

Docker

Back-End	1
MongoDB	1
Visual Studio	4
Front-End	6
Visual Studio	6
Imágenes y contenedores en Docker	7
Docker Desktop	8
De IP Local a IP pública	10
Ngrok	10

Back-End

MongoDB

1. Creamos una cuenta en Mongoddb, nos logueamos con las credenciales y creamos un cluster para almacenar los datos de la aplicación

MONGODB USER: **carlalorenapaz**

MONGODB PASSWORD: **y1YkAJ06CyIPvyCB**

Connect to ClusterSC2-1



You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

1. Add a connection IP address

✓ Your current IP address (186.22.18.93) has been added to enable local connectivity. Only an IP address you add to your Access List will be able to connect to your project's clusters. Add more later in [Network Access](#).

2. Create a database user

This first user will have [atlasAdmin](#) permissions for this project.

We autogenerated a username and password. You can use this or create your own.

i You'll need your database user's credentials in the next step. Copy the database user password.

Username

carlalorenepaz

Password

y1YkAJ06CyIPvyCB

HIDE

Copy

Create Database User

Close

Choose a connection method

Connect to ClusterSC2-1



Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver	Version
Node.js ▼	5.5 or later ▼

2. Install your driver

Run the following on the command line

```
npm install mongodb
```

[View MongoDB Node.js Driver installation instructions.](#)

3. Add your connection string into your application code

Use this connection string in your application

☐ View full code sample ☒ Show Password ⓘ

```
mongodb+srv://carlalorenapaz:y1YkAJ06CyIPvyCB@clustersc2-1.93tv4.mongodb.net/?  
retryWrites=true&w=majority&appName=ClusterSC2-1
```

The password for **carlalorenapaz** is included in the connection string for your first time setup. **This password will not be available again after exiting this connect flow.**

RESOURCES

[Get started with the Node.js Driver](#)

[Access your Database Users](#)

[Node.js Starter Sample App](#)

[Troubleshoot Connections](#)

Go Back

Done

Visual Studio

En la terminal de visual studio, apuntamos a la carpeta donde se encuentra ubicado el desarrollo de back-end

2 - Instalar node js

3- instalar express en el proyecto : `npm i express`

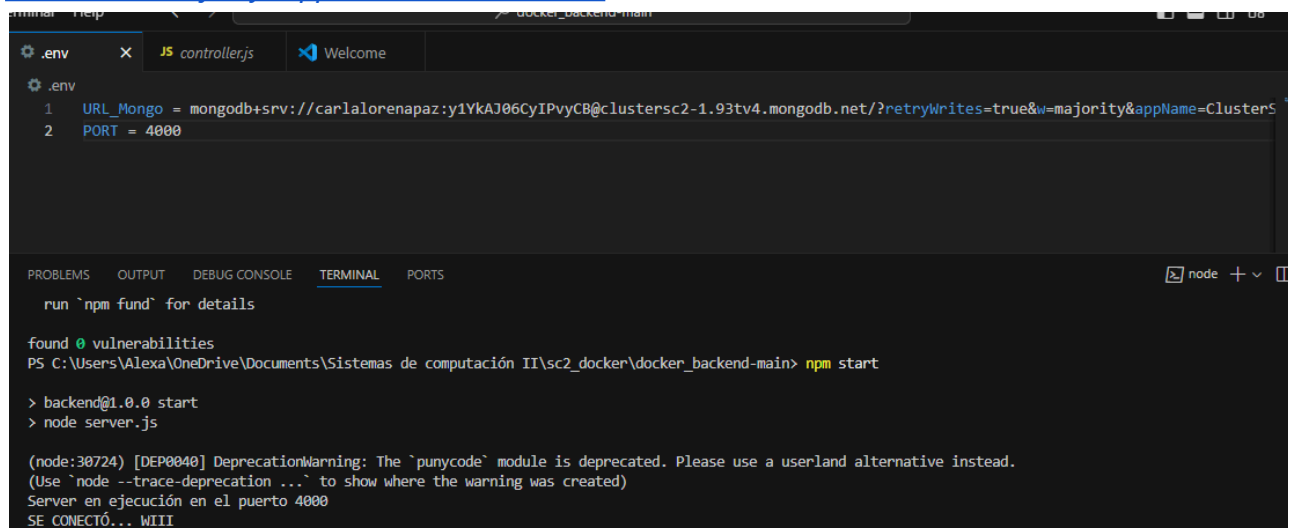
4: instalar mongodb: `npm install mongodb`

5- instalar cors: `npm i cors`

6- Utilizamos la URL y contraseña de usuario de mongodb para conectarnos desde el Back.

Copiamos la url que nos devuelve mongodb y la guardamos en una variable dentro `.env` del Back-end:

`mongodb+srv://carlalorenapaz:<db_password>@clustersc2-1.93tv4.mongodb.net/?retryWrites=true&w=majority&appName=ClusterSC2-1`



The screenshot shows the Visual Studio interface. The top part displays the `.env` file with two lines: `1 URL_Mongo = mongodb+srv://carlalorenapaz:y1YkAJ06CyIPvyCB@clustersc2-1.93tv4.mongodb.net/?retryWrites=true&w=majority&appName=ClusterSC2-1` and `2 PORT = 4000`. The bottom part shows the terminal output for the command `npm start`. The output includes a deprecation warning for the `punycode` module and a message indicating the server is running on port 4000.

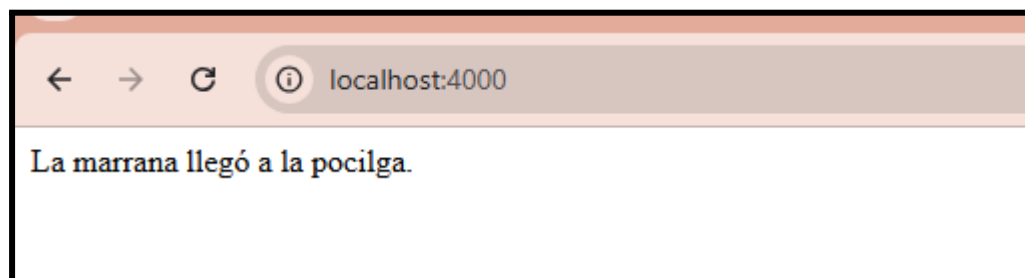
7- Prueba de conexión

En la terminal del proyecto, ejecutamos:

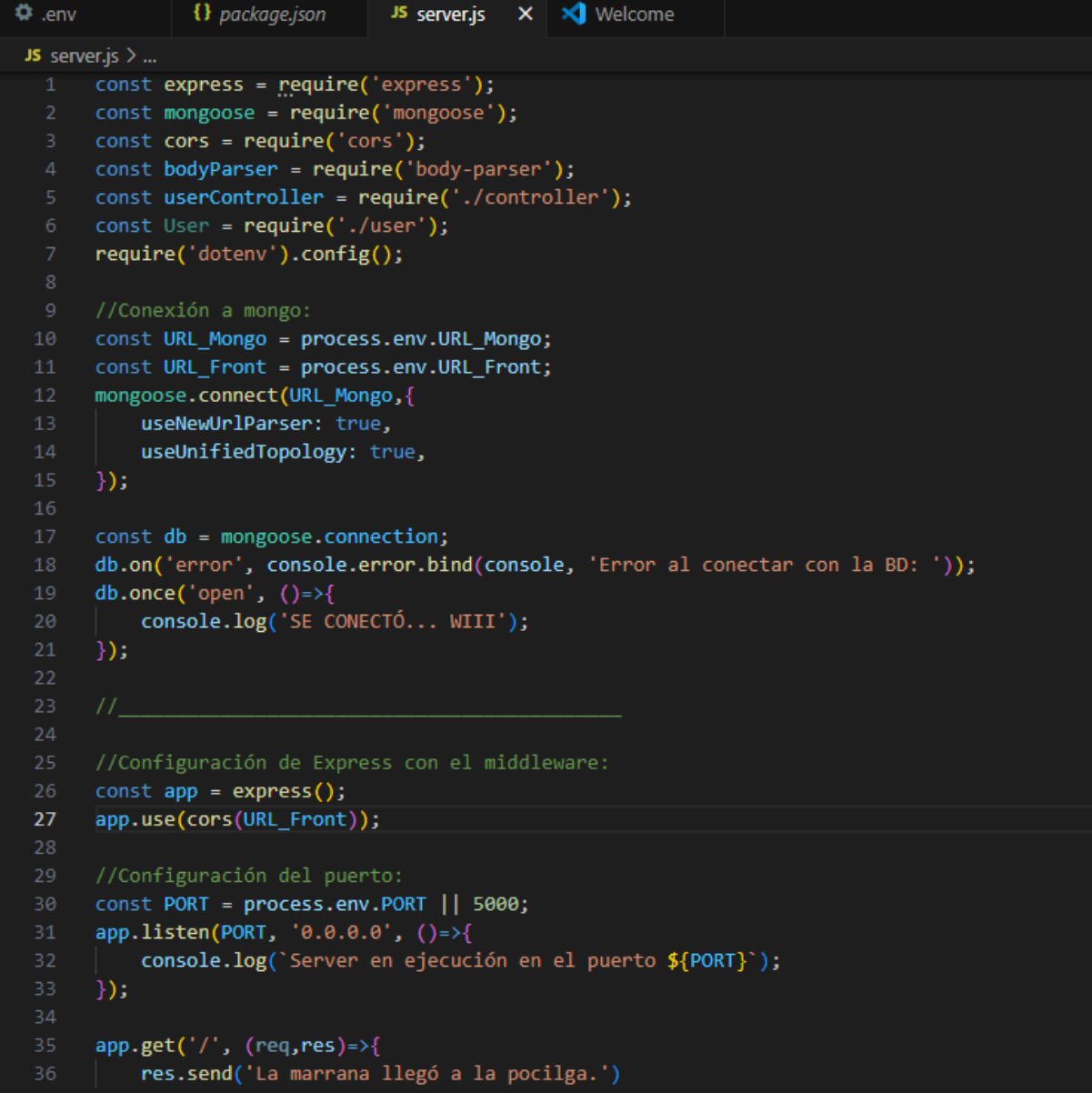
`npm i`

`npm start`

En el buscador escribimos `http://localhost:4000`



8- Agregamos la URL Local que apunta al front-end para correr todo el proyecto desde el Front-end



```
.env package.json JS server.js X Welcome
JS server.js > ...
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const cors = require('cors');
4  const bodyParser = require('body-parser');
5  const userController = require('./controller');
6  const User = require('./user');
7  require('dotenv').config();
8
9  //Conexión a mongo:
10 const URL_Mongo = process.env.URL_Mongo;
11 const URL_Front = process.env.URL_Front;
12 mongoose.connect(URL_Mongo,{
13   useNewUrlParser: true,
14   useUnifiedTopology: true,
15 });
16
17 const db = mongoose.connection;
18 db.on('error', console.error.bind(console, 'Error al conectar con la BD: '));
19 db.once('open', ()=>{
20   console.log('SE CONECTÓ... WIII');
21 });
22
23 // _____
24
25 //Configuración de Express con el middleware:
26 const app = express();
27 app.use(cors(URL_Front));
28
29 //Configuración del puerto:
30 const PORT = process.env.PORT || 5000;
31 app.listen(PORT, '0.0.0.0', ()=>{
32   console.log(`Server en ejecución en el puerto ${PORT}`);
33 });
34
35 app.get('/', (req,res)=>{
36   res.send('La marrana llegó a la pocilga.')
37 })
```

Front-End

Visual Studio

- 1 - Install node JS: `npm install vite --save-dev`
- 2 -Correr en desarrollo: `npm run dev`

```
npm audit fix --force

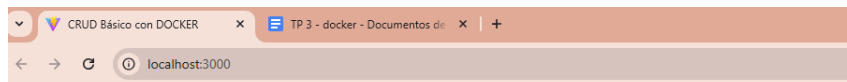
Run `npm audit` for details.
PS C:\Users\Alexa\OneDrive\Documents\Sistemas de computación II\sc2_docker\docker_frontend-main> npm run dev
>>

> dockerfront@0.0.0 dev
> vite

VITE v5.4.10 ready in 458 ms

  → Local:   http://localhost:3000/
  → Network: http://192.168.0.14:3000/
  → press h + enter to show help
PS C:\Users\Alexa\OneDrive\Documents\Sistemas de computación II\sc2_docker\docker_frontend-main>
```

- 3- En el buscador vamos a escribir la url local del Front : <http://localhost:3000>



CRUD de Usuarios

- Crear Usuario
- Ver Todos los Usuarios
- Buscar Usuario
- Actualizar Usuario
- Eliminar Usuario

Imágenes y contenedores en Docker

Para unir las imágenes de frontend y backend utilizamos **Docker Compose**, por lo cual se agregó el archivo `docker-compose.yml` que defina ambos servicios y cómo deben comunicarse.:

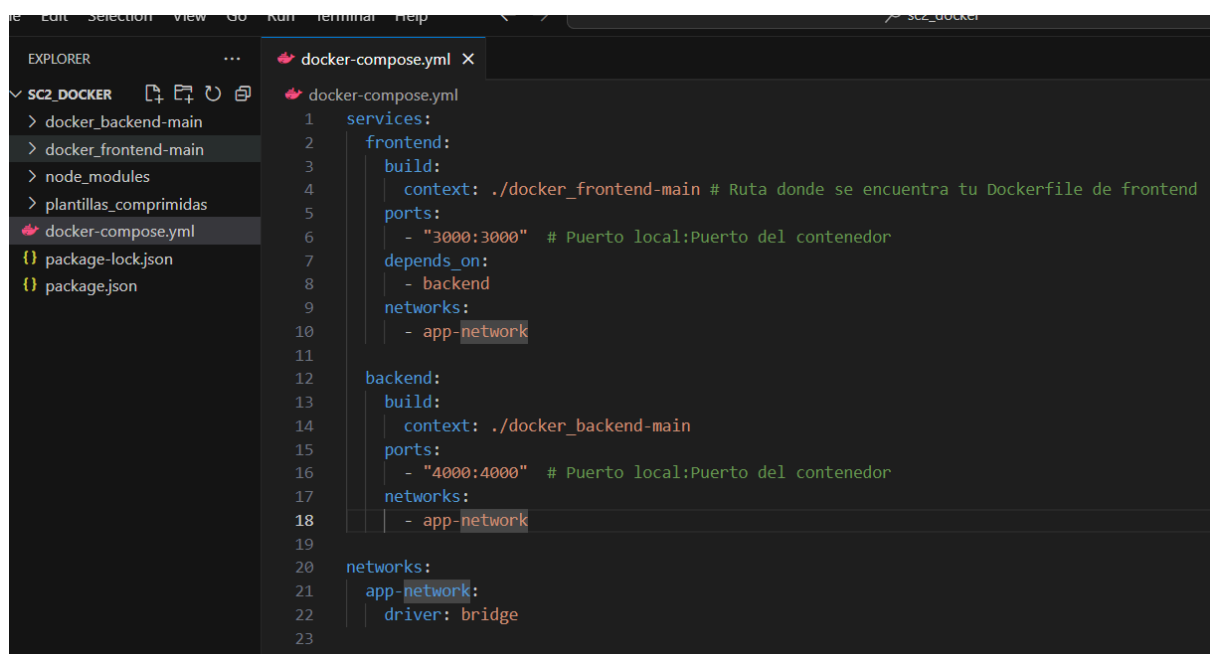
Estructura del proyecto:

```
project-root/
├── docker-compose.yml
├── frontend/
│   └── Dockerfile
└── backend/
    └── Dockerfile
```

Configura los Dockerfiles: Cabe destacar que tanto el back como el front tienen su Dockerfile en cada directorio los cuales construyen las imágenes correctamente.

El Dockerfile del frontend debería exponer el puerto donde se ejecuta (ej. `3000`), y el Dockerfile del backend debería exponer el puerto de su API (ej. `4000`).

Archivo `docker-compose.yml`: En la raíz del proyecto, se creó el archivo con el siguiente código



```
1  services:
2    frontend:
3      build:
4        context: ./docker_frontend-main # Ruta donde se encuentra tu Dockerfile de frontend
5      ports:
6        - "3000:3000" # Puerto local:Puerto del contenedor
7      depends_on:
8        - backend
9      networks:
10       - app-network
11
12    backend:
13      build:
14        context: ./docker_backend-main
15      ports:
16        - "4000:4000" # Puerto local:Puerto del contenedor
17      networks:
18       - app-network
19
20  networks:
21    app-network:
22      driver: bridge
23
```

1. Explicación del archivo `docker-compose.yml`:

- **version:** Especifica la versión de Docker Compose.
- **services:** Define cada servicio (en este caso, `frontend` y `backend`).

- **build.context:** Define el directorio donde se encuentra cada Dockerfile.
- **ports:** Expone los puertos de cada servicio en tu máquina local.
- **depends_on:** Define la dependencia del **frontend** en el **backend**, asegurando que el backend se inicie primero.
- **networks:** Crea una red compartida llamada **app-network** para que los servicios puedan comunicarse entre sí por sus nombres de servicio (**frontend** y **backend**).

2. Conexión entre frontend y backend:

- En el código del frontend, cuando necesites hacer una solicitud a la API del backend, usa **http://backend:4000** en lugar de **localhost:4000**. Esto funciona porque Docker Compose conecta los servicios en la misma red y permite que se comuniquen usando sus nombres de servicio.

Construir y ejecutar los servicios:

Docker Desktop

1 En la terminal de desktop , vamos a la carpeta donde están las dos partes (back-end y front-end) y ejecutamos la siguiente línea:

docker-compose up --build

The screenshot shows the Docker Desktop interface. The 'Containers' tab is active, displaying a table of running containers. Below the table, a terminal window shows the output of the 'docker-compose up --build' command.

Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
sc2_docker	-	-	-	0.16%	33 minutes ago	[Stop] [Restart] [Delete]
backend-1	02d285137a56	sc2_docker-backend	4000:4000	0.01%	33 minutes ago	[Stop] [Restart] [Delete]
frontend-1	630b0effda4d	sc2_docker-frontend	3000:3000	0.15%	33 minutes ago	[Stop] [Restart] [Delete]

```

PS C:\Users\Alexa\OneDrive\Documents\Sistemas de computación II> cd sc2_docker
PS C:\Users\Alexa\OneDrive\Documents\Sistemas de computación II\sc2_docker> docker-compose up --build
[+] Building 3.8s (28/28) FINISHED
=> (backend internal) load build definition from Dockerfile
=> == transferring dockerfile: 971B
=> (frontend internal) load metadata for docker.io/library/node:alpine
=> [backend auth] library/node:pull token for registry-1.docker.io
=> [backend internal] load .dockerignore
=> == transferring context: 60B
=> [frontend 1/5] FROM docker.io/library/node:alpine@sha256:1467ea23cce893347696b155b9e0e7f0ac7d21555eb6f27236f1328212e045e
=> == resolve docker.io/library/node:alpine@sha256:1467ea23cce893347696b155b9e0e7f0ac7d21555eb6f27236f1328212e045e
=> [backend internal] load build context
=> == transferring context: 270B
=> CACHED [frontend 2/5] WORKDIR /app
=> CACHED [backend 3/5] COPY package*.json .
=> CACHED [backend 4/5] RUN npm install
=> CACHED [backend 5/5] COPY . .
=> [backend] exporting to image
=> == exporting layers
  
```

2 - Testamos la conexión, corriendo el buscador **http://localhost:3000/**



CRUD de Usuarios

- [Crear Usuario](#)
- [Ver Todos los Usuarios](#)
- [Buscar Usuario](#)
- [Actualizar Usuario](#)
- [Eliminar Usuario](#)

De IP Local a IP pública

NGROK

En los requisitos del tp, el acceso debe estar en IP Pública ,osea el front debe estar con ip público en ngrok

- 1- Nos logueamos en ngrok con un usairo y contraseña
- 2- Desde el Dashboard descargamos el ejecutable comprimido

Install ngrok via Chocolatey with the following command:

```
choco install ngrok
```



Run the following command to add your authtoken to the default ngrok.yml [configuration file](#).

```
ngrok config add-authtoken 2oiC0Y0TfJNWdX4x1fsTAZioakc_7WrM8r12J5zdRtJo
```



Deploy your app online

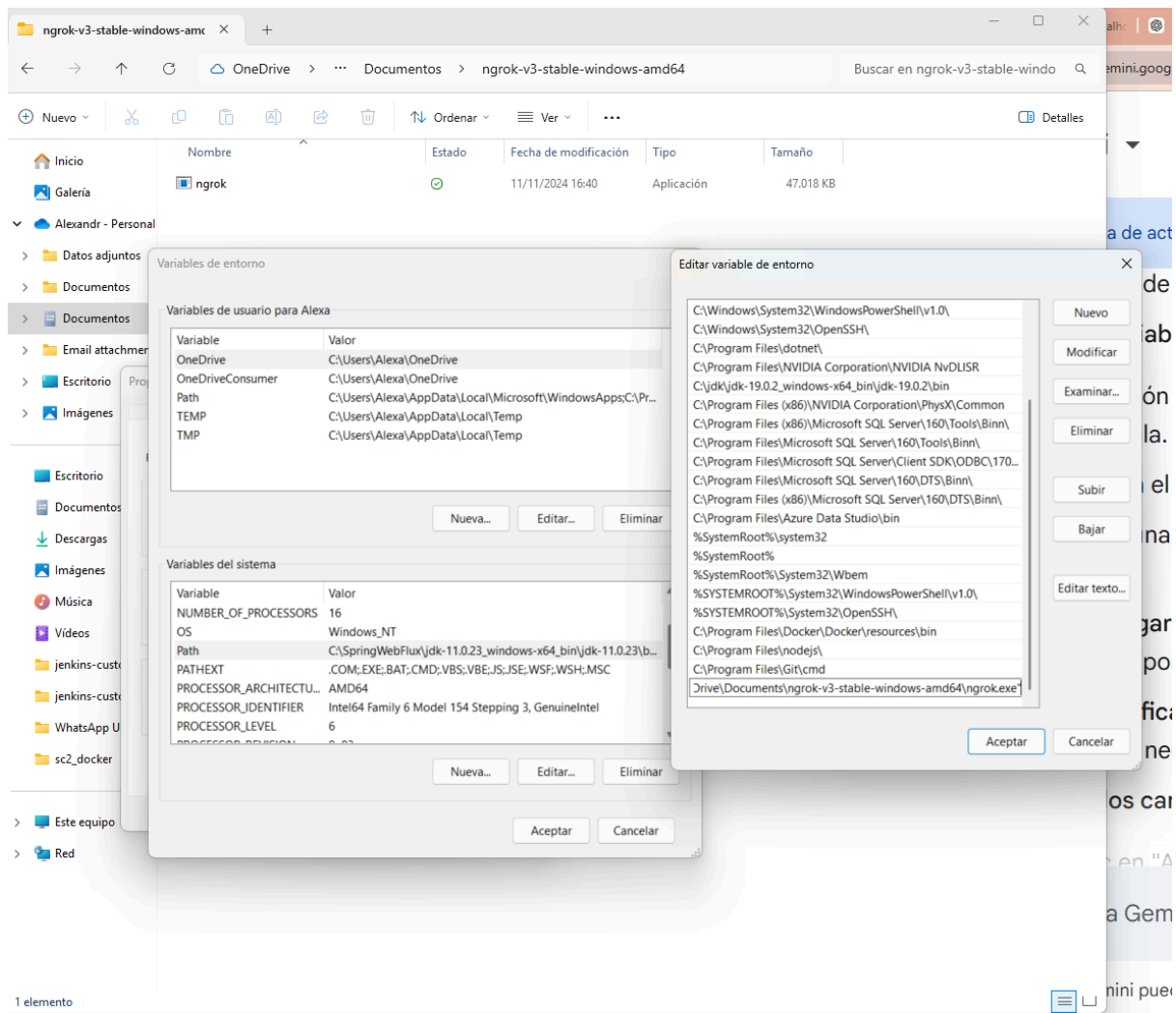
[Ephemeral Domain](#) [Static Domain](#)

Put your app online at an [ephemeral domain](#) forwarding to your upstream service. For example, if it is listening on port `http://localhost:8080`, run:

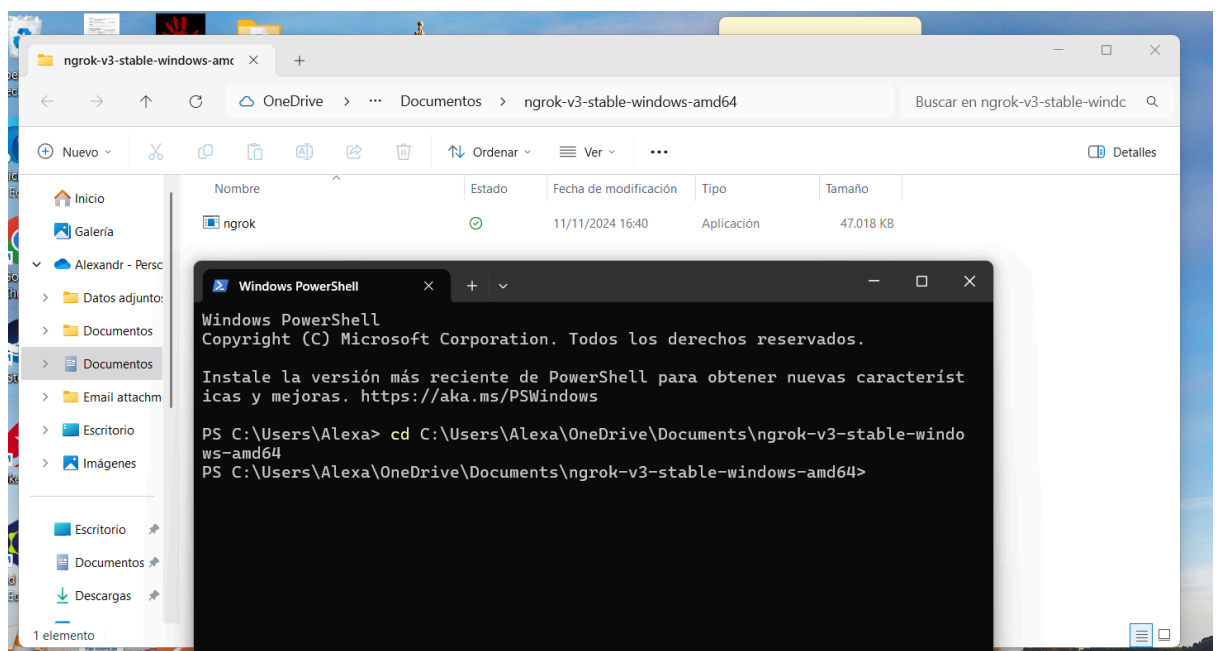
```
ngrok http http://localhost:8080
```



- 3- Descomprimimos el archivo en una carpeta deseada.
- 4- En windows, vamos a las variables de entorno y buscamos la variable PATH. Una vez que hacemos click dentro, agregamos la ruta donde se encuentra el ejecutable ngrok.exe



5 - En la terminal de Powershell apuntamos a esa carpeta y vemos la versión instalada de dicho archivo para verificar que se haya instalado correctamente



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características.

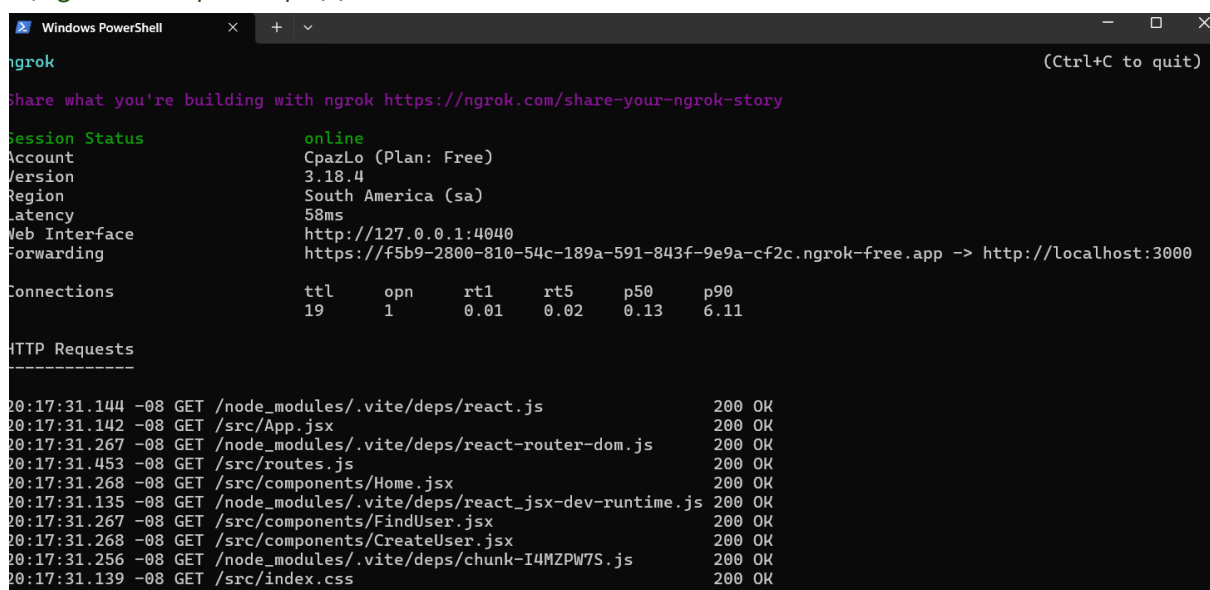
PS C:\Users\Alexa> cd C:\Users\Alexa\OneDrive\Documents\ngrok
PS C:\Users\Alexa\OneDrive\Documents\ngrok> ngrok version
```

4- Una vez comprobada la instalación, nos autenticamos con el siguiente comando:

```
.\ngrok config add-authtoken
2oiCOY0TfJNWdX4x1fsTAZioakc_7WrM8r12J5zdRtJoGGbjE
```

5 - Le indicamos que IP Local queremos convertir a pública con el comando:

```
.\ngrok http http://localhost:3000
```



```
Windows PowerShell
ngrok (Ctrl+C to quit)

Share what you're building with ngrok https://ngrok.com/share-your-ngrok-story

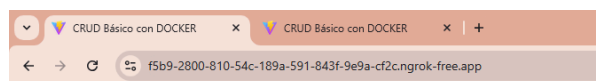
Session Status      online
Account             CpazLo (Plan: Free)
Version             3.18.4
Region              South America (sa)
Latency             58ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://f5b9-2800-810-54c-189a-591-843f-9e9a-cf2c.ngrok-free.app -> http://localhost:3000

Connections
  ttl    opn    rt1    rt5    p50    p90
   19     1     0.01   0.02   0.13   6.11

HTTP Requests
-----
20:17:31.144 -08 GET /node_modules/.vite/deps/react.js          200 OK
20:17:31.142 -08 GET /src/App.jsx                          200 OK
20:17:31.267 -08 GET /node_modules/.vite/deps/react-router-dom.js 200 OK
20:17:31.453 -08 GET /src/routes.js                          200 OK
20:17:31.268 -08 GET /src/components/Home.jsx                       200 OK
20:17:31.135 -08 GET /node_modules/.vite/deps/react_jsx-dev-runtime.js 200 OK
20:17:31.267 -08 GET /src/components/FindUser.jsx                     200 OK
20:17:31.268 -08 GET /src/components/CreateUser.jsx                    200 OK
20:17:31.256 -08 GET /node_modules/.vite/deps/chunk-I4MZPW7S.js    200 OK
20:17:31.139 -08 GET /src/index.css                          200 OK
```

6 - Probamos la conexión con la URL generada:

<https://f5b9-2800-810-54c-189a-591-843f-9e9a-cf2c.ngrok-free.app>



CRUD de Usuarios

- [Crear Usuario](#)
- [Ver Todos los Usuarios](#)
- [Buscar Usuario](#)
- [Actualizar Usuario](#)
- [Eliminar Usuario](#)