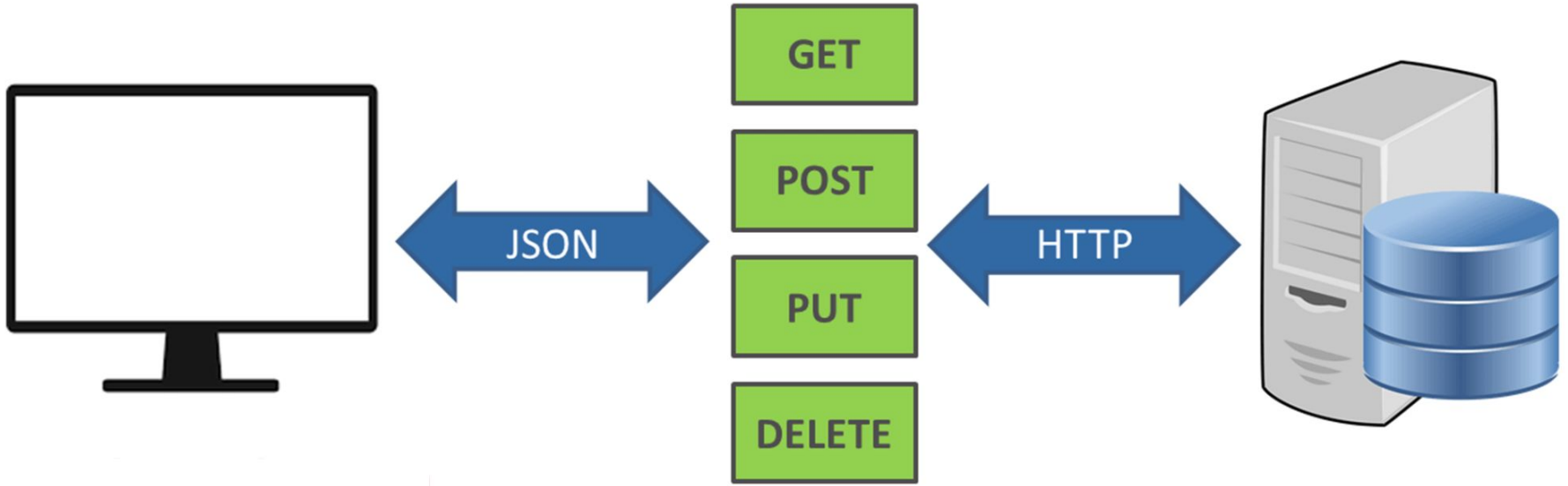
REST

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.

ExpressJS (CRUD)

DEV.F:.
DESARROLLAMOS(PERSONAS);

ExpressJS

- Mayo 2010 (TJ Holowaychuk).
- Es un framework de node para la construcción de aplicaciones web (API Rest y Server Side Render).

Práctica 13

Realizar una copia de la pokeapi en express con las 4 operaciones pero utilizando los 5 verbos http.

Deploys

DEV.F::
DESARROLLAMOS(PERSONAS);

Práctica 14

- Desplegar la API realizada en expressJS en Heroku.