

# Web Interfaces para aplicaciones de ingeniería

Python + Flask + JavaScript

Pablo Garrido Sánchez



# ¿Qué queremos hacer?

**Monitorizar** y **controlar** sistemas...

...mediante tecnologías **estándares** y **actuales**...

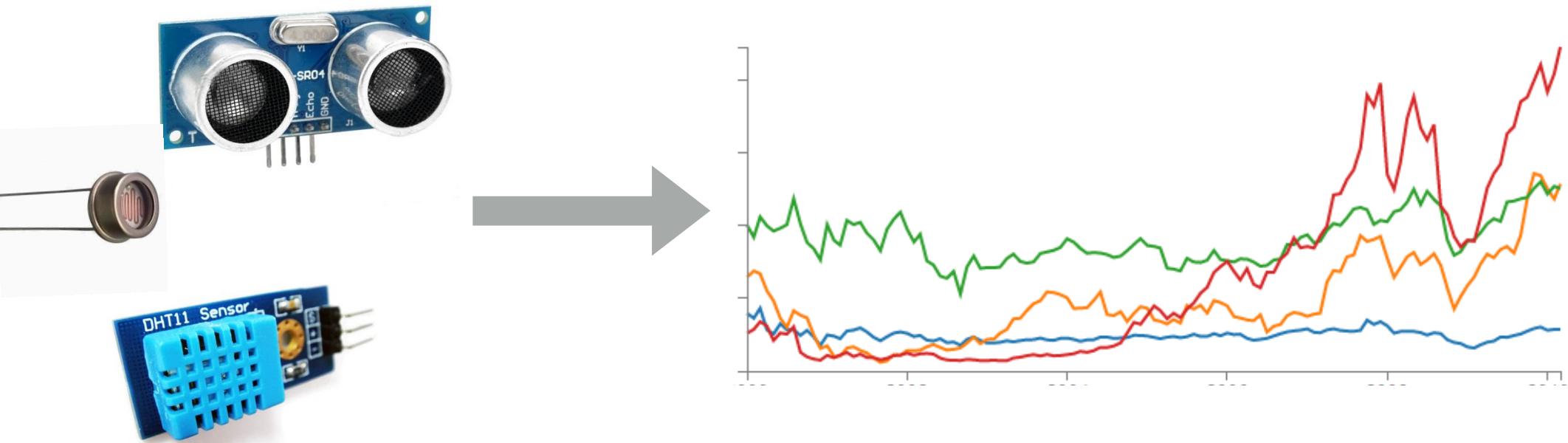
...basadas en paradigmas **web**...

...enfocadas al **desarrollo rápido**.

# Por partes...

## Monitorizar y controlar sistemas...

Es decir, **obtener datos** y **realizar acciones** sobre sistemas normalmente electrónicos. Por ejemplo, leer la información que una placa Arduino conectada a un sensor de temperatura nos puede ofrecer, o actuar sobre unos motores capaces de abrir y cerrar una puerta mecánica.



# Por partes...

... mediante tecnologías **estándares** y **actuales** ...

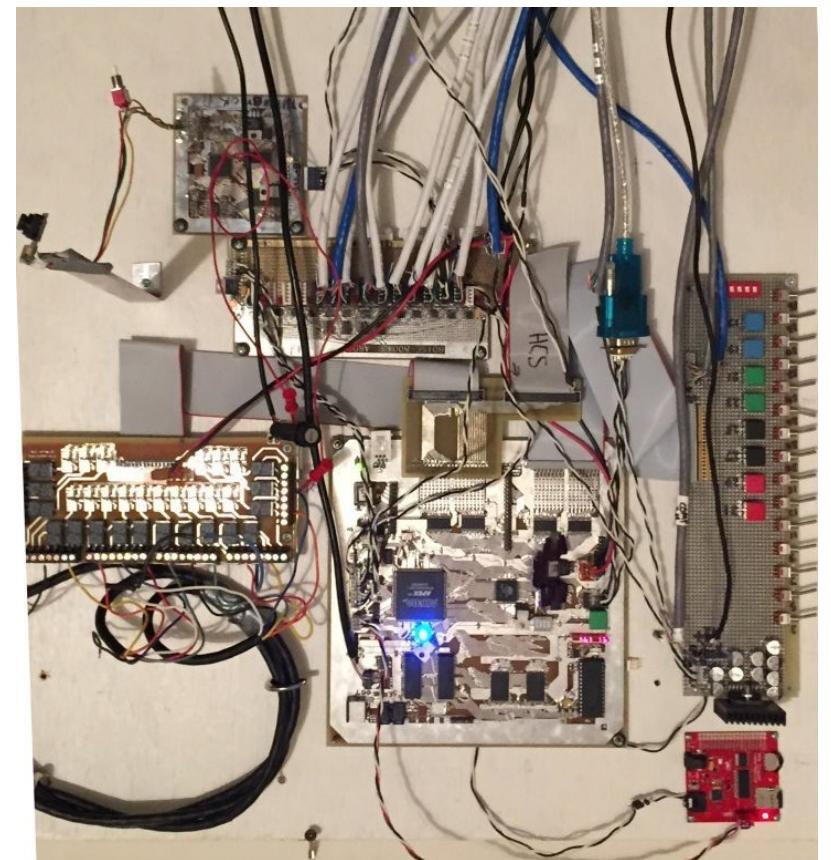
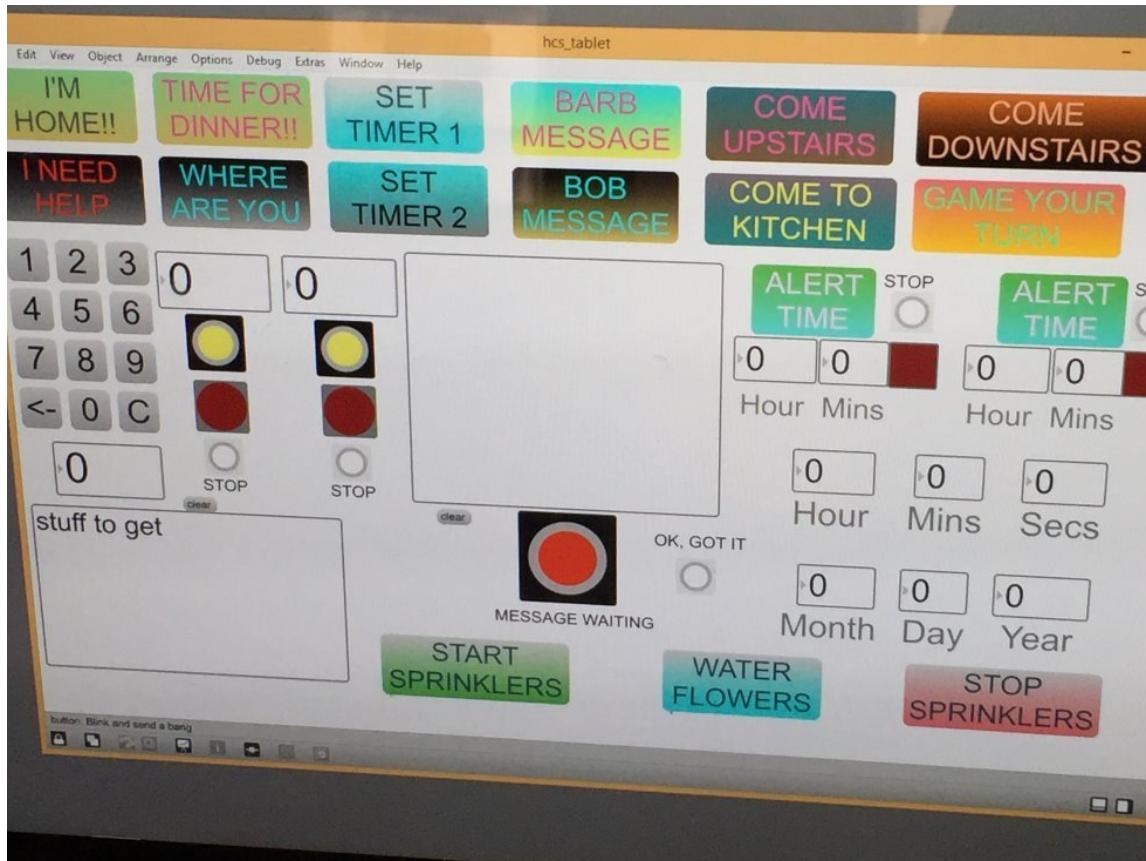
Orientando nuestro proceso de aprendizaje a las **nuevas tecnologías** que van apareciendo y forzando a este proceso de aprendizaje a ser continuo.

Es decir, tratando de estar informados de las soluciones más **actuales** y ser capaces evaluar su campo de aplicación así como sus **pros y contras** respecto a otras ya que conozcamos.

Y aprender a poner en valor los beneficios de los “**estándares de facto**” (estándares aceptados por la comunidad desarrolladora): documentación, comunidad, ejemplos, proyectos existentes...

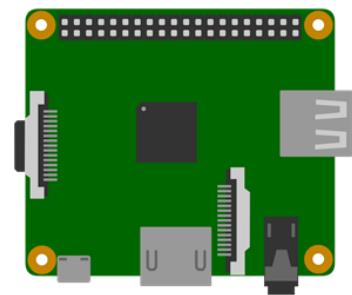
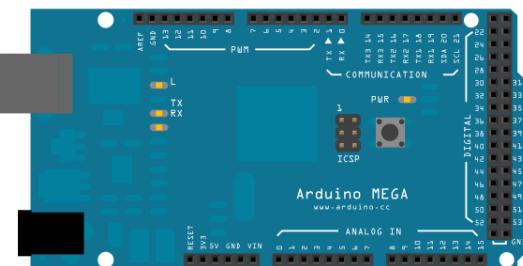
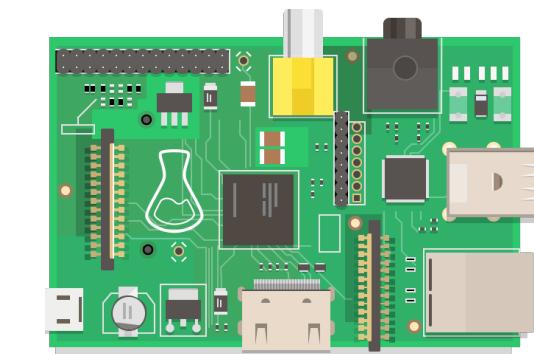
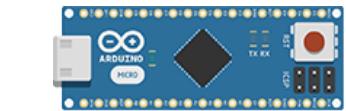
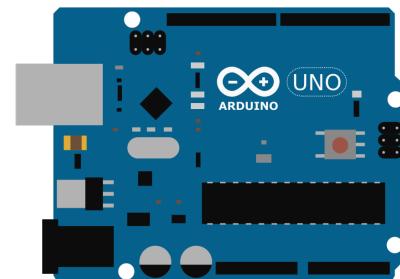
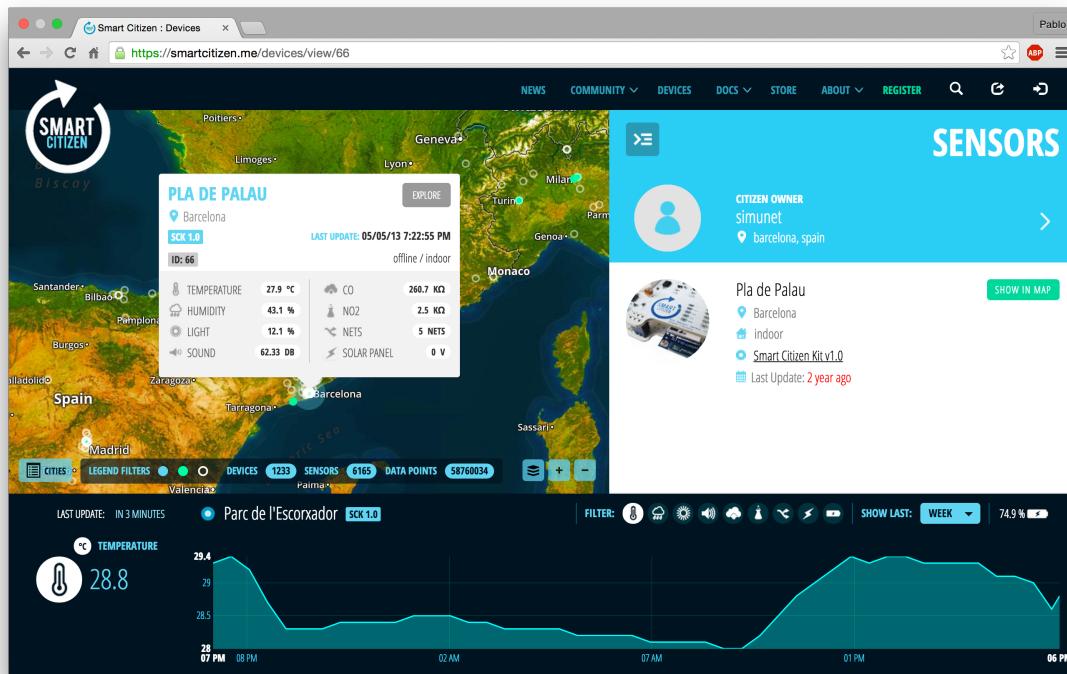
# Por partes...

... no mediante tecnologías **propietarias** y **viejas** ...



# Por partes...

... mediante tecnologías **estándares** y **actuales** ...  
... y a ser posible **abiertas** ...



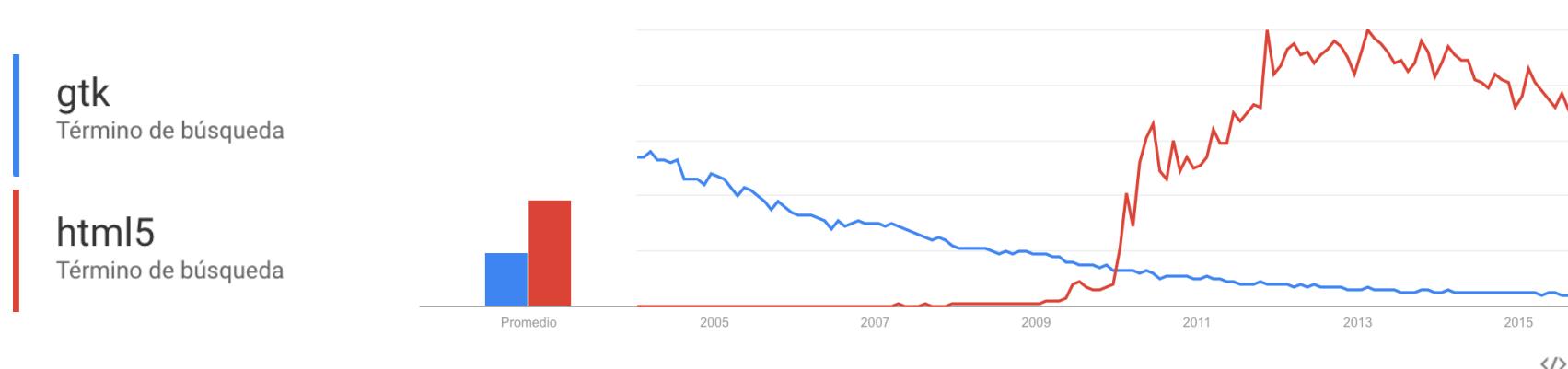
# Por partes...

...basadas en paradigmas **web**...

Orientando nuestros diseños e ideas hacia el **Internet of Things**, no por una cuestión de moda de la industria, si no por una cuestión de utilidad y sencillez.

Internet of Things → Tecnologías Web + Electrónica → Millones de desarrolladores

Millones de desarrolladores → Estándares de facto → Documentación, comunidades, foros, ejemplos, tutoriales, lenguajes de alto nivel muy ricos en funcionalidad, modelos de desarrollo, herramientas abiertas, open software, open hardware... → **Facilidades**



# Por partes...

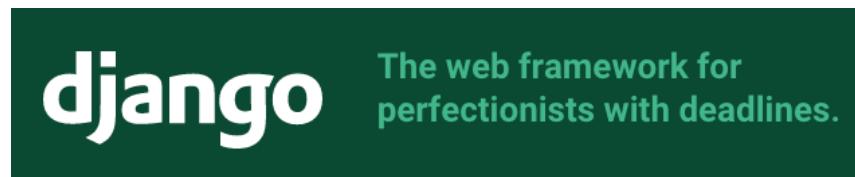
...enfocadas al **desarrollo rápido**...

El desarrollo rápido implica que si la interfaz gráfica no es la parte más importante de nuestro sistema no debería gastar la mayor parte de nuestro tiempo.

Sin embargo, por suerte dispondremos de herramientas cuya filosofía de trabajo nos permite implementar **soluciones muy buenas** en un **tiempo muy razonable**. Véanse los eslóganes de Django y jQuery mas abajo.

La pregunta es: Si no nos dedicamos al desarrollo de interfaces ni a la programación y diseño web ¿por qué debería ocuparme esto demasiado tiempo?

No ocupará demasiado tiempo si sabemos buscar las herramientas correctas.



# Por partes...

... aunque ciertos problemas siempre requerirán mayor esfuerzo.

The module has been programmed in [ANSI C](#) and the database can use [database management system MySQL](#) or [MariaDB](#). The reader will probably think "a web application written in C?" Yes, it is not usual, [but... :-\)](#)

SWAD core has about 185 000 [lines of source code](#). Each executable program (one or each language) has a size of 1.85 [MiB](#) and in most of the [possible actions](#) generates the [page](#) in few milliseconds.

**Technical specifications**

**Server**

SWAD core is a [CGI](#) programmed in [C](#) comprising almost all the functionality of the platform. The core is supplemented with some external programs like photo processing module and chat module. The server runs on a [GNU/Linux](#) system with [Apache](#) and a [MySQL](#) or [MariaDB](#) database.

# Acotando el problema

El problema que vamos a abordar para llevar a cabo nuestro propósito de usar interfaces gráficas web para aplicaciones de ingeniería pasará por:

- **Conocer** las tecnologías que tenemos disponibles,
- **Analizar** las ventajas e inconvenientes de cada una,
- **Seleccionar** las idóneas,
- **Aprender** a utilizarlas en la medida que necesitemos, y
- **Resolver** el problema que tengamos entre manos.

# ¿Y qué aplicaciones de ingeniería?

En general dentro de nuestra aplicación de ingeniería tendremos un sistema que es posible **controlar** y del que es posible **extraer información**. Le vamos a llamar **“Aplicación”**

Esta “Aplicación” va a comunicarse con otro sistema gestor de la información. En nuestro caso concreto gestionar la información va a implicar **leer la información** que produce la “Aplicación”, **almacenarla** y hacerle saber los **comandos o acciones** que tiene que ejecutar. A esta parte la denominaremos **“Servidor”**.

Por último una tercera parte implicada será el **usuario**. O más bien, la **interfaz gráfica** delante la cual se sentará el usuario final a observar el comportamiento de la “Aplicación” y a controlarlo. A esta interfaz gráfica y al usuario final lo englobaremos en el término **“Cliente”**.

# Concreta...

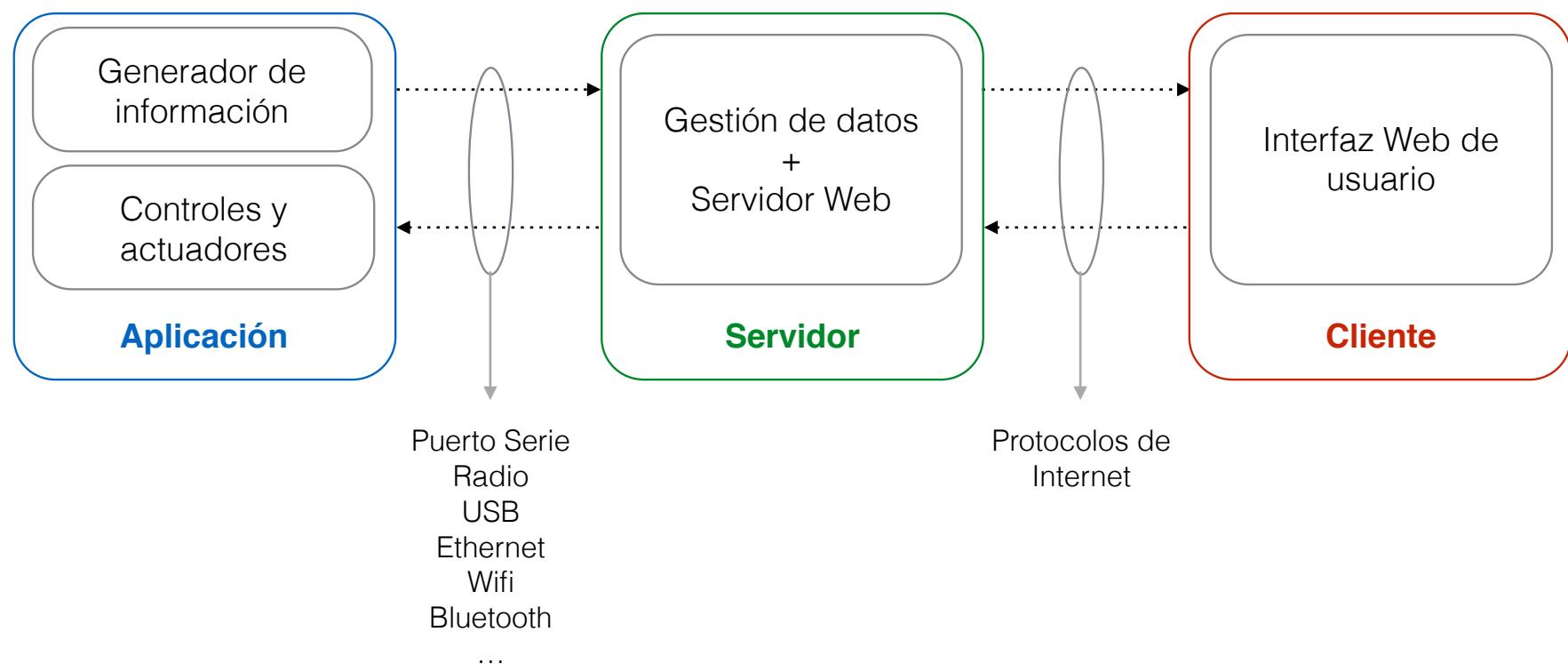
**Aplicación** = Electrónica con sensores, Arduino, placas de prototipo, Raspberry Pi, electrónica custom, dispositivos existentes con módulos de comunicación... (¿Existen? Sí, sonómetros con puerto serie, cámaras de video vigilancia con conexión Ethernet, juguetes con radiocontrol, estaciones meteorológicas baratas con comunicación radio...)

**Servidor** = Un computador que pueda comunicarse con la Aplicación y que se pueda programar. (¿Y podemos entender por computador una Raspberry Pi ? Definitivamente SI).

**Cliente** = Cualquier persona que tenga en sus manos un dispositivo con navegador web y conexión a Internet. (O con conexión a la red donde esté el Servidor).

# Arquitectura básica

De forma esquemática tenemos:



# Arquitectura básica

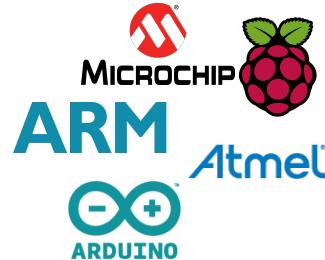


Pero... ¿Qué tecnologías escoger?



# Arquitectura básica

Primero hay que conocerlas, no manejarlas, pero al menos conocerlas



Plataformas hardware



Lenguajes de programación



Frameworks



Formatos estándares



Librerías de JavaScript



Librerías gráficas de JavaScript

## EJEMPLO 1

# Vamos a un caso real...

Para concretar la idea vamos a poner un caso práctico basado en un dispositivo electrónico real que mide mediante sensores una magnitud física concreta.

En este caso nuestra **Aplicación** será un dispositivo electrónico hecho a medida para un propósito específico. Se comunica con el **Servidor** mediante un puerto RS232.

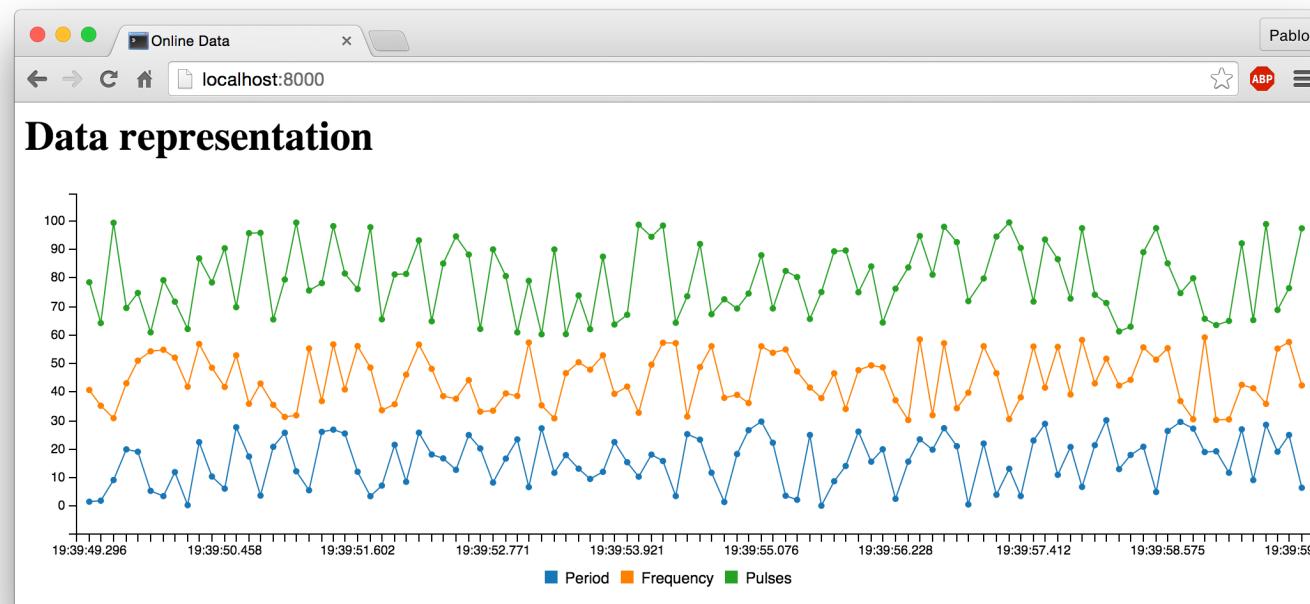
El **Servidor** lo constituirán varias capas dentro de un ordenador convencional. Como lenguaje de programación usaremos Python, como framework web utilizaremos Flash y utilizaremos una base de datos SQLite. La comunicación con el **Cliente** estará basada en el formato JSON y el estándar AJAX.

Por último el **Cliente** ejecutará en un navegador web moderno tecnologías web estándar como HTML, CSS y JavaScript junto con las librerías jQuery y C3.js.

# Vamos a un caso real...

EJEMPLO 1

Al final pretendemos obtener una interfaz web en un navegador accesible desde la red similar a esta.



# Vamos a un caso real...

EJEMPLO 1

Todo el código utilizado está en un repositorio de GitHub.

Es recomendable ir observando cada parte del código junto con las explicaciones de este documento a medida que se estudia este procedimiento.

**<https://github.com/pablogs9/SensorWebInterface>**



# ¿Alternativas?

EJEMPLO 1

**LAMP** = Linux + Apache + MySQL + PHP

**MEAN** = MondoDB + Express.js + Angular.js + Node.js



Puerto Serie



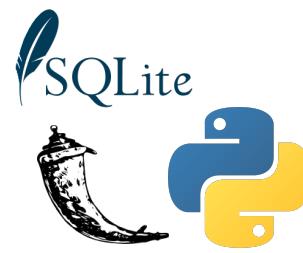
# Organización, por favor.

EJEMPLO 1

Las tecnologías seleccionadas se explicarán a posterior, pero primero observemos donde se ubica cada una dentro de la estructura general que hemos visto antes.



Puerto Serie

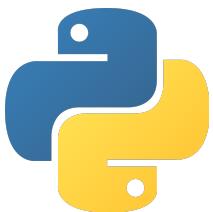


# ¿Qué son estas tecnologías?

EJEMPLO 1



Los Microchip PICs son microcontroladores ampliamente utilizados.



Python es un lenguaje de programación interpretado y orientado a objetos sencillo de utilizar con gran cantidad de extensiones o módulos para realizar de forma fácil muchas tareas que en otros lenguajes son tediosas.



SQLite es un sistema de gestión de bases de datos SQL relacionales sencillo de utilizar e implementar.

# ¿Qué son estas tecnologías?

EJEMPLO 1



Flask es un *framework* de desarrollo web basado en Python. Pretende mantenerse simple y extensible.



JSON es un formato ligero de intercambio de datos. Algo parecido a XML. Originalmente compatible con JavaScript y actualmente compatible con multitud de sistemas.



Paradigma de comunicación web asíncrono. Permite al cliente recuperar información del servidor en cualquier instante (ante cualquier evento) sin recargar la web completa.

# ¿Qué son estas tecnologías?

EJEMPLO 1



HTML 5 es el lenguaje mediante el cual se estructuran las páginas web.



CSS 3 es el lenguaje mediante el cual se provee de diseño a las páginas web.



JavaScript es, entre otras muchas cosas, el único lenguaje que los navegadores web pueden ejecutar. Es capaz, entre otras muchas cosas, de interaccionar con HTML o CSS, así como de implementar comunicaciones AJAX.

# ¿Qué son estas tecnologías?

EJEMPLO 1



jQuery es un librería de JavaScript que facilita enormemente el acceso a las diferentes partes de la estructura de una web, también conocida como DOM.



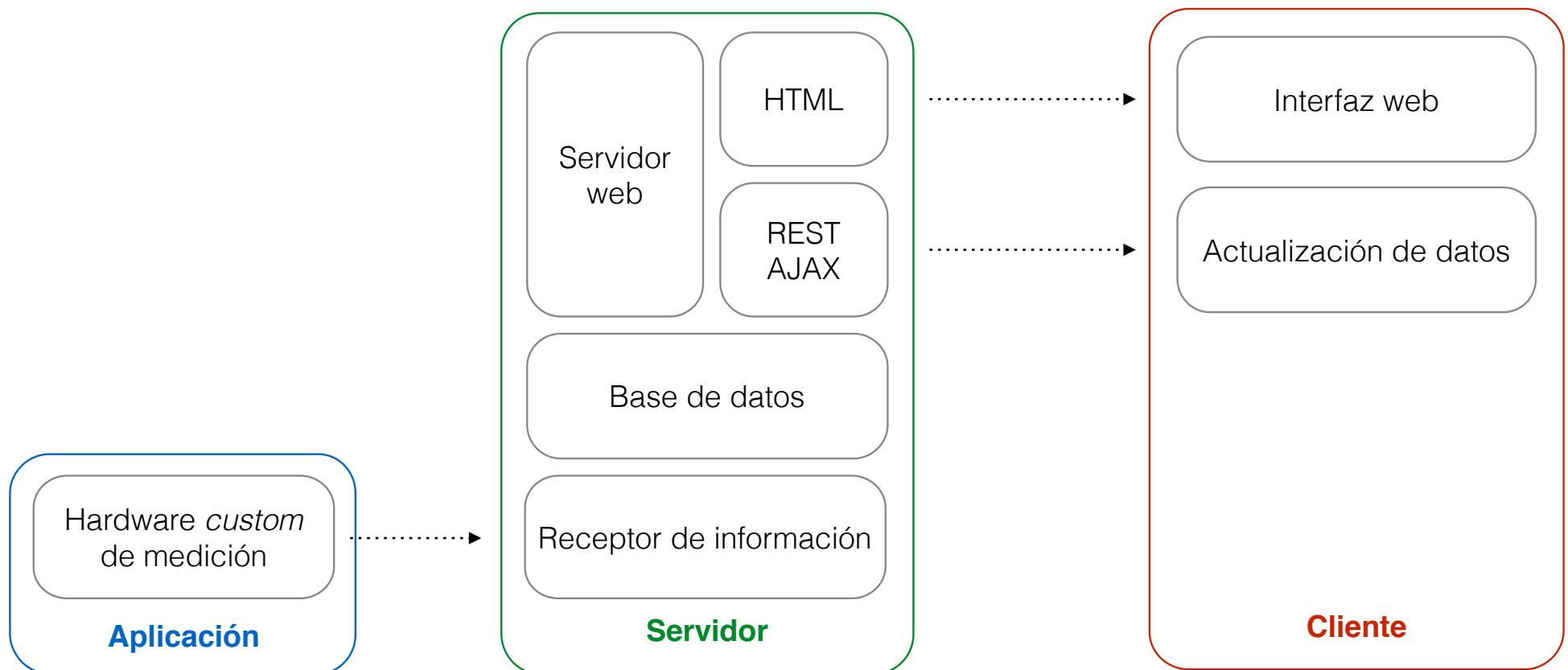
D3.js es un librería de JavaScript que permite realizar gráficos e animaciones de cualquier tipo.

C3.js

C3.js es una extensión de D3.js y librería de JavaScript que permite dibujar gráficos de representación de información: barras, líneas, puntos, histogramas...

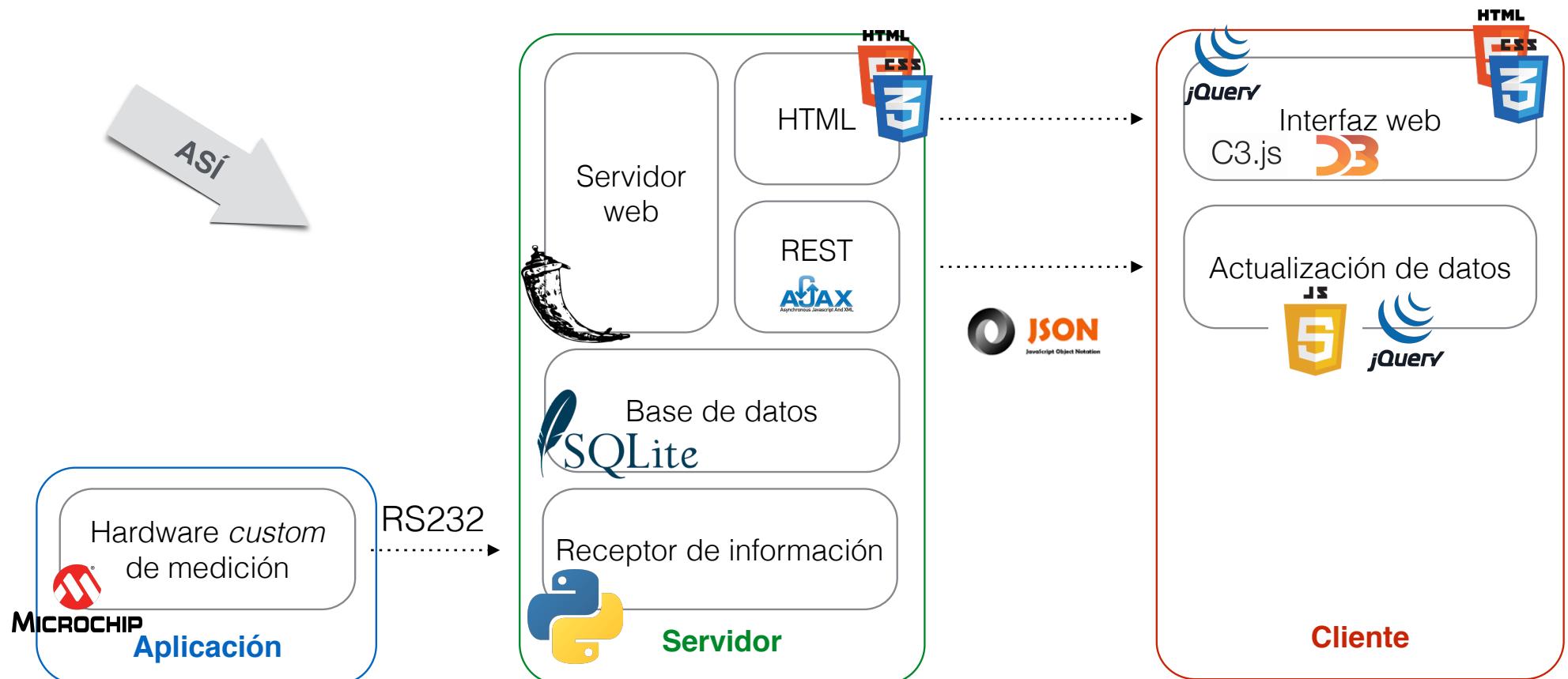
# ¿Y como queda la arquitectura?

EJEMPLO 1



# ¿Y como queda la arquitectura?

EJEMPLO 1



## EJEMPLO 1

# Pues manos a la obra...

En primer lugar debemos diseñar el hardware, seleccionar componentes, revisar el hardware, revisar el hardware otra vez, fabricar un prototipo, comprar los componentes, soldar el prototipo, comprobar que cada parte funciona como debería, comprobar que nada se calienta... Es decir, hacer una **Aplicación**.

Por suerte, para este primer ejemplo ya tenemos esta parte solucionada y disponemos de una electrónica capaz de medir.

En este punto lo único que nos interesa es que se puede conectar a nuestro ordenador, o mejor dicho, nuestro **Servidor** mediante un cable USB o serie y que alguien haya probado que esta comunicación funciona.

# Ten un buen entorno de trabajo

EJEMPLO 1

Antes de comenzar a programar nuestra interfaz web necesitamos un entorno de trabajo cómodo, es decir, un ordenador con un sistema operativo para ejecutar nuestro **Servidor** y un editor de código cómodo.

[Discusiones acerca de sistemas operativos]

[Discusiones acerca de editores de código]

Si no tienes ninguna opción favorita yo utilizo:

- Debian/Ubuntu como sistema operativo
- Atom como editor de código.

# Representación gráfica de datos de sensores

## Ejemplo 1: Python y SQLite

Receptor de información



Base de datos



Partimos de un sistema GNU/Linux clásico como Debian o Ubuntu. Instalamos las dependencias de **Python** y **SQLite** necesarias:

```
sudo apt-get install sqlite3 python python-pip python-dev build-essential
```

PIP es el gestor de paquetes de Python. Mediante él podemos instalar los paquetes que necesitamos para el desarrollo. En primer lugar lo utilizaremos para instalar en el sistema **virtualenv**:

```
sudo pip install virtualenv virtualenv-wrapper
```

Virtualenv es un módulo de Python que nos permite crear un entorno aislado de ejecución. Es decir, en el sistema podremos tener instalados los módulos Python que queramos, pero dentro de un entorno virtual concreto solo estarán instalados los necesarios para la ejecución de dichos programas. Es imprescindible para tener control sobre las dependencias de un software y para no mezclar las dependencias de diferentes proyectos.

# Representación gráfica de datos de sensores

## Ejemplo 1: Python y SQLite

Receptor de información



Base de datos



Enlaces:

[initServer.sh](#)

[requeriments.txt](#)

Una buena metodología de trabajo es mantener ciertos *scripts* que realicen tareas rutinarias, por ejemplo, la instalación del sistema.

A continuación se muestra el *script* `initServer.sh`, el cual se ejecuta, una vez dentro de la carpeta del proyecto con:

`. initServer.sh`

En el se crea (o actualiza en caso de existir) el entorno virtual y se instalan las dependencias necesarias. Estas dependencias están en el fichero `requeriments.txt`.

Una buena práctica de desarrollo es ir actualizando el fichero `requeriments.txt` mediante el comando:

`pip freeze > requeriments.txt`

A medida que vayamos instalando manualmente paquetes que nos sean necesarios mediante:

`pip install [nombre del paquete]`

# Representación gráfica de datos de sensores

## Ejemplo 1: Python y SQLite



Receptor de información



Base de datos

### Enlaces:

[startServer.sh](#)

[database/schema.sql](#)

[serialService.py](#)

[webService.py](#)

Otro de los *scripts* que utilizaremos será **startServer.sh**, mediante el cual iniciaremos el servidor con sus diferentes funciones y la base de datos.

Se ejecuta mediante:

`. startServer.sh`

En el podemos observar como se genera un base de datos nueva mediante comandos de SQLite. Esta base de datos responde a un esquema, el fichero **database/schema.sql** muestra como se crea una base de datos con una tabla y varias columnas de datos, identificador único y marca de tiempo.

A continuación **startServer.sh** ejecuta dos procesos independientes, uno para leer y almacenar la información del puerto serie (**serialService.py**) y otro para servir los contenidos web (**webService.py**).

La forma en la que **serialService.py** recupera y almacena la información está documentada en los comentarios del código.

# Representación gráfica de datos de sensores

## Ejemplo 1: Flask



Servidor  
web

### Enlaces:

[webService.py](#)  
[templates/index.html](#)

Si analizamos el código del servidor en **webService.py** observamos que utiliza el módulo Flask para levantar un servicio web en el puerto 8000 y únicamente responde a dos URLs.

Para acceder desde la maquina local, estas URLs son:

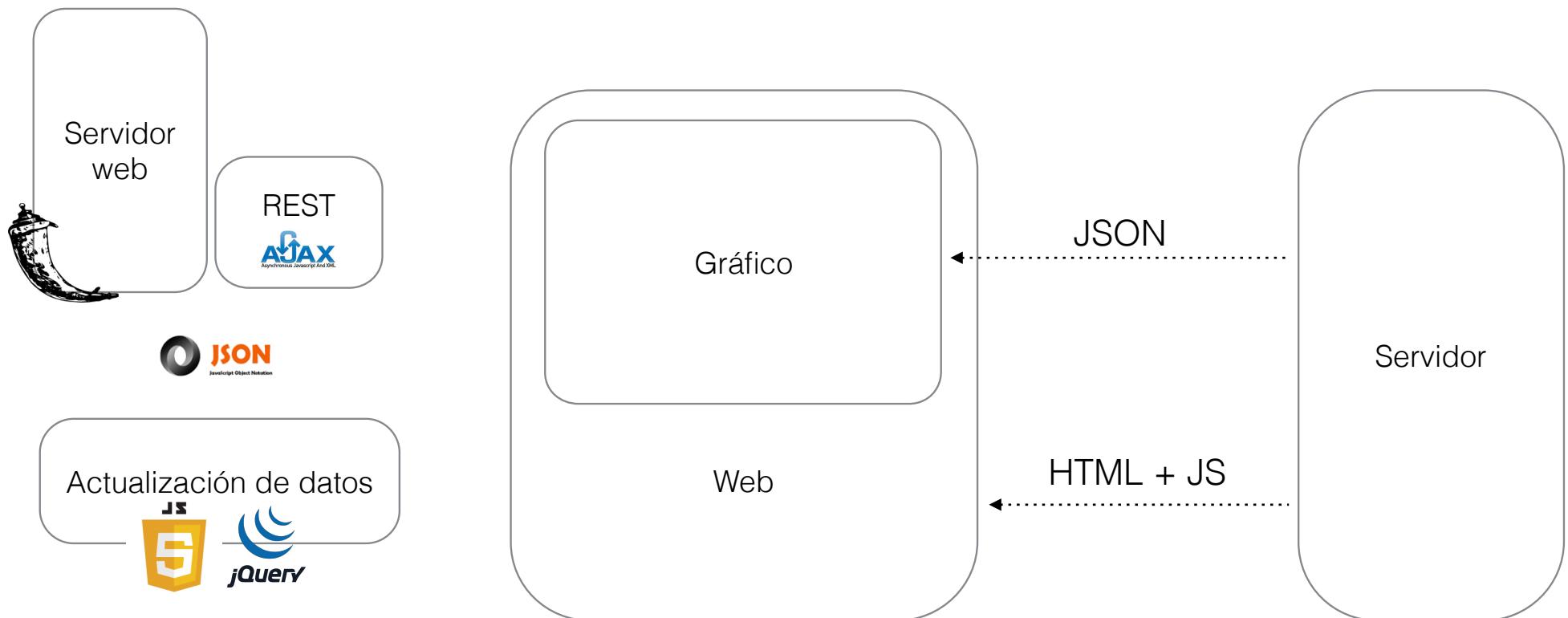
<http://localhost:8000/> → Obtienes código HTML y Javascript de **templates/index.html**

<http://localhost:8000/chartData/20> → Obtienes las últimas 20 entradas de la base de datos en formato JSON

A continuación debemos implementar en el cliente mediante HTML, CSS y JavaScript una arquitectura AJAX que permita una vez cargada la información de **templates/index.html** únicamente ir actualizando los datos de la gráfica mediante llamadas a la interfaz REST de /chartData

# Representación gráfica de datos de sensores

## Ejemplo 1: Implementación AJAX



# Representación gráfica de datos de sensores

## Ejemplo 1: Implementación AJAX



En este punto tenemos que analizar qué le envia el servidor al cliente.

Por una parte, si el cliente llama a la URL <http://localhost:8000/chartData/20> obtiene información de los últimos 20 datos en formato JSON.

Sin embargo, cuando requiere la URL <http://localhost:8000/> es cuando obtiene el fichero **templates/index.html** que le indica:

- Como representar la información: HTML
- Y, como actualizar la información de la gráfica: JavaScript

En el <head> cargamos las librerías JS necesarias: jQuery, C3 y D3. Estas librerías se encuentran almacenadas en la carpeta **static/** que es lugar donde Flask almacena por defecto los ficheros estáticos a servir.

En el <body> rellenamos la web, creando un <div> donde se cargará la gráfica.

### Enlaces:

[templates/index.html](#)

[static/](#)

# Representación gráfica de datos de sensores

## Ejemplo 1: Utilización de C3.js



**Enlaces:**  
[templates/index.html](#)

Por último, en el <script> final del documento definimos en JavaScript la funcionalidad:

- Genera un Chart de la libreria C3.js con las características indicadas.
- Y, planifica una tarea mediante la función setInterval( function, interval) que ejecute el comando GET de jQuery hacia nuestra interfaz REST y actualice los datos.