Programación Web 2 Servidor HTTP y Enrutamiento

U-TAD

Modelo cliente-servidor

Modelo que usamos para acceder a Internet y obtener recursos e información.

Cliente: El **navegador** desde el cual se realizan las solicitudes a un servidor. Pero podría ser una **app**, o un **script**.

Servidor: **Programa** que se ejecuta en un servidor físico (on-premise o cloud) para ofrecer un servicio al cliente. Envía información.

El servidor **conoce el formato** esperado del mensaje que envía el cliente. Y el cliente **conoce el formato** esperado del mensaje que recibe del servidor.

El **protocolo HTTP** (capa de aplicación) define el formato de los mensajes.

Protocolo: Conjunto de **reglas** que permiten transmitir información entre dispositivos de una red.

Solicitudes HTTP

HTTP: Hyper**t**ext **T**ransfer **P**rotocol

Request (req): Cuando un cliente solicita información de un servidor o desencadena algún evento para que el servidor ejecute algún proceso.

- Métodos HTTP: GET, POST, PUT, DELETE
- PATH: http[s]://domain[:port]/**path**
- Versión de HTTP
- Headers: Proveen información adicional sobre la solicitud (**metadatos**).
- Body: Contiene los **datos** que deben ser enviados al servidor para procesar la solicitud (se envían en JSON). **No se incluye en todas las solicitudes** (solo en las que requieran enviar información, como POST o PUT).

Métodos HTTP

Verbo o sustantivo que indica la intención de la solicitud.

- GET: Verbo para **solicitar** un recurso específico. Por ejemplo, un archivo html, css o una imagen.
- POST: Verbo para **crear** un recurso específico. Por ejemplo, agregar un usuario nuevo a una base de datos.
- PUT: Verbo para **modificar** un recurso específico. Por ejemplo, hacer un cambio en la base de datos.
- DELETE: Verbo para **eliminar** un recurso específico. Por ejemplo, eliminar un usuario de una base de datos.
- Otros: Head, Connect, Options, Trace, Patch (no los vamos a ver).

Respuestas HTTP

Response (res): Cuando el servidor responde a la petición previa del cliente.

- Código de estado: 200, 302, 403, 404, 500
- Texto de estado: OK, Redirect, Forbidden, Not Found, Internal Server Error
- Versión de HTTP
- Headers: Son opcionales y proveen información adicional sobre la respuesta.
- Body: Contiene los **datos** que deben ser enviados desde el servidor hacia el cliente.

Códigos de Estado HTTP

Le permite saber al Cliente qué pasó con su solicitud en el Servidor.

Es un número que indica si se ha completado exitosamente la solicitud HTTP:

- Respuestas informativas (100-199)
- Respuestas satisfactorias (200-299)
- Redirecciones (300-399).
- Errores de los clientes (400-499).
- Errores de los servidores (500-599).

https://developer.mozilla.org/es/docs/Web/HTTP/Status

Con Node.js y Express podemos especificar el código de estado de la respuesta HTTP en nuestro servidor.

Crear un servidor con Node.js

Módulo http: le permite a Node.js escuchar solicitudes y transmitir información con el protocolo HTTP.

```
const http = require('http');
```

Crear un servidor con http:

```
const servidor = http.createServer((req, res) => {
    … //Lógica del proceso leyendo req
    res.end(...); //Permite enviar la respuesta al cliente cuando termina el proceso
});
```

Estará escuchando en un end-point (IP:Puerto)

Puerto: Ubicación virtual del Sistema Operativo en la cual se puede acceder a una aplicación, o a un proceso específico, que se esté ejecutando (escuchando) en ese puerto. En nuestros ejemplos de Node.js trabajaremos con el puerto **3000**

Crear un servidor con Node.js

```
const port = 3000;
servidor.listen(port, () => { //Puerto y qué queremos hacer al inicializarse
    console.log("Servidor escuchando en localhost puerto" , port);
});
```

Lo arrancamos con “*node index.js”* y accedemos desde un navegador a “*localhost:3000*”

Podemos salir con *Ctrl+C*

NOTA para no tener que reiniciar el servidor cuando cambiamos algo en el código: **Nodemon**

npm init –yes

npm install nodemon –save-dev (o si lo queremos global “npm install -g nodemon”)

En package.json (en “scripts”), añadir:

“start”: “nodemon index.js”

Arrancar con “npm start” (o desde Command Prompt con “nodemon index.js” si lo instalaste con -g)

req y res

Propiedades más importantes de Request (req):

- req.url
- req.method

Extensiones de VS Code recomendadas para instalar:

- ESLint (opcional, para analizar estáticamente tu código JS)
- REST Client (lo usaremos para realizar peticiones y probar nuestra API)
  - Crea el fichero **index.http** dentro de tu directorio
  - Añade el contenido `POST http://localhost:3000/`
  - Y al tener la extensión REST Client, se nos habilita la opción “Send Request” arriba.
  - Ver la respuesta obtenida en la pestaña “Response” del VS Code.
  - Con el navegador solo podemos hacer peticiones GET. Pero con REST Client, haremos además de GET, POST, PUT, DELETE, etc.

req y res

Propiedades más importantes de Response (res):

- res.statusCode
- res.setHeader(...)
- res.getHeaders()

Estructura de una URL

URL (Uniform Resource Locator): Dirección de un recurso en la Web.

Ejemplo: https://www.google.es/imghp

- Protocolo: http:// o https://
- Dominio: www.google.es
  - Subdominio (www): Información adicional agregada al inicio del dominio de una Web. Permite a los sitios Web organizar y separar la información para distintos propósitos.
  - Dominio (google): Referencia única a un sitio web en Internet.
  - TLD (es): Top Level Domain (com, it, pt, fr, …, gov, org, edu, ...)
- Path: /imghp
  - Archivo o directorio en el servidor web.
  - Puede tener parámetros para personalizarlo:
    - Parámetros de ruta, separados con "/": http://www.ejemplo.org/usuarios/**14**
    - Parámetros query con "?": https://www.google.es/imghp**?hl=es&authuser=0**
      - Son parámetros usados para obtener contenido dinámico. Por ejemplo, filtrar una lista de recursos.

Usaremos parámetros query para **filtrar** los recursos en las solicitudes GET.

Módulo url

```
const miURL = new URL('https://www.ejemplo.org/cursos/programacion?ordenar=vistas&nivel=1');
```

Imprimir con console.log:

- miURL.hostname
- miURL.pathname
- miURL.searchParams
  - miURL.searchParams.get(‘ordenar’)
  - miURL.searchParams.get(‘nivel’)

Routing

Manejar las solicitudes del cliente en base a ciertos criterios:

- Method de la solicitud HTTP. De esta forma, el servidor sabe qué tipo de operación se realizará (GET, POST, PUT, DELETE, …)
- Path de la solicitud HTTP. De esta forma, el servidor sabe el recurso específico que se usará.

¿Cómo se aplica? Crea un nuevo proyecto en /routing con “npm init –yes”

- Crea un servidor en index.js
- Crea un fichero cursos.js (que simulará la base de datos).

Routing (cursos.js)

```
const infoCursos = {
    'programacion': [
        {
            id: 1,
            titulo: 'Aprende Python',
            lenguaje: 'python',
            vistas: 15000,
            nivel: 'basico'
        },
        {
            id: 2,
            titulo: 'Pyhton intermedio',
            lenguaje: 'python',
            vistas: 13553,
            nivel: 'intermedio'
        },
        {
            id: 3,
            titulo: 'Aprende javascript',
            lenguaje: 'javascript',
            vistas: 102223,
            nivel: 'basico'
        }
    ],
    'matematicas': [
        {
            id: 1,
            titulo: 'Aprende Calculo',
            tema: 'calculo',
            vistas: 12427,
            nivel: 'basico'
        },
        {
            id: 2,
            titulo: 'Aprende Algebra',
            tema: 'algebra',
            vistas: 15722,
            nivel: 'intermedio'
        }
    ]
}

module.exports.infoCursos = infoCursos;
```

Routing (index.js)

```
const http = require('http');

const cursos = require('./cursos');

const servidor = http.createServer((req, res) => {

        switch(req.method) {

                case 'GET':

                        return manejarSolicitudesGET(req, res);

                … // Lo mismo con POST, PUT y DELETE

                default: …

        }

function manejarSolicitudesGET(req, res) {

        const path = req.url;

        if (path === '/') { … }

        else if (path === '/cursos') { …; return res.end(JSON.stringify(cursos.infoCursos)); }

        … // TODO Devuelve /cursos/matematicas además de /cursos/programacion

}

const port = 3000;

servidor.listen(port, () => {...})
```

Routing (index.js)

```
…

function manejarSolicitudesPOST(req, res) {
        if (path === “/cursos/programacion”) {
                let body = “”;
                req.on(‘data’, (content) => {
                        body += content.toString();
                })
                req.on(‘end’, () => {
                        body = JSON.parse(body);
                        …
                        return res.end(“Procesamiento POST finalizado”);
                })
        } else { res.statusCode = 404 }
}

…
```