Práctica 2 - PTI

Web APIs

Arnau Yepes Huguet

Natalia Dai

Grupo 1

26/02/2023

**Índice**

[**1. -Introducción**](#_57r2ajec7c81) **2**

[**2. -Alternativas para resolverla**](#_mnk3vuovdg75) **3**

[**3. -New Rental**](#_7luim73vgxhb) **3**

[**4. -List Rentals**](#_6225uqc7vhth) **5**

[**5. -Dockerization**](#_42amrznusima) **6**

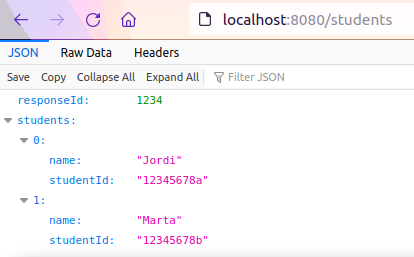
[**6. -CI/CD**](#_60h3ej1ozyn0) **6**

# **1. -Introducción**

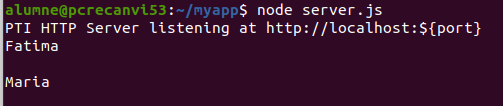
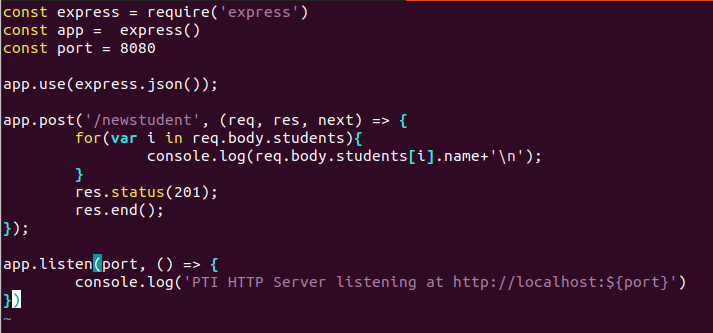
En esta sesión desarrollaremos una Web API sencilla con Node.js, de alquiler de coches como la sesión pasada.

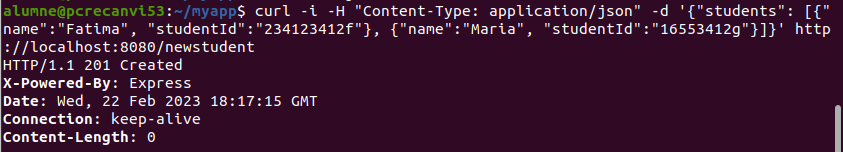
Una vez instalado y configurado todo lo necesario para hacer la práctica según el enunciado (ya teníamos VirtualBox pero nos tuvimos que instalar cURL “curl -sL https://deb.nodesource.com/setup\_14.x | sudo -E bash -” y Node.js “sudo apt-get install -y nodejs”), instalamos también el paquete Express con los valores predeterminados, que registrará las dependencias de nuestra aplicación.

Seguimos el tutorial para familiarizarnos de los *endpoints*, Node.js, JSON y cURL, he aquí unos ejercicios:



Figuras 1, 2 - Ejemplo de *app.get*





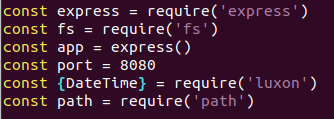
Figuras 3, 4, 5 - Ejemplo de *app.post*

# **2. -Alternativas para resolverla**

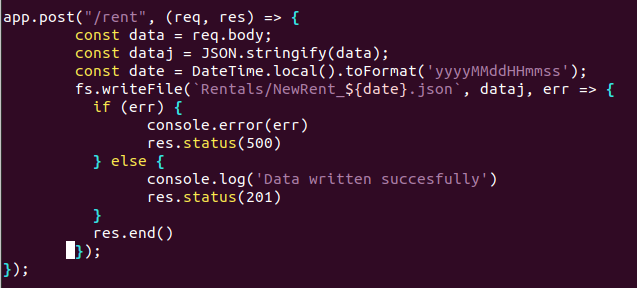
Para poder realizar esta práctica podemos haber usado otro framework web, y otra base de datos para almacenar la información en vez de usar JSON.

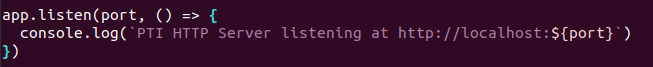
# **3. -New Rental**

La primera funcionalidad que tenemos que implementar es la de solicitar un nuevo alquiler, donde tenemos que usar un *endpoint* para registrarlo, en este caso un *app.post* conteniendo los campos indicados y que se nos devuelva solamente el código de respuesta de estado HTTP 201:

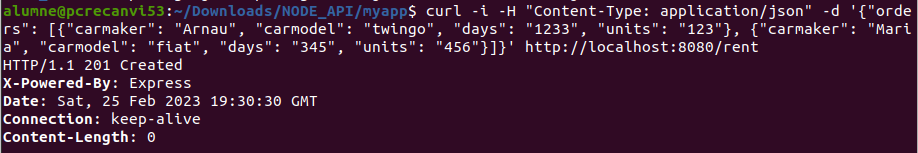






Figuras 6, 7, 8, 9 - Implementación New Rental

Con el código anterior lanzamos las siguientes dos instancias con los valores que se nos piden (*car maker, car model, number of days* y *number of units*) en cada caso, y podemos ver que se nos devuelve el 201 de HTTP y se ha creado un fichero JSON conteniendo los datos de la comanda de cURL:









Figuras 10, 11, 12, 13 - Salida y prueba de New Rental

# **4. -List Rentals**

La segunda funcionalidad que tenemos que implementar es la de listar todos los alquileres previamente hechos, donde tenemos que usar otro *endpoint,* el *app.get*:

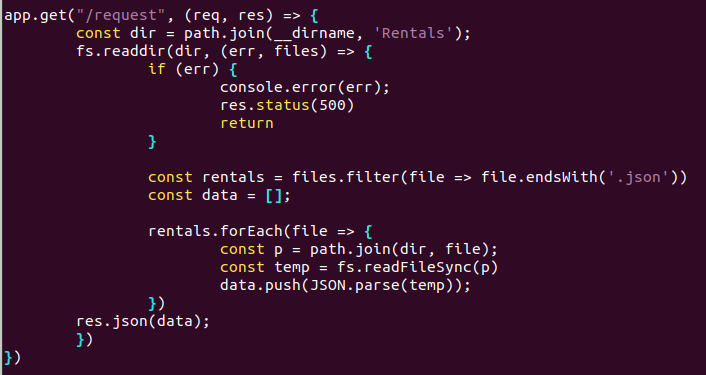


Figura 14 - Implementación de List Rentals

Con el código anterior podemos ver en el <http://localhost:8080/request> que se nos muestra la lista de los coches que hemos alquilado anteriormente:

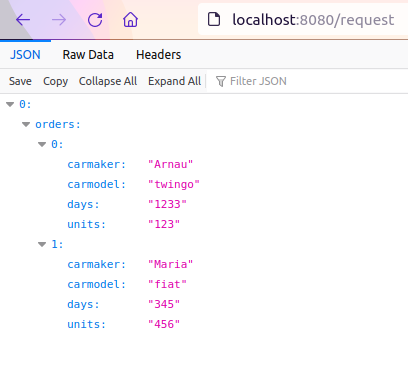


Figura 15 - Salida y prueba de List Rentals

# **5. -Dockerization**

En este apartado, hemos hecho un docker de nuestra aplicación web con la ayuda del Dockerfile que se nos proporciona (el contenido) y unas comandas (“docker build -f Dockerfile -t carrental .” y “docker run --name carrental -d -p 8080:8080 -p 8443:8443 carrental”). Ahora con el Dockerfile y los ficheros “server.js” y “package.json” construimos el tarball para la entrega.

# **6. -CI/CD**

En este apartado, siguiendo los pasos y comandas proporcionados, automatizamos la construcción, prueba y despliegue de nuestra aplicación.

Primero creamos un repositorio para nuestro REST API en GitLab con todo lo que hicimos antes con la ayuda de git commit y push.

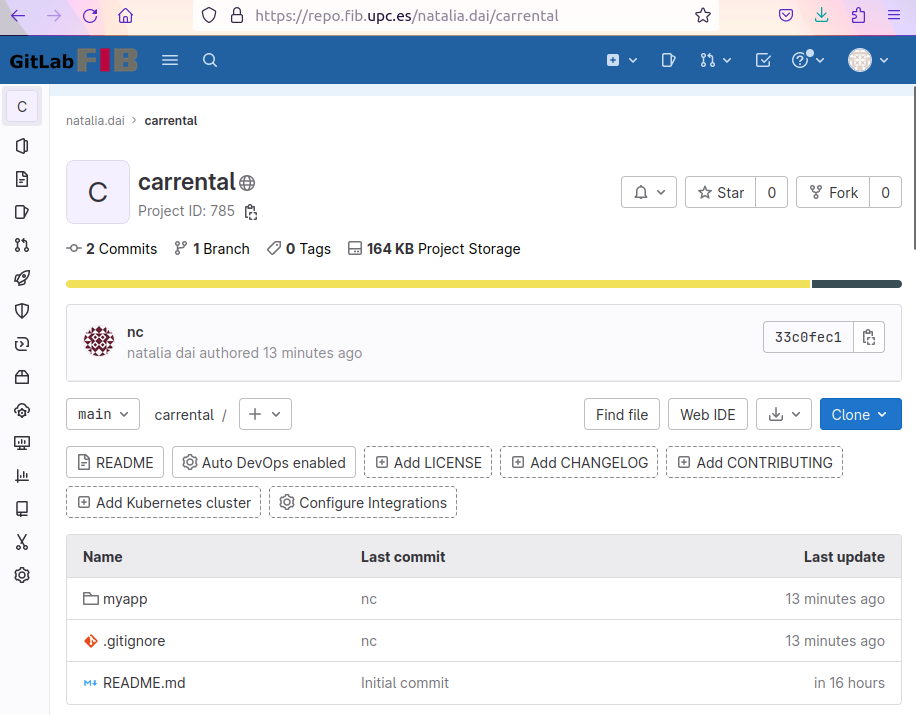


Figura 16 - Repositorio creado de GitLab de carrental

Luego, instalamos un GitLab Runner, que se muestra en la esquina inferior a la izquierda de la siguiente imagen:

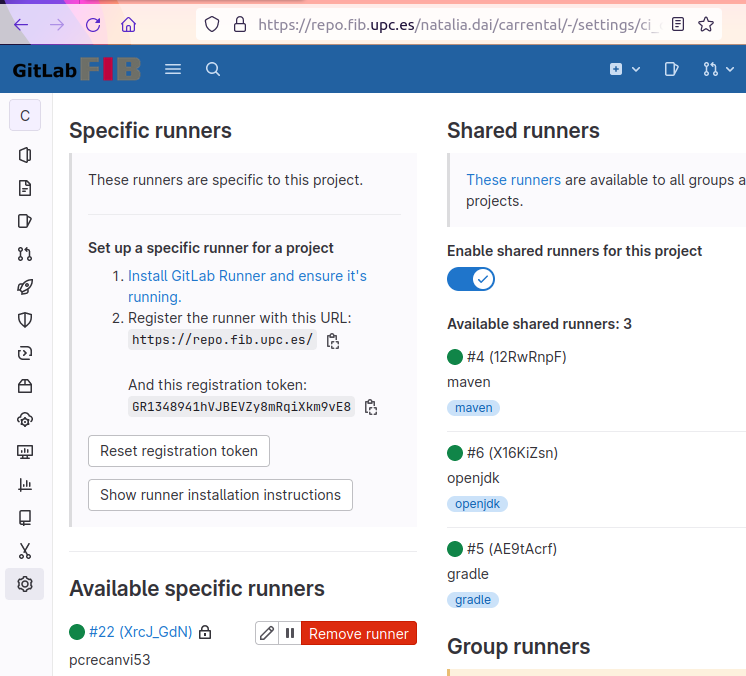


Figura 17 - GitLab Runner instalado

Al tenerlo ya instalado, hacemos una prueba con un “Hello, World!” en el fichero .gitlab-ci.yml, que podemos ver que se ha podido ejecutar bien:

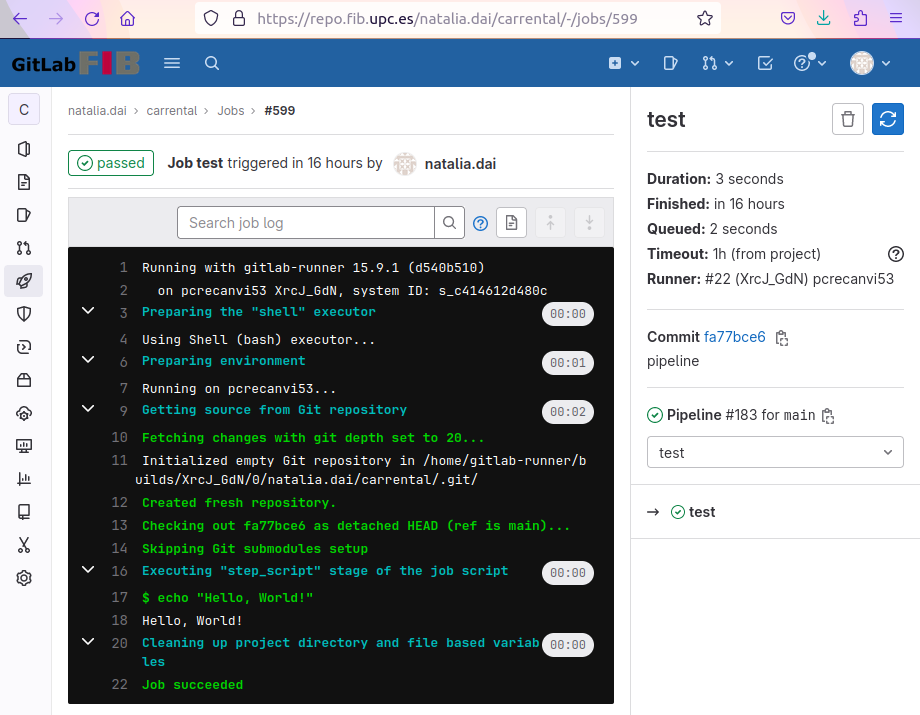


Figura 18 - Prueba archivo .gitlab-ci.yml

Este es otro ejemplo que hicimos para comprobar los diferentes *jobs* de la *pipeline*, cambiando el fichero de .gitlab-ci.yml:

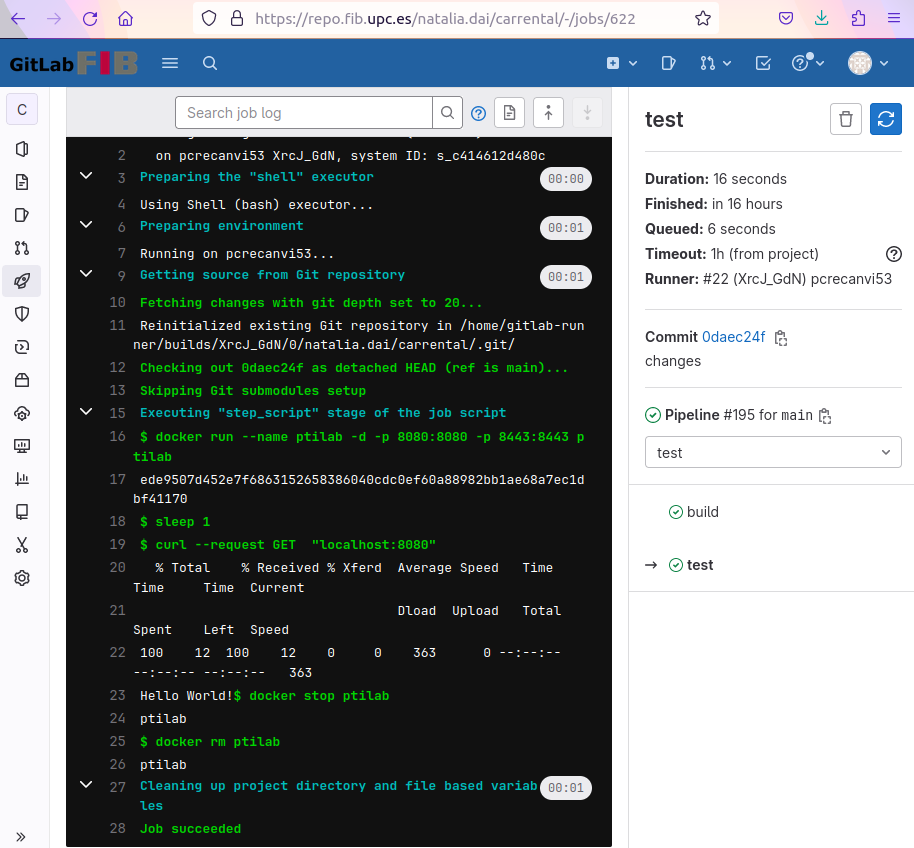
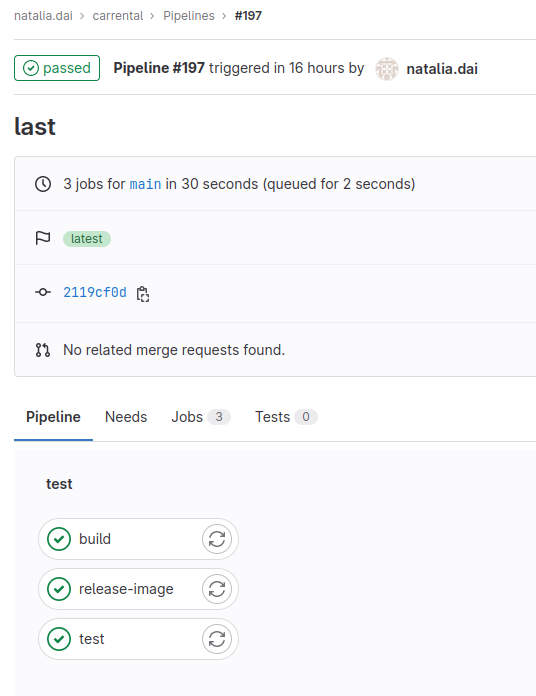
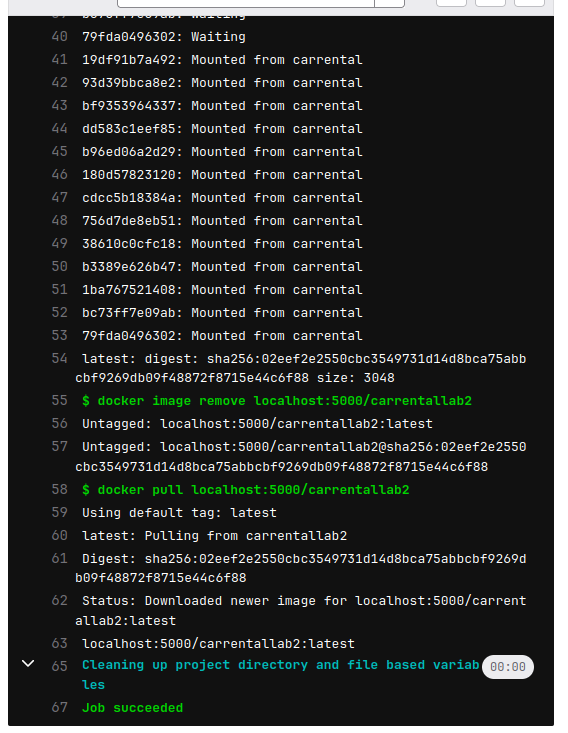


Figura 19 - Segunda prueba del funcionamiento de .gitlab-ci.yml

Ahora cambiando el contenido del archivo mencionado al que se nos proporciona en el último apartado, y haciendo commit y push, se nos genera otra compilación automática y nos dice si ha pasado limpio o contiene errores.

Podemos ver que se ha ejecutado bien en las siguientes imágenes:





Figuras 20, 21 - Última prueba de CI/CD