

José Montoya Guzmán

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

<https://github.com/montoyaguzman>



KATA

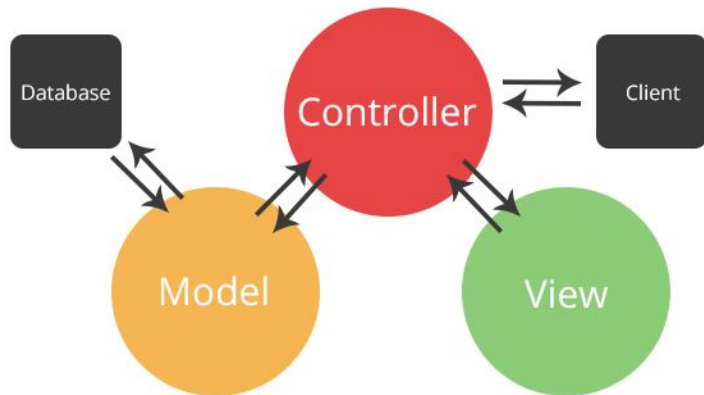
JavaScript Avanzado

DEV.F
DESARROLLAMOS(PERSONAS);

dev

¿Qué veremos en esta 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.



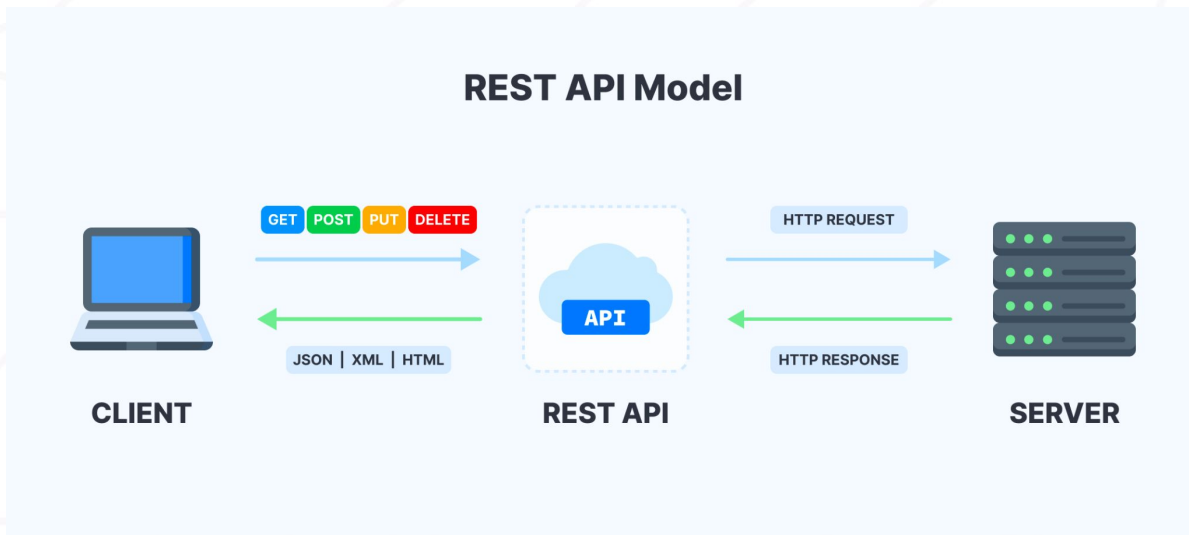
Semana 2

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



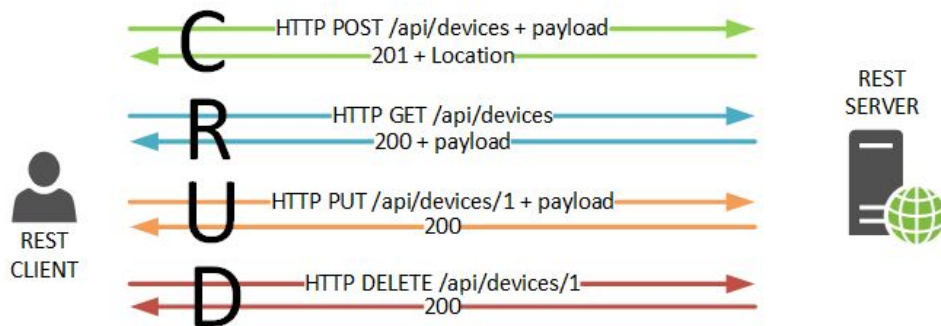
Semana 3

- 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);

dev

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?

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

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.

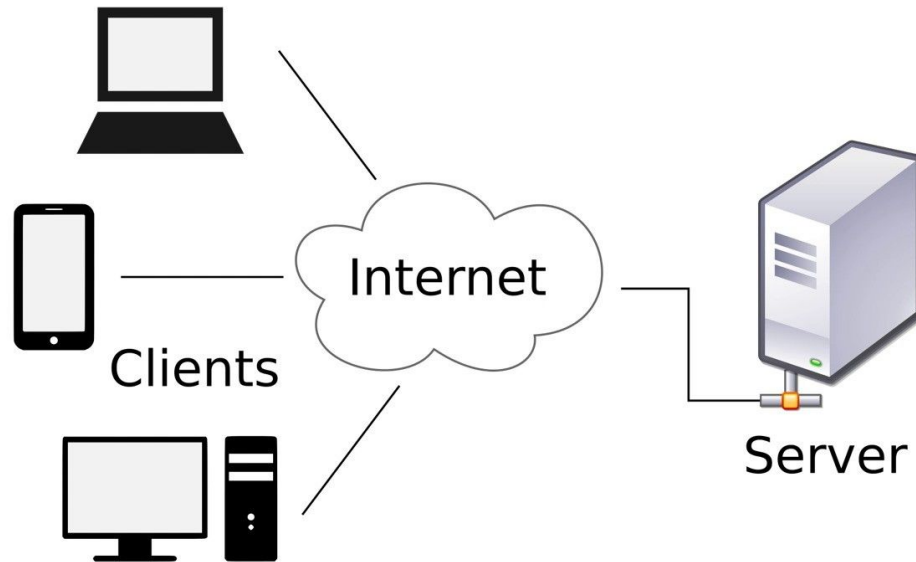


Práctica 3

- 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?queryParams

Ejemplo:

<https://www.tutorialesprogramacionya.com/javascriptya/no-dejsya/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);

dev

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);

dev

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 <package-name>`
- `npm install <package-name>`
- `npm install --save <package-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 esta 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);

dev

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.

1.2.3-beta.1+meta



The diagram illustrates the components of the Semantic Versioning string '1.2.3-beta.1+meta'. It features a horizontal line with five segments. Below each segment is a label: 'Major' under '1', 'Minor' under '2', 'Patch' under '3', 'Pre-release' under 'beta.1', and 'Metadata' under '+meta'. Lines connect each label to its corresponding segment in the version string.

Major Minor Patch Pre-release Metadata

Funcionamiento de semantic versión

Dado un número de versión **MAYOR.MENOR.PARCHÉ**, 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 **PARCHÉ** cuando reparas errores compatibles con versiones anteriores.

MAYOR.MENOR.PARCHÉ = 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);

dev

Arquitecturas más comunes

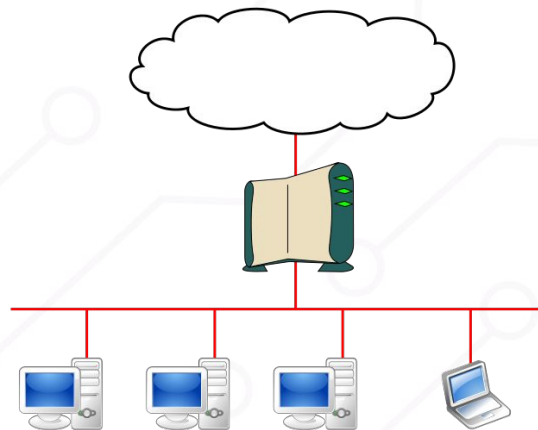
Generalmente, no es necesario inventar una nueva arquitectura de software para cada sistema de información.

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



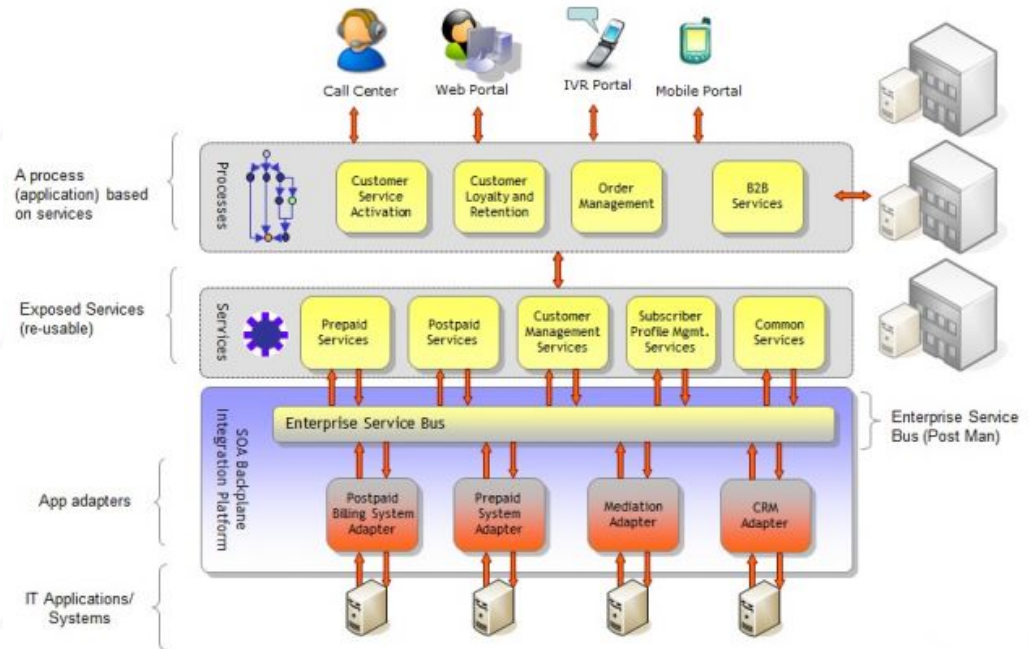
¿Qué es una arquitectura en software?

- 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.

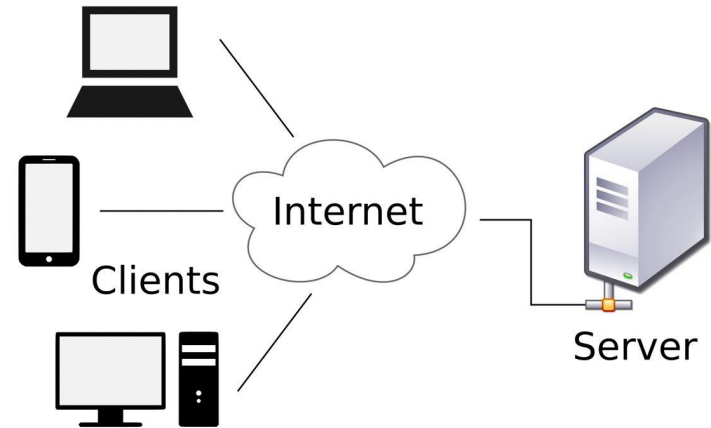


Stacks de desarrollo web

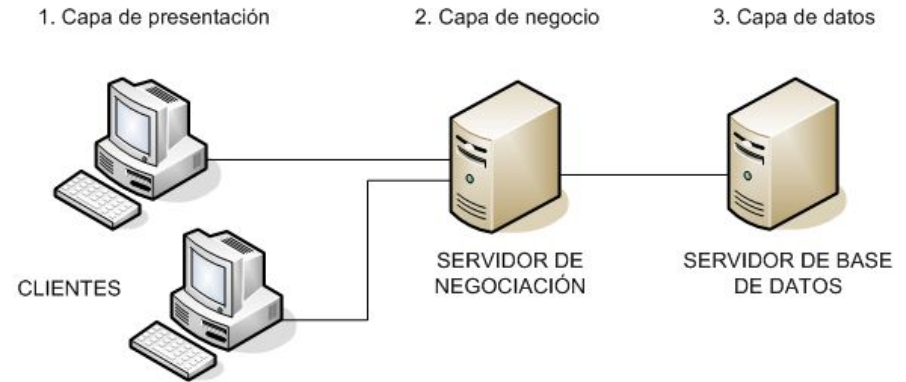
DEV.F
DESARROLLAMOS(PERSONAS);

dev

Cliente Servidor



Arquitectura de Tres Niveles



Arquitectura MVC

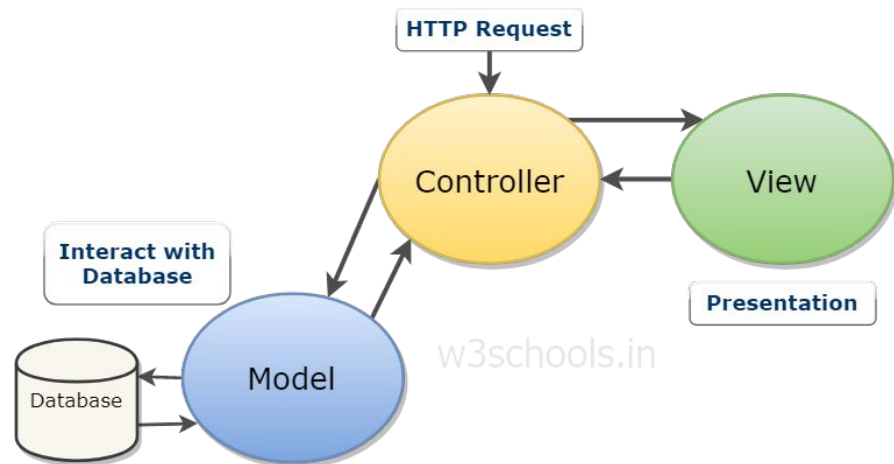


Fig: MVC Architecture

Arquitetura dirigida por eventos

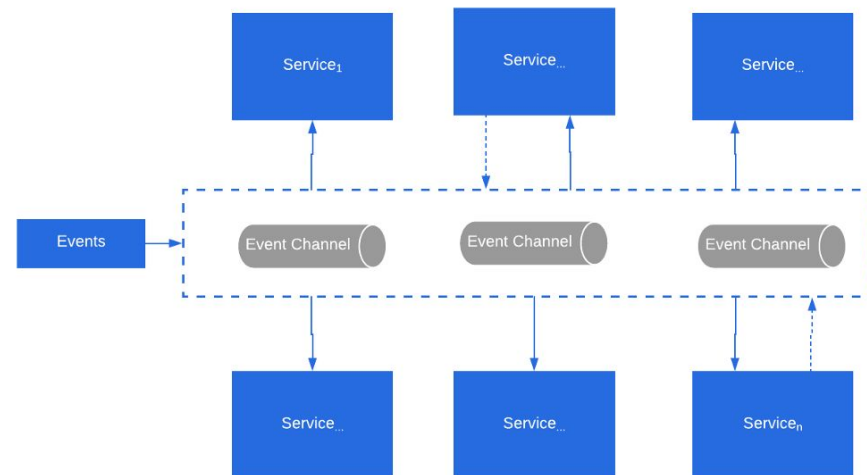
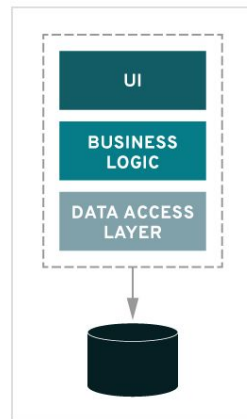


Figure 2 - Broker Topology

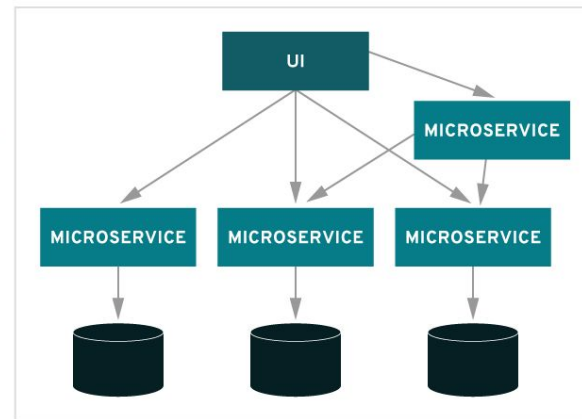
Arquitectura de Micro Servicios

MONOLITHIC



VS.

MICROSERVICES



Serverless

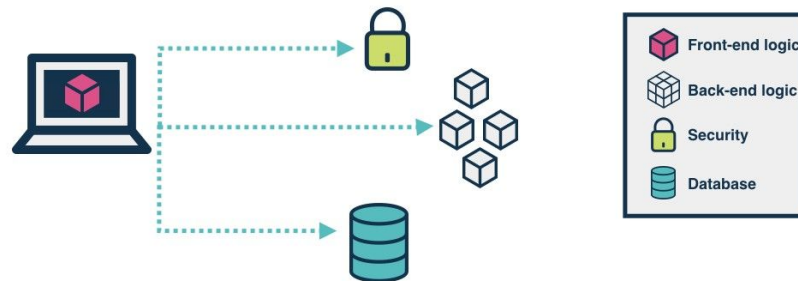
TRADITIONAL vs SERVERLESS

TRADITIONAL



SERVERLESS

(using client-side logic and third-party services)



Stacks de Desarrollo web

DEV.F
DESARROLLAMOS(PERSONAS);

dev

¿Qué es un stack tecnológico?

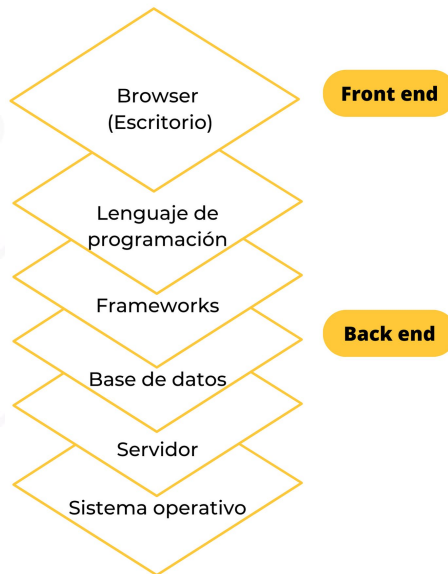
Un stack tecnológico es un **conjunto de servicios de software que se utilizan para el desarrollo de aplicaciones**. Normalmente, un Stack está formada por lenguajes de programación, Frameworks, bibliotecas, herramientas de desarrollo y enfoques de programación.

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:

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



LAMP



ASP.NET



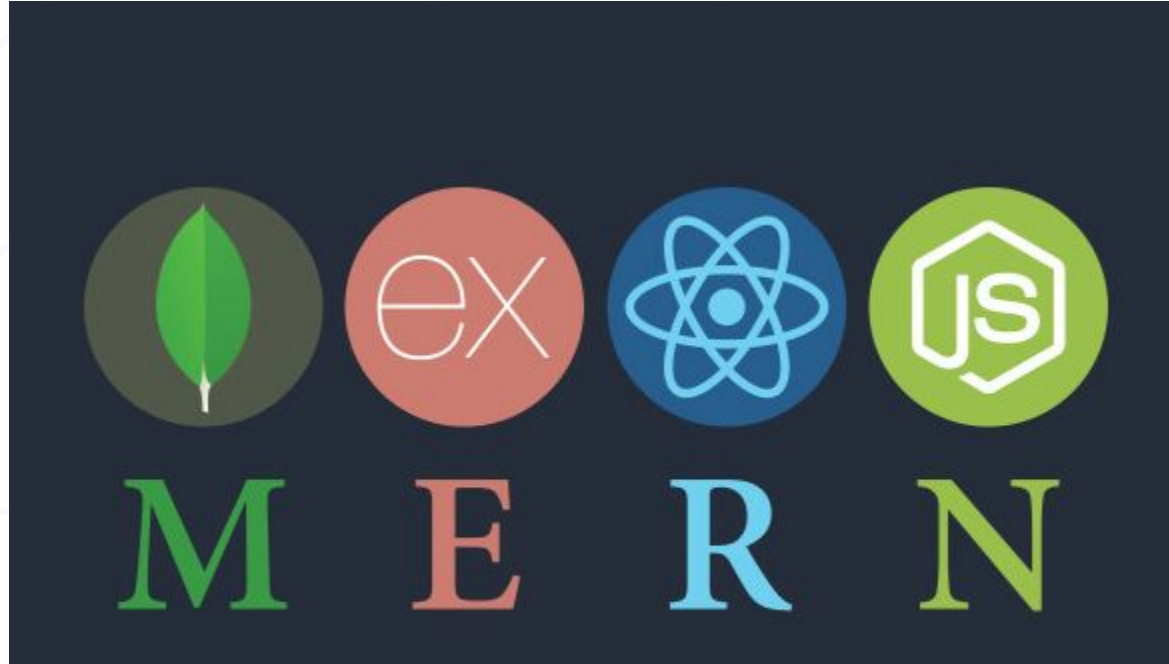
TM

ASP.NET

MEAN



MERN



Event Loop y Asincronía

DEV.F
DESARROLLAMOS(PERSONAS);

dev

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.

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

Comparativa entre Node vs JavaScript

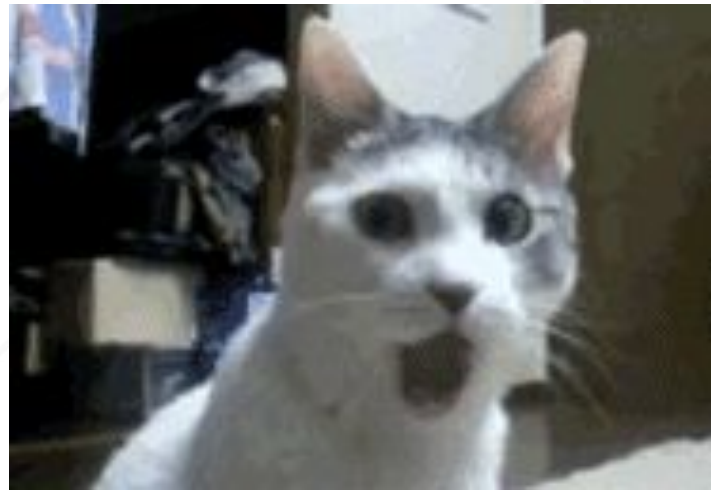
DEV.F
DESARROLLAMOS(PERSONAS);

dev

Notas interesantes

El motor v8 tiene las siguientes características:

- 0 retraso.
- No hay temporizador (setTimeout y setInterval).
- No hay manejo de eventos (addEventListener).
- No hay solicitudes Ajax (consumo de API's).
- Navegadores usan Libevent.
- Node usa Libuv.

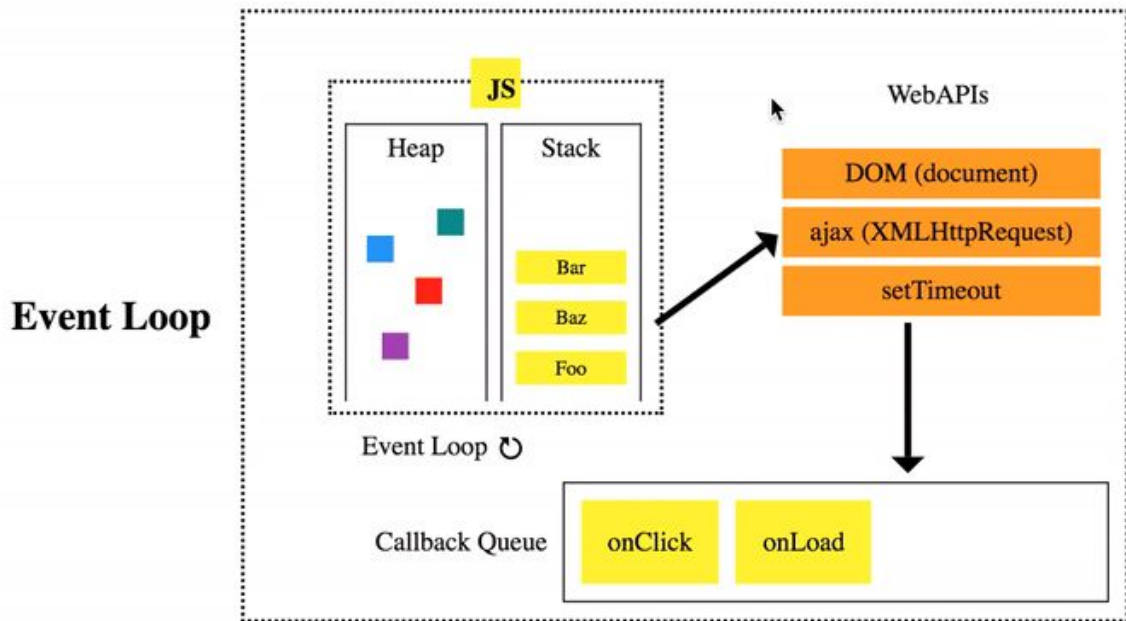


Métodos de manejo de la asincronía

- **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 (ES6):** Función con la sugar syntaxis de es6 y simula que el código es asíncrono (por detrás sigue siendo una promesa).

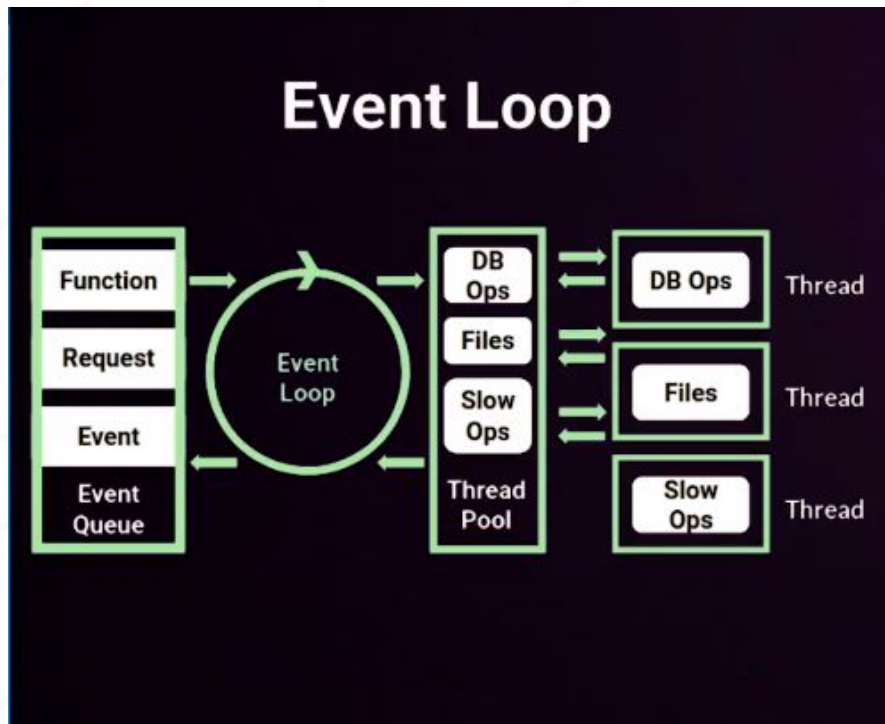
Event Loop JavaScript (libevent)

- Asincronismo y monohilo.



Event Loop Node (libuv)

- Asincronismo y monohilo.



Ejemplo de restaurante para asincronismo



Práctica 12

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





API

DEV.F
DESARROLLAMOS(PERSONAS);

dev

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 tu teléfono “habla” con este sistema a través de la API y le muestra las actualizaciones meteorológicas diarias en pantalla.

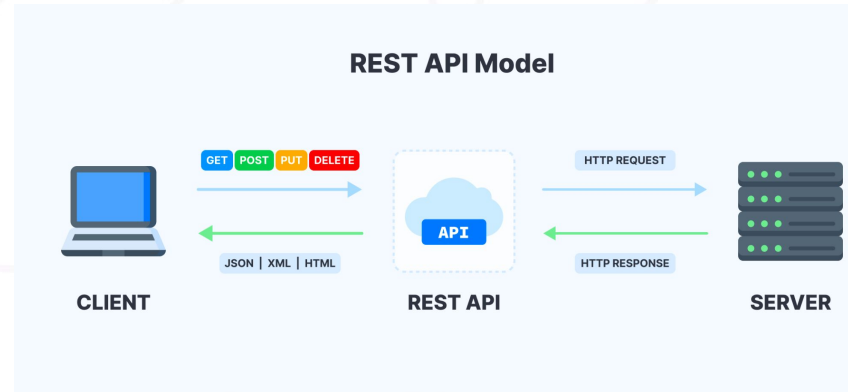
Conceptos

DEV.F
DESARROLLAMOS(PERSONAS);

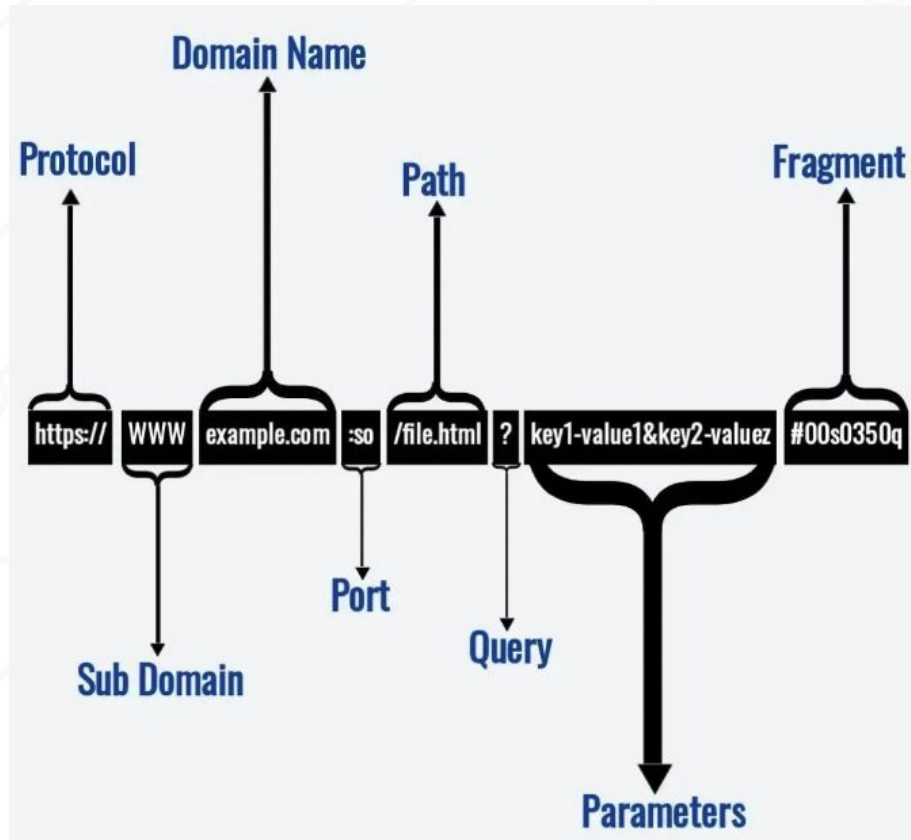
dev

Conceptos de un API

- **Client:** Es quien realiza la petición.
- **Request:** Petición http conformada por url, params, payload.
- **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 teoría, cada URL valida apunta a un único recurso. Dichos recursos pueden ser páginas HTML, documentos CSS, imagenes, etc.

Diferencia entre URI y URL

Quizás un poco lioso, pero para aclararlo y por decirlo de otra manera:


- Un URL es un URI que identifica un recurso y también proporciona los medios para localizar el recurso mediante la descripción de la forma de acceder a él.
- Un URL es un URI.
- Un URI no es necesariamente una URL.
- Un URN sólo define un nombre, no proporciona detalles acerca de cómo obtener el recurso en una red.
- Puedes obtener una «cosa» a través de una URL, usted no puede conseguir cualquier cosa con un URN
- En general, si el URL describe tanto la ubicación y el nombre de un recurso, el término a utilizar es URI.
- Básicamente, un URI es igual a la suma del URL y del URN.

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

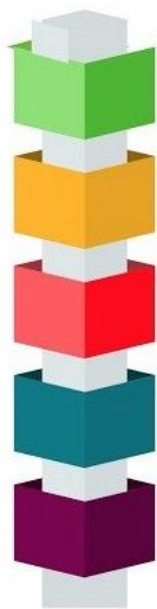
- **TCP/IP** : Transfer control protocol/Internet Protocol.
- **HTTP**: Se encarga de la comunicación entre un servidor web y un navegador web. HTTP se utiliza para enviar las peticiones de un cliente web (navegador) a un servidor web, volviendo contenido web (páginas web) desde el servidor hacia el 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



1XX
INFORMATIONAL

2XX
SUCCESS

3XX
REDIRECTION

4XX
CLIENT ERROR

5XX
SERVER ERROR

Entendiendo conceptos

- Servicio \approx endpoint \approx recurso \approx servicio web.
- API \approx API REST \approx API Restful.
- Request \approx Petición \approx Solicitud.
- Response \approx Respuesta.
- Path \approx Dirección \approx Url \approx URI.
- File \approx Fichero \approx Archivo.
- Terminal = Línea de comandos = CLI = Bash.
- Script \approx Programa \approx Conjunto de líneas de código o instrucciones.



REST

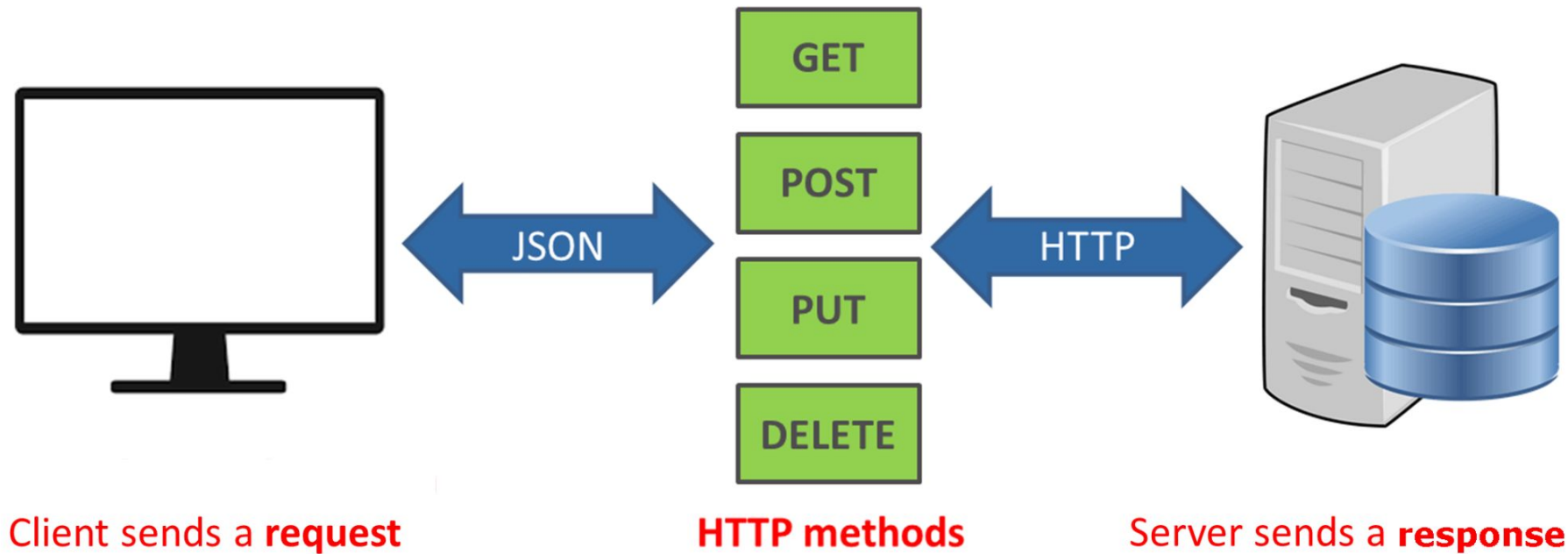
DEV.F
DESARROLLAMOS(PERSONAS);

dev

¿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);

dev

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);

dev

Práctica 14

- Desplegar la API realizada en expressJS en Heroku.

