Programación Web 2 Servidor Introducción

U-TAD

Introducción al Back-end Web

Introducción a Node.js

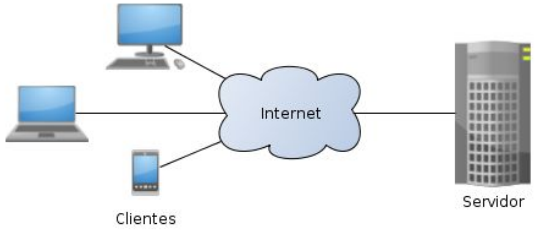
Entorno de ejecución de JavaScript orientado a eventos asíncronos.

- Entorno de ejecución: entorno (infraestructura) en el cual se ejecuta un programa o aplicación. Anteriormente, solo se podía ejecutar JS en el navegador, pero gracias a Node.js se puede ejecutar en un terminal del SO. Por tanto, se puede utilizar en entornos back-end.
- Evento asíncrono: Evento que se ejecuta independientemente del proceso principal de la aplicación. Continúa su ejecución (tareas en paralelo).
- Evento síncrono: Evento que se ejecuta como parte del proceso principal de la aplicación. Bloquea la aplicación (tareas en serie).

Características:

- Open-source (código abierto)
- Multiplataforma
- Basado en el motor V8 de Google (Chrome)
    - Motor que ejecuta JS

Conceptos



Arquitectura (o Modelo) Cliente-Servidor

- Modelo en el cual el servidor envía recursos al dispositivo que los solicita (cliente).

Desarrollo front-end

- Área del desarrollo web que se encarga de los componentes que ve el usuario y con los que interactúa (Interfaz de Usuario).

Desarrollo back-end (registro de usuario, guardar y consultar datos, …)

- Área del desarrollo web que se encarga de la parte servidor (y bases de datos). Lo que el usuario NO ve.
- Intermediarios entre las solicitudes del navegador y las Bases de Datos.

Desarrollo full-stack (desarrollo front y back-end)

Protocolo entre cliente y servidor (por ejemplo, HTTP/S)

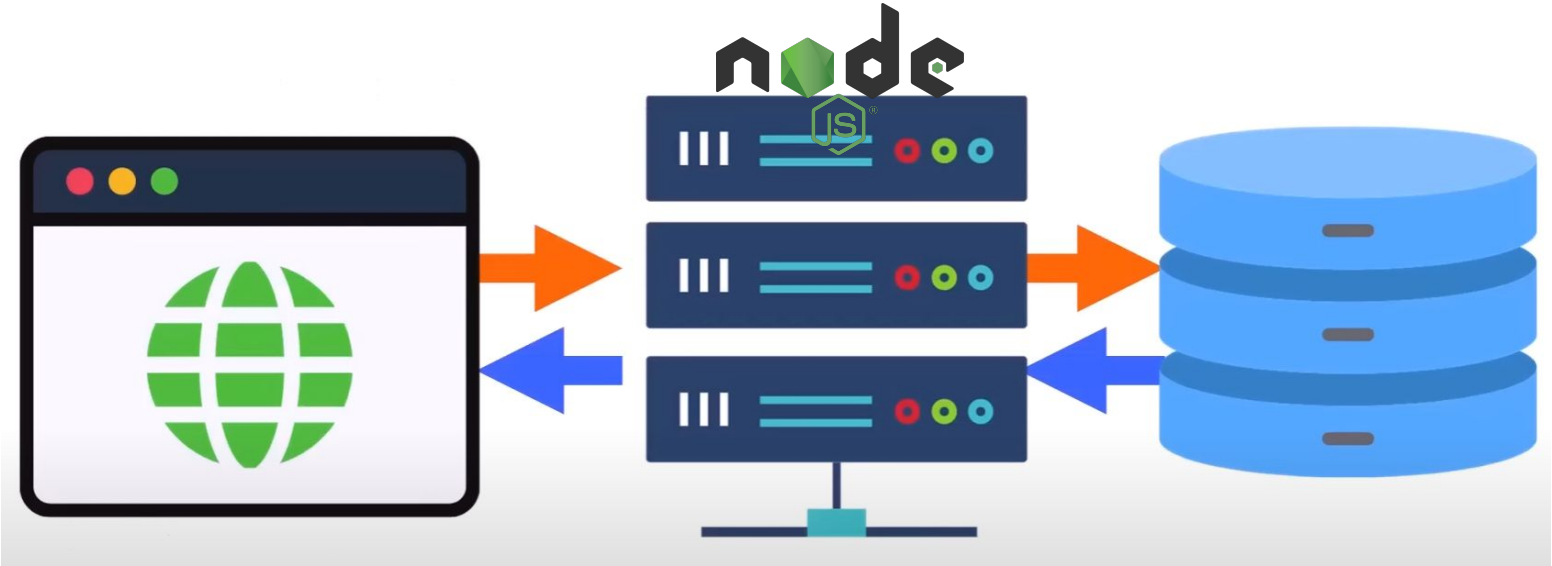
- Reglas que permiten que dos entidades de una red se comuniquen.
- Envían peticiones y respuestas en un formato específico.

Bases de datos

Conjunto de datos estructurados sobre un mismo contexto que se pueden consultar.

Bases de datos relacionales (MySQL, Postgres) y noSQL (MongoDB, Redis)

Webs dinámicas Vs estáticas (blog)



Aplicaciones de Node.js

Se usa para:

- Desarrollo back-end (entorno de ejecución, el lenguaje es JS y su framework puede ser *express* y puede hacer uso de distintas bibliotecas)
- Desarrollar APIs (Application Programming Interface)
  - Interfaz entre programas (frente a la interfaz de usuario del front-end)
  - Conecta la aplicación que se ejecuta en el navegador con el servidor (back-end) para solicitar información a la BD o trabajar en ella. Aunque también puede procesar sin involucrar a la BD.

Ventajas:

- Escalable: su rendimiento se puede adaptar a medida que crece el uso de la aplicación y recibe más solicitudes.
- En tiempo real: establece una conexión bidireccional y dinámica entre el servidor y el cliente. Analiza los eventos que ocurren y reacciona de forma casi inmediata.

Instalación de Node.js

https://nodejs.org/es/

Descargamos LTS (Long-term Support) y lo instalamos (busca tu Sistema Operativo).

En el “Custom Setup” selecciona el “Add to PATH” si no está ya.

Revisar desde consola (VS Code) con “node –version”

REPL (Read Eval Print Loop) de Node.js

Herramienta interactiva para poder escribir código JS y ejecutarlo.

Entramos con el comando “node”.

> .help

.editor

```
function holaMundo(nombre) { return `Hola ${nombre}`; }
holaMundo(“mundo”);
```

Ctrl+D

Ctrl+C

.exit

Primer Programa con Node.js

Environment: VS Code

Github: https://github.com/rpmaya/u-tad-Server

Cread un directorio nuevo.

Crea el fichero “app.js” en él, y copia el código de holaMundo() anterior

Ejecuta “node .\app.js”

Módulos en Node.js

Funcionalidad organizada en uno o varios archivos JavaScript que puede ser **reutilizada** en una aplicación.

Organizaremos el código en módulos, de modo que los podamos reutilizar.

Importamos la funcionalidad de un módulo en otro módulo.

Ventajas:

- Evitar repetición de código, reutilizar código.
- Es más fácil mantener el código (actualizaciones y detección de “bugs”).
- Es más fácil agregar nuevas funcionalidades.

Crear un módulo

Creamos distintos ficheros con funciones, por ejemplo, en *saludo.js* la definición y otro, *app.js*, con las llamadas.

En app.js o cualquier otro módulo, debemos importar (un módulo que se requiera): Darle acceso a funciones y elementos definidos en otro módulo. Para ello, debemos exportarlo desde el módulo a utilizar.

module.exports = { f1: f1, f2: f2, …}

const nombre = require(“path/name.js”)

const { f1, f2 } = require(“path/name.js) (sintaxis de desestructuración)

Módulos principales (core): built-in (incorporados)

Módulos que puedes usar directamente al trabajar con Node.js (sin instalarlos)

- http y https
- fs (file system)
- os (operating system)
- path
- …

Módulo **console:** Implementa funcionalidad similar a la consola de un navegador

- console.log()
- console.warn()
- console.error()
- console.assert()
- console.table()

Módulo **process**: Provee información sobre el proceso de NodeJS que se está ejecutando. Y puede tener cierto control sobre el proceso.

- process.env: Permite acceder al contenido del *enviroment*.
- processs.argv: Permite acceder a los argumentos pasados en la llamada a node app.js arg1 arg2 …
- process.memoryUsage()

Módulos principales (core): built-in (incorporados)

Módulo **OS:** contiene funcionalidad para obtener información sobre el sistema operativo en el cual se ejecuta la aplicación.

No se importa por defecto, por tanto, **const os = require(“os”);**

- os.type()
- os.homedir()
- os.uptime() (en segundos)
- os.userInfo()

Module **timers** (se puede usar para simular operaciones asíncronas): contiene funciones que ejecutan código después de un cierto periodo de tiempo.

- setTimeout(function, timeout(ms), param1, param2, …) : ejecuta una función después de un número específico de ms.
- setImmediate(function, param1, param2, …) : para ejecutar código asíncrono en la próxima iteración del ciclo de eventos (lo antes posible). Se ejecuta después del código síncrono. Y con la mayor prioridad sobre las tareas asíncronas.
- setInterval(function, interval(ms), param1, param2) : para ejecutar código un número infinito de veces con un retrase específico de ms.

Módulos principales (core): built-in (incorporados)

Ejercicios con **timers**, en app.js:

1. Crear función mostrarTema(tema) y llamarla de modo que se ejecute pasados 3 segs.
2. Crear función sumar(x, y) y llamarla de modo que se ejecute pasados 5 segs.
3. Llama a mostrarTema(tema) después del ciclo de ejecución síncrono. Añade una console.log(“antes”) y un console.log(“después”) antes y después de la llamada a mostrarTema respectivamente. ¿Cuál es el orden de ejecución?
4. Llama a mostratema(tema) cada dos segundos.
5. Llama a sumar(x, y) cada tres segundos.

Módulos principales (core): built-in (incorporados)

Module **fs (File System):** Módulo que contiene funcionalidad muy útil para trabajar con el sistema de archivos

Operaciones útiles: leer, modificar, copiar, eliminar, cambiar nombre **(de un fichero o directorio)**

Todos los métodos de este módulo son asíncronos por defecto. Pero se puede seleccionar versiones síncronas de estos métodos agregando **Sync** a sus nombres. *fs.rename() -> fs.renameSync()*

Ejercicio: Crear app.js e index.html (con la estructura básica) y leélo con fs (se debe incluir el módulo previamente). Usa *fs.readFile(file, "utf-8", () => { … });*

```
const fs = require("fs");
```

Módulos principales (core): built-in (incorporados)

Módulo fs (cont)

Asíncronos:

- Cambiar el nombre de un archivo: fs.rename(name, newName, (err)=>{...})
- Añadir contenido al final del archivo: fs.appendFile(name, content, (err)=>{...})
- Reemplazar contenido: fs.writeFile(name, newContent, (err)=>{...})
- Eliminar archivo: fs.unlink(name, (err)=>{...})

Si se ejecutarán todas, no tienen porqué ser en orden (son ejecuciones paralelas y cada una terminará en un tiempo distinto, no ordenado según el código).

Si queremos que sea predecible el orden, usamos las funciones síncronas:

- renameSync, appenFileSync, writeFileSync, unlinkSync (ya no es necesario ()=>{})

Introducción a npm

Es el archivo (repositorio) de software más grande del mundo que contiene paquetes que puedes instalar y usar con Node.js. También es una herramienta de la línea de comandos.

Un paquete es un archivo o directorio descrito por el archivo *package.json*

Este paquete puede ser publicado en el registro de npm.

Un módulo es cualquier archivo o directorio en el directorio *node_modules* que puede ser importado con *require().*

node_modules se creará automáticamente en nuestro directorio de trabajo cuando trabajemos con npm. Ahí estarán los paquetes o módulos descargados y que podremos usar en nuestro proyecto.

Solo los módulos que tienen un archivo *package.json* son paquetes.

npm (continuación)

Dependencia: paquete que otro paquete necesita para funcionar correctamente.

Paquete **A** **—>** Paquete **B**

Dependencia “depende de A” (Cuando instalamos B, también se instalará A)

Crear paquete con npm

- Crear un nuevo directorio, y entra dentro desde la consola.
- Ejecutar: *npm init*
- Escribir o dejar valores por defecto para
  - package name, version, description, **entry point** (nuestro fichero principal: index.js) y test command, git repository (los dejamos vacíos de momento), keywords (si lo vamos a publicar en npm), author (tú) y license (ISC)

- Se guarda automáticamente en el fichero *package.json*

- Si se quiere con los valores por defecto: *npm init –yes*

Repaso de JSON

JavaScript Object Notation

Formato de texto usado para almacenar y transportar información.

Se usa para intercambiar información entre el cliente y el servidor.

JSON está basado en la notación de objetos de JavaScript, pero es solo texto.

Nos permite almacenar **objetos** de Javascript como **texto**.

Es independiente del lenguaje de programación con el cual estás trabajando.

Recordad que en las peticiones HTTP de tipo POST se enviaba un JSON, y el server la *parsearía*. Además, el server respondía a las peticiones de tipo GET también en formato JSON, y después las parseábamos en el cliente.

Características del formato JSON

Los datos se representan como pares clave-valor (propiedades)

```
{
  "name": "paquete",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
  },
  "author": "Ricardo Palacios",
  "license": "ISC"
}
```

También puede estar engloblado por “[…]” si es un array de JSON.

Los elementos deben estar separados por una **coma**.

Características del formato JSON (continuación)

Las claves deben ser cadenas de caracteres.

Los valores pueden contener distintos tipos de datos:

- Cadena de caracteres
- Números
- Arrays
- Boolean (true | false)
- Objetos (otros json anidados)

De JSON a objeto de {JS} y de {JS} a JSON:

- JSON.parse() y JSON.stringify()

Ejercicio: En el directorio anterior, crear un curso.json y parséalo. Comienza usando *require('file.json'), acceso local,* y escribe en consola alguno de sus campos como el título.

Ahora lo trabajamos como un objeto de JS, re-conviértelo a json (con otro nombre) usando stringify() y parse(), por ejemplo, para poderlo enviar o almacenar.

Instalar y desinstalar paquetes con npm

Crear nuevo directorio, entrad en él e inicializalo con "*npm init –yes*" desde consola.

Crearemos el fichero index.js

En package.json, modificar:

- description: **"Mi primer paquete de Node.js"**
- author: **"<Tu nombre>"**
- scripts: { "test": "...", **"start": "node index.js"** }

Si necesitamos instalar un paquete:

- npm install <paquete>, por ejemplo: ***npm install express*** (se añade automáticamente a package.json en "dependencies").
- Se descargan todas las dependencias en el directorio *node_modules/* y se crea el fichero *package-lock.json* (metadatos de los paquetes instalados)

Instalar y desinstalar paquetes con npm

Si queremos desinstalar el paquete: ***npm uninstall express*** (desaparace de package.json)

Si queremos una versión específica de un paquete: ***npm install express@4.15.1***

node_modules/ no se incluye en los repos de Github. Cuando compartimos código con terceros, estos se descargan las dependencias haciendo "npm install" que leerá de package.json.

Si instalaste express v.4.15.1, desinstálalo con ***npm uninstall express***

Tendremos algunas dependencias que solo usaremos durante la fase de desarrollo, y no serán necesarias cuando esté nuestra Web en producción:

npm install express –save-dev (en package.json, ahora aparecerá bajo la categoría "devDependencies" en su versión más reciente).

package-lock.json

Se genera automáticamente cuando npm modifica el árbol de *node_modules* o *package.json* (mantiene un registro con el árbol de dependecias).

Describe el árbol generado para que futuras instalaciones puedan generar exactamente el mismo árbol. De esta forma, otros desarrolladores pueden instalar exactamente las mismas dependencias (asegurando así la funcionalidad del código JS).

Propiedades de package-lock.json:

- “name”: El nombre de nuestro paquete. Es el mismo nombre que incluimos en package.json
- “version”: La versión del paquete. Es la misma versión que incluimos en package.json
- “lockfileVersion”: La versión del archivo package-lock.json (cuantas veces se modificó)
- “packages”: Objeto que asocia la ubicación de cada paquete con un objeto que contiene información sobre ese mismo paquete. Relaciona la ubicación dentro de node_modules con un objeto que describe ese paquete: versión, origen de descarga (resolved), checksum, sus dependencias, …