Universidad de La Laguna

Práctica 3

Inicialmente me gustaría dejar un recordatorio, pues durante la realización de la práctica a nivel de puertos, pues al reiniciar mi dispositivo me suelen salir de forma predeterminada ocupados por ello muestro los comandos que suelo utilizar para arreglar estas situaciones:

```
usuario@debian:~/mean-stack-example/server$ npx ts-node src/server.ts
Error: listen EADDRINUSE: address already in use :::5200
    at Server.setupListenHandle [as _listen2] (node:net:1811:16)
    at listenInCluster (node:net:1859:12)
    at Server.listen (node:net:1947:7)
    at Function.listen (/home/usuario/mean-stack-example/server/node_modules/express/lib/application.js:635:24)
    at /home/usuario/mean-stack-example/server/src/server.ts:26:9
    at processTicksAndRejections (node:internal/process/task_queues:95:5) {
  code: 'EADDRINUSE',
  errno: -98,
  syscall: 'listen',
  address: '::',
  port: 5200
}
usuario@debian:~/mean-stack-example/server$ npx ts-node src/server.ts
Error: listen EADDRINUSE: address already in use :::5200
    at Server.setupListenHandle [as _listen2] (node:net:1811:16)
    at listenInCluster (node:net:1859:12)
    at Server.listen (node:net:1947:7)
    at Function.listen (/home/usuario/mean-stack-example/server/node_modules/express/lib/application.js:635:24)
    at /home/usuario/mean-stack-example/server/src/server.ts:26:9
    at processTicksAndRejections (node:internal/process/task_queues:95:5) {
  code: 'EADDRINUSE',
  errno: -98,
  syscall: 'listen',
  address: '::',
  port: 5200
}
usuario@debian:~/mean-stack-example/server$ sudo lsof -i :5200
[sudo] contraseña para usuario:
COMMAND   PID    USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
node    11489 usuario   23u  IPv6  29568      0t0  TCP *:5200 (LISTEN)
usuario@debian:~/mean-stack-example/server$ sudo kill -9 11489
usuario@debian:~/mean-stack-example/server$ sudo lsof -i :5200
```

https://www.mongodb.com/docs/atlas/cli/current/verify-packages/#std-label-verify-packages COMANDOS DE INSTALACIÓN DE ATLAS PRIMEROS

Comenzamos con l a

Ejecución de los primeros pasos básicos, como lo es la comprobación de la versión del node y la creación de los archivos necesarios para el host que se realizará a la base de datos desde el back end. También me he tenido que instalar MongoDB para ello he tenido que realizar lo siguiente.

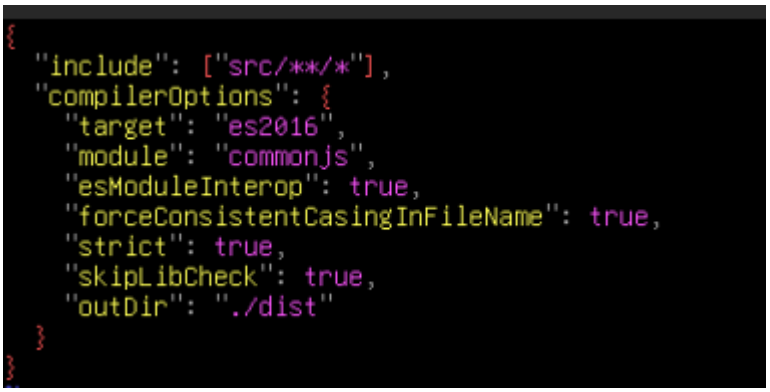
→ **npm install cors dotenv express mongodb**

Además necesitaremos de un un compilador TypeSrcipt y los correspondientes paquetes para su desarrollo:

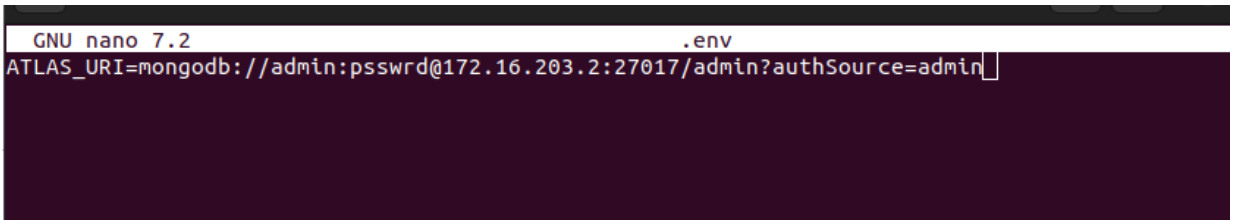
→ **npm install --save-dev typescript @types/cors @types/express @types/node ts-node**

```
usuario@debian:~$ node version
node:internal/modules/cjs/loader:1137
  throw err;
  ^

Error: Cannot find module '/home/usuario/version'
    at Module._resolveFilename (node:internal/modules/cjs/loader:1134:15)
    at Module._load (node:internal/modules/cjs/loader:975:27)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:128:12)
    at node:internal/main/run_main_module:28:49 {
  code: 'MODULE_NOT_FOUND',
  requireStack: []
}

Node.js v18.19.0
usuario@debian:~$ node --version
v18.19.0
usuario@debian:~$ mkdir mean-stack-example
usuario@debian:~$ cd mean-stack-example/
usuario@debian:~/mean-stack-example$ mkdir server
usuario@debian:~/mean-stack-example$ mkdir server/src
usuario@debian:~/mean-stack-example$ cd server/
usuario@debian:~/mean-stack-example/server$ npm init -y
Wrote to /home/usuario/mean-stack-example/server/package.json:

{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}


usuario@debian:~/mean-stack-example/server$ touch tsconfig.json .env
usuario@debian:~/mean-stack-example/server$ (cd src && touch database.ts employee.routes.ts employee.ts server.ts)
usuario@debian:~/mean-stack-example/server$ _
```

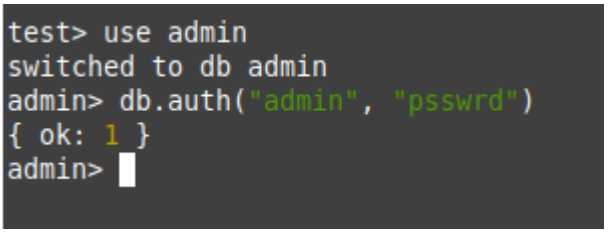
Configuración del archivo tsconfig.json



Conservaremos nuestros datos en un clúster de MongoDB Atlas. Para conectarnos a nuestro clúster, necesitaremos configurar una variable de entorno ATLAS_URI que contenga la cadena de conexión. Esta variable de entorno la añadiremos en el archivo server./env.



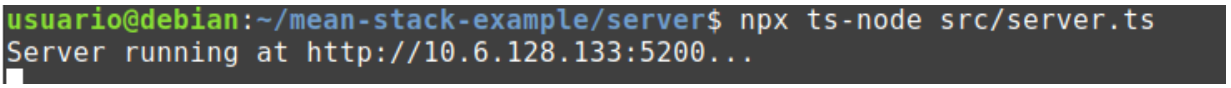
Como comprobación vemos efectivamente desde la tercera máquina virtual de base de datos que podemos acceder a la base de datos con dichos credenciales:



Dado que el archivo server.ts será nuestro punto de entrada, carguemos las variables de entorno desde él, conectémonos a la base de datos e iniciemos el servidor. Toda esta configuración queda realizada en el archivo server.ts.

Ejecutemos la aplicación y veamos que todo funciona:

Tras la configuración y la ejecución del comando vemos que funciona:



A continuación proseguimos con la implementación de GET, POST, PUT y DELETE endpoints para nuestros empleados. Toda esta configuracion la realizamos en el archivo:

mean-stack-example/server/src/employee.routes.ts

Ahora, debemos indicarle al servidor Express que use las rutas que hemos definido. Para ello realizamos una serie de modificaciones al archivo /**mean-stack-example/server/src/server.ts,** de forma que quede de la siguiente forma:

```
  GNU nano 7.2                                  src/server.ts *
import { employeeRouter } from "./employee.routes";
import * as dotenv from "dotenv";
import express from "express";
import cors from "cors";
import { connectToDatabase } from "./database";

// Load environment variables from the .env file, where the ATLAS_URI is configured
dotenv.config();

const { ATLAS_URI } = process.env;

if (!ATLAS_URI) {
  console.error(
    "No ATLAS_URI environment variable has been defined in config.env"
  );
  process.exit(1);
}

connectToDatabase(ATLAS_URI)
  .then(() => {
    const app = express();
    app.use(cors());

    app.use("/employees", employeeRouter);
    // start the Express server
    app.listen(5200, () => {
      console.log(`Server running at http://localhost:5200...`);
    });
  })
  .catch((error) => console.error(error));
```

Reiniciamos y vemos que la configuración ha sido realizada correctamente.

```
usuario@debian:~/mean-stack-example/server$ npx ts-node src/server.ts
Server running at http://10.6.128.133:5200...
```

Dejamos nuestro servidor en ejecución y en otra terminal procedemos a la instalación de Angular CLI mediante el siguiente comando:

→**sudo npm install -g @angular/cli**

A continuación procedemos a la creación de un nuevo proyecto Angular (client) desde el directorio root de nuestro proyecto, así pues obtendremos lo siguiente:

```
usuario@debian:~/mean-stack-example$ ng new client --inline-template --inline-style --minimal
--routing --style=css
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation
(SSG/Prerendering)? yes
CREATE client/README.md (1068 bytes)
CREATE client/.gitignore (587 bytes)
CREATE client/angular.json (2947 bytes)
CREATE client/package.json (1012 bytes)
CREATE client/tsconfig.json (1012 bytes)
CREATE client/tsconfig.app.json (485 bytes)
CREATE client/server.ts (1729 bytes)
CREATE client/.vscode/extensions.json (130 bytes)
CREATE client/.vscode/launch.json (297 bytes)
CREATE client/.vscode/tasks.json (531 bytes)
CREATE client/src/main.ts (250 bytes)
CREATE client/src/index.html (292 bytes)
CREATE client/src/styles.css (80 bytes)
CREATE client/src/main.server.ts (264 bytes)
CREATE client/src/app/app.component.ts (320 bytes)
CREATE client/src/app/app.config.ts (404 bytes)
CREATE client/src/app/app.routes.ts (77 bytes)
CREATE client/src/app/app.config.server.ts (350 bytes)
CREATE client/public/favicon.ico (15086 bytes)
✔ Packages installed successfully.
    Directory is already under version control. Skipping initialization of git.
```

Vemos que esto nos genera un nuevo directorio llamado client

A continuación accedemos a dicho directorio y ejecutamos

→ng serve --host 172.16.202.2 --port 4200

De esta forma nos aseguramos iniciar el servidor angular en la IP 172.16.202.2 y puerto 4200 mostrándonos por pantalla algo de la forma:

```
^Cusuario@debian:~/mean-stack-example/clientng serve --host 172.16.202.2 --port 420000
^[[C^[[D
Warning: This is a simple server for use in testing or debugging Angular applications
locally. It hasn't been reviewed for security issues.

Binding this server to an open connection can result in compromising your application or
computer. Using a different host than the one passed to the "--host" flag might result in
websocket connection issues. You might need to use "--disable-host-check" if that's the
case.

Browser bundles
Initial chunk files     | Names               |  Raw size
polyfills.js            | polyfills           |  90.20 kB |
main.js                 | main                |   1.79 kB |
styles.css              | styles              |  95 bytes |

                        | Initial total       |  92.08 kB


Server bundles
Initial chunk files     | Names               |  Raw size
polyfills.server.mjs    | polyfills.server    | 572.91 kB |
main.server.mjs         | main.server         |   2.23 kB |
render-utils.server.mjs | render-utils.server | 472 bytes |

Application bundle generation complete. [5.198 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  ➜  Network: http://172.16.202.2:4200/
  ➜  press h + enter to show help
```

Y efectivamente vemos como funciona y nos muestra por pantalla una ventana donde aparece lo siguiente al acceder al siguiente enlace: http://10.6.128.133/

A continuación nosotro usaremos Angular Material por lo que desde el directorio client ejecutamos lo siguiente para instalarlo:

→ng add @angular/material

A continuación crearemos la interfaz Angular para nuestros empleados. Para ello desde una nueva terminal ejecutamos:

→ng generate interface employee

Una vez realizado, configuramos el archivo:
mean-stack-example/client/src/app/employee.ts

```
export interface Employee {
  name: string;
  position: string;
  level: 'junior' | 'mid' | 'senior';
  :_id?: string;
}
```

A continuación creamos el servicio de empleados mediante el comando:

→ng generate service employee

Y configurando el archivo
mean-stack-example/client/src/app/employee.service.ts

como aparece en el tutorial.

Así pues estamos usando el servicio HttpClient para realizar solicitudes HTTP a nuestra API. El método refreshEmployees() se usa para obtener la lista completa de empleados y la guarda en una señal llamada employee$ que será accesible para el resto de la aplicación.

Acerca de esto, es importante decir que estamos usando el servicio HttpClient para realizar solicitudes HTTP a nuestra API. Este servicio HttpClient lo proporciona Angular a través de la función provideHttpClient y no es parte de la aplicación por defecto, por lo que debemos importarlo en el archivo app.config.ts. Por lo tanto, debemos tener la siguiente configuración:

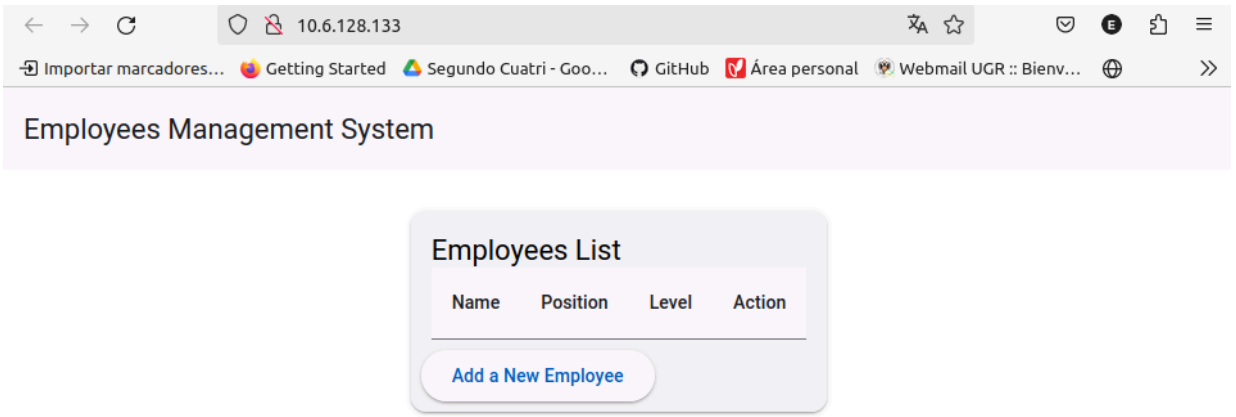
```
import { provideHttpClient, withFetch } from '@angular/common/http';
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideClientHydration } from '@angular/platform-browser';
import { provideAnimationsAsync } from '@angular/platform-browser/animations/async';

export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes),
    provideHttpClient(withFetch()),
    // ...
  ],
};
```

Ahora realizamos pasos similares con la lista de empleados:

→ng generate component employees-list

Y detallamos el siguiente archivo con la configuración especificada en el tutorial:
mean-stack-example/client/src/app/employees-list/employees-list.component.ts

Vemos que efectivamente tras hacer todo esto y nuestra página los cambios se ven reflejados:



Finalmente seguiríamos los siguientes pasos.

Crear una componente EmployeeForm y lo vamos a usar para añadir y editar:

→ng g c employee-form

Y posteriormente configuramos el archivo creado **mean-stack-example/client/src/app/employee-form/employee-form.component.ts.**

2. Implementamos AddEmployeeComponent:

→ng generate component add-employee

Y reemplazamos el contenido del archivo creado: **mean-stack-example/client/src/app/add-employee/add-employee.component.ts**

3. Implementamos la componente para editar empleados:

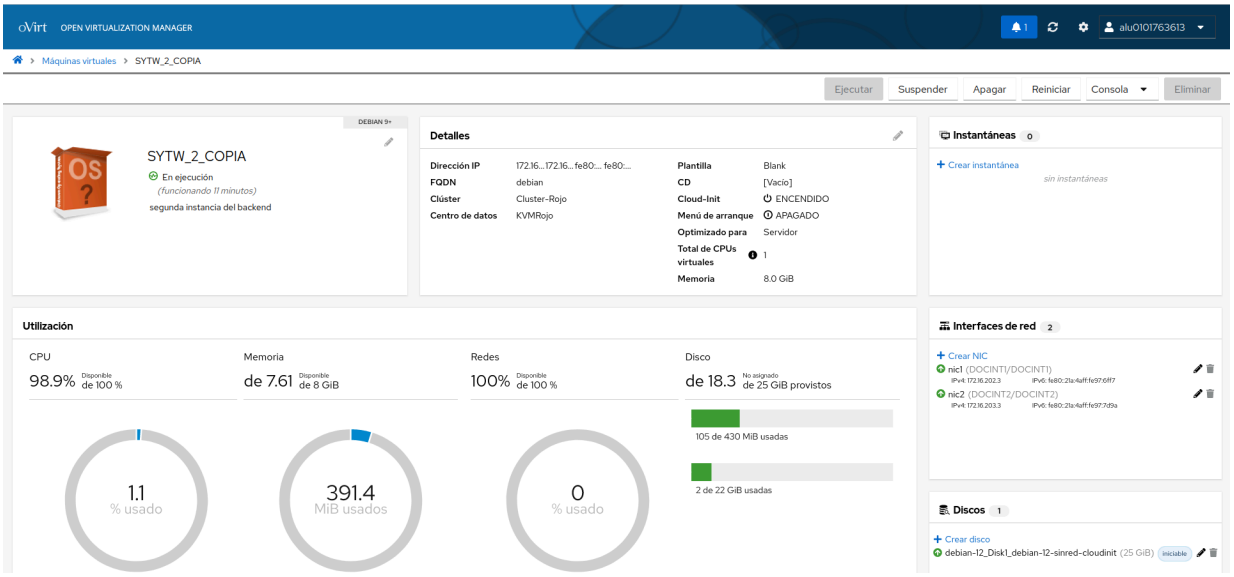
→ng generate component edit-employee

Y editamos el archivo creado: **mean-stack-example/client/src/app/edit-employee/edit-employee.component.ts**

Finalmente nos quedaría añadir la navegación de nuestra página configurando el archivo: **mean-stack-example/client/src/app/app-routing.module.ts**

Y ya tendríamos finalizado el tutorial.

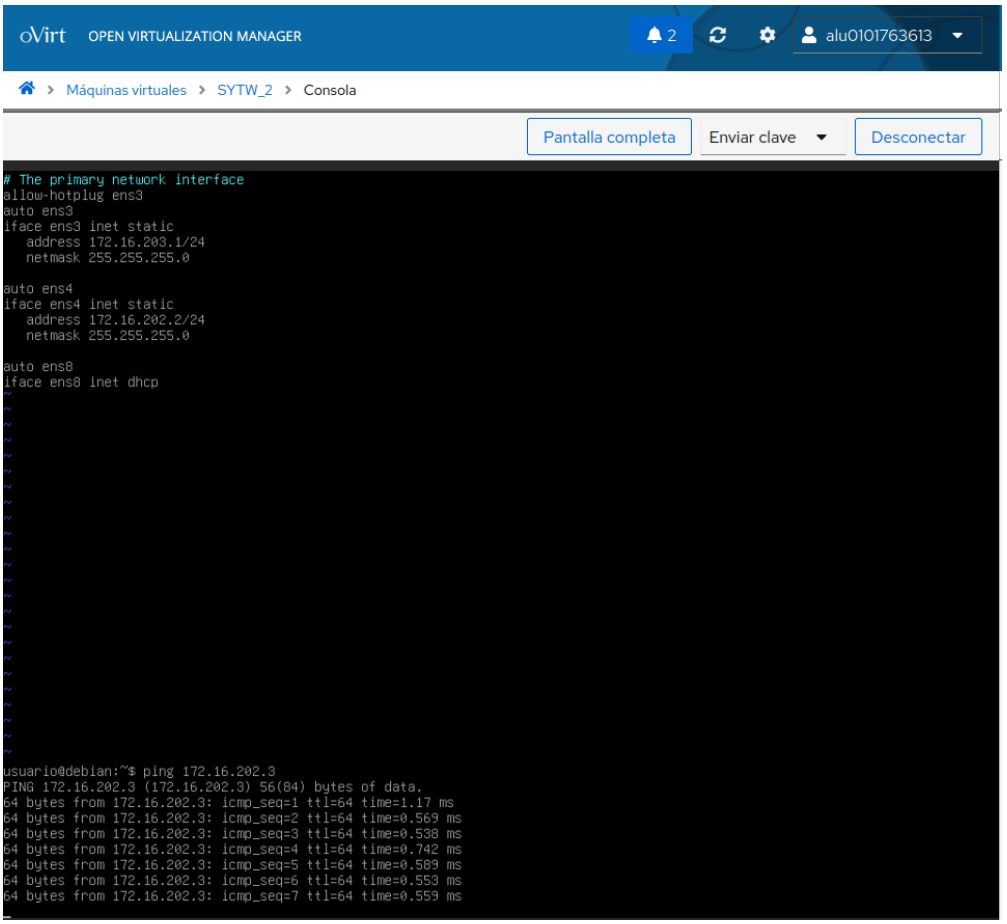
A continuación paso a la parte del clonado del backend, para ello creo una nueva máquina virtual con la misma configuración que en la práctica uno.



Una vez habiéndola creado edito el archivo /etc/network/interfaces de la siguiente forma:

```
source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto ens4
iface ens4 inet static
    address 172.16.202.3/24
    netmask 255.255.255.0

auto ens7
iface ens7 inet static
    address 172.16.203.3/24
    netmask 255.255.255.0
~
~
```

Una vez hecho esto levanto las interfaces de red y ejecuto **sudo systemctl restart networking** y vemos que efectivamente la configuración se ha realizado de forma correcta, permitiéndome hacer ping desde la primera máquina virtual hacia la nueva que he creado



Comprimimos el directorio de la aplicación desplegada y nos lo pasamos por sftp a la máquina virtual nueva que hemos creado. Vemos efectivamente que en la nueva máquina creada tenemos nuestra aplicación:

```
usuario@debian:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:1a:4a:97:6f:f7 brd ff:ff:ff:ff:ff:ff
    altname enp0s4
    inet 172.16.202.3/24 brd 172.16.202.255 scope global ens4
       valid_lft forever preferred_lft forever
    inet6 fe80::21a:4aff:fe97:6ff7/64 scope link
       valid_lft forever preferred_lft forever
3: ens7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:1a:4a:97:7d:9a brd ff:ff:ff:ff:ff:ff
    altname enp0s7
    inet 172.16.203.3/24 brd 172.16.203.255 scope global ens7
       valid_lft forever preferred_lft forever
    inet6 fe80::21a:4aff:fe97:7d9a/64 scope link
       valid_lft forever preferred_lft forever
usuario@debian:~$ ls
mean-stack-example  mean-stack-example.tar.gz
```

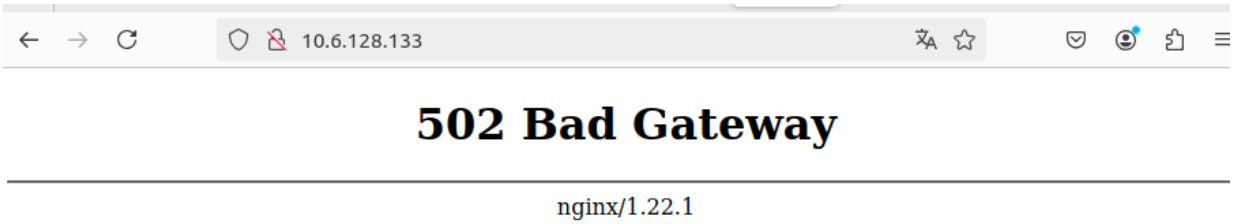
Una vez configurado todo esto he procedido a instalar todas las dependencias y programas necesarios (npm, node, npx,...)

Nuevamente vemos como podemos poner a correr nuestro servidor:

```
usuario@debian:~/mean-stack-example/server$ npx ts-node src/server.ts
(node:12426) ExperimentalWarning: CommonJS module /home/usuario/.nvm/versions/node/v23.1.0/lib/node_modules/npm/node_modules/
debug/src/node.js is loading ES Module /home/usuario/.nvm/versions/node/v23.1.0/lib/node_modules/npm/node_modules/supports-co
lor/index.js using require().
Support for loading ES Module in require() is an experimental feature and might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
Server running at http://10.6.128.133:5200...
```

```
usuario@debian:~/mean-stack-example/client$ ng serve --host 172.16.202.3 --port 4200

Warning: This is a simple server for use in testing or debugging Angular applications
locally. It hasn't been reviewed for security issues.

Binding this server to an open connection can result in compromising your application or
computer. Using a different host than the one passed to the "--host" flag might result in
websocket connection issues. You might need to use "--disable-host-check" if that's the
case.

Browser bundles
Initial chunk files     | Names               |  Raw size
polyfills.js            | polyfills           |  90.20 kB |
styles.css              | styles              |  76.79 kB |
main.js                 | main                |  11.77 kB |

                        | Initial total       | 178.76 kB


Server bundles
Initial chunk files     | Names               |  Raw size
polyfills.server.mjs    | polyfills.server    | 572.91 kB |
main.server.mjs         | main.server         |  12.25 kB |
render-utils.server.mjs | render-utils.server | 472 bytes |

Application bundle generation complete. [8.351 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  ➜  Network: http://172.16.202.3:4200/
  ➜  press h + enter to show help
```

Finalmente vemos que desde la red pública podemos acceder pero el error 502 Bad Gateway en Nginx nos indica que el servidor de backend no está respondiendo correctamente al proxy Nginx. Resolver esto sería similar a rehacer algunos ya hechos anteriormente.



A continuación pasamos al tercer ejercicio. Para este, accedo desde la primera máquina virtual que tengo configurada como proxy al archivo /etc/nginx/sites-available/default y añado el bloque upstream:

```
upstream backend_servers {
    server 172.16.202.2:5000;  # IP de la MV original del backend
    server 172.16.202.3:5000;  # IP de la MV clonada del backend
}
```

A continuación reinicio nginx para que se apliquen los cambios con **sudo systemctl restart nginx.** Una vez hecho esto accedo a **/var/log/nginx/access.log** y vemos los logs de acceso donde podemos verificar si las peticiones están llegando al balanceador de carga.