

API REST

Alumno: Víctor Gallego Izquierdo

Alumno: Jordi Solé García

Grupo 11.

Creación de la web API de alquiler de coches.

A partir del ejemplo “server.js” que hemos creado y modificado antes de empezar el laboratorio hemos generado un “index.js”.

Primero añadimos la herramienta Express que nos permite utilizar la función `app.use(..)` de forma que nos facilita varias funciones del back-end de la aplicación.

También añadimos la herramienta “fs” que nos facilitará la lectura y escritura del fichero json de los coches de alquiler.

```
const express = require('express')
const fs = require('fs')
const port = 8080
const app = express()
```

A partir de aquí, generamos 3 endpoints:

- “/”, que lo hemos generado para comprobar si la aplicación funciona.
- “/new”, de forma que este añade un nuevo coche de alquiler al fichero “rentals.json” utilizando POST. Así, en el endpoint que explicamos a continuación podemos descargar la lista de los coches.
- “list”, que realiza la descarga de la lista de coches leyendo el fichero “rentals.json” y utilizando GET.

Endpoint “/new”.

```
app.post('/new', (req, res) => {
  let rental = {
    maker: req.body.maker,
    model: req.body.model,
    days: req.body.days,
    units: req.body.units
  };
  let rentals = JSON.parse(fs.readFileSync(data_path));
  rentals['car'].push(rental);
  fs.writeFileSync(data_path, JSON.stringify(rentals), (err) => {
    if (err) return console.log(err);
    console.log('added new car');
  });
});
```

```
res.status(201);  
res.send(rental);  
res.end();
```

- Creamos el body del post con un json que contiene el maker, modelo, días y unidades alquiladas
- A partir de aquí, leemos el archivo json que utilizamos de BBDD para los coches.
- Finalmente escribimos en él, retornamos 201 si se ha creado correctamente y hacemos el send.

Endpoint **"/list"**.

```
app.get('/list', (req, res) => {  
  res.send(JSON.parse(fs.readFileSync(data_path)));  
});
```

En este caso simplemente leemos el fichero y lo devolvemos parseado en formato json.

Listen

Finalmente, en nuestro "index.js" contamos con una función listen, que nos sirve para que cuando el servidor este en marcha este escuchando el puerto que hemos definido (8080). Además, si es la primera vez que iniciamos la API, este crearía nuestro fichero json para la BBDD de los coches nuevos a añadir.

```
app.listen(port, () => {  
  if (!fs.existsSync(data_path)) {  
    fs.appendFile(data_path, JSON.stringify({car: []}), (err)  
=> {  
      if (err) return console.log(err);  
    });  
  }  
  console.log(`Listening: http://localhost:${port}`);  
});
```

Dockerization

Creamos el siguiente Dockerfile para dockerizar la aplicación.

```
FROM node:16
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD [ "node", "index.js" ]
```

CI/CD (Lab extension)

El primer paso para llevar a cabo esta tarea es crear un repositorio en GitLab de la FIB. Para ello, entramos en repo.fib.upc.edu.es y creamos un repositorio público, lo clonamos en nuestra máquina y copiamos la carpeta **carrental** allí. Accedemos a ella y comprobamos que es correcto todo lo necesario. Como opción, podemos añadir un archivo **.gitignore** con la carpeta *node_modules/* en el interior para no añadir esta carpeta al git. Tendremos que configurar nuestras credenciales para poder acceder al repositorio de GitLab con los comandos:

- `git config --global user.name "FIRST_NAME LAST_NAME"`
- `git config --global user.email "USERNAME@upc.edu"`
- `git config --global credential.helper store`

Añadimos los cambios con **git add .** , hacemos un commit con **git commit -m “algo”** y hacemos el push al repositorio remoto con **git push**.

The screenshot shows the GitLab interface for a project named 'API REST' (Project ID: 778). The repository is on the 'main' branch, with 4 commits, 1 branch, and 0 tags. It has 78.4 MB of project storage. The user 'prueba2' (victor.gallego.izquierdo) is the author of the latest commit. Below the repository information, there are buttons for 'Find file', 'Web IDE', and 'Clone'. A section for 'CI/CD configuration' includes links to 'Add README', 'Add LICENSE', 'Add CHANGELOG', and 'Add CONTRIBUTING'. There are also links for 'Auto DevOps enabled', 'Add Kubernetes cluster', and 'Configure Integrations'. A table lists the files in the repository, showing their names, the last commit they were updated in, and the time since the last update.

| Name | Last commit | Last update |
|--------------------|-------------|---------------|
| node_modules | prueba2 | just now |
| .dockerignore | prueba2 | just now |
| gitlab-ci.yml | prueba1 | 6 minutes ago |
| Dockerfile | prueba2 | just now |
| docker-compose.yml | prueba2 | just now |
| index.js | prueba2 | just now |
| package-lock.json | prueba2 | just now |
| package.json | prueba2 | just now |
| rentals.json | prueba2 | just now |

A partir de aquí, hay que configurar la automatización que compruebe que el proyecto funciona correctamente. Esta herramienta viene incluida en GitLab. Para ello hay que instalar esta herramienta en la máquina y seguir las instrucciones que se encuentran en **Settings > CI / CD > Runners > Show runner installation instructions** :

Specific runners

These runners are specific to this project.

Set up a specific runner for a project

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:

`https://repo.fib.upc.es/`

And this registration token:

`GR1348941p99FaH2XJekzKtaVBNhZ`

Reset registration token

Show runner installation instructions

Una vez instalado, comprobamos que está funcionando en **Check that the runner status in Settings > CI / CD > Runners**

El siguiente paso es definir una pipeline por la que esta tarea realizará las pruebas que le indiquemos.

Para ello, creamos el archivo **.gitlab-ci.yml** con el siguiente contenido en el interior:

```
test:
  script:
    - echo "Hello, World!"
```

Hacemos un commit y push con esto al repositorio. Si accedemos a **CI / CD > Pipelines**, podemos ver como se ha ejecutado la tarea de test y que funcione correctamente:

Victor Gallego Group > APLREST > Pipelines

All 3 Finished Branches Tags Clear runner caches CI lint Run pipeline

Filter pipelines 🔍 Show Pipeline ID ▾


| Status | Pipeline | Triggerer | Stages |
|-------------------------------------|---------------------------------|-----------|--------|
| passed 00:00:01 just now | prueba2 #166 main → 5a743511 | | |
| passed 00:00:01 7 minutes ago | prueba1 #165 main → 551c0ea7 | | |
| passed 00:00:17 7 minutes ago | test #164 main → d89b51ce | | |


Ahora modificamos el código del runner y ponemos lo siguiente para que compruebe de manera automática que el contenedor Docker se ejecuta de manera correcta:

```
build:
  script:
    - docker build . -t carrental
test:
  script:
    - docker run --name carrental -d -p 8080:8080
carrental
  - sleep 1
  - curl --request GET "localhost:8080"
  - docker stop carrental
  - docker rm carrental
```


De nuevo vamos a ver la Pipeline si ha ejecutado bien el job que hemos creado:



Victor Gallego Group > APL_REST > Pipelines > #168


 **passed**

Pipeline #168 triggered 2 days ago by  victor.gallego.izquierdo

prueba4



 2 jobs for **main** in 22 seconds



 **a24cc8a8** 

 No related merge requests found.

Pipeline Needs Jobs **2** Tests **0**

test



 build 

 test 


Y vemos que ambos han funcionado correctamente. Por último, tenemos que crear el job de registrar nuestra imagen de docker con un tag, que servirá para ir mostrando las actualizaciones que podamos ir haciendo. Para ello, modificamos de nuevo el archivo **.gitlab-ci.yml** y añadimos lo siguiente:

```
build:
  script:
    - docker build . -t carrental
test:
  script:
    - docker run --name carrental -d -p 8080:8080
carrental
  - sleep 1
  - curl --request GET "localhost:8080"
  - docker stop carrental
  - docker rm carrental
release-image:
  script:
    - docker tag carrental:latest localhost:5000/carrental
    - docker push localhost:5000/carrental
    - docker image remove localhost:5000/carrental
    - docker pull localhost:5000/carrental
```

Hacemos el commit y push de nuevo y vemos como ha funcionado:


 **passed** Pipeline #171 triggered 2 days ago by  victor.gallego.izquierdo

prueba7

 3 jobs for `main` in 40 seconds (queued for 1 second)

 `latest`



 `098e079b` 

 No related merge requests found.

Pipeline Needs Jobs 3 Tests 0

test

 build 

 release-image 

 test 