

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`

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!");
```

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
```

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> .		
-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 estandar de Node. Per exemple, anem a introduir un parell de mòduls estàndar que resulten molt útils.

a) Fem servir el mòdul *path*. Explica quin diferent objectiu aconseguen 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)
```

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 conviure distints 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')
```

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

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.

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

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'))
```

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

- Quina és la funcionalitat del paquet.
- Quines opcions ofereix.
- Quina és la seva darrera versió.
- Quin tipus de llicència té.
- Quines dependències té.

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

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

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.
- Què vol dir el prefix ^.
- Quins altres prefixos podria tenir el número de versió.

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- extremadament 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.

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

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

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

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.

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.

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

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

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"*.

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

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.

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

nodemon app.js

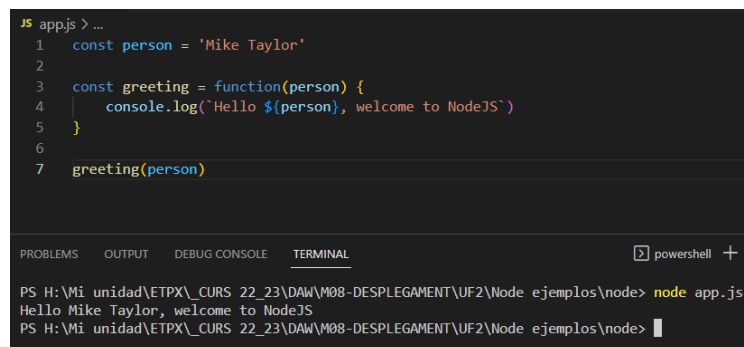
Mentre el servidor està en marxa, fes canvis en el codi, i observa com *nodemon* els detecta i recarrega el codi, de manera que els canvis en el codi font es reflecteixen immediatament en el funcionament de l'aplicació.

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

node app.js

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

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



```
JS 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\ETPX_CURS 22_23\DAW\M08-DESPLEGAMENT\UF2\Node ejemplos\node> node app.js
Hello Mike Taylor, welcome to NodeJS
PS H:\Mi unidad\ETPX_CURS 22_23\DAW\M08-DESPLEGAMENT\UF2\Node ejemplos\node> |

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)
```

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.

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

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.

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?

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')
})
```