¿Qué significa API?

La interfaz de programación de aplicaciones, del inglés, *application programming interface*, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. ¿Para qué capas de abstracción? Simple, cuando podemos abstraer de toda una visión de la realidad un modelo que aplique a próximos cambios o varianzas; tenemos un modelo sólido que se ajusta a la realidad.

¿Qué es Node?

Definición de libro: Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome.

Pero… ¿Qué es?:
Para esto tenemos que pensar un poco más en cómo funciona la herramienta más usada hoy en día los navegadores. Si sabemos un poco podemos fácilmente decir que existe un lenguaje de maquetado HTML el cual sirve para describir interfaces de usuario, que existe un lenguaje que permite facilitar el embellecer estas interfaces de usuario que se llama CSS y finalmente existe un lenguaje que permite describir qué cómo y dónde van a pasar las cosas, javascript.
Pero… ¿Podemos ir más allá? Sí. Podemos entonces decir que el HTML existe dentro del DOM (Document Object Model), podemos decir que todas las interacciones de nuestro HTML terminan por ser cambios en el DOM. Pero… ¿Cuál es la capa que le sigue por debajo a nuestro código javascript? Si hacemos algo de memoria podremos ver que todo código termina en algún momento por ser un binario. ¿Cual es el mecanismo que se usa? La compilación. Me van a decir ah pero yo leí que javascript es un lenguaje no compilado; y si, así es, javascript es interpretado, sin embargo el programa que interpreta ese código es compilado puntualmente el usado por Google es código C++ compilado y se llama V8.
Bien y ahora ¿Qué pasaría si ese código V8 lo sacamos del contexto de un navegador? Esa fué justamente la pregunta que se hizo Ryan Dahl. Y gracias a esa pregunta hoy tenemos Node.js, ¿La respuesta cúal es? Al poder escribir código js que exista fuera del contexto del front-end, nos habilita a escribir del lado del servidor, a escribir js que corra en aplicaciones de escritorio, código que se pueda traducir a código nativo de Android o IOS.
Y ahora que entendemos que es Node.js eso ¿A dónde nos lleva? Devuelta a la definición de libro… Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome.

Genial, pero nosotros somos programadores, **¿Qué onda con el código?**
Primero lo primero tendremos que descargar Node.js, lo podemos hacer desde su página oficial. Una vez descargado e instalado podremos empezar a jugar un poco…

Podemos escribir el comando node en una consola y nos abrirá un intérprete de js.

```
C:\Users\eclur>node
> 2 * 3
6
> console.log("Hello")
Hello
undefined
> Math.pow(2, 20)
1048576
>
```

Y ¿Cómo hacemos para que Node.js ejecute un archivo .js? Fácil, basta con crear el archivo añadir algo de código y ejecutar dentro de la ruta del archivo el comando:
node *<nombre_del_archivo>*

```
main.js ▸ ...
1  function derivative_(f, x) {
2      const h = 0.00001
3      return (f(x + h) - f(x)) / h
4  }
5
6  function derivative(f) {
7      return x => derivative_(f, x)
8  }
9
10 const f  = x => Math.pow(x, 2)
11 const df = derivative(f)
12
13 for(let x = -5.0; x <= 5.0; x += 0.01)
14     console.log(
15         `f(${x}) = ${f(x).toPrecision(3)}\
16         f'(${x}) = ${df(x).toPrecision(3)}`)
```

Ejecutamos node main.js y entonces…

```
f(4.769999999999881) = 22.8        f'(4.769999999999881) = 9.54
f(4.77999999999988) = 22.8        f'(4.77999999999988) = 9.56
f(4.78999999999988) = 22.9        f'(4.78999999999988) = 9.58
f(4.79999999999988) = 23.0        f'(4.79999999999988) = 9.60
f(4.80999999999988) = 23.1        f'(4.80999999999988) = 9.62
f(4.8199999999998795) = 23.2        f'(4.8199999999998795) = 9.64
f(4.829999999999879) = 23.3        f'(4.829999999999879) = 9.66
f(4.839999999999879) = 23.4        f'(4.839999999999879) = 9.68
f(4.849999999999879) = 23.5        f'(4.849999999999879) = 9.70
f(4.859999999999879) = 23.6        f'(4.859999999999879) = 9.72
f(4.869999999999878) = 23.7        f'(4.869999999999878) = 9.74
f(4.879999999999878) = 23.8        f'(4.879999999999878) = 9.76
f(4.889999999999878) = 23.9        f'(4.889999999999878) = 9.78
f(4.899999999999878) = 24.0        f'(4.899999999999878) = 9.80
f(4.909999999999878) = 24.1        f'(4.909999999999878) = 9.82
f(4.919999999999877) = 24.2        f'(4.919999999999877) = 9.84
f(4.929999999999877) = 24.3        f'(4.929999999999877) = 9.86
f(4.939999999999877) = 24.4        f'(4.939999999999877) = 9.88
f(4.949999999999877) = 24.5        f'(4.949999999999877) = 9.90
f(4.9599999999998765) = 24.6        f'(4.9599999999998765) = 9.92
f(4.969999999999876) = 24.7        f'(4.969999999999876) = 9.94
f(4.979999999999876) = 24.8        f'(4.979999999999876) = 9.96
f(4.989999999999876) = 24.9        f'(4.989999999999876) = 9.98
f(4.999999999999876) = 25.0        f'(4.999999999999876) = 10.0
```

Boom! Nuestro código corre en consola.

¿Node nos sirve a la hora de escribir una API?

Spoiler: **Si**

Pero vamos empezar por el principio **¿cómo escribimos nuestro primer server?**
Para el ejemplo vamos a hacer uso del módulo http.

server.js ▸ ...

```
1  const http = require('http')
2
3  const server = http.createServer(
4      (req, res) => {
5          res.write('<h1>Hello Node.js with http</h1>')
6          res.end()
7      }
8  ).listen(80)
```

¿Es eso sólo? **Si**, con estas pocas líneas basta para poder tener un servidor
Este código lo que hace es que cuando se nos haga get al localhost:80 de cómo respuesta un Hello World y finalice.

Ahora si lo ejecutamos con node, nos vamos a un navegador y escribimos localhost vamos a ver la respuesta de nuestro servidor:

Si bien tenemos la opción de utilizar http, para proyectos un poco más extensos que un hola mundo tiende a tener mucho código tedioso o repetitivo, para esto es que existen librerías cómo express que son una capa sobre http que son muy útiles para nosotros.

¿Cómo instalamos una librería en Node?

Por suerte contamos con una herramienta que viene junto a Node llamada npm, esta vendría a ser el equivalente de lo que en C# es NuGet, es un gestor de paquetes. ¿Cómo lo usamos? Fácil solo tenemos que ir a una consola y escribir **npm install** <*nombre_del_paquete*>, para instalar express sólo tendremos que escribir dentro del directorio de nuestro proyecto los siguientes comandos:

>npm init --yes
Este comando lo usaremos para que npm genere un proyecto con los parámetros predeterminados. Ahora si prestamos atención veremos que se nos genero un archivo package.json este es el que guardará todas los nombres de las dependencias de nuestro proyecto.

>npm install express
De esta forma podemos usar express. Una vez instalado si vemos dentro de package.json en dependencias esta express:

package.json ▸ ...

```json
{
  "name": "dev",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

Esto nos informa de que express se instaló correctamente, en adición a esto encontramos que se nos generó una carpeta node_modules y un archivo package-lock.json. La carpeta contiene todos los módulos de node y los que hayamos instalado, el archivo .json contiene más información sobre los paquetes.

Ahora si, vamos a hacer nuestro hola mundo en express:

server.js ▸ ...

```js
const app = require('express')()
app.get('/', (req, res) => res.send('<h1>Hello Node.js with Express</h1>'))
app.listen(80)
```

Con estas tres líneas ya tenemos nuestro servidor:

Cómo podemos construir nuestra primera API, bien si prestamos atención al ejemplo vimos que existe un método get, este método lo que permite es que cuando se haga get a la url (su primer parámetro) se ejecute esa función. Pero así como existe el método get existen post, put delete, etc. en nuestra sencilla API vamos a describir el funcionamiento de una única ruta (/api) con dos métodos distintos get y post, con get obtendremos el json de una “base de datos” que es simplemente un array de objetos y con el método post meteremos objetos a este array.

server.js ...

```
1  const express = require('express')
2  const app = express()
3
4  app.listen(80)
5
6  const db = [{}]
7
8  app.use(express.json())
9  app.use(express.urlencoded({ extended: false }))
10
11 app.get('/api', (req, res) => {
12     res.json(db)
13 })
14
15 app.post('/api', (req, res) => {
16     const { name, dni, vote } = req.body
17     console.log(req.body)
18     if(!name || !dni || !vote) res.status(400).send('Invalid request\'s body')
19     db.push({ name, dni, vote })
20     res.send('All ok')
21 })
```

Con estas sencillas 21 líneas habremos conseguido que nuestra API funcione, lo más destacable o lo nuevo a ver son los métodos use, express.json, express.urlencoded, res.status y res.json.

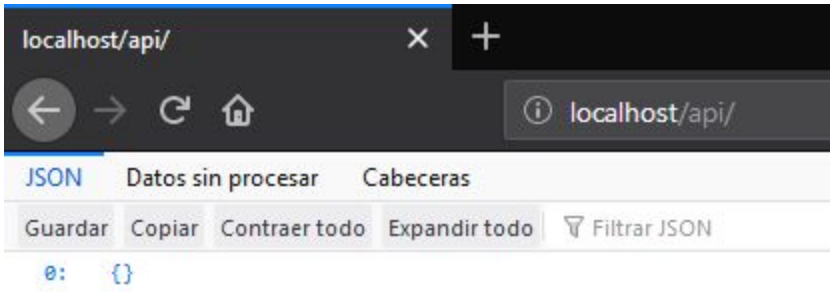
app.use: Con use podemos hacer varias cosas en este caso lo usamos para permitir el uso de **middlewares** (**express.json** y **express.urlencoded**) que en este caso permiten interpretar json.

res.json: Envía una respuesta json.

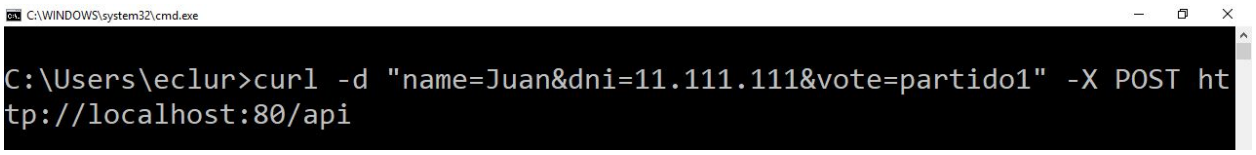
res.status: Podemos indicar el estado de la respuesta, por defecto 200, pero en nuestro ejemplo pusimos que si no se nos indican esos tres parámetros en el cuerpo del request sea 400 (error).

Si corremos nuestro servidor con node server.js, nuestra API ya estará lista para funcionar.

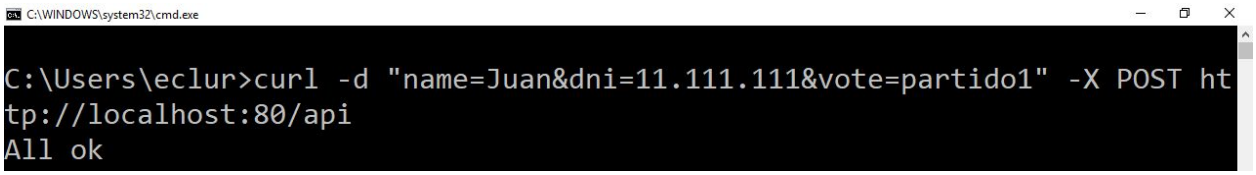
Ahora nos vamos al navegador y escribimos la ruta de la api:



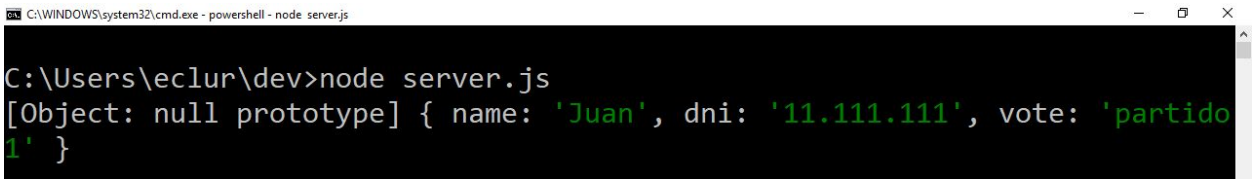
Cómo podemos ver nos da un array de objetos vacío, para meter datos necesitamos usar el método POST.
Podemos ahora chequear cómo funciona con la herramienta curl que viene por defecto instalada en windows 10, esta nos facilitara la tarea de hacer POST.



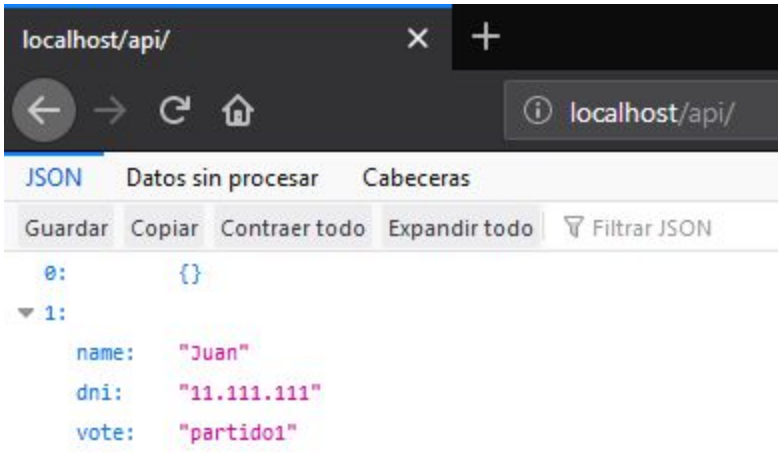
Si todo salió bien nos dará la siguiente respuesta:



“All ok”, que es el mensaje que le dijimos que ponga cuando todo sale bien con code status 200.
Y en la consola de nuestro servidor veremos lo que recibió:



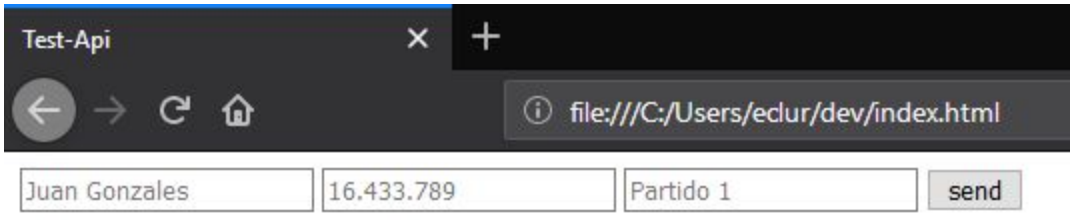
Ahora si volvemos a hacer get a la ruta de la api con un buscador veremos que Juan ha sido agregado al array:

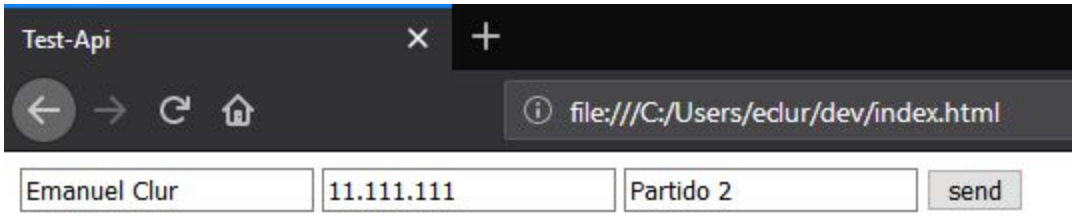


Bien y ahora **¿la podemos "Integrar" a una página web por ejemplo?** Si, con el siguiente ejemplo bien simple lo vas a ver:

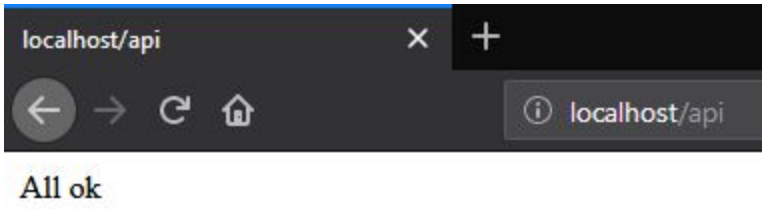
```
index.html ▸ ...
1  <html lang="en">
2  <head>
3      <meta charset="UTF-8">
4      <meta name="viewport" content="width=device-width, initial-scale=1.0">
5      <meta http-equiv="X-UA-Compatible" content="ie=edge">
6      <title>Test-Api</title>
7  </head>
8  <body>
9      <form action="http:localhost/api" method="POST">
10         <input type="text" name="name" placeholder="Juan Gonzales">
11         <input type="text" name="dni" placeholder="16.433.789">
12         <input type="text" name="vote" placeholder="Partido 1">
13         <input type="submit" value="send">
14     </form>
15 </body>
16 </html>
```

Creamos un formulario simple con unos input text con los mismos nombres que indicamos recibía el cuerpo del request y lo fundamental le decimos que el método sea POST y que la acción sea la ruta de nuestra API. Una vez más si todo salió bien veremos algo así:

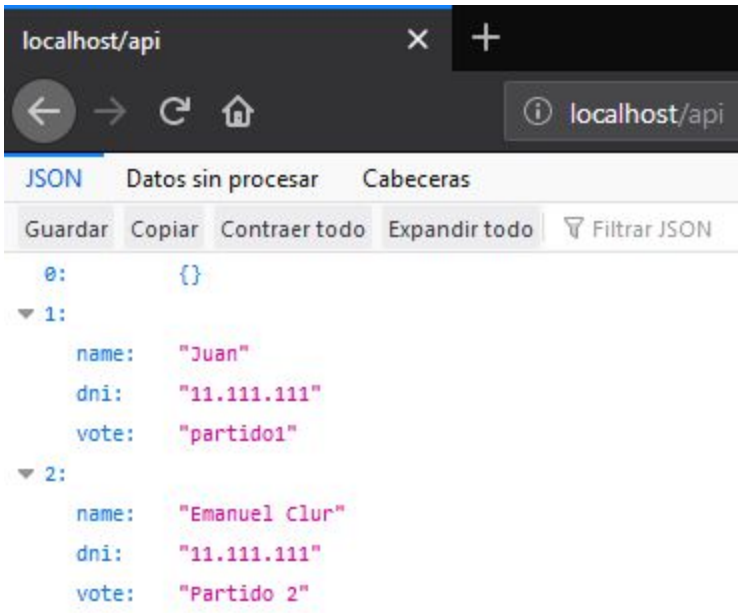




Apretamos send y entonces…



Boom nuestro maravilloso “All ok”.
Y en la ruta de nuestra API mediante el método GET:



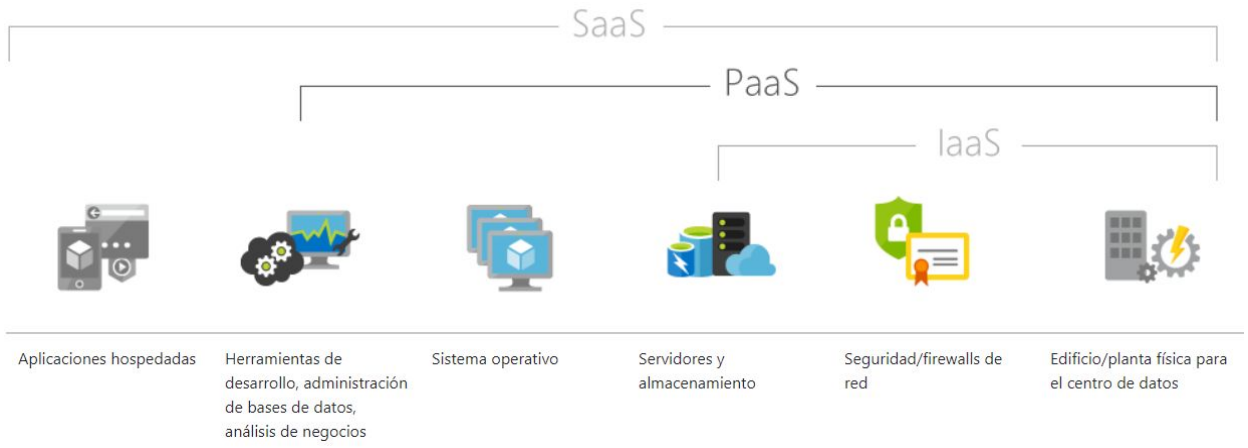
Y así hemos terminado nuestra primer API completa diseñada con solo javascript, estp añadido a las pruebas automatizadas y los test unitarios, los repositorios git como GitHub o Bitbucket, los servidores de integración continua y las PaaS como Heroku o Nodejitsu hacen que el desarrollo web moderno sea más divertido y ágil.

PaaS:

PaaS significa Platform as a service (Plataforma como Servicio) y es similar a Iaas. La diferencia es que en este modelo es posible crear nuevas funciones. Podemos desarrollar aplicaciones propias a partir de la plataforma principal ofrecida. Es flexible y fiable, ya que se

manipula para satisfacer las necesidades específicas de su empresa. Si usted necesita un software, pero por alguna razón se siente limitado por los modelos de SaaS y IaaS, PaaS es la solución. Es práctico, ya que ofrece la personalización que usted tendría en su empresa, pero con la ventaja de ser “como servicio”, liberándonos de la adquisición de hardware, interminables reuniones con el sector de Ti y los costos de mantenimiento.

La automatización de procesos y la aplicación de la metodología BPM se hacen más sencillas cuando se utiliza SaaS, IaaS o PaaS. Toda la información se almacena de forma segura, se puede controlar fácilmente, llega a todos los departamentos y personas al mismo tiempo y puede ser actualizada en el momento que ocurra. Esto resulta en una estandarización muy positiva, ya que todos los procesos están en el mismo lugar y pueden ser fácilmente mapeados y optimizados. Puesto que los pagos se hacen por el uso y mensualmente o anualmente, y los resultados se notan rápidamente, los costos caen y aumenta la productividad, dando a la compañía una gran ventaja sobre los que siguen usando el software tradicional.



Heroku

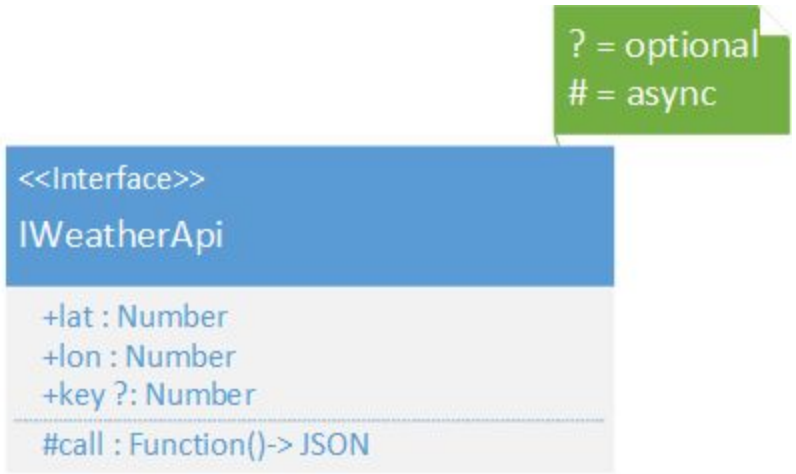
Heroku es una plataforma en la nube como servicio (PaaS) que admite varios lenguajes de programación . Es una de las primeras plataformas en la nube, ha estado en desarrollo desde junio de 2007, cuando sólo admitía el lenguaje de programación Ruby , pero ahora es compatible con Java , Node.js , Scala , Clojure , Python , PHP y Go. Por esta razón, se dice que Heroku es una plataforma Polyglot que tiene características para que un desarrollador cree, ejecute y escale aplicaciones de manera similar en la mayoría de los lenguajes. https://en.wikipedia.org/wiki/Polyglot_(computing)

Cómo funciona nuestra API

Se cuenta con dos rutas /api que redirecciona a una simple respuesta html que nos indica que el servidor está corriendo. Luego para su uso específico se debe hacer un get a /api.

http://api-clima-albor.herokuapp.com/api/lat&lon&lang&key

¿Qué es lo interesante? que con **alborForecast** el único cambio que hay que hacer es agregar la siguiente clase.



La respuesta JSON que espera la app final debe ser la siguiente:

Recomendaciones para escribir una API

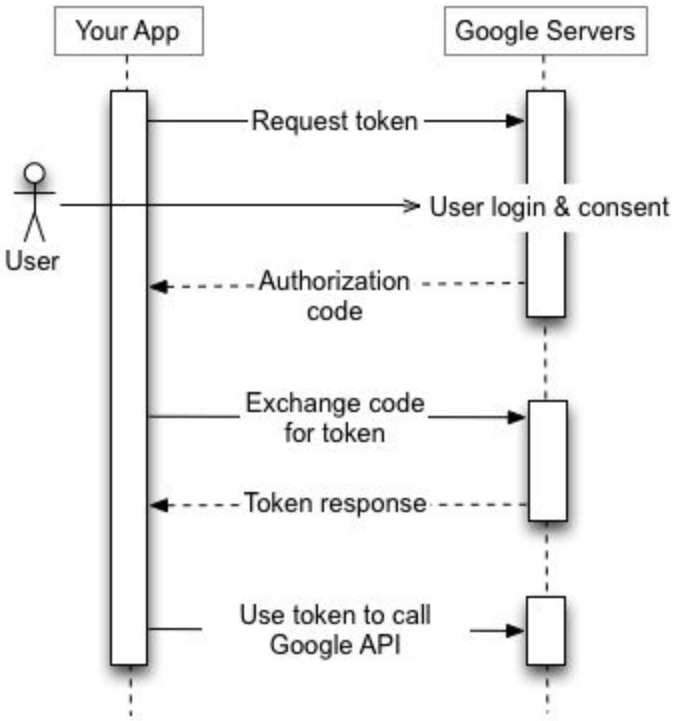
Una de la principales, que seguimos nosotros en la API del clima fué usar **Typescript**, ayuda un montón a encontrar errores, solucionar fallos, etc.
Pensar antes de empezar a escribir el código hacer los diagramas de **secuencia**, **clase**, los que necesites para entender cómo, qué, cuándo y dónde tu API tiene que funcionar.

¿Qué sigue?

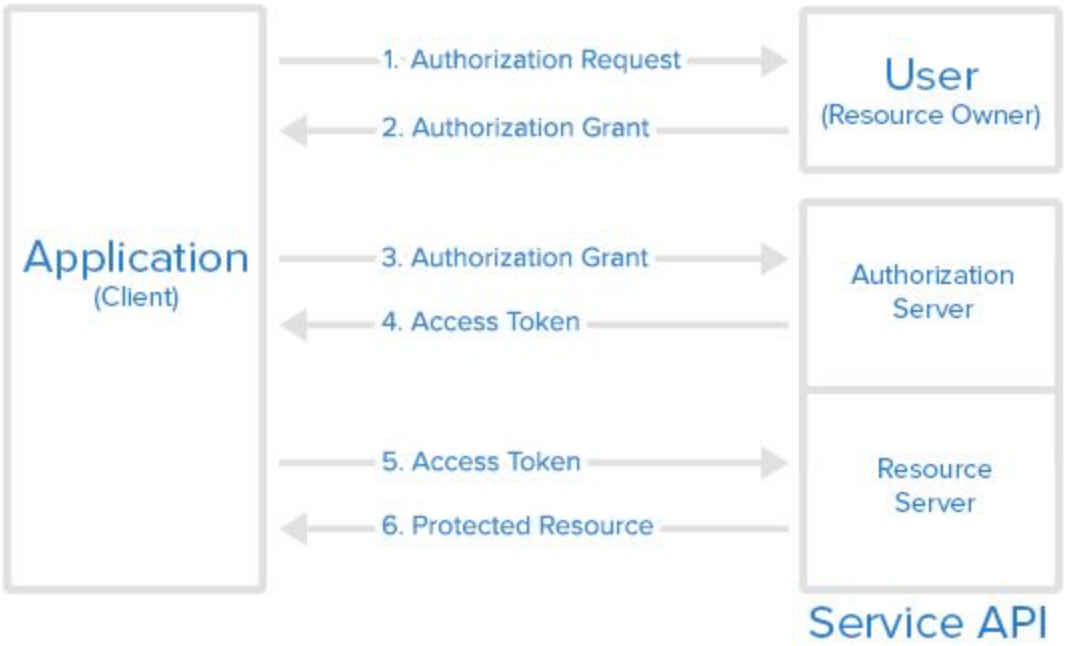
Calendar API

Calendar API permite mostrar, crear y modificar eventos de calendario, así como trabajar con muchos otros objetos relacionados con el calendario, como calendarios o controles de acceso.
Para jugar y ver qué puede hacer la API, sin escribir ningún código, se puede visitar el Explorador de API, para intentar usar calendar.events.list, con algún correo electrónico de Google como ID de calendario. (Deberá habilitar la autorización OAuth 2.0).
Para poder tener un mayor control sobre las funcionalidades que nos proporciona esta api, hay que comprender qué es OAuth 2.0.

¿Qué es OAuth 2.0?
OAuth 2.0 es el protocolo de autorización estándar de la industria. OAuth 2.0 reemplaza el trabajo realizado en el protocolo OAuth original creado en 2006. OAuth 2.0 se centra en la simplicidad del desarrollador del cliente al tiempo que proporciona flujos de autorización específicos para aplicaciones web, aplicaciones de escritorio, teléfonos móviles y dispositivos de sala de estar.
¿Cómo funciona OAuth 2.0?
La información presentada es extraída de la documentación del marco de autorización de OAuth 2.0, a la cual se puede acceder desde el siguiente link: https://tools.ietf.org/html/rfc6749
Pero Google nos informa acerca de este protocolo en su página para



desarrolladores: https://developers.google.com/identity/protocols/OAuth2



1. El cliente solicita autorización del propietario del recurso, puede ser directa o indirectamente.
2. El cliente recibe una concesión de autorización, que es una credencial que representa la autorización del propietario del recurso.
3. El cliente solicita un token de acceso autenticandose con el servidor de autorización y presentando la concesión de autorización.
4. El servidor de autorización autentica al cliente y valida la concesión de autorización y, si es válido, emite un token de acceso.
5. El cliente solicita el recurso protegido del servidor de recursos y se autentica presentando el token de acceso.
6. El servidor de recursos valida el token de acceso y, si es válido, atiende la solicitud.