

ACTIVITAT AVALUABLE AC1**Mòdul:** MP08- Desplegament d'aplicacions web**UF:** UF3 – Aplicaciones web**Professor:** Albert Guardiola**Data límit d'entrega:** 05/02/2023 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-UF3-AC1-Nom_Alumne.doc (o pdf)***

Es valorarà la presentació.

Resultats de l'aprenentatge:

RA 1. Verifica l'execució d'aplicacions web comprovant els paràmetres de configuració de serveis de xarxa.

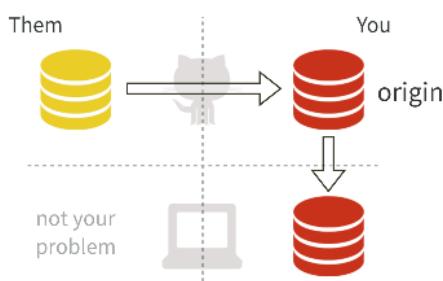
En esta práctica exploraremos algunos aspectos importantes del despliegue online de un proyecto. Para ello, utilizaremos el proyecto en NodeJS con el que veníamos trabajando en la UF2, que proporciono completo en un repositorio.

Es importante, para el correcto desempeño de la práctica, documentar todos los pasos del proceso.

Despliegue del proyecto en local

Tarea 1. Realiza un **fork** del siguiente repositorio a tu GitHub, y clona éste a tu máquina local.

https://github.com/albertetpx/productsapi_UF3AC1



"fork and clc

[productsapi_UF3AC1](#) Public

Forked from albertetpx/productsapi_UF3AC1

JavaScript 6 Updated yesterday

El proyecto que encontrarás en el repositorio necesita de dos pasos previos antes de poder ser ejecutado en local.

- a. Dado que utiliza una conexión a MongoDB, tendrás que configurar una cuenta, un clúster y una base de datos en el *cloud Atlas* de MongoDB (tarea 2).
- b. Dado que no se incluyen las dependencias (módulos de *Node*) en el repositorio, se deberán instalar manualmente mediante *npm* (tarea 3).

Tarea 2. Sigue las instrucciones que encontrarás en el enlace siguiente para crear la base de datos en Atlas.

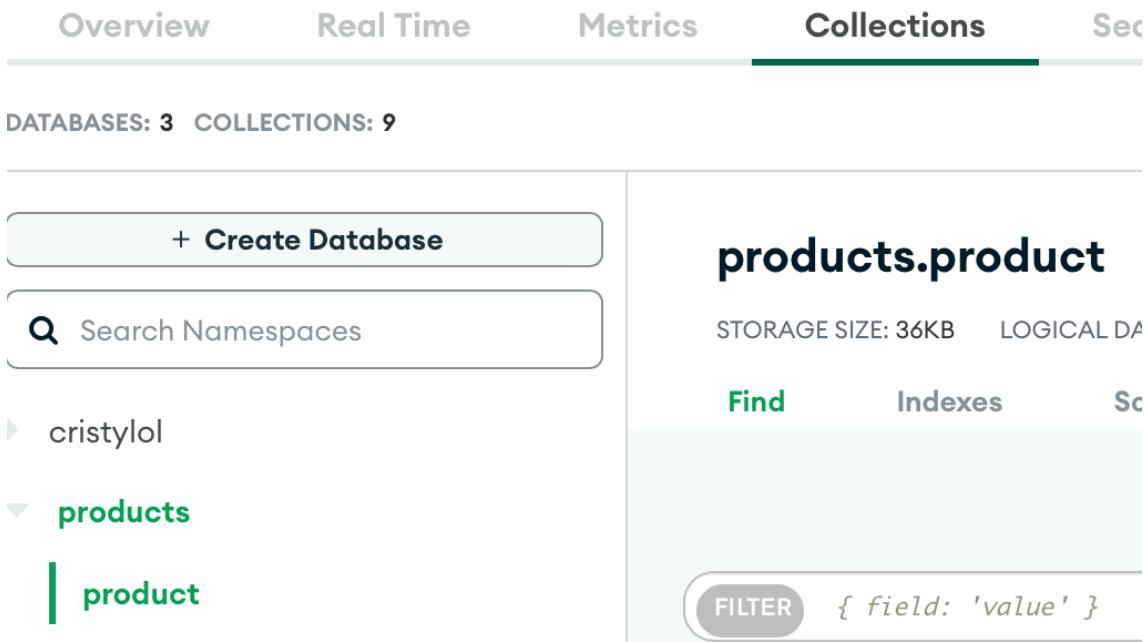
What is MongoDB Atlas?

MongoDB Atlas is a multi-cloud database service by the same people that build MongoDB. Atlas simplifies deploying and managing your databases while offering the versatility you need to build resilient and performant global applications on the cloud providers of your choice.

<https://www.mongodb.com/docs/atlas/getting-started/>

Ten en cuenta que, para que nuestra aplicación pueda utilizar el clúster debes asegurarte de hacer las siguientes operaciones:

- a. Crear un base de datos llamadas *products*. Las colecciones que contenga (el análogo a las tablas en el modelo relacional), las creará la propia aplicación.



The screenshot shows the MongoDB Atlas interface with the 'Collections' tab selected. At the top, it displays 'OVERVIEW' and 'REAL TIME' tabs. Below that, it shows 'DATABASES: 3' and 'COLLECTIONS: 9'. On the left, there's a sidebar with a '+ Create Database' button and a search bar for namespaces. The main area lists databases: 'cristylol' and 'products'. The 'products' database is expanded, showing its collections: 'product'. The 'product' collection is highlighted, showing its details: 'STORAGE SIZE: 36KB' and 'LOGICAL DA'. Below the collection details, there are 'Find', 'Indexes', and 'Sc' buttons. At the bottom of the collection view, there's a 'FILTER' button with the placeholder '{ field: 'value' }'.

- b. **Crear un nuevo usuario.** Es recomendable que la contraseña asociada la genere automáticamente la UI de Atlas. Asegúrate de que el nuevo usuario creado dispone de todos los permisos sobre la base de datos.
- c. **Añadir la IP desde la que vas a trabajar a la whitelist del clúster** (en la UI de Atlas: *Data Services/Security/Network Access*). Puedes saber la IP pública desde la que te estas conectando con esta herramienta, por ejemplo: <https://www.whatismyip.com/es/>.

IP Address	Comment	Status
37.29.187.160/32 (includes your current IP address)	Institut	● Active
0.0.0.0/0 (includes your current IP address)	My IP Address	● Active

-Finalmente, añade la URI de la conexión (la UI de Atlas: *Data Services/Database/Connect*) al fichero */config/config.js* del proyecto.

```
module.exports = {
  mongoURI: "mongodb+srv://admin:shadowesp2@cluster0.juajh.mongodb.net/?retryWrites=true&w=majority"
};
```

Tarea 3. Como puedes observar, el repositorio no contiene la carpeta de dependencias *node_modules*. Esta es una buena práctica en tiempo de desarrollo: ¿para qué subir los paquetes de dependencias, que son de largo los más pesados del proyecto, al repositorio (o a Clickedu...), si puede reconstruirse a partir del fichero *package.json*?

- a) Observa cómo hemos evitado que las dependencias se añadan al control de versiones añadiendo la carpeta *node_modules* al fichero *.gitignore*.

```
node_modules
views/styles/styles.css
```

- b) Observa cómo las dependencias del proyecto están listadas en el fichero `package.json`. Es importante tener este archivo *mantenido*. Si dejamos de utilizar un paquete, hay que *desinstalarlo* para que desaparezca tanto de nuestro directorio de dependencias como del fichero `package.json`.

```
{  
  "name": "productsapi",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.18.2",  
    "mongodb": "^4.14.0",  
    "pug": "^3.0.2"  
  }  
}
```

- c) Instala las dependencias con `npm install`.

```
productsapi_UF3AC1 git:main ▾  
› npm install  
  
added 190 packages, and audited 191 packages in 30s  
  
16 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

- d) Ejecuta la aplicación (p.ej. `nodemon app.js`) y comprueba que la página de bienvenida de la API se sirve en <http://localhost:5000/api>

Verás que en la página no aplican estilos. Nos ocuparemos de esto más adelante.

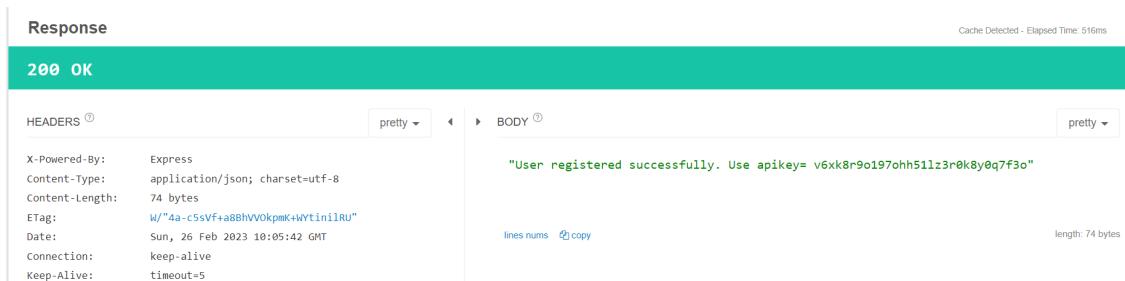
- e) Documenta el correcto funcionamiento de la aplicación:

-Registra un nuevo usuario (observa el formato del JSON):



METHOD: POST
 SCHEME: // HOST [":" PORT] [PATH ["?" QUERY]]
 http://localhost:5000/users/register
 length: 36 byte(s)
 Send
 QUERY PARAMETERS
 HEADERS
 api-key: y3aco8x71cif2vt0hj5uh
 Content-Type: application/json
 BODY
 Text
 "name": "apiuser01",
 "password": "pass09876"

-Verás que en la respuesta HTTP se proporciona una apikey:



Response
 200 OK
 Cache Detected - Elapsed Time: 516ms
 HEADERS
 pretty
 X-Powered-By: Express
 Content-Type: application/json; charset=utf-8
 Content-Length: 74 bytes
 ETag: W/"4a-c5svf+a8BhVVOkpmK+Hytini1RU"
 Date: Sun, 26 Feb 2023 10:05:42 GMT
 Connection: keep-alive
 Keep-Alive: timeout=5
 BODY
 pretty
 "User registered successfully. Use apikey= v6xk8r9o197ohh511z3r0k8y0q7f3o"
 lines num: copy
 length: 74 bytes

-Crea algunos productos de prueba:

```
productsapi_UF3AC1 git:main %
> curl -X POST http://localhost:5000/users/register/ -H "Content-Type: application/json" -d '{"name":"apiuser01","password":"123"}'
"User registered successfully. Use apikey= zyemtq5s4fptwrunj16fdxasgfm2pn"%
```

apikey=zyemtq5s4fptwrunj16fdxasgfm2pn

```
productsapi_UF3AC1 git:main %
> curl -X POST http://localhost:5000/users/register/ -H "Content-Type: application/json" -d '{"name":"joe","password":"123"}'
"User registered successfully. Use apikey= cxfw3g00fi24uqmilj3j0msy2i3i9c"%
```

apikey= cxfw3g00fi24uqmilj3j0msy2i3i9c

METHOD: POST SCHEME // HOST [":" PORT] | PATH ["?" QUERY]
 http://localhost:5000/api/products Send
 length: 34 byte(s)

QUERY PARAMETERS

HEADERS: Form
 api-key : v6xk8r9o197ohh51z3r0k8y0q7f3o
 Content-Type : application/json
 + Add header Add authorization

BODY: Text

```
1 {
2   "id": "1",
3   "name": "Rumba4000",
4   "price": 600
5 }
```

-Consulta la lista de productos

```
productsapi_UF3AC1 git:main ▾
> curl http://localhost:5000/api/products\?apikey=cxfw3g00fi24uqmilj3j0msy2i3i9c
[]%
```

f) Finalmente, sube los cambios del proyecto **a tu repositorio**. Sólo debería haber cambiado el *config.js* (del código fuente). Asegúrate de que no en el *.gitignore* siguen estando la carpeta de dependencias y el fichero CSS.

Rutas del .gitignore

```
2 node_modules
1 views/styles/styles.css
```

```
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
  modified: config/config.js
  new file: package-lock.json
  modified: views/index.pug
```

-Despliegue del proyecto online (Render)

Tarea 4. A continuación, desplegaremos el proyecto *online*. Una buena solución para un aplicación NodeJS sería utilizar el *hosting* de Heroku, pero recientemente han retirado su *free tier*. Por ello, utilizaremos una alternativa parecida:

<https://render.com/>

Sigue las instrucciones en este *post* para crear una cuenta en *render.com*, conectarla con tu cuenta de *GitHub* y desplegar el último *commit* realizado.

<https://www.freecodecamp.org/news/how-to-deploy-nodejs-application-with-render/>

Create a new Web Service

Connect your Git repository or use an existing public repository URL.

Connect a repository

User / Repository	Last updated	Connect
joelforworks / productsapi_UF3AC1	8 minutes ago	Connect
joelforworks / layout-example	a month ago	Connect
joelforworks / cobeer	2 months ago	Connect
joelforworks / m8ac7_python	3 months ago	Connect
joelforworks / m8ac7	3 months ago	Connect
joelforworks / problems	4 months ago	Connect

 GitHub
[@joelforworks](#) • 9 repos

 [Configure account](#)

 GitLab
[+ Connect account](#)

Durante el proceso de instalación, asegúrate de:

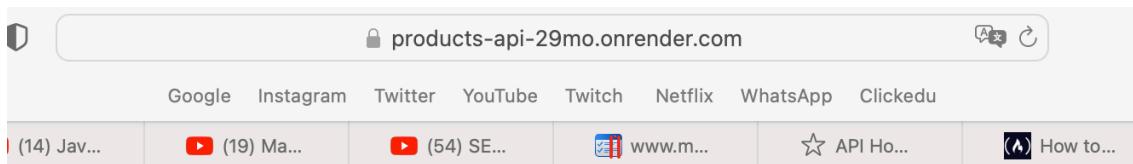
- a) Configurar como comando de compilación (*build command*): *npm install*, ya que las dependencias no se incluyen en el repositorio y deben instalarse en tu sesión del *hosting*.
- b) Configurar como comando de arranque (*start command*):

```
node --unhandled-rejections=strict app.js
```

```
$ node --unhandled-rejections=strict app.js
```

La opción `Unhandled-rejection=strict` soluciona algunos problemas derivados del uso que hacemos del paquete `mongodb` en relación con la versión de NodeJS que hay instalada en el hosting.

- c) Observa como, después de configurar el nuevo servicio web, se despliega (*deploy*) automáticamente el proyecto del repositorio conectado, en la versión del último *commit* (si has marcado la opción `autodeploy` durante el proceso de configuración). Captura el *log* del proceso de *deployment*.
- d) Comprueba el funcionamiento de la api: lo normal sería que pudieras acceder a la página de bienvenida (sin estilos), pero que no pudieras utilizar correctamente el resto de *endpoints*.



Wellcome to Products API

Register to get API key!

(to provide it, place it on header 'api-key' or as URL param 'apikey')

Operation	API endpoint
Register	POST /api/users/register
List all products	GET /api/products
List products by ID	GET /api/products/:productID
Create product	POST /api/products
Update product	PUT /api/products/:productID
Delete product	DELETE /api/products/:productID

- e) Para que éstos funcionen, deberás añadir las IPs del servicio web a la *whitelist* de Atlas. Puedes saber cuáles son las IPs públicas del servicio web en el botón *Connect* del panel de control principal del servicio.

Static Outbound IP Addresses

Network requests from your service to the public internet will come from one of the following IP addresses.



White list atlas

3.125.183.140/32	render2
3.75.158.163/32	render1
35.157.117.28/32	render3

- f) Documenta el correcto funcionamiento de la API, de manera exhaustiva.

Create user endpoint

```
j curl -X POST https://products-api-29mo.onrender.com/users/register/ -H "Content-Type: application/json" -d '{"name": "joelRender", "password": "123"}'  
User registered successfully. Use apikey= mtyij7squxgbk7ct9d193amlkg2nn6
```

```
_id: ObjectId('6404cd02d73fff63558bc892')  
name: "joelRender"  
password: "123"  
apiKey: "mtyij7squxgbk7ct9d193amlkg2nn6"
```

Create product endpoint

```
j curl -X POST https://products-api-29mo.onrender.com/api/products/?apikey=mtyij7squxgbk7ct9d193amlkg2nn6 -H "Content-Type: application/json" -d '{"name": "productFromRender", "price": "123"}'  
"Product created successfully"
```

```
_id: ObjectId('6404cdf0d73fff63558bc893')
id: NaN
name: "productFromRender"
price: "123"
```

A background image of a young girl with short brown hair, wearing a dark blue school uniform with a white collar and a light blue bow tie. She is looking slightly to her left with a neutral expression. The background is a plain, light color.

```
> curl -X POST https://products-api-29mo.onrender.com/api/products\?apikey\=mtyij7squxgbk7ct9d193amlkg2nn6 -H "Content-Type: application/json" -d '{"name": "productFromRender2", "price": "123"}'
"Product created successfully"\n
~
> curl -X POST https://products-api-29mo.onrender.com/api/products\?apikey\=mtyij7squxgbk7ct9d193amlkg2nn6 -H "Content-Type: application/json" -d '{"name": "productFromRender3", "price": "1000"}'
"Product created successfully"\n
~
> curl -X POST https://products-api-29mo.onrender.com/api/products\?apikey\=mtyij7squxgbk7ct9d193amlkg2nn6 -H "Content-Type: application/json" -d '{"name": "productFromRender4", "price": "2222"}'
"Product created successfully"\n
```

```
_id: ObjectId('6404cec1d73fff63558bc894')
id: NaN
name: "productFromRender2"
price: "123"
```

```
_id: ObjectId('6404cecbd73fff63558bc895')
id: NaN
name: "productFromRender3"
price: "1000"
```

```
_id: ObjectId('6404ced7d73fff63558bc896')
id: NaN
name: "productFromRender4"
price: "2222"
```

Product with id

```
> curl -X POST https://products-api-29mo.onrender.com/api/products?apikey=mtyij7squxgbk7ct9d193amlkg2nn6 -H "Content-Type: application/json" -d '{"id": "123", "name": "productFromRender4", "price": "2222"}'  
"Product created successfully"
```

```
_id: ObjectId('6404cf75d73fff63558bc897')  
id: 123  
name: "productFromRender4"  
price: "2222"
```

GET ALL PRODUCTS ENDPOINT

```
> curl https://products-api-29mo.onrender.com/api/products?apikey=mtyij7squxgbk7ct9d193amlkg2nn6  
[{"id": "6404cf0d73fff63558bc893", "id": null, "name": "productFromRender", "price": "123"}, {"id": "6404cecd73fff63558bc894", "id": null, "name": "productFromRender2", "price": "123"}, {"id": "6404cecb73fff63558bc895", "id": null, "name": "productFromRender3", "price": "1000"}, {"id": "6404ced7d73fff63558bc896", "id": null, "name": "productFromRender4", "price": "2222"}, {"id": "6404cf75d73fff63558bc897", "id": 123, "name": "productFromRender4", "price": "2222"}]
```

GET PRODUCT BY ID ENDPOINT

```
> curl https://products-api-29mo.onrender.com/api/products/123?apikey=mtyij7squxgbk7ct9d193amlkg2nn6  
[{"_id": "6404cf75d73fff63558bc897", "id": 123, "name": "productFromRender4", "price": "2222"}]
```

UPDATE PRODUCT ENDPOINT

```
> curl -X PUT https://products-api-29mo.onrender.com/api/products/123?apikey=mtyij7squxgbk7ct9d193amlkg2nn6 -H "Content-Type: application/json" -d '{"name": "productUpdated", "price": "200"}'  
"Product updated successfully"
```

```
_id: ObjectId('6404cf75d73fff63558bc897')  
id: 123  
name: "productUpdated"  
price: "200"
```

Tarea 5. El hecho de que el web service se compile y despliegue automáticamente desde el repositorio de *GitHub* facilita enormemente el proceso de actualización del proyecto. Hagamos un cambio tonto para tener claro el proceso:

- a) Instala en el proyecto de tu máquina local la dependencia *slugify*. Observa como se ha añadido al *package.json*.

```
productsapi_UF3AC1 git:main
> npm install slugify

added 1 package, and audited 192 packages in 1s

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

found 0 vulnerabilities
```

- b) Usa *slugify* para *slugizar* con asteriscos el mensaje de bienvenida del servidor Express (en *app.js*).

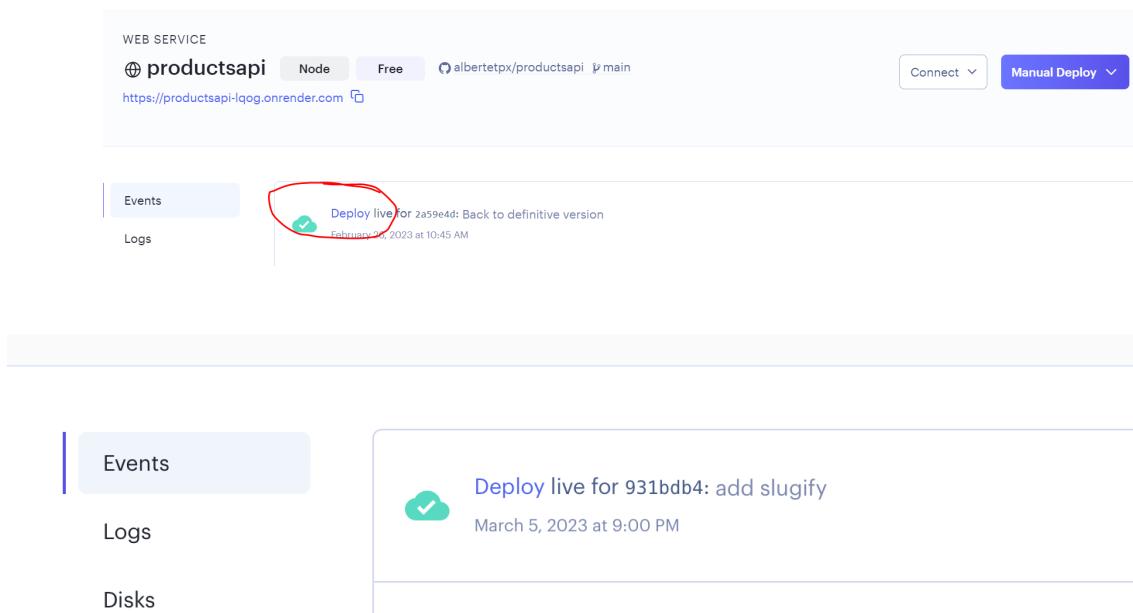
```
1 // Bienvenida
2 const welcome = slugify('WELCOME TO THE SERVER', '*');
3
4
5 //Server startup
6 app.listen(5000, () => {
7   console.log(welcome);
8   console.log(`server is listening on port ${PORT}`);
9 })
```

- c) Sube cambios a tu repositorio e *GitHub*.

```
productsapi_0F3AC1 git:main
> git commit -m "add slugify"
[main 867e4e9] add slugify
  3 files changed, 20 insertions(+), 2 deletions(-)

productsapi_UF3AC1 git:main
> git push
Upstream fast-forwarded to https://github.com/alberttpx/productsapi
```

- d) En *render*, comprueba cómo se lanza automáticamente el despliegue del *commit* que acabas de realizar. Puedes acceder a las trazas del proceso de compilación y despliegue clicando sobre el *deploy* que toque en el *dashboard* de tu servicio web:



The screenshot shows the Render dashboard for a service named 'productsapi'. At the top, it displays the service name, Node.js runtime, and a 'main' branch. Below this are 'Connect' and 'Manual Deploy' buttons. The main area has tabs for 'Events' (selected) and 'Logs'. Under 'Events', there is a log entry: 'Deploy live for 2a59e4d: Back to definitive version' dated 'February 26, 2023 at 10:45 AM'. A red circle highlights this entry. Below this, under 'Logs', there is another log entry: 'Deploy live for 931bdb4: add slugify' dated 'March 5, 2023 at 9:00 PM'. A red circle highlights this entry.

- e) Accede a la consola del servicio web (sección *logs* del *dashboard*) para comprobar que se han aplicado los cambios correctamente (*slugificación* del *prompt* de bienvenida del servidor Express).

```
Mar 5 08:58:29 PM [4/4] Building fresh packages...
Mar 5 08:58:29 PM success Saved lockfile.
Mar 5 08:58:29 PM Done in 12.00s.
Mar 5 08:58:29 PM => Generating container image from build. This may take a few minutes...
Mar 5 08:59:35 PM => Uploading build...
Mar 5 08:59:51 PM => Build uploaded in 12s
Mar 5 08:59:51 PM => Build successful ✨
Mar 5 08:59:51 PM => Deploying...
Mar 5 09:00:24 PM => Starting service with 'node --unhandled-rejections=strict app.js'
Mar 5 09:00:27 PM WELCOME*TO*THE*SERVER
Mar 5 09:00:27 PM server is listening on port 5000
```

-Automatización de la compilación

Tarea 6. Habitualmente, el proceso de despliegue de una aplicación web habrá de venir precedido no solo de la instalación de las dependencias, sino de la compilación de ciertos ficheros. Estos son algunos ejemplos:

- Compilación de SCSS a CSS.
- Compilación de *frameworks JS*: REACT, Vue...
- Transpilación de Typescript a Javascript (en Angular, por ejemplo).
- Post-procesado de CSS (autoprefixing, por ejemplo).
- Minificación de ficheros.
- Linting de código.

En esta ocasión, necesitamos automatizar el proceso de compilación de CSS a partir de ficheros SASS. Para ello:

- a) Instala el paquete *node-sass* en tu proyecto local, como depedencia de desarrollo: *npm install --save-dev node-sass*. Observa como se añade al *package.json*.

```
productsapiclient git:main
> npm install --save-dev node-sass
npm WARN deprecated @npmcli/move-file@1.1.2: This functionality has been moved to @npmcli/fs
npm WARN deprecated @npmcli/move-file@2.0.1: This functionality has been moved to @npmcli/fs
added 206 packages, and audited 398 packages in 60s
32 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

```
"devDependencies": {
  "node-sass": "^8.0.0"
}
```

- b) Lanza la compilación de estilos mediante *npx*:

```
npx node-sass ./views/styles/main.scss ./views/styles/styles.css
```

```
productsapiclient git:main ▾ 60s
> npx node-sass views/styles/main.scss views/styles/styles.css
Rendering Complete, saving .css file...
Wrote CSS to /Users/joel/Desktop/FP/2/M8/MP08_UF3_AC1/productsapiclient/views/styles/styles.css
```

- c) Observa cómo se genera el fichero de estilos. Ahora la página de bienvenida de la API aparecerá correctamente estilada.



- d) Convertiremos ahora el comando de compilación de estilos en un *script npm*. Para ello, añade una nueva clave en el apartado de *scripts* del *package.json*:

```
"build:css": "node-sass ./views/styles/main.scss ./views/styles/styles.css"
```

```
"test": "echo \"Error: no test specified\" && exit 1",
"build:css": "node-sass ./views/styles/main.scss ./views/styles/styles.css"
```

- e) Haz algún cambio en el SASS y recompila los estilos, esta vez usando el script npm: `npm run build:css`.

```
productsapi_UF3AC1 git:main ✘
> npm run build:css
> productsapi@1.0.0 build:css
> node-sass ./views/styles/main.scss ./views/styles/styles.css
Rendering Complete, saving .css file...
Wrote CSS to /Users/joel/Desktop/FP/2/M8/MP08_UF3_AC1/productsapi_UF3AC1/views/styles/styles.css
```

- f) Finalmente, empaquetaremos las dos operaciones (instalación de dependencias y compilación de estilos) en un único *script npm*:

```
"build": "npm install && npm run build:css",
```

```
built.css . node-sass ./views/styles/main.s
"build": "npm install && npm run build:css"
```

- g) Actualiza el comando de compilación (*build command*) en los *settings* del servicio web en render a: *npm run build*.

```
> npm run build
> productsapi@1.0.0 build
> npm install && npm run build:css

up to date, audited 398 packages in 595ms

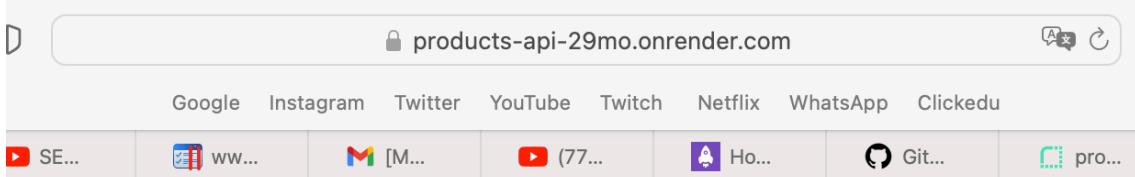
32 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

> productsapi@1.0.0 build:css
> node-sass ./views/styles/main.scss ./views/styles/styles.css

Rendering Complete, saving .css file...
Wrote CSS to /Users/joel/Désktop/FP/2/M8/MP08_UF3_AC1/productsapi_UF3AC1/views/styles/styles.css
```

- h) Sube los campos al repo, observa cómo se lanza un nuevo despliegue en render, y como, ahora sí, se genera el fichero de estilos (que, recordemos, NO SE SUBE AL REPO), y la página de bienvenida de la API *online* aparece estilada correctamente.



Wellcome to Products API

Register to get API key!

(to provide it, place it on header 'api-key' or as URL param 'apikey')

Operation	API endpoint
Register	POST /api/users/register
List all products	GET /api/products
List products by ID	GET /api/products/:productID
Create product	POST /api/products
Update product	PUT /api/products/:productID
Delete product	DELETE /api/products/:productID

En el siguiente enlace puedes ver otros *scripts npm* útiles para la automatización de procesos de despliegue: <https://css-tricks.com/why-npm-scripts/>

```

1 > productsapi@1.0.0 build /opt/render/project/src
1 > npm run build:css && node --unhandled-rejections=strict app.js
1
1
1 > productsapi@1.0.0 build:css /opt/render/project/src
1 > node-sass ./views/styles/main.scss ./views/styles/styles.css
1
1 Rendering Complete, saving .css file...
1 Wrote CSS to /opt/render/project/src/views/styles/styles.css
1 WELCOME*TO*THE*SERVER
1 server is listening on port 10000

```

Ocultación de secretos:

Tarea 7. Lo más probable es que *GitHub* te haya avisado, mediante un correo, de que existe en tu repositorio una vulnerabilidad de seguridad. Concretamente, de que la URI de la conexión a la base de datos de MongoDB está expuesta, en texto plano, en el repositorio. Esto es gravísimo.

- Vamos a convertir la URI a una variable de entorno para que no quede almacenada en el código. Para ello, desde la terminal integrada de VSCode, usa:

`$ENV:MONGOURI='tuURI'`

He en puesto en el apartado `environment` de render una variable MONGOURI

- Ahora, haz que en el fichero `config.js` se lea esta URI desde la variable de entorno:

`process.env. MONGOURI;`

```

module.exports = {
    mongoURI: process.env.MONGOURI
};

```

- Comprueba que la API sigue pudiendo conectar correctamente a la base de datos.

ESTA PARTE NO ME FUNCIONA EN EL SERVIDOR PERO SI EN LOCAL

- d) En el panel de control del servicio web en render, en la sección *Environment*, añade la variable de entorno a la sesión del *hosting*.
- e) Sube los cambios locales al repositorio, observa como se lanza otro *deploy*, y comprueba que la API *online* sigue funcionando. La URI de MongoDB ha dejado de estar expuesta en el repositorio.

Esta manera de trabajar con variables de entorno no es la óptima. Existen algunos paquetes (p.ej. *dotenv*) que automatizan la configuración en el entorno de estas variables a partir de ficheros dedicados de extensión *.env*, que OBVIAMENTE, debenadirse al *.gitignore*, para no quedar expuestos en el repo, y subirse a mano al *hosting*. Investiga en qué lugar se suben este tipo de ficheros en el caso de render.