

Part A. Introducció a NodeJS

1) Instalar Nodejs

a) Instal·la NodeJS desde <https://nodejs.org/en/>



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

```
C:\Users\aaarat>node -v
v20.18.0
```

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:

```
C:\Users\aaarat>node
Welcome to Node.js v20.18.0.
Type ".help" for more information.
> console.log("Hola món!");
console.log("Hola món!");

Uncaught SyntaxError: Invalid or unexpected token
> |
```

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:

A screenshot showing a file explorer window with a file named 'app.js' and a terminal window. The terminal shows the command 'C: > Users > aaarat > JS app.js' and the output '1 console.log("Hola món!");'.

```
C: > Users > aaarat > JS app.js
1 console.log("Hola món!");
```

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\Actividad 1> node app.js
Hola món!
```

- 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...	Si	
-S'executa en la màquina servidor.		Si
-És possible accedir el DOM.	Si	
-És possible accedir el sistema de fitxers		Si
-Fa servir mòduls ES6	Si	
-Fa servir mòduls CommonJS.		Si
-Fa servir Globals, objectes accessibles a tot el codi, com ara __dirname, __filename, require, module, process...		Si

- 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:

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

data_folder = 'data/';
products_folder = 'products/';
products_file = 'products.json';

const full_path = path.join(data_folder, products_folder, products_file);
console.log(full_path);
```

- El método path.join se usa para unir múltiples rutas seguidas de forma segura y coherente. Se asegura de usar el separador correcto dependiendo del sistema operativo. Si accidentalmente pones varios / seguidos, los corrige automáticamente.
- Solo una parte de la ruta (relativa o absoluta) sin saber donde está trabajando.
- Retorna una ruta relativa

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

data_folder = 'data/';
products_folder = 'products/';
products_file = 'products.json';

const full_path = path.resolve(data_folder, products_folder, products_file);
console.log(full_path);
```

- El método path.resolve crea una ruta absoluta (es decir, que empieza desde la raíz del sistema de archivos).
- Si no hay fragmento absolutos, usa el directorio actual (el lugar donde está ejecutándose el programa) como base.
- Genera una ruta absoluta usando el lugar donde está ejecutando el programa como referencia.
- Retorna una ruta absoluta

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

- path.join: Retorna una ruta relativa

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\Actividad 1> node app.js
data\products\products.json
```

- path.resolve: Retorna una ruta absoluta

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\Actividad 1> node app.js
C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\Actividad 1\data\products\products.json
```

- Ninguno de los códigos usa funciones asíncrones. Solo trabajan con cadenas de texto, no necesita esperar a que algo termine.

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

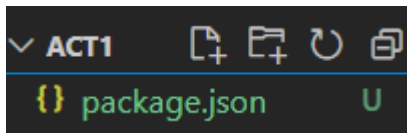
- Función Síncrones
 - El programa espera que termine cada tarea antes de seguir. Todo ocurre en orden. Si un programa toma mucho tiempo, el programa se detiene.
- Función Asíncrona
 - El programa no espera y sigue con otras tareas, aprovechando mejor el tiempo. Cuando la tarea termina, avisan al programa (usando un callback, una promesa o async/await) Son útiles para tareas que pueden tardar (como leer archivos o pedir datos a un servidor)

3) Tasca 3.

- 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:

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1> npm init -y
Wrote to C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1\package.json:
```

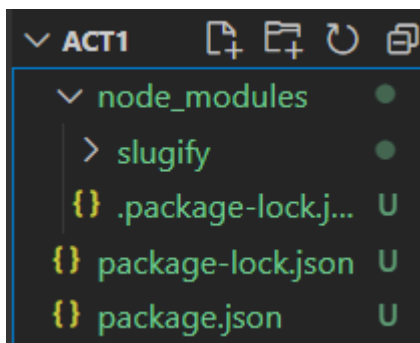
```
{
  "name": "act1",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```



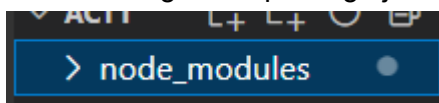
b) A continuació, instal·la el paquet slugify, mitjançant la comanda

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1> npm install slugify
added 1 package, and audited 2 packages in 929ms

found 0 vulnerabilities
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1> 
```



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:

```

JS app.js  U X
JS app.js > ...
1  const slugify = require('slugify')
2
3  console.log(slugify('My New Web Site'));

```

```

PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1> 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.
 - Slugify convierte cadenas de texto en slugs, que son versiones simplificadas y aptas para URLs. Por ejemplo, convierte “Hola Munto!” en “hola-mundo”.
 - Quines opcions ofereix.
 - **lower** = Convierte el slug a minúsculas.
 - **strict** = Elimina caracteres especiales.
 - **locale** = Soporta idiomas específicos, como alemán o español.
 - **customReplacements** = Permite definir reglas personalizadas para reemplazos.
 - Quina és la seva darrera versió
 - Es 1.6.6
 - Quin tipus de llicència té.
 - Slugify utiliza la licencia MIT, que es de uso libre y abierto.
 - Quienes dependències té.
 - Actualmente, el paquete slugify no tiene dependencias externas. Esto 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 slugify`

```

PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1> npm list slugify
act1@1.0.0 C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1
└─ slugify@1.6.6

```

- Actualitzar la versió d'un paquet.
 - `npm update slugify`

```

PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1> npm update slugify
up to date, audited 2 packages in 867ms
found 0 vulnerabilities

```

- Eliminar un paquet.
 - `npm install slugify@latest`
- 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** = Versión principal (major). Cambia si hay modificaciones importantes que no son compatibles con versiones anteriores.
 - **6** = Versión secundaria (minor). Cambia si se añaden nuevas funcionalidades de forma compatible.
 - **5** = Versión de parche (patch). Cambia si se corrigen errores menores.
 - Què vol dir el prefix ^.
 - Indica que puedes usar versiones compatibles. En este caso, permite actualizaciones dentro de la misma versión principal (1.6.6, 1.7.0, etc)
 - Quins altres prefixos podria tenir el número de versió.
 - **~** = Permite actualizaciones solo dentro de la misma versión secundaria (por ejemplo, 1.6.6, pero no 1.7.0).
 - ***** = Permite cualquier versión disponible
 - **Sin prefijo** = solo usa la versión específica mencionada

4) Tasca 4

- 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.

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1 - Copy> node app.js
node:internal/modules/cjs/loader:1228
  throw err;
  ^

Error: Cannot find module 'slugify'
Require stack:
- C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1 - Copy\app.js
  at Module._resolveFilename (node:internal/modules/cjs/loader:1225:15)
  at Module._load (node:internal/modules/cjs/loader:1051:27)
  at Module.require (node:internal/modules/cjs/loader:1311:19)
  at require (node:internal/modules/helpers:179:18)
  at Object.<anonymous> (C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1 - Copy\
app.js:1:17)
  at Module._compile (node:internal/modules/cjs/loader:1469:14)
  at Module._extensions..js (node:internal/modules/cjs/loader:1548:10)
  at Module.load (node:internal/modules/cjs/loader:1288:32)
  at Module._load (node:internal/modules/cjs/loader:1104:12)
  at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:174:12) {
  code: 'MODULE_NOT_FOUND',
  requireStack: [
    'C:\\Users\\aaarat\\OneDrive\\Desktop\\2daw for git\\M08_deplegament\\UF2\\act1 - Copy\\app.js'
  ]
}
```

Node.js v20.18.0

b) Per fer-ho, fes npm install.

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1 - Copy> npm install

added 1 package, and audited 2 packages in 873ms

found 0 vulnerabilities
```

```
▼ ACT1 - ... [L] [F] [C] [B]
  > node_modules ●
  JS app.js      U
  {} package-lock.json U
  {} package.json U
```

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

- El archivo que informa a npm de las dependencias es **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.

- Comprobar la versión actual

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1> npm list slugify
act1@1.0.0 C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1
└─ slugify@1.6.6
```

- Buscar versión anterior el la pagina oficial de npm para slugify
<https://www.npmjs.com/package/slugify?activeTab=versions>

<u>1.5.3</u>	16,822	4 years ago
--------------	--------	-------------

- Instalar una versión anterior especificada

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1 - Copy> npm install slugify@1.5.3

changed 1 package, and audited 2 packages in 756ms

found 0 vulnerabilities
```

5) Tasca 5

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

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1 - Copy> npm install -g cowsay

changed 41 packages in 3s

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

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\act1 - Copy> cowsay "Hola, mundo!"

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

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

- Desde diferente proyecto

```
PS C:\Users\aarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> cowsay "Esto funciona sin instalación local"

< Esto funciona sin instalación local >
-----
      \   ^__^
       \  (oo)\_______
          (__)\       )\/\
             ||----w |
             ||     ||
```

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

- Para saber en que directorio están los paquetes global

```
PS C:\Users\aarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> npm root -g
C:\Users\aarat\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\aarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> npm uninstall -g cowsay

removed 41 packages in 650ms
PS C:\Users\aarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> cowsay "Ya no debería funcionar"
cowsay : The term 'cowsay' is not recognized as the name of a cmdlet, function, script file, or operable
program. Check the spelling of the name, or if a path was included, verify that the path is correct and try
again.
At line:1 char:1
+ cowsay "Ya no debería funcionar"
+ ~~~~~
    + CategoryInfo          : ObjectNotFound: (cowsay:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\aarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project>
```

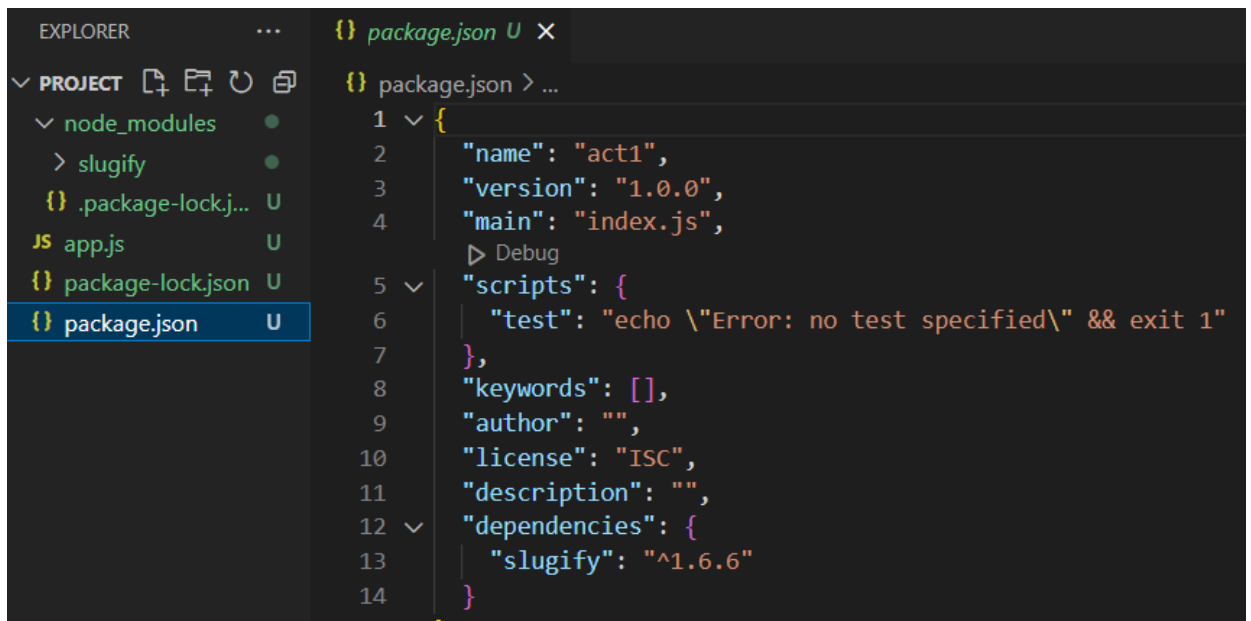
- e) Per exemple, executa npx cowsay "Hello".

```
PS C:\Users\aarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> npx cowsay "Hello"
Need to install the following packages:
cowsay@1.6.0
Ok to proceed? (y) y

< Hello >
-----
      \   ^__^
       \  (oo)\_______
          (__)\       )\/\
             ||----w |
             ||     ||
```

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

- Local

The image shows a VS Code window with the Explorer sidebar on the left and the package.json file open in the editor. The Explorer sidebar shows a project structure with 'node_modules' expanded, showing 'slugify', '.package-lock.j...', 'app.js', 'package-lock.json', and 'package.json'. The 'package.json' file is selected and its content is displayed in the editor. The content of package.json is: { "name": "act1", "version": "1.0.0", "main": "index.js", "scripts": { "test": "echo \"Error: no test specified\" && exit 1" }, "keywords": [], "author": "", "license": "ISC", "description": "", "dependencies": { "slugify": "^1.6.6" } }

```
EXPLORER  ...  {} package.json U X
PROJECT  [?] [?] [?] [?]
  node_modules  ●
    > slugify  ●
    {} .package-lock.j... U
    JS app.js U
    {} package-lock.json U
    {} package.json U

{} package.json > ...
1 {
2   "name": "act1",
3   "version": "1.0.0",
4   "main": "index.js",
5   > Debug
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "description": "",
13  "dependencies": {
14    "slugify": "^1.6.6"
15  }
16 }
```

- Global

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> npm list -g cowsay
C:\Users\aaarat\AppData\Roaming\npm
└─ (empty)
```

6) Tasca 6.

a) Instal·la nodemon a nivell global. y verificar

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> npm install -g nodemon
added 29 packages in 3s

4 packages are looking for funding
  run `npm fund` for details
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> nodemon --version
3.1.9
```

b) Torna a executar el codi que requeria http (tasca 4) amb: nodemon app.js

```
const slugify = require('slugify')

console.log(slugify('My New Web Site'));

const slugify = require('slugify')

console.log(slugify('Holaa'));
```

```

PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> nodemon 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
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Holaa
[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
- **node app.js** = Requiere un reinicio manual del servidor para aplicar cambios.
 - **nodemon app.js** = Se encarga de la recarga automática del servidor cuando detecta cambios en el código.

7) 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)

```

```

PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> 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),

```

s  U      JS greeting.js U X
ting.js > ...
const greeting = function (person) {
  console.log(`Hello ${person}, Welcome to NodeJS`);
};

module.exports = greeting;

```

i afageix la instrucció d'exportació per aquesta funció:

```
app.js  U X  JS greeting.js U
$ app.js > ...
1  const greeting = require('./greeting.js')
2  const person = 'Mike Taylor'
3  greeting(person)

PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> node app.js
Hello Mike Taylor, Welcome to NodeJS
```

- 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.

- greeting.js:

```
.js  U  JS greeting.js U X
eeting.js > ...
const greeting = function (person) {
  console.log(`Hello ${person}, Welcome to NodeJS`);
};

const farewell = function(person) {
  console.log(`Goodbye ${person}, see you next time!`);
};

module.exports = { greeting, farewell };
```

- app.js

```
JS app.js  U X  JS greeting.js U
JS app.js > ...
1  const { greeting, farewell } = require('./greeting.js')
2  const person = 'Mike Taylor'
3
4  greeting(person)
5  farewell(person);

PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> node app.js
Hello Mike Taylor, Welcome to NodeJS
Goodbye Mike Taylor, see you next time!
```

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

- **CommonJS** = son específicos de NodeJS y utilizan **module.exports** y **require**.

```
// greeting.js
module.exports = function(person) {
  console.log(`Hello ${person}`);
};

// app.js
const greeting = require('./greeting');
greeting('Mike');
```

- **módulos ES6** = tienen una sintaxis más moderna con export e import, y son más utilizados en aplicaciones front-end y también soportados en NodeJS moderno.

```
// greeting.js
export function greeting(person) {
  console.log(`Hello ${person}`);
};

// app.js
import { greeting } from './greeting.js';
greeting('Mike');
```

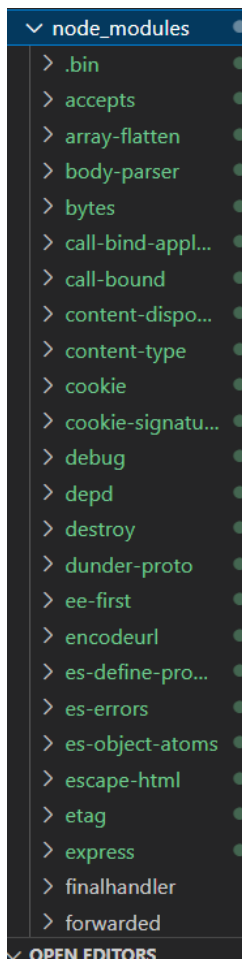
8) Tasca 8

- Instal·la el paquet express amb l'eina npm.

```
PS C:\Users\aarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> npm install express
added 69 packages, and audited 71 packages in 3s

14 packages are looking for funding
  run `npm fund` for details

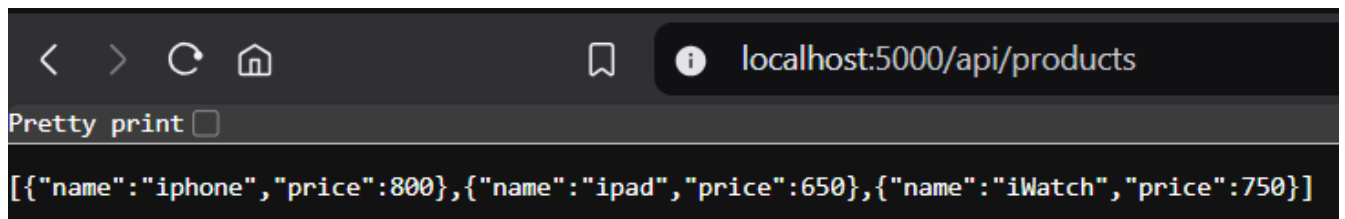
found 0 vulnerabilities
```



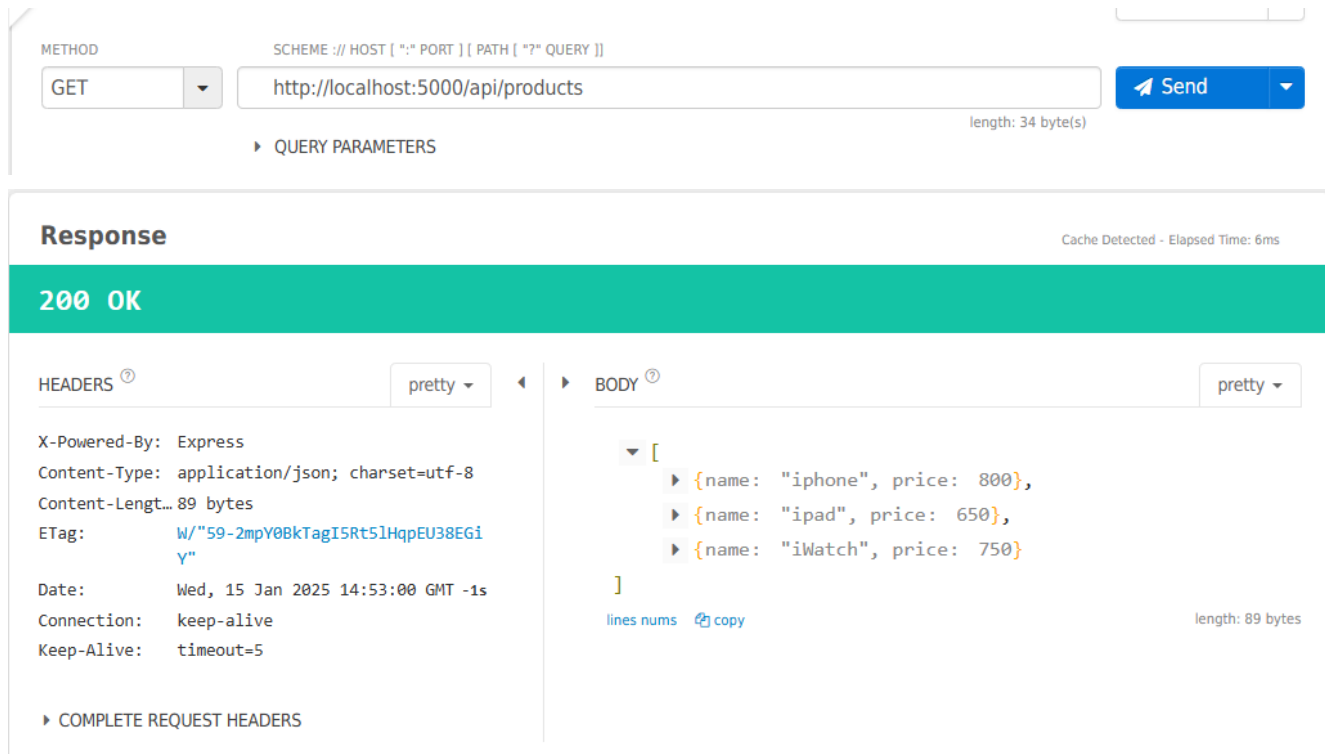
b) Executa el següent codi:

```
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 });
```

```
PS C:\Users\aaarat\OneDrive\Desktop\2daw for git\M08_deplegament\UF2\project> node app.js
server is listening on port 5000
█
```



- 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`):

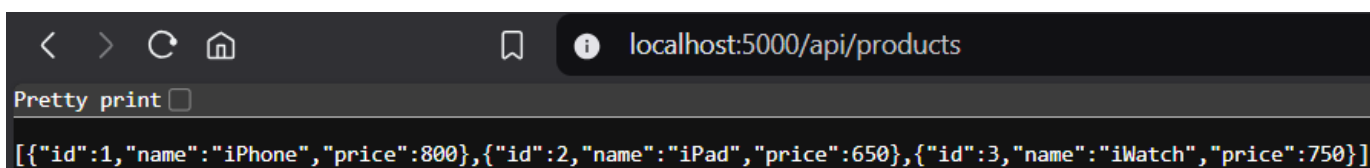
```
JS data.js U X
a.js > ...
const products = [
  { id: 1, name: 'iPhone', price: 800 },
  { id: 2, name: 'iPad', price: 650 },
  { id: 3, name: 'iWatch', price: 750 }
]

module.exports = products;
```

9) 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:

```
JS app.js U X JS data.js U
JS app.js > ...
1  const express = require('express');
2  const products = require('./data');
3
4  const app = express();
5
6  app.listen(5000, () => {
7    console.log('server is listening on port 5000');
8  });
9
10 app.get('/api/products', (req, res) => {
11   res.json(products);
12 });
```



- Aparece con el id

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

```

const express = require('express');
const products = require('./data.js');

const app = express();

app.listen(5000, () => {
  console.log('server is listening on port 5000');
});

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

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)
})

```

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

```

// Endpoint pel mètode PUT (per actualitzar un producte):
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):

```
// Endpoint pel mètode DELETE (per esborrar un producte):
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')
})
```