



## **Proyecto HelloZUM**

**Servicio REST realizado con NodeJS, Express JS  
y Docker.**

**Realizado por Antony Dixon Albites Tapia**

## Contenido

<b>1-Tecnologías Por Utilizar:</b> .....	3
<b>2-Creando el Servicio Rest:</b> .....	3
2.1 Creando un repositorio local .....	4
2.2 Iniciando con NodeJS y ExpressJS : .....	4
<b>3- “Dockerizando” Nuestro servicio.</b> .....	7
<b>4-Subiendo nuestro proyecto a GITHUB</b> .....	10

## 1-Tecnologías Por Utilizar:

- **NodeJS y ExpressJS:** Para el desarrollo del servicio rest donde solo expondremos una ruta “/rest” para retornar un Json: {helloZum: 'Bienvenido a HelloZUM'}.
- **Docker:** Para poder encapsular todo en un contenedor, primero crearemos una Imagen con un Dockerfile y después la ejecutaremos con docker-compose.
- **GitHub:** Una vez terminado con ayuda de git subiremos lo realizado a un repositorio en github.

## 2-Creando el Servicio Rest:

En esta sección crearemos el servicio con ayuda de NodeJS y ExpressJS, muy importante tener instalado NODE.JS antes de realizar los siguientes pasos, puedes instalar node.js de este enlace: <https://nodejs.org/es/>.

Descargar la versión recomendada y verificar si todo se instalo correctamente, abrir un CMD y escribir lo siguiente:

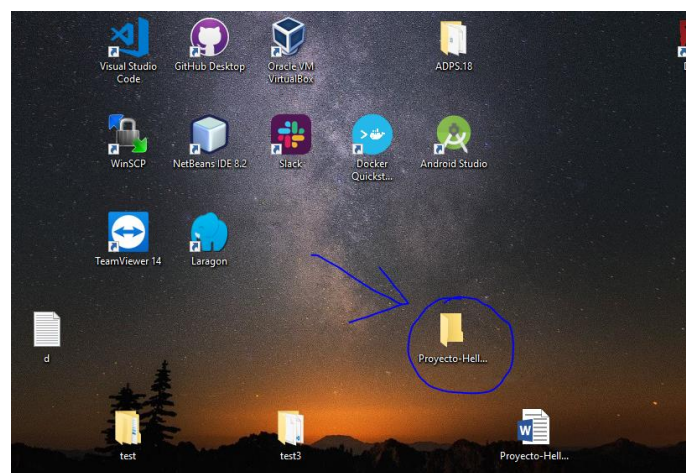
```
C:\Windows\System32\cmd.exe

C:\Windows\system32>node -v
v10.15.3

C:\Windows\system32>
```

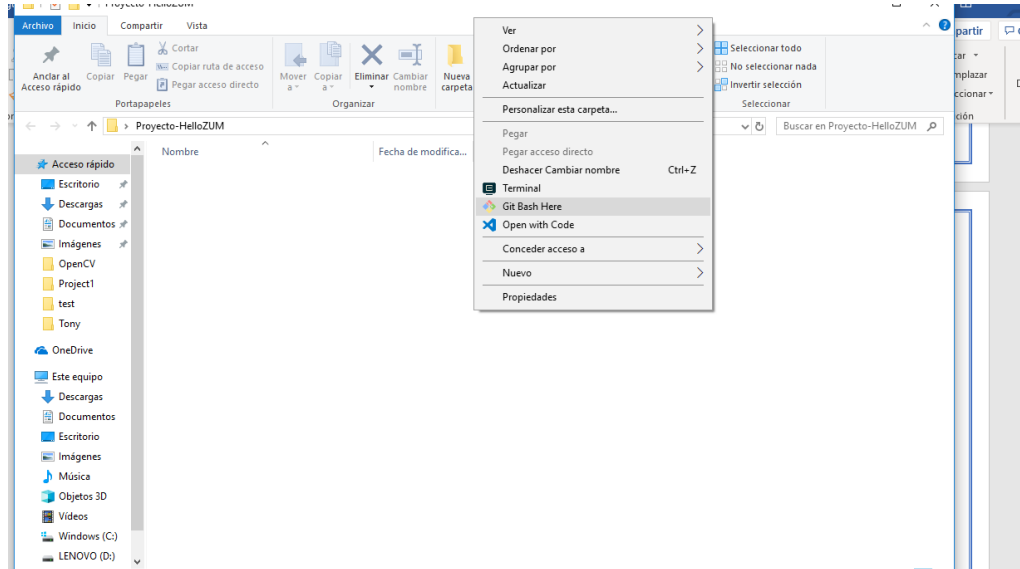
Si se muestra como la imagen, todo está correcto.

Crearemos una carpeta en el directorio de preferencia, yo lo haré en el escritorio.

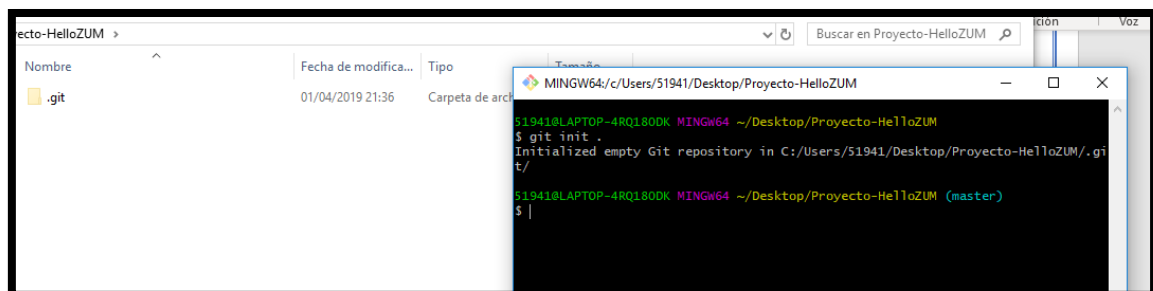


## 2.1 Creando un repositorio local

Dentro antes de iniciar con NODE.JS, crearemos un repositorio local con GIT, recuerda tenerlo instalado previamente, puedes instalarlo en el siguiente enlace: <https://git-scm.com/>. Instalar y después realizar lo siguiente dentro de la carpeta creada.



Iniciaremos un bash y dentro escribiremos lo siguiente.



Iniciaremos un repositorio local, por ahora es todo a realizar, puedes dejar el bash abierto, después con ayuda de él podremos subirlo a github.

## 2.2 Iniciando con NodeJS y ExpressJS :

Abrimos un CMD en la carpeta actual y escribiremos los siguientes comandos en este orden:

```
npm install express-generator -g
```

```
express --view=ejs api
```

```
cd api && npm install
```

```
npm start
```

## Proyecto HelloZUM

```
create : api\public\stylesheets\style.css
create : api\routes\
create : api\routes\index.js
create : api\routes\users.js
create : api\views\
create : api\views\error.ejs
create : api\views\index.ejs
create : api\app.js
create : api\package.json
create : api\bin\
create : api\bin\www

change directory:
> cd api

install dependencies:
> npm install

run the app:
> SET DEBUG=api:* & npm start

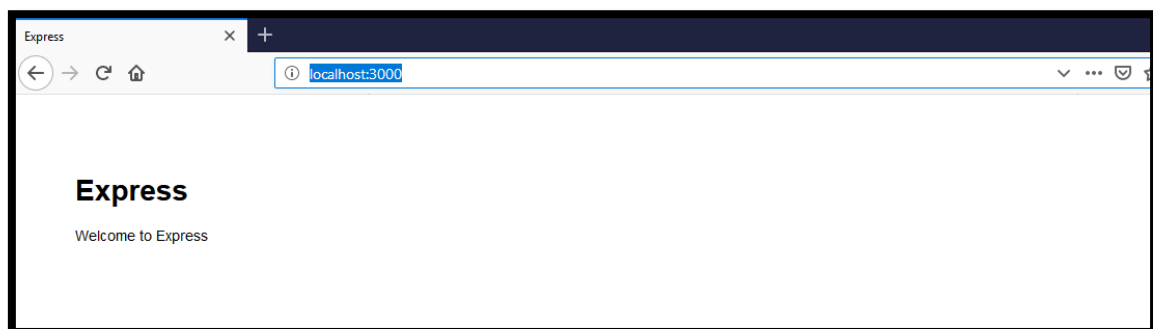
C:\Users\51941\Desktop\Proyecto-HelloZUM>cd api && npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
added 53 packages from 38 contributors and audited 141 packages in 9.586s
found 0 vulnerabilities

C:\Users\51941\Desktop\Proyecto-HelloZUM\api>npm start

C:\Users\51941\Desktop\Proyecto-HelloZUM\api>npm start

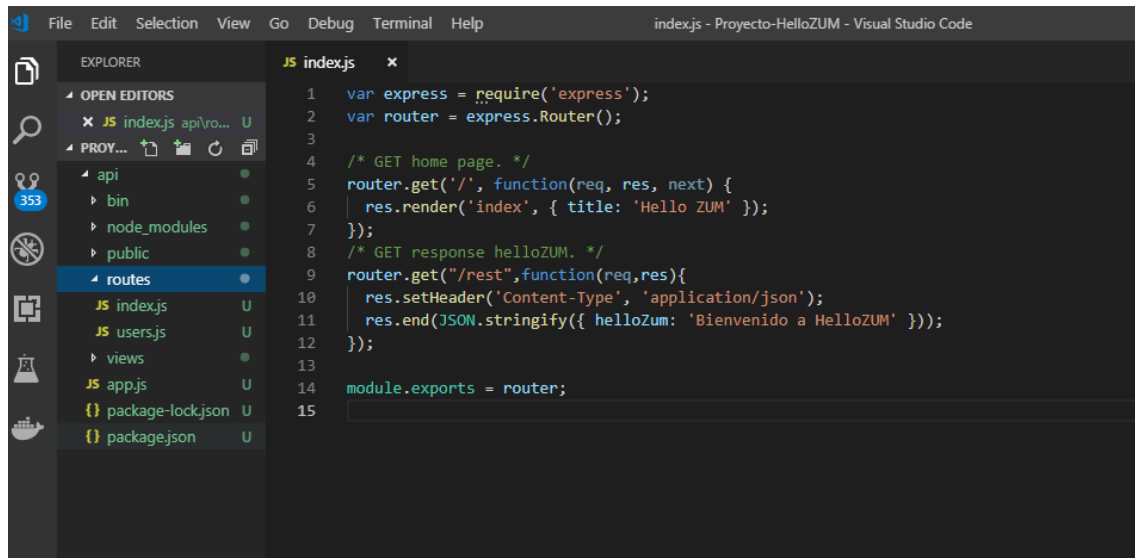
> api@0.0.0 start C:\Users\51941\Desktop\Proyecto-HelloZUM\api
> node ./bin/www
```

Ya se habrá creado nuestra aplicación Express, podemos acceder a ella mediante esta dirección : <http://localhost:3000/>



Ahora entraremos a la carpeta “API” y modificaremos unos archivos, puedes usar el editor de código de preferencia, en mi caso será Visual Studio Code.

## Proyecto HelloZUM

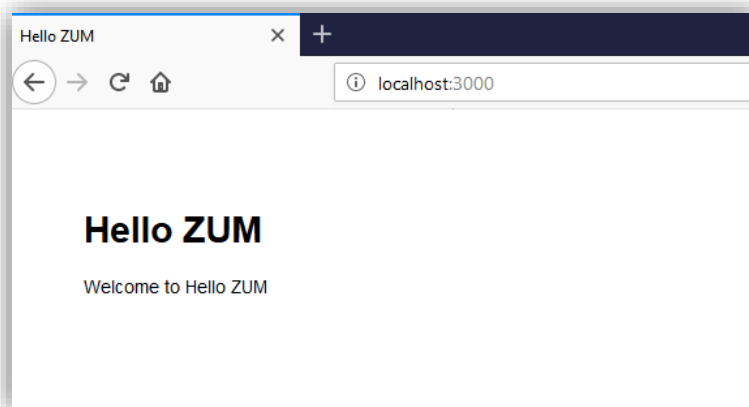


The screenshot shows the Visual Studio Code editor with the 'index.js' file open in the 'routes' folder. The code defines an Express application with two routes: a home page and a REST API endpoint.

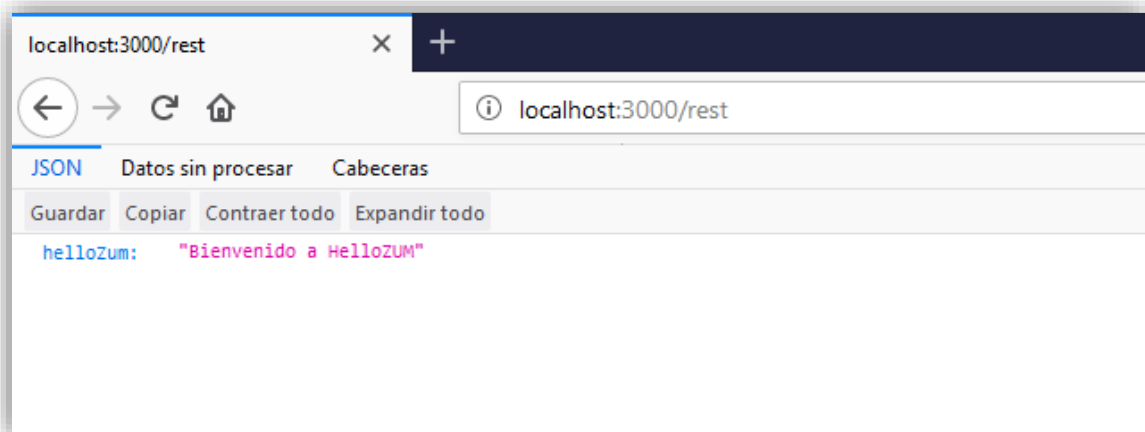
```
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('index', { title: 'Hello ZUM' });
7 });
8
9 /* GET response helloZUM. */
10 router.get("/rest",function(req,res){
11   res.setHeader('Content-Type', 'application/json');
12   res.end(JSON.stringify({ helloZum: 'Bienvenido a HelloZUM' }));
13 });
14
15 module.exports = router;
```

Ubicaremos el archivo que se visualiza en la imagen y lo modificaremos como se encuentra en la captura.

Después detendremos el comando “npm start” con CTRL+C y lo volveremos a ejecutar, accedemos a la ruta y se apreciará lo siguiente.



Entramos a la ruta ‘localhost:3000/rest’ y veremos lo siguiente.

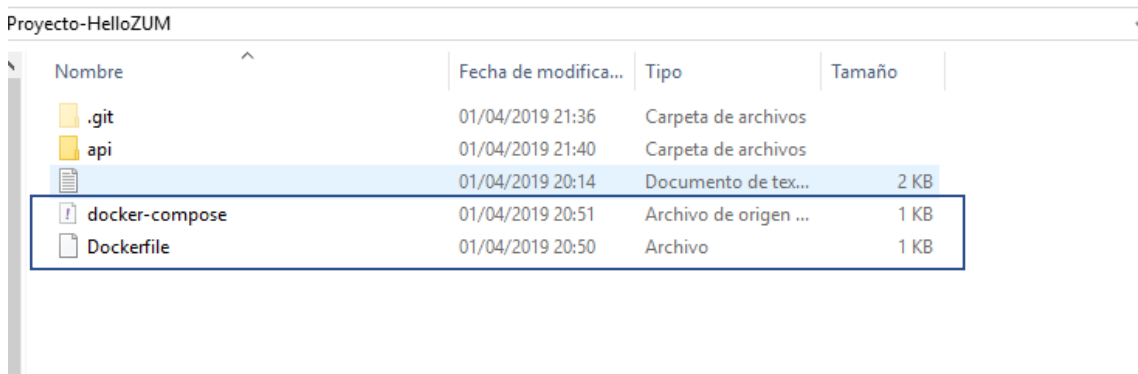


Por ahora ya tenemos terminado nuestro servicio, es hora de “dockerizarlo” ¡.

### 3- “Dockerizando” Nuestro servicio.

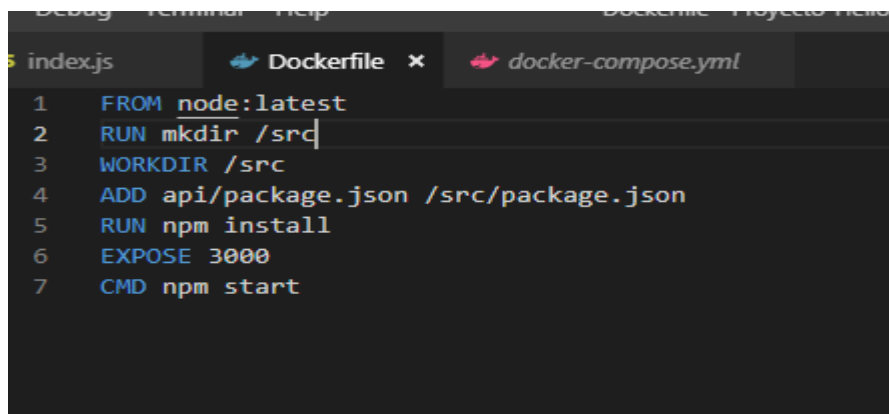
Como ya sabemos Docker trabaja con contenedores, y hace mucho más fácil el desarrollo en un equipo e incluso facilita el envío a producción. En esta sección veremos cómo configurar el dockerfile y el Docker-compose.yml para esta mini App.

Primero crearemos estos 2 archivos que se muestran en la imagen.



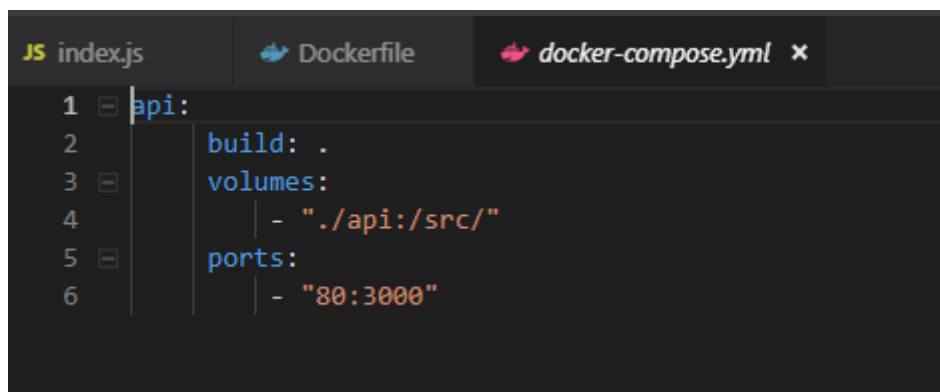
Nombre	Fecha de modifica...	Tipo	Tamaño
.git	01/04/2019 21:36	Carpeta de archivos	
api	01/04/2019 21:40	Carpeta de archivos	
	01/04/2019 20:14	Documento de tex...	2 KB
docker-compose	01/04/2019 20:51	Archivo de origen ...	1 KB
Dockerfile	01/04/2019 20:50	Archivo	1 KB

- 1- Docker-compose.yml
- 2- Dockerfile



```
1 FROM node:latest
2 RUN mkdir /src
3 WORKDIR /src
4 ADD api/package.json /src/package.json
5 RUN npm install
6 EXPOSE 3000
7 CMD npm start
```

Este es el contenido del Dockerfile. Donde básicamente es crear una imagen de node:latest, donde “latest” es el tag, que significa que traerá la última versión de Dockerhub. Creará un directorio llamado “src” donde ahí estará nuestra aplicación, donde justamente con el comando “add” copiaremos de nuestro archivo local “package.json” al del contenedor, después instalaremos nuestras dependencias con el archivo “package.json”, expone el puerto 3000 y por último ejecutará NODE.JS.



```
1 api:
2   build: .
3   volumes:
4     - "./api:/src/"
5   ports:
6     - "80:3000"
```

El archivo docker-compose básicamente construye nuestro Dockerfile, “volumes”, permite copiar los archivos de nuestra máquina al Docker para que de esta forma podamos trabajar desde fuera sin la necesidad de meter los archivos al Docker cada que los actualizamos y por último con “ports” es como accederemos a este, mediante el puerto 80 ingresaremos al puerto 3000 de nuestro contenedor.

Y listo es hora de ejecutar el docker-compose, ejecutaremos los siguientes comandos en este orden:

```
docker-compose build
```

```
docker-compose up -d
```

```
docker ps
```



```
Successfully tagged proyectohellozum_api:latest
C:\Users\51941\Desktop\Proyecto-HelloZUM>docker-compose up -d
Creating proyectohellozum_api_1 ... done
C:\Users\51941\Desktop\Proyecto-HelloZUM>docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED        STATUS        PORTS                    NAME
d20435e38073   proyectohellozum_api  "/bin/sh -c 'npm sta..." 4 seconds ago  Up 3 seconds  0.0.0.0:80->3000/tcp     proy
```

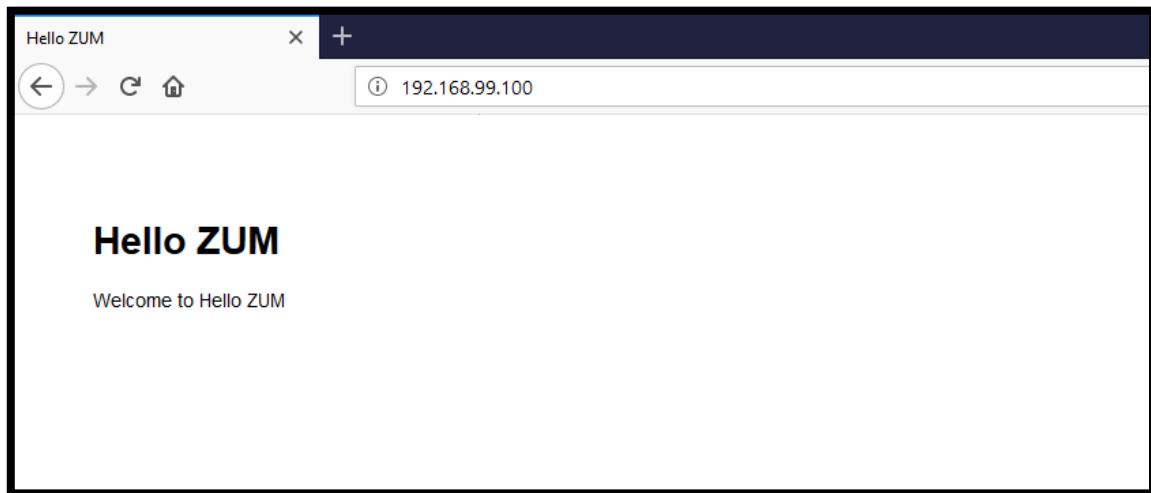
Tendremos una sección como esta, donde nos dice que el contenedor está ejecutado y está exponiendo un puerto 3000, nosotros accederemos por el puerto 80.



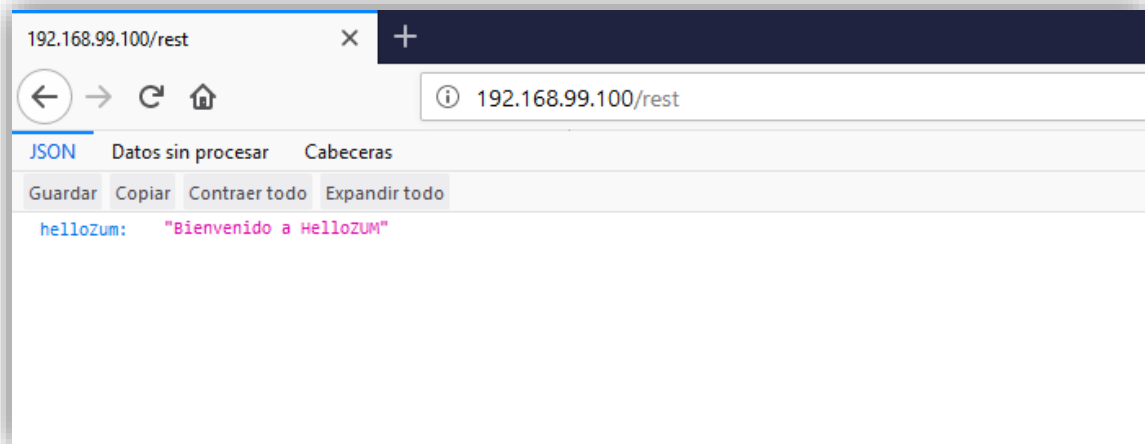
## Proyecto HelloZUM

```
C:\Windows\system32>docker-machine ip  
192.168.99.100
```

Dependiendo de la configuración en algunos casos se accede con “localhost:80” o en mi caso con accederé con la ip que muestra el comando “docker-machine ip”.



Ahora accederemos a la ruta “/rest”:



Y listo ya está todo funcionando ;

## 4-Subiendo nuestro proyecto a GITHUB

Ahora con nuestro bash de git abierto procederemos a realizar los siguiente.

Primero recuerda haber creado una cuenta en GITHUB, la puedes crear desde su página oficial: <https://github.com/>.

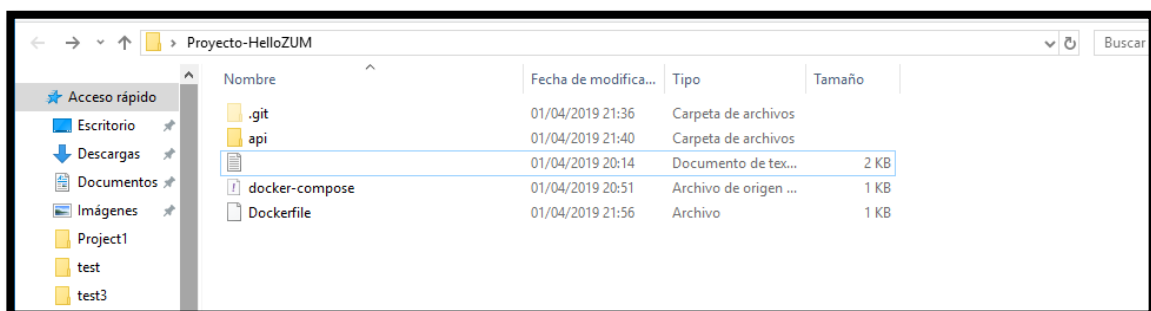
Una vez creado tendremos que registrar nuestros datos en nuestro GIT.

```
51941@LAPTOP-4RQ180DK MINGW64 ~/Desktop/Proyecto-HelloZUM (master)
$ git config --global user.name DevTony322
51941@LAPTOP-4RQ180DK MINGW64 ~/Desktop/Proyecto-HelloZUM (master)
$ git config --global user.email 2016236988@unfv.edu.pe
```

Con el comando “git config -l”, verificamos que todo este correcto.

```
user.name=DevTony322
user.email=2016236988@unfv.edu.pe
user.user=
color.ui=true
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

Antes de realizar nuestro commit, agregaremos un archivo “.gitignore”, para evitar subir archivos innecesarios a nuestro repositorio en github.



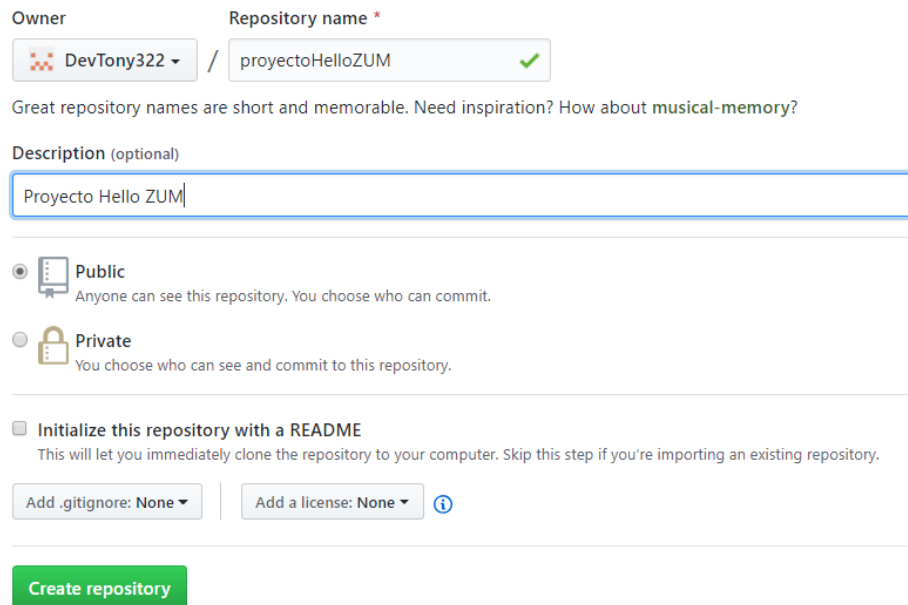
Ahora haremos nuestro commit

```
51941@LAPTOP-4RQ180DK MINGW64 ~/Desktop/Proyecto-HelloZUM (master)
$ git add .
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory
```

## Proyecto HelloZUM

```
51941@LAPTOP-4RQ180DK MINGW64 ~/Desktop/Proyecto-HelloZUM (master)
$ git commit -m "First Commit"
[master (root-commit) ad641d3] First Commit
12 files changed, 693 insertions(+)
```

Ahora, tendremos que crear un repositorio en github.



The screenshot shows the GitHub 'Create new repository' form. The 'Owner' is 'DevTony322' and the 'Repository name' is 'proyectoHelloZUM', which is marked as valid with a green checkmark. The 'Description' field contains 'Proyecto Hello ZUM'. The 'Visibility' is set to 'Public'. The 'Initialize this repository with a README' checkbox is checked. At the bottom, there are dropdowns for '.gitignore' (set to 'None') and 'License' (set to 'None'), followed by a green 'Create repository' button.

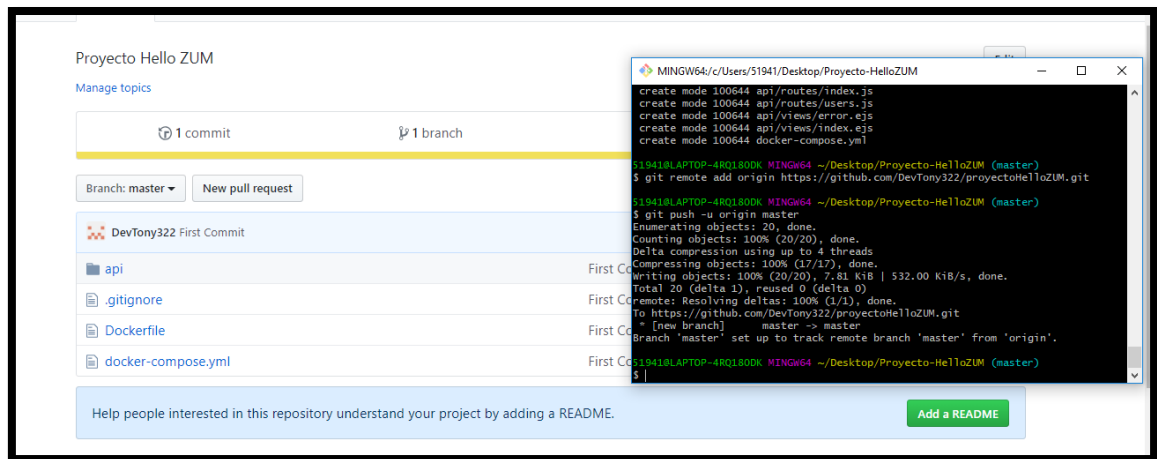
Creamos el repositorio y por último escribiremos los siguientes comandos:

```
51941@LAPTOP-4RQ180DK MINGW64 ~/Desktop/Proyecto-HelloZUM (master)
$ git remote add origin https://github.com/DevTony322/proyectoHelloZUM.git

51941@LAPTOP-4RQ180DK MINGW64 ~/Desktop/Proyecto-HelloZUM (master)
$ git push -u origin master
```

Donde agregamos nuestro repositorio en github y subimos todos nuestros cambios a la rama máster.

## Proyecto HelloZUM



Y vemos que todo se subió correctamente.

:D