EL PROTOCOLO HTTP

Es usado para enviar y recibir datos de la web.

Características

- Sencillo: es en modo texto y fácil de usar por una persona.
- Extensible: se pueden enviar más metadatos que los que están por defecto (Ej. nº de página).
- Sin estado: cada petición es independiente. Eso es un problema en sitios como por ejemplo un carrito de la compra.

Ventajas

- Caché: mejora la velocidad al controlar la cache de las páginas.
- Autenticación: permite identificar usuarios.
- Proxys: permite de forma transparente el uso de proxys.
- Sesiones: Gracias a las cookies podemos mantener el estado entre peticiones.
- Formatos: Permite indicar el formato de lo que se envía, se pide y se retorna.

Formato

Una petición HTTP tiene la siguiente forma:

```
GET /index.html HTTP/1.1
Host: www.fpmislata.com
Accept-Language: fr
```

- Donde cada parámetro nos da cierta información:
  - `GET` : Es el método por el que se piden los datos. Entre sus valores está: GET, PUT,POST, DELETE.
  - `/index.html` : Es la ruta dentro del servidor del documento que estamos pidiendo
  - `HTTP/1.1` : La versión del protocolo. Prácticamente siempre es 1.1
  - `Host: www.fpmislata.com` : Indica el nombre del host al que va dirigida la petición.
  - `Accept-Language: fr` : Indica en que idioma queremos que nos retorne los datos. En este caso es en francés.

La respuesta del servidor es la siguiente:

```
HTTP/1.1 200 OK
Content-Length: 29769
Content-Type: text/html; charset=utf-8

<!DOCTYPE html... (los 29769 bytes de la página)
```

- Donde cada parámetro nos da cierta información:
  - `HTTP/1.1` : La versión del protocolo con la que responde. Prácticamente siempre es 1.1
  - `200 OK` : Si ha sido exitosa o no la petición.
  - `Content-Length: 29769` : Indica las bytes que ocupan los datos que se retornan
  - `Content-Type: text/html; charset=utf-8` : Indica el formato MIME type de los datos que retornan y su codificación. En este caso es en HTML y en formato UTF-8.
  - `<!DOCTYPE html...` : Son finalmente los datos que se han pedido.

Cabeceras HTTP

Las cabeceras se dividen entre las que se envían en la petición y las que se retorna en la respuesta.

Petición:

Cabeceras que se pueden enviar en la petición

- `Accept` : El formato MIME type en la que queremos que se retornen los datos. Ej: En `text/html` , en `text/xml` , `application/json` , `application/pdf` , etc. Luego el

servidor los retornará en el formato que quiera/pueda

- `Accept-Language` : El idioma en el que queremos que nos retorne los datos. Luego el servidor los retornará en el idioma que quiera/pueda.
- `Host` : El dominio al que se está enviando la petición. Esta cabecera el muy útil ya que permite en un mismo servidor tener alojados varios dominios.
- `Content-Type` : El formato de los datos que envían al servidor. Ej: En `text/html` , en `text/xml` , `application/json` , `application/pdf` , etc. Y como están codificado. Normalmente los formatos son `utf-8` o `ISO-8859-1` .

Respuesta

Cabeceras que se pueden enviar en la respuesta

- `Content-Type` : El formato de los datos que se retorna. Ej: En `text/html` , en `text/xml` , `application/json` , `application/pdf` , etc. Y como están codificado. Normalmente los formatos son `utf-8` o `ISO-8859-1` . No tiene porque coincidir con `Accept` .
- `Content-Language` : El idioma de los datos que se retorna.
- `Content-Length` : Tamaño en bytes de los datos
- `Cache-Control` : Cuanto tiempo pueden estar cacheado los datos.

> **Note**
>
> La cabecera Content-Type es importante para el programador ya que el servidor puede no saber exactamente el formato de los datos y es necesario que lo indiquemos nosotros. Muchas veces hay además problemas con la codificación si es utf-8 o ISO-8859-1 por lo que también se debe indicar.
>
> Por otro lado notar que Content-Type se puede usar tanto en la petición como en la respuesta. Se usa en la petición si se envían datos en la petición

Estados HTTP

Es estado es lo que indica si una petición HTTP ha tenido éxito o no. Sus principales valores son:

- 200-299: La petición ha tenido éxito
- 300-399: Redirección de los datos.
- 400-499: Los datos que ha enviado el cliente no son correctos
- 500-599: Se ha producido un error en el servidor.

De entre todos los código están algunos que solemos ver a menudo:

- 200: Todo ha ido bien.
- 201: Se ha creado el recurso (Suele ser en un INSERT).
- 204: La petición no retorna datos. (Suele ser en un DELETE).
- 400: Los datos que ha enviado el cliente no son correctos.
- 401: Hay que estar logueado.
- 403: El usuario está logueado pero tiene Prohibido el acceso al documento.
- 404: No encuentra el documento.
- 500: Error del servidor.

Métodos

Los métodos (o verbos) HTTP indican que acción queremos hacer con los datos. Al navegar normalmente se usa siempre el `GET` .

- `GET` : Queremos obtener los datos
- `POST` : Queremos añadir los datos.
- `PUT` : Queremos actualizar nuevos datos.
- `DELETE` : Queremos borrar los datos.

REST

REST es una interfaz para conectar varios sistemas basados en el protocolo HTTP (uno de los protocolos más antiguos) y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON *(actualmente el más usado)*.

Las operaciones a realizar

Vamos a ver 4 método HTTP que coinciden con los 4 métodos de un CRUD o con operaciones de SQL:

Método HTTP	Descripción	Metodo CRUD	Metodo SQL
GET	Este método HTTP lo usaremos para cuando queremos leer datos del servidor	Read	SELECT
POST	Este método HTTP lo usaremos para añadir datos al servidor	Create	INSERT
PUT	Este método HTTP lo usaremos para actualizar datos del servidor	Update	UPDATE
DELETE	Este método HTTP lo usaremos para borrar datos del servidor	Delete	DELETE

La Estructura de la URL

Veamos la estructura de la URL de las peticiones en un supuesto ejemplo de una base de datos de usuarios:

Descripción	URL	Método HTTP	JSON Enviado	JSON Retornado
Obtener un libro	`/libro/{idLibro}`	GET	Ninguno	Libro leido
Listado de libros	`/libro`	GET	Ninguno	Array de Libros
Añadir un libro	`/libro`	POST	Libro a insertar	Libro insertado
Actualizar un libro	`/libro/{idLibro}`	PUT	Libro a actualizar	Libro actualizado
Borrar un libro	`/libro/{idLibro}`	DELETE	Ninguno	Ninguno

Servidor REST en NodeJS

```
const app = express()
```

```
const port = 80

app.get('/', (request, response) => {
  response.set('Content-Type', 'text/plain');
  response.status(200);

  if (request.header('Accept-Language').startsWith("ca-ES")) {
    response.send("Hola mon");
  } else if (request.header('Accept-Language').startsWith("en-EN")) {
    response.send("Hello World");
  } else  {
    response.send("Hola mundo");
  }


});
app.post('/', (request, response) => {
  response.status(200);
  response.send('Hello from post!');
});
app.delete('/', (request, response) => {
  response.status(200);
  response.send('Hello from delete!');
});
app.delete('/libro/38', (request, response) => {
  response.status(200);
  response.send('Borrado libro 38');
});
app.delete('/libro/39', (request, response) => {
  response.status(404);
  response.send('El libro 39 no existe');
});

app.listen(port, (err) => {
  console.log(`server is listening on ${port}`)
})
```