



UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

---

Facultat d'Informàtica de Barcelona



# Informe PTI Lab 3: REST\_API

Arnau Garcia Gonzalez

PTI Quadrimestre tardor Curs 2025-26

# Índex

<b>1. Introducció.....</b>	<b>3</b>
<b>2. Tutorial Inicial.....</b>	<b>4</b>
2.1 Setup.....	4
2.2 Servidor web simple.....	4
2.3 Routing de URL.....	5
2.4 Implementació JSON.....	6
2.4.1 Resposta JSON.....	6
2.4.2 Petició JSON.....	7
<b>3. Realització de la pràctica.....</b>	<b>10</b>
3.1 Creació de la API web de lloguer de cotxes.....	10
3.1.1 Preparació de l'entorn.....	10
3.1.2 Explicació del codi Server.js.....	10
3.1.3 Testing de l'API.....	12
3.2 Extensió 1: Dockerització.....	15
3.3 Extensió 2: CI/CD.....	16
3.3.1 Creació del repositori a GitLab.....	16
3.3.2 Instal·lar, registrar i executar a GitLab Runner.....	16
3.3.3 Definir una pipeline CI/CD.....	17
<b>4. Annex.....</b>	<b>18</b>
4.1 Codí Server.js de l'API de car-rental.....	18

## **1. Introducció**

En l'actualitat, la majoria d'aplicacions web i mòbils es basen en arquitectures client-servidor on la comunicació es realitza mitjançant Web APIs. Aquestes APIs utilitzen el protocol HTTP com a mecanisme de transport i normalment JSON com a format d'intercanvi de dades, a causa de la seva simplicitat i compatibilitat amb diferents llenguatges de programació.

El desenvolupament d'APIs ha evolucionat des de solucions més complexes (com SOAP i WSDL) cap a enfocaments més lleugers, influenciats per l'estil arquitectònic REST. Encara que no totes les APIs són estrictament RESTful, els principis bàsics (ús de mètodes HTTP, recursos i representacions en JSON) s'han consolidat com l'estàndard de facto.

En aquest context, Node.js s'ha convertit en una plataforma molt utilitzada per implementar Web APIs, ja que permet desenvolupar serveis lleugers i escalables mitjançant JavaScript al costat del servidor. A més, frameworks com Express simplifiquen la definició de rutes, la gestió de peticions i respostes, i el tractament de dades en JSON.

Finalment, la integració d'aquest tipus d'aplicacions amb pràctiques modernes com la contenidorització amb Docker i els sistemes de CI/CD permet desplegar-les de manera més senzilla, ràpida i automatitzada, seguint l'enfocament DevOps que predomina en la indústria.

## **2. Tutorial Inicial**

Primer que tot s'ha seguit el tutorial proporcionat pel professor a [https://repo.fib.upc.es/felix.freitag/pti/-/tree/master/REST\\_API](https://repo.fib.upc.es/felix.freitag/pti/-/tree/master/REST_API). S'han seguit aquests passos per preparar l'entorn abans de realitzar la pràctica.

### **2.1 Setup**

Primer que tot s'ha comprovat que es tenien els programes necessaris, i en cas de no tenir-los s'han instal·lat.

```
Shell
curl --version #Per comprobar que curl ja era instal·lat
sudo apt-get install -y nodejs #Per instal·lar Node.js
```

Després s'ha preparat un directori per l'aplicació (s'ha decidit fer-lo dins del directori de la pràctica) i s'ha creat un package.json per registrar les dependències. A continuació s'ha instal·lat el package Express i s'ha inclòs com a dependència. Per a fer això s'han seguit els següents passos.

```
Shell
#Creem el directori per l'aplicació
mkdir myapp
cd myapp

npm init #Crea package.json, fitxer per registrar les dependències

#Instal·la el paquet express i l'afegeix a dependències
npm install express --save

#Per assegurar que s'ha afegit correctament i veure la versió instal·lada
npm list
```

### **2.2 Servidor web simple**

Després dels passos inicials s'ha programat un servidor web bàsic amb [Node.js](#). Això s'ha fet creant el fitxer server.js al directori de l'aplicació. El codi del servidor és el següent:

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

```

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`PTI HTTP Server listening at http://localhost:${port}`)
})

```

En aquest exemple una aplicació Express. Quan rep una petició HTTP GET, respon amb el missatge “*Hello World!*” al client (navegador). Finalment, obre un socket i escolta connexions al port 8080.

Després llancem el servidor amb `node server.js` i ho comprovem en <http://localhost:8080>.

la primera vegada no s’ha pogut executar amb èxit. Això ha estat perquè la versió de node estava desactualitzada, així que s’ha actualitzat a una versió superior. Per fer l’actualització s’han fet les següents comandes.

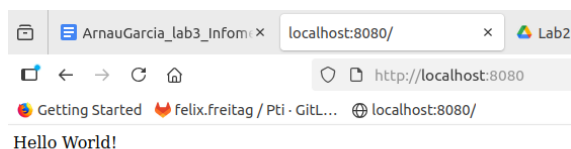
```

Shell
#S'instal·la nvm per tenir diverses versions, instal·lar-la donava conflicte
amb anteriors versions
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh |
bash

#S'instala la versió 18 i es posa com a default i en ús
nvm install 18
nvm use 18
nvm alias default 18

```

Una vegada fet això s’ha pogut executar el servidor web.



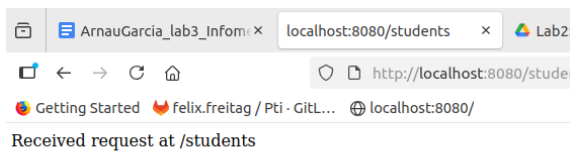
## 2.3 Routing de URL

Un cop creat el servidor bàsic, s’han afegit noves rutes per poder gestionar diferents peticions HTTP. A Express, una ruta s’especifica amb el mètode `app.METHOD` (per exemple `app.get`) i el camí on escoltarà.

Al fitxer `server.js` s'ha afegit la següent ruta:

```
JavaScript
app.get("/students", (req, res, next) => {
  res.send('Received request at /students')
});
```

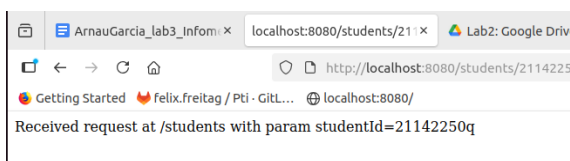
Això vol dir que quan fem una petició GET a <http://localhost:8080/students>, el servidor respon amb el text *Received request at /students*.



També s'ha afegit un exemple amb paràmetres dins la ruta:

```
JavaScript
app.get('/students/:studentId', function (req, res) {
  res.send('Received request at /students with param studentId='+req.params.studentId)
});
```

En aquest cas, si visitem <http://localhost:8080/students/21142250q>, el servidor detecta el valor del paràmetre `studentId` i el mostra a la resposta.



## 2.4 Implementació JSON

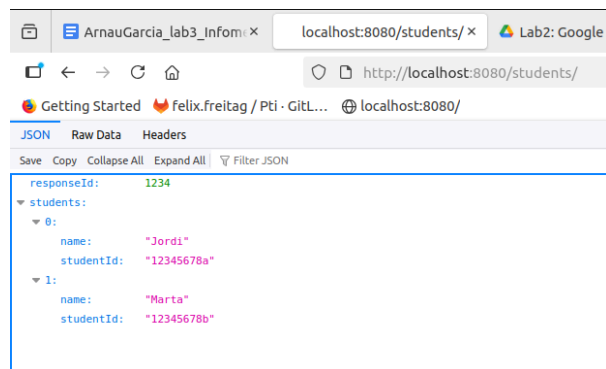
En molts casos, un endpoint ha de gestionar dades més complexes. Per això, en lloc d'enviar una cadena de text, s'utilitza el format JSON tant per l'entrada com per la sortida. Així que això és el següent que s'ha implementat.

### 2.4.1 Resposta JSON

Primer s'ha modificat el endpoint anterior per incloure una resposta JSON:

```
JavaScript
app.get("/students", (req, res, next) => {
  res.json({
    responseId: 1234,
    students: [
      {name: "Jordi", studentId: '12345678a'},
      {name: "Marta", studentId: '12345678b'}
    ]
  });
});
```

Aleshores s'ha provat d'accedir des del navegador a <http://localhost:8080/students> i a més a més també provat de fer una petició mitjançant `curl -H "Content-Type: application/json" http://localhost:8080/students`.



```
alumne@nipygon:~/Documents/PTI-FIB/Practicas_lab/Lab3/REST_API$ curl -H "Content-Type: application/json" http://localhost:8080/students
{"responseId":1234,"students":[{"name":"Jordi","studentId":"12345678a"}, {"name":"Marta","studentId":"12345678b"}]}alumne@nipygon:~/Documents/PTI-FIB/Practicas_lab/Lab3/REST_API$
```

### 2.4.2 Petició JSON

A més de retornar JSON, el servidor també pot rebre dades en format JSON. Perquè Express pugui interpretar el cos de la petició cal activar el mòdul integrat `express.json()` amb `app.use(express.json());`

S'ha creat un nou endpoint amb el mètode POST i el path `/newstudent` que llegeix dades JSON del cos de la petició:

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

```

app.use(express.json());

app.post('/newstudent', (req, res, next) => {
  console.log(req.body.name);
  res.end();
})

app.listen(port, () => {
  console.log(`PTI HTTP Server listening at http://localhost:${port}`)
})

```

Lavors després de relançar el servidor i executar `curl -i -H "Content-Type: application/json" -d '{"name":"Fatima", "studentId":"234123412f"}' http://localhost:8080/newstudent` obtenim el següent resultat indicant que el servidor ha rebut correctament la informació:

```

alunne@nlpigon:~/Documents/PTI-FIB/Practicas_lab/Lab3/REST_API/myapp$ node server.js
PTI HTTP Server listening at http://localhost:8080
PTI HTTP Server listening at http://localhost:8080
Fatima

```

Una vegada fet el pas anterior s'ha complicat una mica fent que es pugui enviar un array d'estudiants. Per això s'ha modificat l'anterior codi. A aquest codi també s'ha afegit que retorni el codi 201, això és perquè quan una petició POST afegeix recursos nous envia aquest codi (generalment si rep dades les guardarà a algun lloc):

```

JavaScript
app.post('/newstudent', (req, res, next) => {
  for(var i in req.body.students){
    console.log(req.body.students[i].name+'\n');
  }
  res.status(201);
  res.end();
})

```

I després d'executar `curl -i -H "Content-Type: application/json" -d '{"students": [{"name": "Fatima", "studentId": "234123412f"}, {"name": "Maria", "studentId": "16553412g"}]}' http://localhost:8080/newstudent` obtenim el següent al servidor i al client que envia la petició respectivament:



r.js PTI HTTP Server listening at http://lo Fatima Maria	HTTP/1.1 201 Created X-Powered-By: Express Date: Sat, 27 Sep 2025 12:00:46 GMT Connection: keep-alive Keep-Alive: timeout=5 Content-Length: 0
---	--

## **3. Realització de la pràctica**

Per la realització de la pràctica s'utilitzarà com a base el codi del tutorial anterior, fent-ne les modificacions i addicions necessàries. Tot i això, per motius d'ordre s'ha decidit treballar en un directori diferent, en el directori `car-rental-api`.

### **3.1 Creació de la API web de lloguer de cotxes**

Aquesta API permet amb el mètode post enviar un nou lloguer de cotxe. Cada lloguer té un fabricant (maker), model de cotxe, nombre de dies i unitats. A més a més, s'ha cregut convenient d'incloure un ID en tots els lloguers, determinat pel moment de fer la comanda, per tal de ser únic. I també s'ha afegit la data de creació del lloguer com a element. (Aquestes últimes dues propietats són automàtiques, no les introdueix l'usuari).

#### **3.1.1 Preparació de l'entorn**

Primer que tot s'ha preparat l'entorn de l'api, aquest pas es similar a l'inicial en el tutorial.

```
Shell
mkdir car-rental-api
cd car-rental-api

npm init      #Crea package.json, fitxer per registrar les dependències
npm install express --save #Instala el paquet i l'afegeix a dependències
```

Ara cal començar a crear el fitxer [Server.js](#). Per explicar el codi s'anirà seguint poc a poc per seccions, posant el codi pertinent i explicant la funció de cada secció. Per veure el codi complet es pot veure [l'annex 4.1](#).

#### **3.1.2 Explicació del codi Server.js**

Primer que tot, seguint l'esquelet del tutorial s'han importat els mòduls necessaris.

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

app.use(express.json());
```

La següent secció del codi és l'encarregada de configurar el fitxer JSON. L'objecte rentalsJSON és el que s'utilitzarà per fer addicions o per enviar. Si el fitxer JSON no existeix en crea un de nou amb l'objecte arrel rentals. Si existeix s'agafa el seu contingut.

```
JavaScript
const filepath = 'rentals.json';
let rentalsJSON;

//Comprovar si el fitxer JSON existeix
if (!fs.existsSync(filepath)) {
  // Crear document inicial
  rentalsJSON = { "rentals": [] };
  fs.writeFileSync(filepath, JSON.stringify(rentalsJSON, null, 2));
} else {
  // Llegir document existent
  const rawData = fs.readFileSync(filepath);
  rentalsJSON = JSON.parse(rawData);
}
```

El següent fragment és l'endpoint de POST a /rentals, per registrar un nou lloguer. Això captura els paràmetres passats per l'usuari i els guarda a l'objecte JSON, per posteriorment escriure aquest objecte al fitxer JSON. Aleshores retorna un codi 201.

```
JavaScript
// Endpoint POST: registrar un nou lloguer
app.post('/rentals', (req, res) => {
  const { maker, model, days, units } = req.body;

  //Si falta algun parametre --> Error
  if (!maker || !model || typeof days !== 'number' || typeof units !==
'number') {
    return res.status(400).json({ error: "Payload invàlid" });
  }

  const rental = {
    id: Date.now().toString(),
    maker,
    model,
    days,
    units,
    //Aqui seria millor "new Date().toISOString()", però per
llegibilitat poso el que hi ha
    createdAt: new Date().toLocaleString()
  };

  rentalsJSON['rentals'].push(rental);
});
```

```

    fs.writeFileSync(filepath, JSON.stringify(rentalsJSON, null, 2));

    res.status(201).json(rental);
  });

```

El següent és l'endpoint GET a /list. Aquest llista totes les ordres, això ho fa enviant el fitxer JSON.

```

JavaScript
// Endpoint GET: retornar tots els lloguers
app.get('/rentals', (req, res) => {
  res.json(rentalsJSON);
});

```

La inicialització del servidor és igual que la del tutorial previ.

```

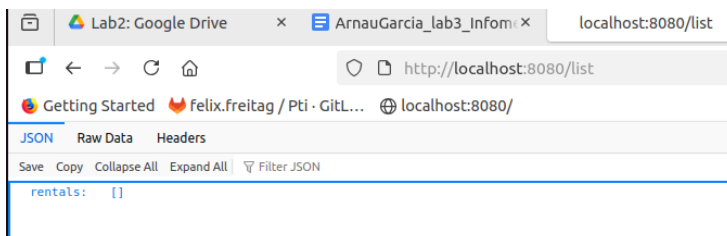
JavaScript
// Iniciar servidor
app.listen(port, () => {
  console.log(`Servidor HTTP escoltant a http://localhost:${port}`);
});

```

### 3.1.3 Testing de l'API

Per comprovar que tot funciona correctament s'ha realitzat un petit testing.

Primer que tot s'ha posat en marxa el servidor amb `node server.js` provat de fer una petició GET a /list per comprovar que si no hi ha fitxer json no hi ha cap error. Després d'accedir a <http://localhost:8080/list> veiem el següent.



Aquesta acció a més a més ens ha generat el rentals.json de forma local. Una vegada vist que es rep el json correctament toca testejar l'endpoint de POST. Per això s'ha executat la següent comanda:

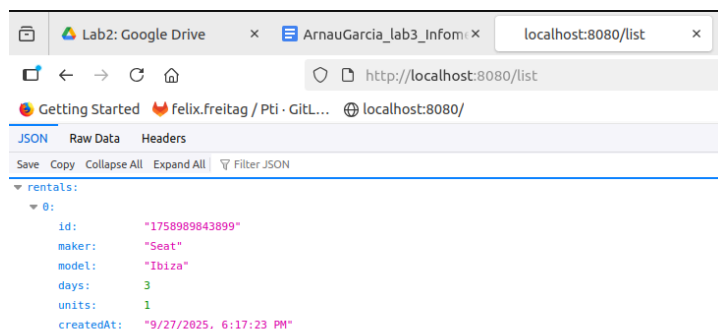
```
Shell
curl -i -H "Content-Type: application/json" -d '{"maker":"Seat","model":"Ibiza","days":3,"units":1}' http://localhost:8080/rentals
```

S'ha rebut el següent missatge:

```
alumni@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab3/REST_API/car-rental-api$ curl -i -H "Content-Type: application/json" -d '{"maker":"Seat","model":"Ibiza","days":3,"units":1}' http://localhost:8080/rentals
HTTP/1.1 201 Created
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 108
ETag: W/"6c-ob5F06J3Yj1aby1/pVx2i17PRAQ"
Date: Sat, 27 Sep 2025 16:17:23 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"id":"1758989843899","maker":"Seat","model":"Ibiza","days":3,"units":1,"createdAt":"9/27/2025, 6:17:23 PM"}
```

Aleshores s'ha comprovat que si ara es visita <http://localhost:8080/list> la llista es troba actualitzada.



```
JSON
Raw Data
Headers

Save Copy Collapse All Expand All Filter JSON

rentals:
  0:
    id: "1758989843899"
    maker: "Seat"
    model: "Ibiza"
    days: 3
    units: 1
    createdAt: "9/27/2025, 6:17:23 PM"
```

Com a última prova s'ha provat d'afegir cinc lloguers més, per això s'ha executat el següent script:

```
Shell

#Crea el primer lloguer
curl -i -H "Content-Type: application/json" \
  -d '{"maker":"Seat","model":"Ibiza","days":3,"units":1}' \
  http://localhost:8080/rentals

sleep 2      # Fa pausa per tenir diferent temps

#Crea el segon lloguer
curl -i -H "Content-Type: application/json" \
  -d '{"maker":"Toyota","model":"Corolla","days":5,"units":2}' \
  http://localhost:8080/rentals
```

```

sleep 2      # Fa pausa per tenir diferent temps

#Crea el tercer lloguer
curl -i -H "Content-Type: application/json" \
      -d '{"maker":"Volkswagen","model":"Golf","days":7,"units":1}' \
      http://localhost:8080/rentals
sleep 2      # Fa pausa per tenir diferent temps

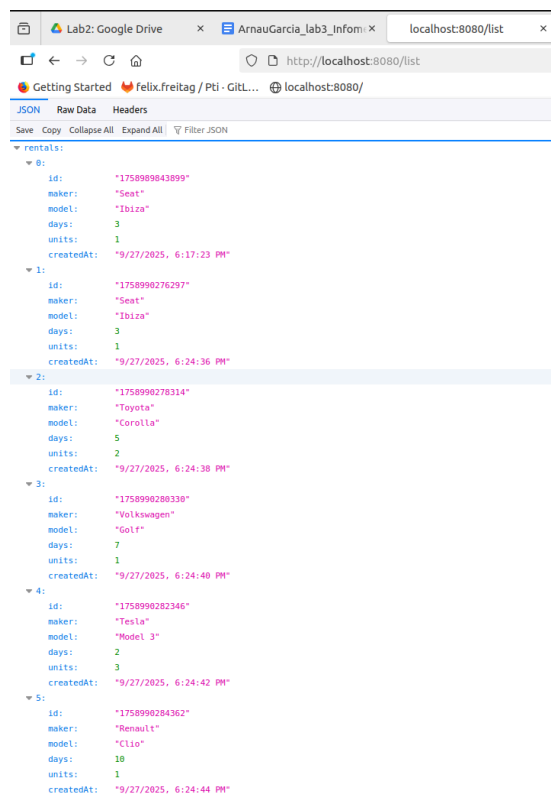
#Crea el quart lloguer
curl -i -H "Content-Type: application/json" \
      -d '{"maker":"Tesla","model":"Model 3","days":2,"units":3}' \
      http://localhost:8080/rentals
sleep 2      # Fa pausa per tenir diferent temps

#Crea el cinquè lloguer
curl -i -H "Content-Type: application/json" \
      -d '{"maker":"Renault","model":"Clio","days":10,"units":1}' \
      http://localhost:8080/rentals

echo Finished!

```

Després d'executar i visitar altra vegada <http://localhost:8080/list> veiem el següent resultat:



Per tant, la nostra API ja està funcionant correctament.

## 3.2 Extensió 1: Dockerització

Per dockeritzar l'aplicació primer que tot cal crear un Dockerfile i posar el contingut següent. S'ha inclòs COPY rentals.json perquè al contenidor es guardi aquest fitxer.

```
Shell
FROM node:18

# Create app directory
WORKDIR /usr/src/app

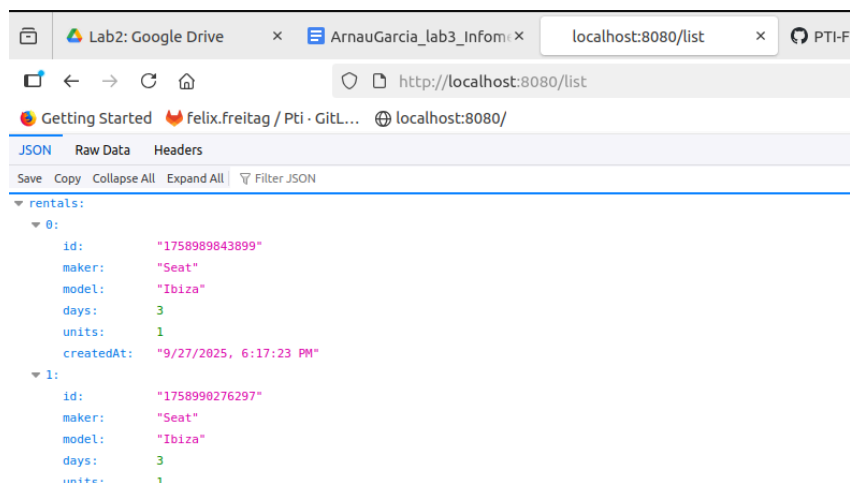
# Install app dependencies
COPY package*.json ./

RUN npm install

# Bundle app source
COPY server.js .
COPY rentals.json .

CMD [ "node", "server.js" ]
```

Després s'ha fet el build amb `"docker build -f Dockerfile -t carrental" .`. Aleshores ja s'ha pogut córrer el contenidor amb `docker run --name carrental_P3 -d -p 8080:8080 -p 8443:8443 carrental`.



Com es pot veure el procés s'ha realitzat amb èxit i podem fer les consultes http.

### 3.3 Extensió 2: CI/CD

En aquest apartat es treballarà amb CI/CD aplicat al projecte de lloguer de cotxes (carrental). L'objectiu és entendre i posar en pràctica com es pot automatitzar tot el cicle de vida de l'aplicació: des de la construcció de la imatge Docker fins a les proves i la publicació en un registre.

A la pràctica, es configurarà un pipeline de GitLab CI/CD que, cada vegada que fem un commit al repositori:

- Construirà la imatge Docker de l'aplicació.
- Executarà proves bàsiques (arrencar el contenidor i comprovar que l'API respon).
- Publicarà la imatge a un registre Docker (en aquest cas local, ja que no tenim accés al GitLab Container Registry).

#### 3.3.1 Creació del repositori a GitLab

Primer que tot s'ha creat el repositori pti-carrental a <https://repo.fib.upc.es/arnau.garcia.gonzalez/pti-carrental>. Una vegada fet això s'ha preparat l'estructura del repositori, i s'ha creat un `.gitignore` amb “`node_modules/`”.

#### 3.3.2 Instal·lar, registrar i executar a GitLab Runner

Gitlab Runner és una eina que permetrà executar les tasques CI/CD. Per tant el primer que s'ha hagut de fer es instal·lar el programa. Tot el procediment s'ha seguit de la guia a [https://repo.fib.upc.es/felix.freitag/pti/-/blob/master/REST\\_API/cicd/CICD.md](https://repo.fib.upc.es/felix.freitag/pti/-/blob/master/REST_API/cicd/CICD.md). Per instal·lar el programa s'han introduït les següents comandes:

```
Shell

curl -L
"https://packages.gitlab.com/install/repositories/runner/gitlab-runner/scrip
t.deb.sh" | sudo bash

sudo apt install gitlab-runner
```

Després s'ha instal·lat un runner des de la configuració de GitLab. I s'ha comprovat que aquest es trobava actiu.

● #259 (Ei1woyttY)

  Remove runner



Després s'ha afegit gitlab-runner al grup docker i s'ha executat també una comanda per evitar el "ERROR: Job failed: prepare environment".

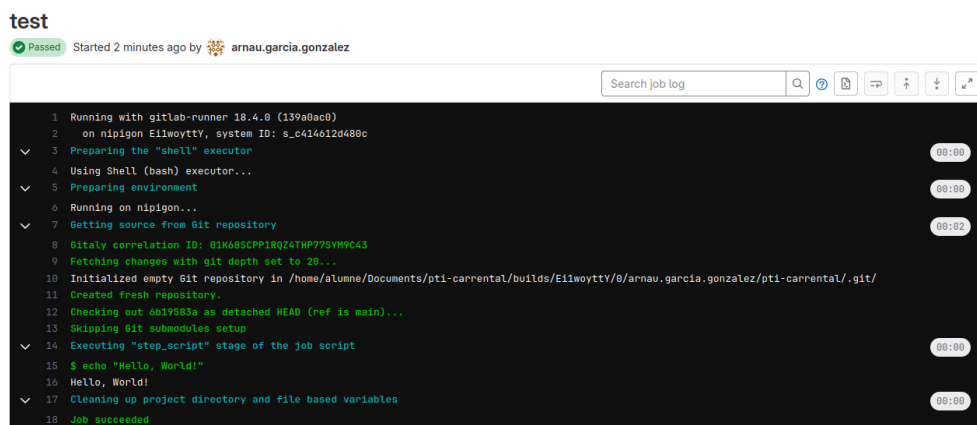
Shell

```
sudo usermod -aG docker gitlab-runner #Afegeix git-lab runner al grup
sudo rm -r /home/gitlab-runner/.bash_logout #Evita l'error mencionat
```

### 3.3.3 Definir una pipeline CI/CD

En aquest apartat es configurarà un pipeline de CI/CD a GitLab. El pipeline permet automatitzar la construcció, prova i publicació de la imatge Docker cada cop que es fa un *commit* al repositori.

Primer que tot com a prova inicial s'ha omplert un `.gitlab-ci.yml`, fitxer que serà el clau en aquest procés. Podem veure a continuació el resultat de la pipeline després d'haver fet push d'aquest canvi.



Després hem modificat aquest fitxer perquè ens construeixi la imatge docker. Aquest és el contingut introduït.

```
XML
build:
  script:
    - docker build -t carrental myapp
test:
  script:
    - docker run --name carrental -d -p 8080:8080 -p 8443:8443 carrental
    - sleep 1
    - curl --request GET "localhost:8080"
    - docker stop carrental
    - docker rm carrental
```

Després d'això, en revisar la pipeline podem veure si s'ha dockeritzat amb èxit.

```
1 Running with gitlab-runner 18.4.0 (139a0ac0)
2   on nipigon EilwoyTTY, system ID: s_c414612d480c
3   ✓ Preparing the "shell" executor
4   Using Shell (bash) executor...
5   ✓ Preparing environment
6   Running on nipigon...
7   ✓ Getting source from Git repository
8   Gitly correlation ID: 01K68T228785JQ625F6KRRK0349
9   Fetching changes with git depth set to 20...
10  Reinitialized existing Git repository in /home/alumne/Documents/pti-carrental/builds/EilwoyTTY/0/arnau.garcia.gonzalez/pti-carrental/.git/
11  Checking out 06203638 as detached HEAD (ref is main)...
12  Skipping Git submodules setup
13  > Executing "step_script" stage of the job script
147 ✓ Cleaning up project directory and file based variables
148 Job succeeded
```

Finalment, s'ha modificat una altra vegada el fitxer per “*pushejar*” la imatge a un registre, en aquest cas local. Per això primer que tot s'ha executat la comanda “`docker run -d -p 5000:5000 --restart=always --name registry registry:2.`”

A continuació s'ha fet la modificació al `.gitlab-ci.yml`:

```
XML
build:
  script:
    - docker build -t carrental myapp
test:
  script:
    - docker run --name carrental -d -p 8080:8080 -p 8443:8443 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
```

Després d'això, com podem veure s'ha realitzat correctament i ja tenim automatitzat per CI el nostre repo.

```
1 Running with gitlab-runner 18.4.0 (139a0ac0)
2   on nipigon EilwoyTTY, system ID: s_c414612d480c
3   ✓ Preparing the "shell" executor
4   Using Shell (bash) executor...
5   ✓ Preparing environment
6   Running on nipigon...
7   ✓ Getting source from Git repository
8   Gitly correlation ID: 01K68TJE8C7X0Z94E1W8FWJ0S8
9   Fetching changes with git depth set to 20...
10  Reinitialized existing Git repository in /home/alumne/Documents/pti-carrental/builds/EilwoyTTY/0/arnau.garcia.gonzalez/pti-carrental/.git/
11  Checking out 327c30fb as detached HEAD (ref is main)...
12  Skipping Git submodules setup
13  > Executing "step_script" stage of the job script
62 ✓ Cleaning up project directory and file based variables
63 Job succeeded
```

## 4. Annex

### 4.1 Codi Server.js de l'API de car-rental

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

app.use(express.json());

const filepath = 'rentals.json';
let rentalsJSON;

//Comprovar si el fitxer JSON existeix
if (!fs.existsSync(filepath)) {
  // Crear document inicial
  rentalsJSON = { "rentals": [] };
  fs.writeFileSync(filepath, JSON.stringify(rentalsJSON, null, 2));
} else {
  // Llegir document existent
  const rawData = fs.readFileSync(filepath);
  rentalsJSON = JSON.parse(rawData);
}

// Endpoint POST: registrar un nou lloguer
app.post('/rentals', (req, res) => {
  const { maker, model, days, units } = req.body;

  //Si falta algun parametre --> Error
  if (!maker || !model || typeof days !== 'number' || typeof units !==
'number') {
    return res.status(400).json({ error: "Payload invàlid" });
  }

  const rental = {
    id: Date.now().toString(),
    maker,
    model,
    days,
    units,
    //Aqui seria millor "new Date().toISOString()", però per
llegibilitat poso el que hi ha
    createdAt: new Date().toLocaleString()
  };

  rentalsJSON['rentals'].push(rental);
});
```

```
    fs.writeFileSync(filepath, JSON.stringify(rentalsJSON, null, 2));

    res.status(201).json(rental);
  });

  // Endpoint GET: retornar tots els lloguers
  app.get('/list', (req, res) => {
    res.json(rentalsJSON);
  });

  // Iniciar servidor
  app.listen(port, () => {
    console.log(`Servidor HTTP escoltant a http://localhost:${port}`);
  });
```