

**ACTIVITAT AVALUABLE AC1****Mòdul:** MP08- Desplegament d'aplicacions web**UF:** UF2 – Servidors d'aplicacions web**Professor:** Albert Guardiola**Data límit d'entrega:** 19/01/2025 23:59**Mètode d'entrega:** Per mitjà del Clickedu de l'assignatura. Les activitats entregades més enllà de la data límit només podran obtenir una nota de 5.**Instruccions:** S'ha d'entregar un únic document amb el nom:***MP08-UF2-AC1-Nom\_Alumne.doc (o pdf)***

Es valorará la presentació.

**Resultats de l'aprenentatge:**

RA1. Implanta aplicacions web en servidors d'aplicacions avaluant i aplicant criteris de configuració per al seu funcionament segur.

**Tasques a realitzar:****Part A. Introducció a NodeJS**

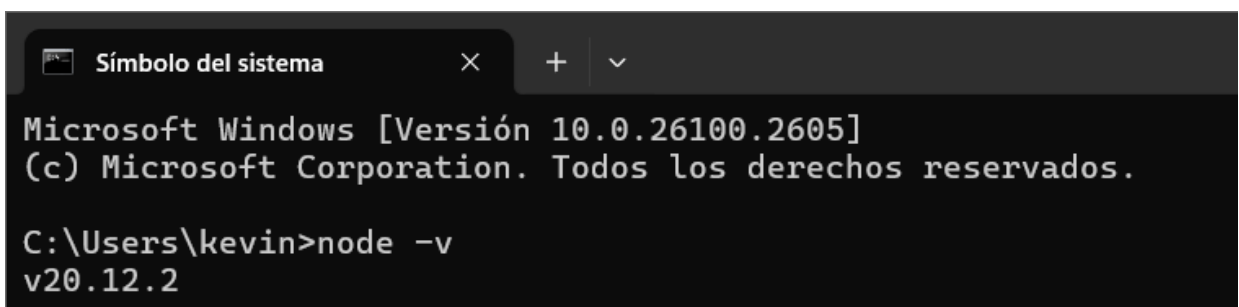
**NodeJS** es un entorno de ejecución de JavaScript para backend, de carácter multiplataforma y con licencia *open source*. Actualmente, compite en popularidad con los navegadores web como principal entorno de ejecución de JS.

NodeJS ha permitido que el lenguaje JavaScript se pueda utilizar en todo el *stack* de una aplicación web, de manera que el desarrollo se haga más rápido y eficiente.

Además, gracias a la herramienta de gestión de paquetes *npm*, que se instala junto con NodeJS, es posible utilizar en los proyectos propios una multitud creciente de módulos desarrollados, mantenidos y documentados por la comunidad de desarrolladores.

Tasca 1. a) Instal·la *NodeJS* desde <https://nodejs.org/en/> (hay documentación detallada del producto en <https://nodejs.org/en/docs/>).

b) Comprova la versió instal·lada amb la comanda:

`node -v`

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.26100.2605]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\kevin>node -v
v20.12.2
```

c) *Node* permet dos modes d'interpretació i execució de comandes JavaScript. Primer veiem el REPL, l'interpret de línia de comandes, que s'arrenca amb la comanda de CMD:

*node*

i executa una instrucció JavaScript. Per exemple:

*console.log("Hola món!");*

```
Welcome to Node.js v20.12.2.  
Type ".help" for more information.  
> console.log("Hola, mundo")  
Hola, mundo  
undefined  
>
```

Com es pot veure, és un anàleg de l'interpret interactiu *iPython*.

d) Ara veiem l'altra manera (i de lluny la més habitual) d'executar JavaScript a *Node*. Guarda el codi JS de la tasca anterior en un fitxer (*app.js*) i executa'l com un *script*, amb la comanda:

*node app.js*

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1>node app.js  
Hola, mundo
```

e) Abans de continuar, parem un moment per entendre bé les diferències entre els dos entorns d'execució de JavaScript: els navegadors web vs *NodeJS*. Per fer-ho, marca a quin dels dos entorns aplica cada afirmació.

	Navegador	NodeJS
-S'executa en la màquina client.		
-S'executa en la màquina servidor.		
-És possible accedir el <i>DOM</i> .		
-És possible accedir el sistema de fitxers.		
-Fa servir mòduls <i>ES6</i> .		(Si está configurado)
-Fa servir mòduls <i>CommonJS</i> .		
-Fa servir <i>Globals</i> , objectes accessibles a tot el codi, com ara <i>__dirname</i> , <i>__filename</i> , <i>require</i> , <i>module</i> , <i>process</i> ...		

Tasca 2. En les nostres aplicacions Node podem fer servir els mòduls de la llibreria estàndard de Node. Per exemple, anem a introduir un parell de mòduls estàndard que resulten molt útils.

a) Fem servir el mòdul *path*. Explica quin diferent objectiu aconseguixen aquests dos fragment de codi:

```
1  const path = require('path')
2
3  data_folder = 'data/'
4  products_folder = 'products/'
5  products_file = 'products.json'
6
7  const full_path = path.join(data_folder, products_folder, products_file)
8  console.log(full_path)
```

```
1  const path = require('path')
2
3  data_folder = 'data/'
4  products_folder = 'products/'
5  products_file = 'products.json'
6
7  const full_path = path.resolve(data_folder, products_folder, products_file)
8  console.log(full_path)
```

Podríamos dividir las diferencias en que a/ usa el método join del objeto path mientras que b/ usa el método resolve.

Las diferencias a nivel de uso es que join se suele usar para crear rutas relativas ya que este devuelve un string combinando los segmentos de ruta, respetando el separador del sistema operativo correspondiente.

En cambio resolve te devuelve una ruta absoluta combinando los fragmentos de ruta pasados combinándolos con tu CWD.

Com dèiem abans, Node, a diferència del navegador web, té accés (en principi irrestrict) al sistema de fitxers de la màquina amfitriona. Bona part del treball amb fitxers es fa, a baix nivell, per mitjà del mòdul *fs*. La seva documentació explica totes les operacions que pot fer sobre fitxers i directoris:

<https://nodejs.org/api/fs.html>

Ens interessa aquí, però, entendre la diferència entre treball síncron i treball asíncron sobre fitxers. Aquesta distinció és important a Node i, en general, en tot l'àmbit *back-end*, en la que acostumen a congregar diversos entorns de programació, diferents fonts d'informació i, generalment, també diverses màquines i que, per tant, fa necessari en moltes ocasions l'ús de funcions asíncrones.

b) Executa aquests dos fragments de codi i compara els seus resultats. Quin dels dos fa servir funcions asíncrones?

```
1  const fs = require('fs')
2
3  const data = fs.readFileSync('info.txt', 'utf-8')
4  console.log(data) // file content
5  console.log('The file has been read')
```

```
1  const fs = require('fs')
2
3  const info = fs.readFile('info.txt', 'utf-8', (err, data) => {
4    console.log(data)
5  })
6  console.log('The file has been read')
```

1: Hace servir una funció síncrona, hasta que no lee el archivo , el hilo de ejecución del programa no continúa.

```
C:\Users\kevin\Desktop\DAW\2º\M8.Despliegament d'aplicacions web\UF2\AC1>node
app.js
Prueba1
The file has been read
```

2: Hace servir una funció asíncrona, el programa continúa sin tener que esperar a que termine la lectura sin bloquear el hilo de ejecución.

```
C:\Users\kevin\Desktop\DAW\2º\M8.Despliegament d'aplicacions web\UF2\AC1>node
app.js
The file has been read
Prueba1
```

c) Explica, en general, la diferència entre funcions síncrones i funcions asíncrones.

- Síncronas: Se ejecutan de manera secuencial, esperando a que cada tarea termine antes de continuar con la siguiente. Esto puede bloquear el flujo si una tarea tarda mucho.
- Asíncronas: Permiten que el programa continúe ejecutándose mientras una tarea se realiza en segundo plano. Cuando la tarea finaliza, ejecuta una acción (como un callback o una promesa).

**Més endavant, dedicarem temps a estudiar en detall com es treballa de manera asíncrona a Node, ja que és una tècnica que es fa servir habitualment en entorn servidor.**

Tasca 3. En Node es fan servir, constantment, no només els mòduls estàndar sinó també els dels repositoris de la comunitat. A Node, el gestor de paquets per defecte és npm (node package manager), tot i que el seu autor diu que es tracta d'un retroacrònim).

A més de l'eina de gestió de paquets, *npm* és un repositori online, mantingut pel que fa a la compatibilitat i a la seguretat dels paquets. La web oficial <https://www.npmjs.com/> inclou documentació sobre els paquets.

I, a més, *npm* és una eina de gestió de la configuració d'un projecte Node, per mitjà del que es coneixen com a *scripts npm*. **Més endavant en el curs explorarem la potència d'automatització que tenen aquests *scripts npm*.** Però ara demanarem a l'eina que ens creï un *script* de configuració mínim.

a) Crea un projecte nou de Node: per fer-ho, tanca el directori (*folder*) existent a VSCode i obre'n un altre. A la terminal, executa la comanda:

```
npm init -y
```

i observa el fitxer *package.json* que s'ha creat.

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1>npm
init -y
Wrote to C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2
\AC1\package.json:

{
  "name": "ac1",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

b) A continuació, instal·la el paquet *slugify*, mitjançant la comanda

```
npm install slugify
```

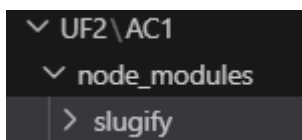
Observa com l'eina *npm* descarrega el paquet i l'instal·la.

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1>npm
install slugify

added 1 package, and audited 2 packages in 2s

found 0 vulnerabilities
```

c) Observa com s'han afegit noves dependències del projecte al fitxer de configuració *package.json* (carpeta *node\_modules*).



3/8



CFGS DESENVOLUPAMENT D'APLICACIONS WEB

d) Ara ja pots incloure el paquet en el teu codi. Per exemple, executa:

```
1  const slugify = require('slugify')
2
3  console.log(slugify('My New Web Site'))
```

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1>node app.js
My-new-web-site
```

e) Busca a la web de NPM la documentació del paquet *slugify* i explica:

-Quina és la funcionalitat del paquet.

- Se utiliza para convertir cadenas de texto en "slugs", es decir, versiones de una cadena que son aptas para ser utilizadas en URLs, nombres de archivos o identificadores, eliminando o reemplazando caracteres no válidos y espacios.

-Quines opcions ofereix.

- **replacement**: Caracter que reemplaza a los espacios en blanco; por defecto es ' - '.
- **remove**: Expresión regular que define qué caracteres eliminar; por defecto es **undefined**.
- **lower**: Convierte la cadena a minúsculas si se establece en **true**; por defecto es **false**.
- **strict**: Elimina caracteres especiales excepto el de reemplazo si se establece en **true**; por defecto es **false**.
- **locale**: Código de idioma para utilizar reglas específicas de transliteración; por defecto es **undefined**.

-Quina és la seva darrera versió.

- La última versión publicada es la 1.6.6, lanzada hace 2 años.

-Quin tipus de llicència té.

- El paquete está licenciado bajo la licencia MIT.

-Quines dependències té.

No tiene dependencias, lo que significa que no requiere otros paquetes para funcionar

f) Busca quines són les opcions de la comanda *npm* que permeten:

-Saber la versió instal·lada d'un paquet.

- `npm list <nombre_paquete>`

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1>npm list slugify
ac1@1.0.0 C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1
`-- slugify@1.6.6
```

-Actualitzar la versió d'un paquet.

- `npm update <nombre_paquete>`

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1>npm update slugify

up to date, audited 2 packages in 2s

found 0 vulnerabilities
```

-Eliminar un paquet.

- `npm uninstall <nombre_paquete>`

g) Consulta la secció de dependències del fitxer *package.json*. A l'entrada

**"slugify": "^1.6.5"** (o equivalent)

explica:

-Què vol dir cadascun dels dígit.

1: La primera xifra (major) indica la versió principal del paquet.

6: La segona xifra (menor) indica una actualització de funcionalitat que no trenca l'API existent.

5: La tercera xifra (de correcció) indica una actualització de seguretat o correcció d'errors

-Què vol dir el prefix ^.

El prefix ^ abans del número de versió (^1.6.5) indica que npm permetrà actualitzar a qualsevol versió compatible amb la versió principal especificada.

-Quins altres prefixos podria tenir el número de versió.

- ~: Indica que npm pot actualitzar a qualsevol versió compatible amb el número menor especificat
- \*: Permet qualsevol versió disponible del paquet.
- x o X: Similar a l'asterisc (\*), també permet qualsevol versió. Exemple: "slugify": "1.x" permet qualsevol versió de la sèrie 1.x.x.
- < o >: Permet versions menors o majors a la indicada

**Tasca 4.** Fins ara, hem estat instal·lant els mòduls npm de manera *local*. Com dèiem, les llibreries instal·lades localment es poden trobar a la carpeta *node\_modules* del projecte.

Aquesta carpeta pot arribar a ser -i molt ràpidament- extr emadament voluminosa. Això dificulta la realització de còpies del projecte, la sincronització amb repositoris, etcètera. Fes una còpia del projecte anterior (el que fa servir el mòdul *slugify*) i elimina la carpeta *node\_modules* en el nou projecte.

a)Executa el codi i comprova que el compilador no troba les dependències necessàries.

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1-copi
a>node app.js
node:internal/modules/cjs/loader:1146
  throw err;
  ^

Error: Cannot find module 'slugify'
```

```
code: 'MODULE_NOT_FOUND',
requireStack: [
  "C:\\Users\\kevin\\Desktop\\DAW\\2º\\M8.Desplegament d'aplicacions web\\
UF2\\AC1-copia\\app.js"
]
```

Es pot demanar a *npm* que reconstrueixi en el nou projecte totes les seves dependències.

b)Per fer-ho, fes *npm install*.

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1-copi
a>npm install

added 1 package, and audited 2 packages in 1s

found 0 vulnerabilities
```

c)Quin fitxer informa a *npm* de les dependències del projecte?

El fichero que se encarga de informar a npm de las dependencias del proyecto es el archivo package.json

d) Investiga com fer un *downgrade* de la versió del paquet *slugify*, fes-lo, i observa com canvia la informació sobre les dependències del projecte.

```
12     "dependencies": {
13       "slugify": "^1.6.6"
14     }
15   }
16
```

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1-copi
a>npm install slugify@1.5.0

changed 1 package, and audited 2 packages in 1s
```

Para hacer un downgrade de la version de algun paquete hay que usar el comando `npm install <nombre-del-paquete>@<la-version-deseada>`

```
12     "dependencies": {
13       "slugify": "^1.5.0"
14     }
15   }
16
```

Tasca 5. Els paquets *npm* també es poden instal·lar de manera global. Això és útil per paquets d'eines que es facin servir habitualment a més d'un projecte

a) Instal·la el paquet *cowsay* de manera global, afegint l'opció *-g* a la comanda.

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1-copi
a>npm install cowsay -g

added 41 packages in 3s

3 packages are looking for funding
  run 'npm fund' for details
```

b) Comprova que pots fer servir el paquet *cowsay* des de diferents projectes, tot i que no incloguin les llibreries corresponents de manera local.

```
C:\Users\kevin\Desktop\SpaceInvader>cowsay "Hola, mundo!"

-----
< Hola, mundo! >
-----
      /\
     (oo)\_______
      (__)\       )\/\
           ||----w |
           ||


```

```
C:\Users\kevin\Desktop\EntregaPracticas>cowsay "Adios, mundo cruel!"

-----
< Adios, mundo cruel! >
-----
      /\
     (oo)\_______
      (__)\       )\/\
           ||----w |
           ||


```



c) En quin directori del sistema operatiu es guarden els paquets instal·lats de manera global?

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1-copi  
a>npm root -g  
C:\Users\kevin\AppData\Roaming\npm\node_modules
```

d) Desinstal·la el paquet *cowsay* a nivell global i comprova que han desaparegut els seus arxius font.

```
PS C:\Users\kevin\appdata\roaming\npm\node_modules\cowsay> |
```

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1-copi  
a>npm -g uninstall cowsay  
  
removed 41 packages in 613ms
```

```
C:\Users\kevin\AppData\Roaming\npm\node_modules>cd cowsay  
El sistema no puede encontrar la ruta especificada.  
  
C:\Users\kevin\AppData\Roaming\npm\node_modules>|
```

L'eina *npx* és una alternativa a *npm* que inclou tota una sèrie de funcionalitats no incloses en aquest darrer. Entre elles, la possibilitat de executar (per fer proves) mòduls del repositori *npm* sense necessitat d'instal·lar-los a la màquina local. Només es necessari que els paquets puguin ser invocats des de la línia de comandes.

e) Per exemple, executa *npx cowsay "Hello"*.

```
C:\Users\kevin\AppData\Roaming\npm\node_modules>npx cowsay "hOLi"  
Need to install the following packages:  
cowsay@1.6.0  
Ok to proceed? (y) y  
  
  -----  
< hOLi >  
  -----  
      \      ^__^  
       (oo)\_____  
        (__)\       )\/\  
           ||----w |  
           ||     ||  
  
C:\Users\kevin\AppData\Roaming\npm\node_modules>|
```

f) Comprova que no s'ha instal·lat el paquet, ni local ni globalment.

```
C:\Users\kevin\AppData\Roaming\npm\node_modules>cd cowsay  
El sistema no puede encontrar la ruta especificada.
```

Tasca 6. Un paquet que ja trigueu a instal·lar-vos, si no el teniu ja, és el *nodemon*. Aquest paquet substitueix la comanda *node* que fem servir al executar un *script* JS, permetent que el codi es recarregui en memòria cada cop que pateix canvis.

a) Instal·la *nodemon* a nivell global.

```
PS C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1> npm -g install nodemon  
  
added 29 packages in 5s
```

b) Torna a executar el codi que requeria *http* (tasca 4) amb:

*nodemon app.js*

```
C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament d'aplicacions web\UF2\AC1>node  
mon app.js  
[nodemon] 3.1.9  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node app.js`  
My-new-web-site  
[nodemon] clean exit - waiting for changes before restart
```

Mentre el servidor està en marxa, fes canvis en el codi, i observa com *nodemon* els detecta i recarrega el codi, de manera que els canvis en el codi font es reflecteixen immediatament en el funcionament de l'aplicació.

```
[nodemon] restarting due to changes...  
[nodemon] starting `node app.js`  
My-new-web-site  
Buenas tardes!  
[nodemon] clean exit - waiting for changes before restart
```

c) Compara aquest comportament amb el que es dona en una execució simple amb

*node app.js*

La diferencia en el comportamiento principalmente es que al ejecutar con nodemon, este se irá actualizando según hayan cambios y es bastante útil a la hora de desarrollo.

- *nodemon*: es más adecuado durante el desarrollo, ya que automatiza el reinicio del servidor al detectar cambios en el código, lo que mejora la productividad y facilita las pruebas inmediatas.
- *node*: es útil para ejecutar aplicaciones en entornos más estáticos o de producción, donde los cambios en el código no son frecuentes ni necesitan ser actualizados constantemente.

## Tasca 7. Els usuaris també poden crear els seus propis mòduls.

a) Executa el següent *script* de JavaScript a Node. Pots fer servir el terminal integrat a

VSCode:

```
app.js > ...
1  const person = 'Mike Taylor'
2
3  const greeting = function(person) {
4    console.log(`Hello ${person}, welcome to NodeJS`)
5  }
6
7  greeting(person)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL powershell + -

PS H:\MI unidad\ETPA\_CURS 22\_23\DAW\M8-DESPLIEGAMENT\UF2\node ejemplos\node> node app.js  
Hello Mike Taylor, welcome to NodeJS  
PS H:\MI unidad\ETPA\_CURS 22\_23\DAW\M8-DESPLIEGAMENT\UF2\node ejemplos\node> |

```
PS C:\Users\kevin\Desktop\DAW\2º\M8.Despliegament  
d'aplicacions web\UF2\AC1> node app.js  
Hello Mike Taylor, welcome to NodeJS
```

b) Ara modularitzarem la funció *greeting*: mou la seva declaració a un mòdul (a un altre fitxer *.js*), i afegeix la instrucció d'exportació per aquesta funció:

```
1  const greeting = function(person) {
2    console.log(`Hello ${person}, welcome to NodeJS`)
3  }
4
5  module.exports = greeting;
```

Al fitxer principal *app.js*, requereix aquest mòdul:

```
1  const greeting = require('./greeting.js')
2
3  const person = 'Mike Taylor'
4  greeting(person)
```

```
59  ///
60
61  //a
62  const person = 'Mike Taylor';
63
64  // const greeting = function(person) {
65  //   console.log(`Hello ${person}, welcome to NodeJS`)
66  // }
67
68  // greeting(person)
69
70  //b
71
72  const greeting = require('./greeting.js')
73
74  greeting(person)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS + v ... ^ x

PS C:\Users\kevin\Desktop\DAW\2º\M8.Despliegament  
d'aplicacions web\UF2\AC1> node app.js  
Hello Mike Taylor, welcome to NodeJS  
PS C:\Users\kevin\Desktop\DAW\2º\M8.Despliegament  
d'aplicacions web\UF2\AC1> |

powershell  
powershell

c) Investiga i posa un exemple de com es faria per exportar més d'un objecte d'un mòdul, i de com se'ls requeriria des del fitxer principal.

Se puede exportar a más de un objeto de un módulo haciendo uso de un objeto. Y a este se le llama como un atributo del objeto creado

```
1  const greeting = function(person) {
2    |   console.log(`Hello ${person}, welcome to NodeJS`)
3  }
4
5  const farewell = function(person) {
6    |   console.log(`Goodby ${person}, it surely wasnt a plesure to meet you`)
7  }
8
9  module.exports = {
10   |   greeting,
11   |   farewell
12 };
```

```
62  const person = 'Mike Taylor';
63  const person2 = "Lema"
64
65  // const greeting = function(person) {
66  //   console.log(`Hello ${person}, welcome
67  // }
68
69  // greeting(person)
70
71  //b
72
73  const greeting = require ('../greeting.js')
74
75  greeting.greeting(person)
76  greeting.farewell(person2)
```

PROBLEMAS    SALIDA    CONSOLA DE DEPURACIÓN    TERMINAL    P

```
PS C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament
d'aplicacions web\UF2\AC1> node app.js
Hello Mike Taylor, welcome to NodeJS
Goodby Lema, it surely wasnt a plesure to meet yo
u
PS C:\Users\kevin\Desktop\DAW\2º\M8.Desplegament
d'aplicacions web\UF2\AC1> █
```

d) En aquest exercici estem fent servir la sintaxi dels mòduls *CommonJS*. Investiga la diferència entre aquests i els mòduls *ES6*.

Característica	CommonJS (CJS)	ES6 Modules (ESM)
Palabras clave	require, module.exports, exports	import, export
Sincronía	Sincrónico	Asincrónico
Compatibilidad	Nativo en Node.js	Navegadores y Node.js moderno
Extensiones	.js, .json, etc.	.mjs o .js (con "type": "module")
Scope global	Puede acceder al espacio global	Scope estrictamente modular

### Part B. Introducció a Express

*Express* és el framework estàndard *de facto* per muntar servidors web (és a dir, servidors HTTP) en l'entorn Node.

*Express* es pot fer servir per muntar APIs o per muntar servidors de pàgines web dinàmiques (en anglès *SSR*, *Server-Sider Rendering*). En l'àmbit Python, seria un equivalent a Flask / Django.

En aquesta pràctica, muntarem una senzilla API amb *Express*.

Tasca 8 a) Instal·la el paquet *express* amb l'eina *npm*.

b) Executa el següent codi:

```
1  const express = require('express')
2  const app = express()
3
4  app.listen(5000, () => {
5    console.log('server is listening on port 5000')
6  })
7
8  app.get('/api/products', (req, res) => {
9    res.json([
10     { name: 'iPhone', price: 800 },
11     { name: 'iPad', price: 650 },
12     { name: 'iWatch', price: 750 }
13   ])
14 })
```

i observa com s'arrenca el servidor web.

c) Fes servir l'extensió de Chrome *Talend Tester* (o un altre d'equivalent), per demanar a l'API, pel mètode GET, el llistat de productes. Observa que, tal com es demana en el codi, els resultats es serveixen en format JSON.

<http://localhost:5000/api/products>



d) Observa també com la funció *get* d'*Express* encapsula a alt nivell la resposta del servidor a les peticions pel mètode GET. Com es faria el mateix amb el paquet *http* que hem fet servir abans?

```
97  const http = require('http')
98
99  const server = http.createServer((req, res) => {
100    if (req.method === 'GET' && req.url === '/api/products') {
101      const products = [
102        { name: 'iphone', price: 800 },
103        { name: 'ipad', price: 650 },
104        { name: 'appleWatch', price: 750 }
105      ]
106      res.writeHead(200, { 'Content-Type': 'application/json' })
107      res.end(JSON.stringify(products))
108    } else {
109      res.writeHead(404, { 'Content-Type': 'text/plain' })
110      res.end('Not Found')
111    }
112  })
113
114  server.listen(5000, () => {
115    console.log('Server is listening on port 5000')
116  })
117
```

e)Finalment, mou el llistat de productes a un mòdul separat (*data.js*):

```
const products = [
  { id: 1, name: 'iPhone', price: 800 },
  { id: 2, name: 'iPad', price: 650 },
  { id: 3, name: 'iWatch', price: 750 }
]
```

, exporta'l, i fes que la funció *app.get* del codi principal el faci servir.

Tasca 9. Completa l'API amb els següents *endpoints*, i comproba'ls des del navegador:

a)Endpoint pel mètode *GET* (per servir productes). Substitueix a l'anterior:

```
app.get('/api/products', (req, res) => {
  res.json(products)
})
```

b)Endpoint pel mètode *POST* (per afegir productes mitjançant el cos (*body*) d'un missatge HTTP):

```
app.use(express.json()) // parse json body content

app.post('/api/products', (req, res) => {
  const newProduct = {
    id: products.length + 1,
    name: req.body.name,
    price: req.body.price
  }
  products.push(newProduct)
  res.status(201).json(newProduct)
})
```

(Observa que, a més d'afegir el mètode *post*, hem afegit una altra funció *app.use*. Aquesta funció és part del que es coneix com *middleware* d'Express, i s'aplica a TOTES les peticions HTTP. Les funcions de *middleware*, que tractarem més endavant en el curs, són molt pràctiques per automatitzar operacions en el servidor: en aquest cas, per exemple, parsegen els JSON que arriben en el cos dels missatges HTTP).

c)Endpoint pel mètode *PUT* (per actualitzar un producte):

```
app.use(express.json()) // parse json body content

app.put('/api/products/:productID', (req, res) => {
  const id = Number(req.params.productID)
  const index = products.findIndex(product => product.id === id) if (index ===
-1) {
    return res.status(404).send('Product not found') }
  const updatedProduct = {
```

```

    id: products[index].id,
    name: req.body.name,
    price: req.body.price
  }
  products[index] = updatedProduct
  res.status(200).json('Product updated')
})

```

d) *Endpoint* pel mètode *DELETE* (per esborrar un producte):

```

    app.use(express.json()) // parse json body content

    app.delete('/api/products/:productID', (req, res) => {
      const id = Number(req.params.productID)
      const index = products.findIndex(product => product.id === id) if (index ===
-1) {
        return res.status(404).send('Product not found') }
      products.splice(index,1)
      res.status(200).json('Product deleted')
    })

```