

# FORMULARIO DE CONTACTO EN SPA CON SERVIDOR DE CORREO EN DOCKER

## 1. Objetivo del Proyecto

Crearemos dos máquinas virtuales basada en Linux.

En la primera máquina instalaremos un servicio de correo con interfaz gráfica mediante un Docker o gestor de contenedores.

En la segunda máquina levantaremos una SPA en otro Docker. SPA que irá de la mano con el framework vue.js, obteniendo como resultado un formulario de contacto.

Como objetivo de proyecto conectaremos dicho servicio de correo con el SPA, de tal forma que, si un usuario o cliente decide contactar con nosotros por correo, nos llegue ese correo de dicho cliente a nuestra bandeja de entrada de nuestro servicio de correo.

## Desarrollo de Máquina virtual Servidor de Correo

### Puntos a configurar en esta máquina:

1. Redireccionamiento de puertos en el router, reglas del firewall de Windows y registros DNS del dominio.
2. Instalación de Docker y Docker-compose.
3. Creación de imagen, contenedor y lanzamiento del mismo.
4. Configuración del servidor de correos (GUI en el navegador web) y primer acceso a servidor de correo poste.io.
5. Instalación de Nginx.
6. Configuración de Nginx.
7. Instalación de certbot y creación de certificados LetsEncrypt.
8. Comprobación de envíos/recibos de correos.

## Herramientas a utilizar

**-Software VirtualBox:** Lo usaremos para virtualizar ambas máquinas y conectarlas entre sí, mediante adaptadores puentes. Usaremos la iso de Ubuntu 22.04.1 Desktop, para desarrollar el servidor. (Esta iso será descargada de la fuente oficial de Linux para evitar futuros problemas).

**-Instalación de Docker:** En este caso usaremos Docker compose, el cual será el gestor de contenedores que contendrá los ficheros más importantes e imprescindibles de las aplicaciones que vamos a usar posteriormente.

**-Instalación de Posteo:** Será el servidor de correos que usemos. Servidor con interfaz gráfica, el cual levantaremos mediante un contenedor docker y el servidor proxy nginx.

**-Instalación de Nginx:** Lo usaremos para que sirva de proxy, este software está diseñado especialmente para protocolos de correo electrónico. Además, es un software de código abierto.

**-Instalación de Certbot:** Usaremos este software, para generar certificados SSL a nuestro dominio, mediante la autoridad certificadora LetsEncrypt. Dominio, en el que se instalará el servidor de correos posteo.

## Desarrollo de Máquina virtual SPA Formulario de Contacto

### Puntos a configurar en esta máquina:

1. Instalación de un editor de códigos (vscode o sublimetext).
2. Instalación de Nginx.
  - 2.1. Configuración del firewall para Nginx.
3. Instalación de Docker y Docker-compose.
4. Instalación de la aplicación Vue-cli.
5. Creación y configuración del Dockerfile.
6. Creación y configuración de docker-compose.yml.
7. Creación de imagen de Docker a partir de los ficheros Dockerfile y docker-compose.yml.
8. Ejecución de nuestra Vueapp en el contenedor docker.
9. Comprobación desde el navegador que accedemos al portal de Bienvenida de Vueapp.
10. Registro en EmailJS, creación de plantilla y servicios de correo electrónicos.
11. Implementación de formulario en código.
12. Comprobación final de envío de correo a través del formulario.

### Herramientas a utilizar

**-Software VirtualBox:** Lo usaremos para virtualizar ambas máquinas y conectarlas entre sí. Usaremos la iso de Ubuntu 22.04.1 Desktop, para desarrollar el servidor. (Esta iso será descargada de la fuente oficial de Linux para evitar futuros problemas).

**-Software Visual Code Studio:** es un editor de código open source el cual usaremos para visualizar ficheros mejor y posteriormente, dar un estilo a nuestro SPA.

**-Instalación de Nginx:** Lo usaremos para que sirva de proxy, este software esta, diseñado especialmente para protocolos de correo electrónico. Además, es un software de código abierto.

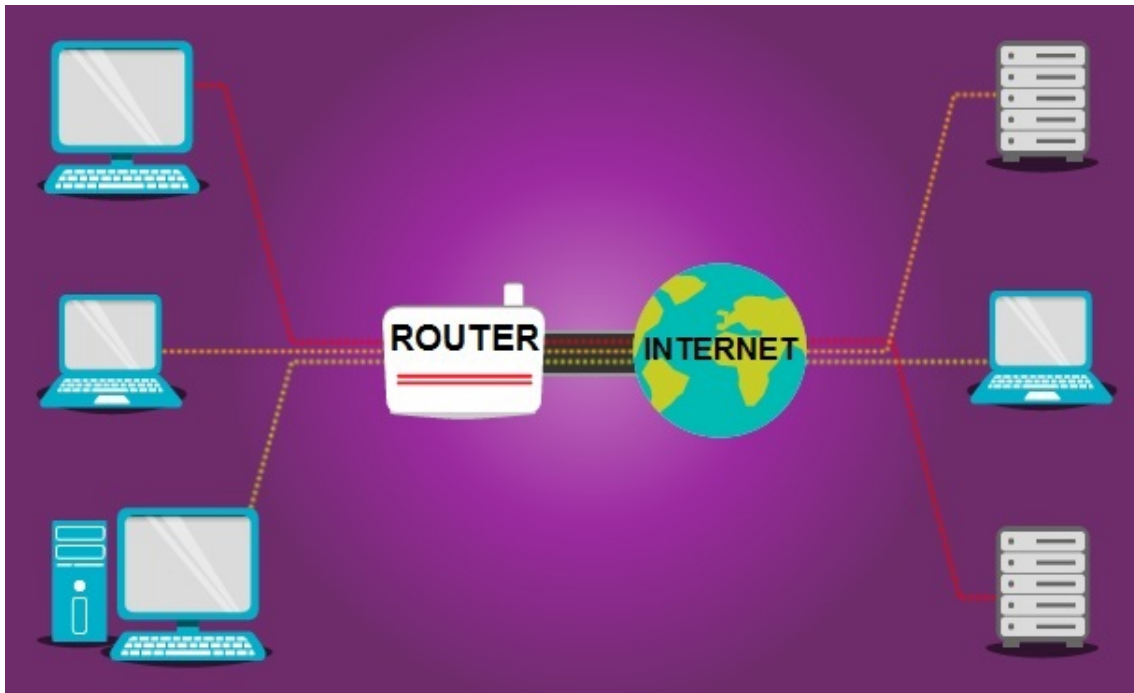
**-Instalación de Docker:** En este caso usaremos Docker compose, el cual será el gestor de contenedores que contendrá los ficheros más importantes e imprescindibles de las aplicaciones que vamos a usar posteriormente.

**-Instalación de VueJS:** Este será el framework, en el que levantaremos nuestro SPA con el formulario. Dicho framework, lo levantaremos mediante un contenedor docker.

**-Darse de alta en EmailJS:** Nos daremos de alta en este servicio, el cual hará de plugin, para enviar correos a nuestro servidor de correos posteio mediante la plantilla de formulario de contacto.

## Máquina virtual Servidor de Correo

Redireccionamiento de puertos en el router, reglas del firewall de Windows y registros DNS del dominio.



Como primer paso, tendremos que redireccionar los puertos necesarios en nuestro router, de forma que apunten a la dirección IP de nuestra máquina. Si no hacemos esto, Spamhaus bloqueará y meterá nuestra IP pública en una lista negra... Prohibiendo de esta forma el tráfico de correos.

Creación de imagen, contenedor y lanzamiento del mismo.



En este paso, crearemos la imagen, el contenedor del servidor de correo posteio y lo lanzaremos.

Ahora ejecutaremos el siguiente comando, para levantar el servidor de correos con sus respectivas configuraciones, que explicaré más adelante:

```
docker run \  
--net=host \  
-e TZ=Europe/Madrid \  
-v /home/posteio:/data \  
--name "mailserver" \  
-h "mail.proyectoasir.com" \  
-e "HTTP_PORT=8080" -e "HTTPS_PORT=4433" -t analogic/poste.io
```

```
root@usuario:/home/posteio# docker run \  
--net=host \  
-e TZ=Europe/Madrid \  
-v /home/posteio:/data \  
--name "mailserver" \  
-h "mail.proyectoasir.com" \  
-e "HTTP_PORT=8080" -e "HTTPS_PORT=4433" -t analogic/poste.io  
Unable to find image 'analogic/poste.io:latest' locally  
latest: Pulling from analogic/poste.io  
b6d6a76ebdbe: Pull complete  
1e62430a5879: Pull complete  
fb8c2a6f6e96: Pull complete  
255b95083370: Pull complete  
3048a31bb4a2: Pull complete  
5a16764df522: Pull complete
```

Explicación de parámetros, del comando anterior:

**--net:** con este parámetro, le indicaremos al comando docker sobre qué dirección IP vamos a trabajar.

**-e TZ:** con este parámetro, le indicaremos al comando docker, la zona horaria que se establecerá para nuestro servidor de correos posteio.

**-v /home/posteio:/data \:** con este parámetro, le indicaremos al comando docker, la ruta del directorio donde se guardarán los datos y configuraciones, para que el servidor de correos posteio funcione correctamente.

**--name:** con este parámetro, le indicaremos al comando docker, el nombre del contenedor que vamos a crear.

**-h "mail.proyectoasir.com" \:** con este parámetro, le indicaremos al comando docker, el nombre del host que va a tener nuestro servidor de correos posteio.

**-e "HTTP\_PORT=8080":** con este parámetro, le indicaremos al comando docker, que será el puerto HTTP personalizado, en el cual LetsEncrypt. Tendrá sus solicitudes.

**-e "HTTPS\_PORT=4433":** con este parámetro, le indicaremos al comando docker, que será el puerto HTTPS personalizado, en el cual accederemos a la interfaz web de nuestro servidor de correos posteio.

**-t analogic/poste.io:** con este parámetro, le indicaremos al comando docker, el nombre de la imagen que vamos a crear.

Aquí vemos como se ha levantado el servidor de correos, con sus direcciones ip y sus puertos de acceso, configurados en el comando anterior

```
[cont-init.d] 33-domains.sh: executing...
* initializing settings for domains
[cont-init.d] 33-domains.sh: exited 0.
[cont-init.d] 34-clamav.sh: executing...
[cont-init.d] 34-clamav.sh: exited 0.
[cont-init.d] 97-randoms: executing...
[cont-init.d] 97-randoms: exited 0.
[cont-init.d] 98-timezone.sh: executing...
* setting timezone to Europe/Madrid
[cont-init.d] 98-timezone.sh: exited 0.
[cont-init.d] 99-custom-plugins: executing...
[cont-init.d] 99-custom-plugins: exited 0.
[cont-init.d] done.
[services.d] starting services
[services.d] done.
2022-12-09 20:35:15 #755(main) <1bbd04>; main; main: rspamd 3.4 is loading configuration, build id: release

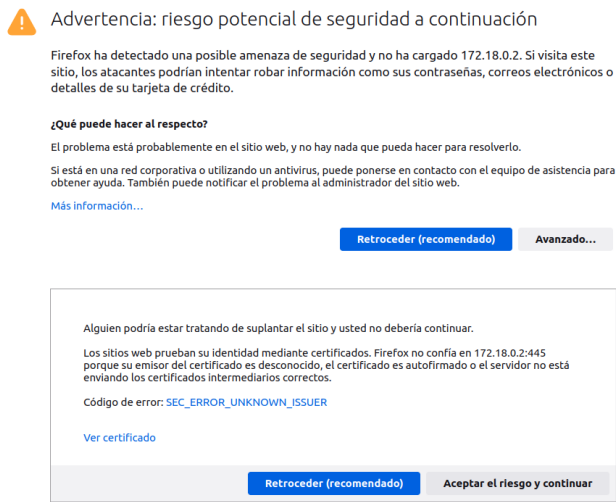
Poste.io administration available at https://80.31.178.18:4433 or http://80.31.178.18:8080

[09-Dec-2022 20:35:16] NOTICE: fpm is running, pid 703
[09-Dec-2022 20:35:16] NOTICE: ready to handle connections
[09-Dec-2022 20:35:16] NOTICE: systemd monitor interval set to 10000ms
```

## Configuración del servidor de correos (GUI en el navegador web) y primer acceso a servidor de correo poste.io.

Ahora accederemos a través del navegador por primera vez, a la interfaz gráfica de nuestro servidor de correos para realizar las primeras configuraciones. Accederemos por la dirección y el puerto pertenecientes al protocolo https. Para ello introduciremos en el navegador, nuestro nombre de subdominio que apunta a la dirección ip pública asignada en el paso anterior:

<https://mail.proyectoasir.com:4433>



Nos saldrá esta ventana, en la que tendremos que darle en Avanzado > Aceptar el riesgo y continuar. Esto sucede porque no tenemos certificados (que los crearemos más adelante, entonces nos lo reconoce como un sitio inseguro).

Una vez hayamos aceptado en la anterior ventana, llegaremos a esta:

## First poste.io configuration

There is no "server.ini" in your data folder, we will try create one. You can update it later in your data folder.

Mailserver hostname \*

mail.proyectoasir.com

Administrator email (email address managed by mailserver) \*

admin@proyectoasir.com

Password \*

Generate

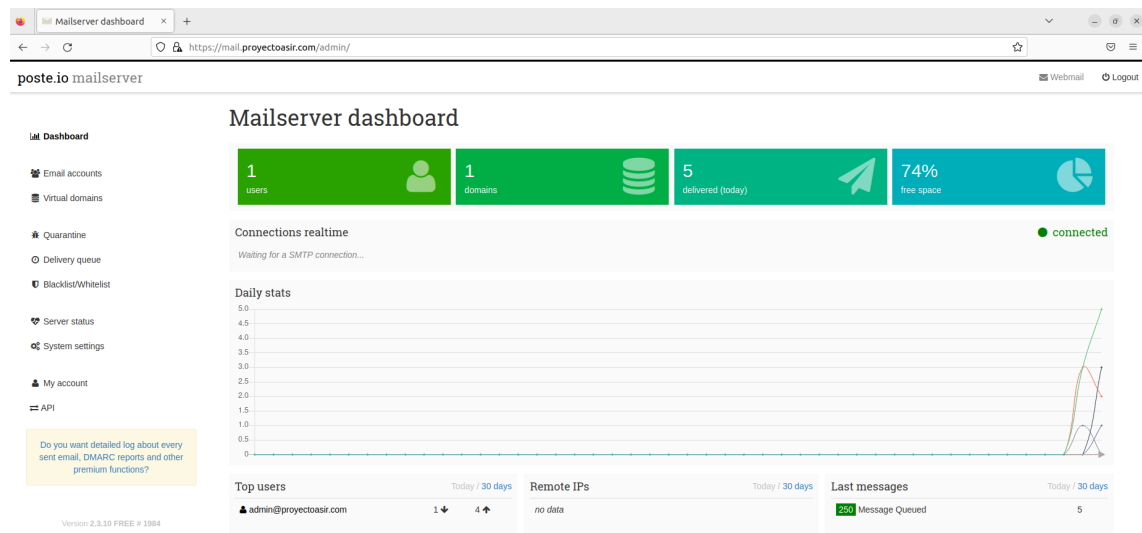
\*\*\*\*\*

Show password

Submit

En este apartado estableceremos nuestro nombre de host, crearemos la dirección de email con la que vamos a trabajar (nuestro nombre de usuario) y crearemos una contraseña.

Y este es el panel que tenemos, cuando accedemos por primera vez:





## Configuración de Nginx

Una vez instalado Nginx pasaremos a configurarlo entre otras cosas, para la posterior creación de certificados LetsEncrypt mediante la herramienta certbot

Para ello, crearemos un archivo nuevo de configuración en la ruta de sitios disponibles de nginx:

**nano /etc/nginx/sites-available/mail**

Introduciremos el siguiente código, que explicaré un poco a continuación:

```
server {  
    listen [::]:80;  
  
    server_name mail.proyectoasir.com;  
  
    proxy_buffering off;  
    proxy_http_version 1.1;  
    proxy_cache_bypass $http_upgrade;  
  
    proxy_set_header Host          $host;  
    proxy_set_header Connection    "upgrade";  
    proxy_set_header Upgrade       $http_upgrade;  
    proxy_set_header X-Real-IP     $remote_addr;  
    proxy_set_header X-Forwarded-Server $host;  
    proxy_set_header X-Forwarded-Proto $scheme;  
    proxy_set_header X-Forwarded-Port $server_port;  
    proxy_set_header X-Forwarded-Host $host:$server_port;  
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
  
    location / {  
        proxy_pass https://0.0.0.0:4433/;  
    }  
}
```

```

GNU nano 6.2 /etc/nginx/sites-available/mail
server {
    listen [::]:80;
    server_name mail.proyectoasir.com;

    proxy_buffering off;
    proxy_http_version 1.1;
    proxy_cache_bypass $http_upgrade;

    proxy_set_header Host                $host;
    proxy_set_header Connection          "upgrade";
    proxy_set_header Upgrade             $http_upgrade;
    proxy_set_header X-Real-IP           $remote_addr;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-Proto  $scheme;
    proxy_set_header X-Forwarded-Port   $server_port;
    proxy_set_header X-Forwarded-Host   $host:$server_port;
    proxy_set_header X-Forwarded-For    $proxy_add_x_forwarded_for;

    location / {
        proxy_pass https://0.0.0.0:4433/;
    }
}

```

Explicación de parámetros de más importancia, del código anterior:

**listen [::]:80:** con este parámetro le indicaremos al archivo de configuración de nginx, que el puerto por el que va escuchar nginx será el 80.

**server\_name mail.proyectoasir.com:** con este parámetro le indicaremos al archivo de configuración de nginx, sobre que nombre de servidor se realizará la configuración.

**proxy\_pass <https://0.0.0.0:4433/>:** con este parámetro le indicaremos al archivo de configuración de nginx, cuál será la dirección IP y el puerto sobre los que se comunicará nginx con el servidor de correos.

Una vez guardada la configuración del archivo, crearemos un enlace simbólico del fichero creado anteriormente, en el directorio sites-enabled. Esto lo hacemos porque certbot, actualizará en el directorio sites-enabled el fichero creado anteriormente, cuando generemos los certificados añadiendo la configuración de los certificados:

**ln -s /etc/nginx/sites-available/mail /etc/nginx/sites-enabled/**

```

root@usuario:/home/usuario# ln -s /etc/nginx/sites-available/mail /etc/nginx/sites-enabled/
root@usuario:/home/usuario#

```

Ahora reiniciamos nginx, para que los ficheros configurados tengan efecto con el siguiente comando:

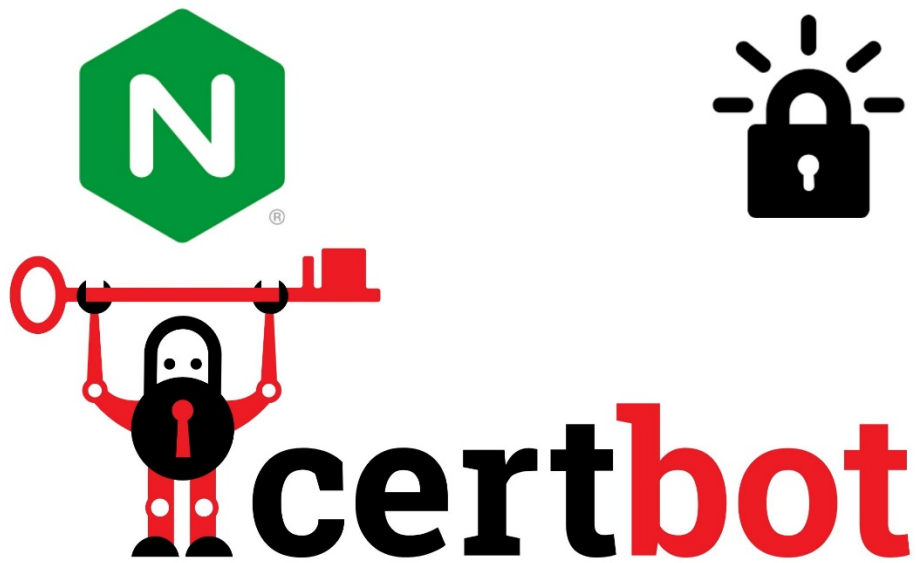
**nginx -s reload**

```

root@usuario:/home/usuario# nginx -s reload
root@usuario:/home/usuario#

```

## Instalación de certbot y creación de certificados LetsEncrypt



En este punto instalaremos certbot y luego generaremos los certificados TLS LetsEncrypt haciendo uso de él.

En mi caso accedí a la web oficial de certbot, donde tuve que seleccionar el proxy que usaba y mi distribución de Linux

Generaremos los certificados con el siguiente comando:

### **certbot --nginx**

Nos pedirá que proporcionemos una dirección de correo, para notificarnos de lo que pueda pasar.

```
root@usuario:/home/usuario# certbot --nginx
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Enter email address (used for urgent renewal and security notices)
(Enter 'c' to cancel): admin@proyectoasir.com

- - - - -
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.3-September-21-2022.pdf. You must
agree in order to register with the ACME server. Do you agree?
- - - - -
(Y)es/(N)o: ☐
```

Aceptaremos los términos de LetsEncrypt.

Ahora, nos preguntará para que dominio queremos instalar los certificados. Si tenemos más dominios disponibles, tendremos que seleccionar entre las distintas opciones que nos darán a escoger.

```
Which names would you like to activate HTTPS for?
We recommend selecting either all domains, or all domains in a VirtualHost/server block.
- - - - -
1: mail.proyectoasir.com
- - - - -
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel):
```

En mi caso, como yo solo tengo un dominio disponible, seleccionaremos la opción 1

```
Which names would you like to activate HTTPS for?
We recommend selecting either all domains, or all domains in a VirtualHost/server block.
- - - - -
1: mail.proyectoasir.com
- - - - -
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel): 1
Requesting a certificate for mail.proyectoasir.com

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/mail.proyectoasir.com-0002/fullchain.pem
Key is saved at: /etc/letsencrypt/live/mail.proyectoasir.com-0002/privkey.pem
This certificate expires on 2023-03-08.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for mail.proyectoasir.com to /etc/nginx/sites-enabled/mail
Congratulations! You have successfully enabled HTTPS on https://mail.proyectoasir.com

- - - - -
If you like Certbot, please consider supporting our work by:
* Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
* Donating to EFF: https://eff.org/donate-le
- - - - -
root@usuario:/home/usuario#
```

En poco segundos, nos habrá generado los certificados con éxito y nos lo habrá guardado en la ruta **/etc/letsencrypt/live/nombrededominio/**

Por otra parte, se habrá actualizado el archivo de configuración de nginx, situado en la ruta **/etc/nginx/sites-enabled/** como expliqué en pasos anteriores.

Archivo de configuración nginx actualizado automáticamente (después de generar los certificados):

```
GNU nano 6.2 /etc/nginx/sites-enabled
server {
    server_name mail.proyectoasir.com;

    proxy_buffering off;
    proxy_http_version 1.1;
    proxy_cache_bypass $http_upgrade;

    proxy_set_header Host $host;
    proxy_set_header Connection "upgrade";
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-Port $server_port;
    proxy_set_header X-Forwarded-Host $host:$server_port;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    location / {
        proxy_pass https://0.0.0.0:443/;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/mail.proyectoasir.com-0002/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/mail.proyectoasir.com-0002/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}
server {
    if ($host = mail.proyectoasir.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

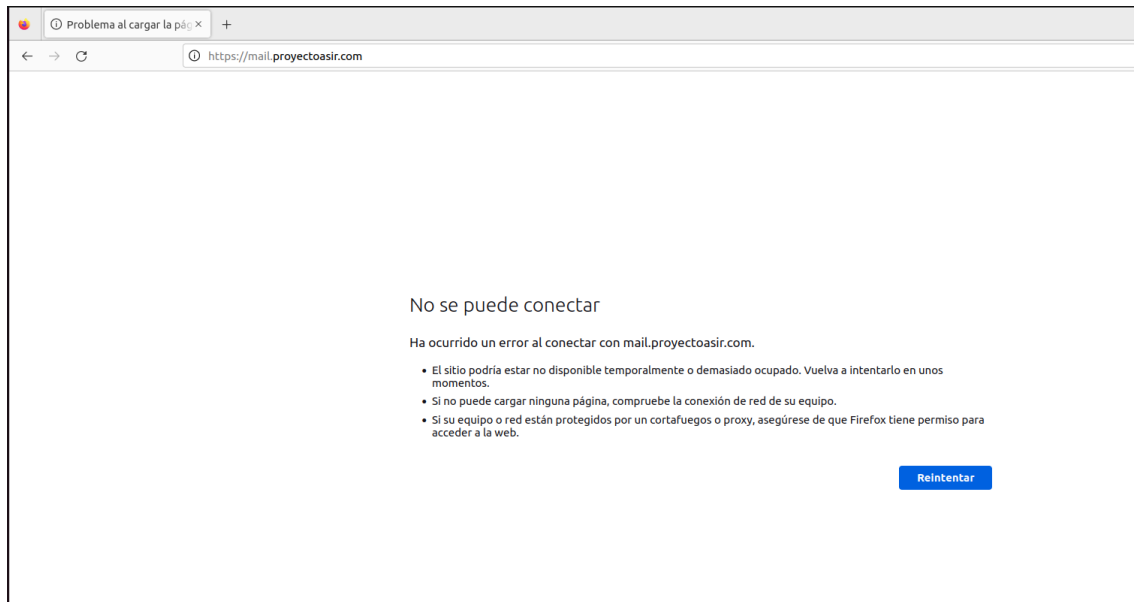
    listen [::]:80;
    server_name mail.proyectoasir.com;
    return 404; # managed by Certbot
}
```

Ubicación de los certificados generados anteriormente:

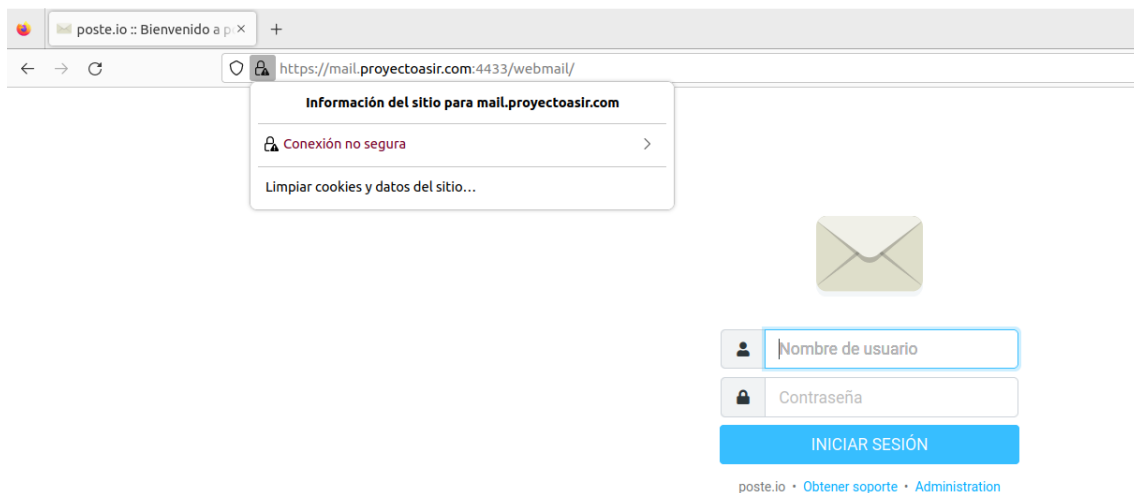
**ls /etc/letsencrypt/mail.proyectoasir.com-0002**

```
root@usuario:/home/usuario# ls /etc/letsencrypt/live/mail.proyectoasir.com-0002/
cert.pem chain.pem fullchain.pem privkey.pem README
root@usuario:/home/usuario#
```

**NOTA DE ERROR:** Cuando accedo a la url, sobre la cual certbot nos indica que se han instalado los certificados (<https://mail.proyectoasir.com>), obtengo esto:



Y cuando accedo a la misma dirección, especificándole el puerto sobre el que hemos configurado el servidor de correos, puedo acceder sin problemas, pero no se me ha instalado el certificado:



Sospecho que esto se pueda deber a que no se haya redireccionado de http a https en nginx... Aunque he intentado redireccionar de http a https y no he conseguido instalar el certificado.

## Máquina virtual SPA Formulario de Contacto

### Creación de la aplicación vue.

A continuación, configuraremos la aplicación vue para posteriormente, instalarla.

#### DOCKER VUEJS

#### RUN VUEJS WEB APPLICATION IN DOCKER

#### CONTAINERIZE VUEJS

[www.middlewareinventory.com](http://www.middlewareinventory.com)

[www.devopsjunction.com](http://www.devopsjunction.com)



Creamos la aplicación vue con el nombre que le queramos dar (en mi caso vueappnginx):

**vue create vueappnginx**

```
root@spaportcfolio:/home/spaportcfolio# vue create vueappnginx
```

Ahora, seleccionamos la versión de vue en la que queremos trabajar, (yo he seleccionado vue 2 para evitar futuros problemas de incompatibilidades con plugins):

```
Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Esperamos a que se cree nuestro proyecto y se instalen los plugins necesarios:

```
Vue CLI v5.0.8
🌟 Creating project in /home/spaportcfolio/vueappnginx.
📦 Initializing git repository...
⚙️ Installing CLI plugins. This might take a while...

) : idealTree:vueappnginx: sill idealTree buildDeps
```

Ahora accedemos al directorio que se nos ha creado automáticamente, al crear nuestro proyecto con el nombre que le hayamos asignado:

```
root@usuario:/home/usuario# cd vueappnginx/
root@usuario:/home/usuario/vueappnginx#
```

Ejecutamos el siguiente comando, el cual nos permitirá compilar los archivos de configuración de nuestro vueapp, creando un directorio llamado dist con los archivos de configuración:

### npm run build

```
root@usuario:/home/usuario/vueappnginx# npm run build
> vueappnginx@0.1.0 build
> vue-cli-service build

All browser targets in the browserslist configuration have supported ES module.
Therefore we don't build two separate bundles for differential loading.

Building for production...
DONE Compiled successfully in 16753ms

File                                Size                                Gzipped
dist/js/chunk-vendors.679c0db5.js    86.30 KiB                          30.60 KiB
dist/js/app.e4d45d38.js             13.08 KiB                          8.40 KiB
dist/css/app.2cf79ad6.css            0.33 KiB                           0.23 KiB

Images and other types of assets omitted.
Build at: 2022-11-19T21:26:20.086Z - Hash: e07636ed567a1364 - Time: 16753ms
DONE Build complete. The dist directory is ready to be deployed.
INFO Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html
root@usuario:/home/usuario/vueappnginx#
```

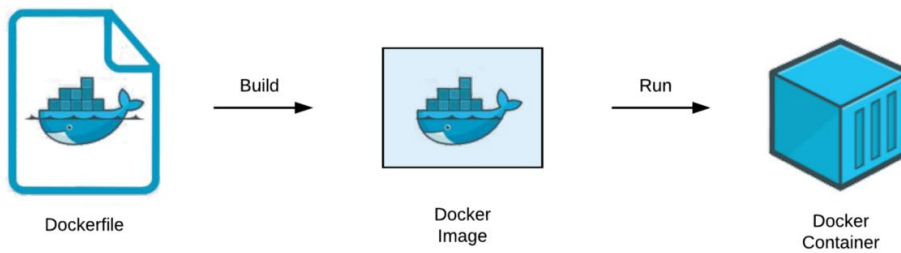
En la siguiente imagen, vemos todos los archivos y directorios que se nos ha creado automáticamente (incluido el directorio dist, que nos ha creado el comando del paso anterior, dentro del directorio que lleva el nombre de proyecto que le hemos dado:

### ls -la

```
root@usuario:/home/usuario/vueappnginx# ls -la
total 836
drwxr-xr-x  7 root    root    4096 nov 19 22:25 .
drwxr-xr-x 19 usuario usuario 4096 nov 19 22:20 ..
-rw-r--r--  1 root    root     73 nov 19 19:38 babel.config.js
drwxr-xr-x  4 root    root    4096 nov 19 22:26 dist
-rw-r--r--  1 root    root    212 nov 19 19:48 docker-compose.yml
-rw-r--r--  1 root    root    311 nov 19 21:46 Dockerfile
drwxr-xr-x  7 root    root    4096 nov 19 19:38 .git
-rw-r--r--  1 root    root    231 nov 19 19:38 .gitignore
-rw-r--r--  1 root    root    279 nov 19 19:38 jsconfig.json
drwxr-xr-x 562 root    root   20480 nov 19 21:37 node_modules
-rw-r--r--  1 root    root    905 nov 19 19:38 package.json
-rw-r--r--  1 root    root  775198 nov 19 19:38 package-lock.json
drwxr-xr-x  2 root    root    4096 nov 19 19:38 public
-rw-r--r--  1 root    root    323 nov 19 19:38 README.md
drwxr-xr-x  4 root    root    4096 nov 19 19:38 src
-rw-r--r--  1 root    root    118 nov 19 19:38 vue.config.js
root@usuario:/home/usuario/vueappnginx#
```



## Creación y configuración del Dockerfile.



Ahora, crearemos el fichero de configuración Dockerfile con las configuraciones necesarias organizadas por etapas, entre ellas la de nginx como proxy para nuestra vuejsapp:

**nano Dockerfile**

**# build stage**

**FROM node:lts-alpine as build-stage**

**WORKDIR /vueappnginx**

**COPY package\*.json ./**

**RUN npm install**

**COPY . .**

**RUN npm run build**

**# production stage**

**FROM nginx:stable-alpine as production-stage**

**COPY --from=build-stage /vueappnginx/dist /usr/share/nginx/html**

**EXPOSE 80**

**CMD ["nginx", "-g", "daemon off;"]**

```
GNU nano 6.2
# build stage
FROM node:lts-alpine as build-stage
WORKDIR /vueappnginx
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# production stage
FROM nginx:stable-alpine as production-stage
COPY --from=build-stage /vueappnginx/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Guardamos los cambios del fichero.

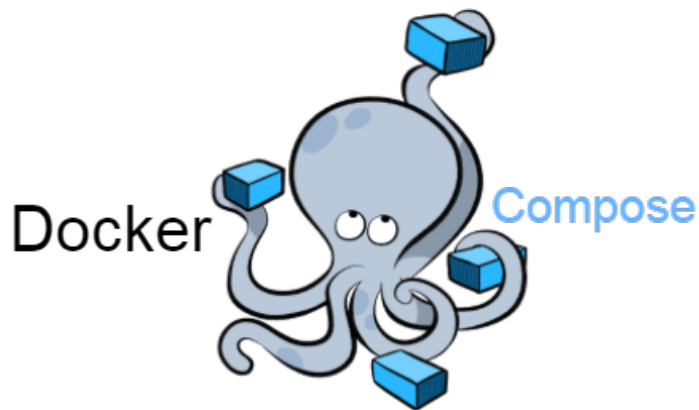
### #Etapa de construcción (build stage)

- FROM node:lts-alpine as build-stage:** indica que imagen docker (en este caso node), vamos a descargar y con el as la bautiza con otro nombre.
- WORKDIR /vueappnginx:** indica que la carpeta /vueappnginx será nuestro directorio de trabajo actual.
- COPY package\*.json ./:** copia los paquetes.json o librerías (si existen), de nuestro directorio actual o los directorios a partir de nuestro directorio actual.
- RUN npm install:** El gestor npm instala las dependencias del proyecto.
- COPY . .:** copia ficheros o directorios del proyecto a nuestro directorio actual.
- RUN npm run build:** compila los directorios y ficheros de configuraciones de nuestra vueapp.

### #Etapa de producción (production stage)

- FROM nginx:stable-alpine as production-stage:** indica que imagen docker (en este caso nginx), vamos a descargar y con el as la bautiza con otro nombre.
- COPY --from=build-stage /vueappnginx/dist /usr/share/nginx/html:** copia partiendo de la etapa de producción, los cambios realizados en la carpeta dist a la carpeta html del proxy nginx.
- EXPOSE 80:** indica el puerto para el que estará configurado el proxy nginx.
- CMD ["nginx", "-g", "daemon off;"]:** indica que la configuración para el proxy nginx, se ejecutará en primer plano.

## Creación y configuración de docker-compose.yml.



En este paso, crearemos el fichero docker-compose.yml necesarias para posteriormente, crear la imagen docker de nuestra vuejsapp con docker compose. Esta imagen se creará a su vez, llamando o haciendo referencia al Dockerfile creado anteriormente:

```
nano docker-compose.yml
```

```
version: '3.7'
```

```
services:
```

```
vueappnginx:
```

```
  container_name: vueappnginx
```

```
  build:
```

```
    context: .
```

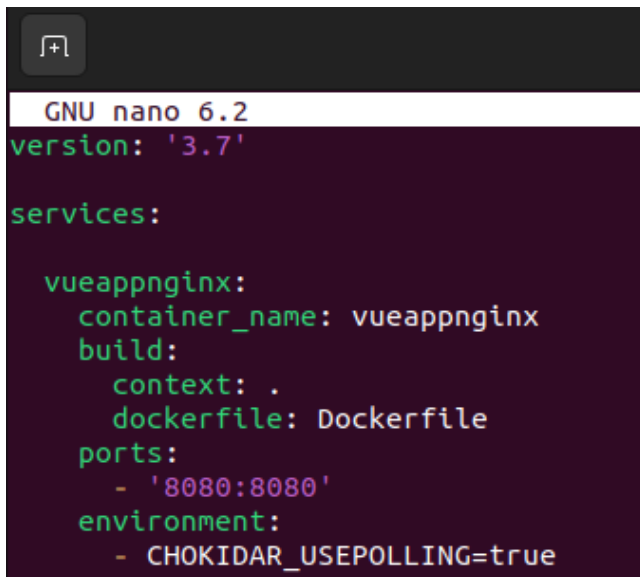
```
    dockerfile: Dockerfile
```

```
  ports:
```

```
    - '8080:8080'
```

```
  environment:
```

```
    - CHOKIDAR_USEPOLLING=true
```



```
GNU nano 6.2
version: '3.7'

services:

  vueappnginx:
    container_name: vueappnginx
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - '8080:8080'
    environment:
      - CHOKIDAR_USEPOLLING=true
```

Guardamos los cambios del fichero.

**-container\_name: vueappnginx:** indica el nombre del contenedor que se va a crear.

**-context: .:** indica el directorio donde se ubica el dockerfile, en este caso es el directorio actual (el directorio vueappnginx de nuestra app).

**-dockerfile:** indica el nombre de nuestro archivo dockerfile.

**-ports:** indica el puerto con el que trabajará nuestro contenedor que crearemos, el 8080. Por lo tanto, para acceder a la vueapp por medio de nuestro navegador, tendremos que usar el puerto 8081 (que esta, abierto) ya que el 8080 estará ocupado por nuestro contenedor.

**-CHOKIDAR\_USEPOLLING=true:** con este parámetro, le indicaremos que nos permita refrescar automáticamente, los cambios que guardemos en el archivo de configuración.

## Creación de imagen de Docker a partir de los ficheros Dockerfile y docker-compose.yml.

Para la creación de la imagen docker a partir de los ficheros Dockerfile y docker-compose.yml, ejecutaremos el siguiente comando:

**docker compose -f docker-compose.yml up -d --build**

```
root@usuario:/home/usuario/vueappnginx# docker compose -f docker-compose.yml up -d --build
[+] Building 280.7s (7/13)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 334B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:stable-alpine
=> [internal] load metadata for docker.io/library/node:lts-alpine
=> [build-stage 1/6] FROM docker.io/library/node:lts-alpine@sha256:9eff44230b2fdcca57a73b8f908c8029e72d24dd05cac5339c79d3dedf6b208b
=> => resolve docker.io/library/node:lts-alpine@sha256:9eff44230b2fdcca57a73b8f908c8029e72d24dd05cac5339c79d3dedf6b208b
=> => sha256:9eff44230b2fdcca57a73b8f908c8029e72d24dd05cac5339c79d3dedf6b208b 1.43kB / 1.43kB
=> => sha256:67373bd5d90eaa00cb5f0fa58d7a5a4eeebf50b8e85c58c1d1cc22df5134db43 1.16kB / 1.16kB
=> => sha256:0fa08f92ee4b03a1a455b2153a50eac8b8e134ab352dbe751229b970b0e05e4f 6.44kB / 6.44kB
=> => sha256:ca7dd9ec2225f2385955c43b2379305acd51543c28cf1d4e94522b3d94cce3ce 2.81MB / 2.81MB
=> => sha256:55371e6747ebe4327c7a293b77a0b46632f2249d0c89c79f830c4f565c538000 46.36MB / 46.36MB
=> => sha256:694d6b1b2d1b452b735d925ef7912fe264d4f03c7ef77effed89d76a086dafd1 2.35MB / 2.35MB
=> => extracting sha256:ca7dd9ec2225f2385955c43b2379305acd51543c28cf1d4e94522b3d94cce3ce
=> => sha256:71f41f5ff77d8eca2e4800eb9001495106991811e45a005c4e59d967d0f40334 452B / 452B
```

**-f:** indica que la imagen se creará a partir de un fichero tipo file.

**-up:** indica que se va a crear e iniciar un contenedor.

**-d:** indica que el servicio se ejecutará en segundo plano.

**--build:** indica que se va a construir un servicio.


Registro en EmailJS, creación de plantilla y servicios de correo electrónicos.



# EmailJS

Cuando pinchemos en agregar nuevo servicio, nos saldrá la siguiente ventana:

Seleccionar servicio

 **Gmail**  
Servicio Personalizado | 500correos electrónicos por día ?

Nombre \*

ID de servicio \*

Conexión de Gmail  

Conectado como sanchezalonsoemiliano@gmail.com 

Desconectar

i

Permita el permiso "Enviar correo electrónico en su nombre" durante la conexión.  
Se admiten las cuentas de Gmail y Google Apps.

☒ Enviar correo electrónico de prueba para verificar la configuración

Cancelar

Servicio de actualización

Aquí es donde vincularemos nuestra cuenta de Gmail. Luego enviaremos un correo de prueba a nuestro Gmail, para verificar que EmailJS se ha vinculado con éxito con nuestra cuenta Gmail.

Cuando pinchemos en crear nueva plantilla, nos saldrá la siguiente ventana:

Aquí tendremos que rellenar la información, que nos llegara a nuestro servidor de correos posteio cuando enviemos un mail. También tendremos que rellenar, las direcciones de correo de origen y destino entre otras cosas.

Antes de implementar dicha plantilla en el código, haremos una prueba de envío de mail una vez configurada y guardada la plantilla de la imagen anterior. Para ello, pulsamos en la pestaña que dice: pruébalo (en la imagen anterior) y llegaremos a la ventana de la siguiente imagen:

Aquí. Seleccionamos el servicio de correos Gmail, previamente configurado. Rellenamos los campos con nuestra información y vemos como al mismo tiempo, se nos muestra el código en JavaScript como referencia, con los datos que vamos rellenando en los campos.

Plantilla de prueba "Formulario vue" X

NOTA: Probando para una plantilla guardada . Guarde los últimos cambios para probarlos.

Usar servicio

Gmail

Parámetros de plantilla ?

Email

admin@proyectoasir.com

nombre

Emiliano

mensaje

Prueba plantilla formulario vue

Código JavaScript

```
emailjs .send ( " servicio_2m7qrxr " , " plantilla  
correo electrónico : " admin@proyectoasir.com "  
nombre : " E " ,  
mensaje : " P " ,  
}) ;
```

Enviar correo electrónico de prueba

Resultado

200 bien

Cancelar



## Implementación de formulario en código

Dicho esto, una vez creada la app del formulario como en pasos anteriores, nos ubicamos en el directorio y le daremos todos los permisos con `chmod`, al directorio donde se encuentra el formulario:

**`chmod -R 777 formulariodecontactoemailjs/`**

```
root@usuario:/home/usuario# chmod -R 777 formulariodecontactoemailjs/
root@usuario:/home/usuario#
```

**NOTA:** Sino le damos los permisos al directorio, nos dará error de permisos cuando queramos guardar los cambios en los archivos de código desde el editor `vscode`.

Después de dar los permisos y ubicarnos en el directorio, ejecutamos el siguiente comando para instalar paquetes y dependencias de la app del formulario. Este comando se aconseja ejecutar, cada vez que hagamos o guardemos un cambio para evitar futuros errores:

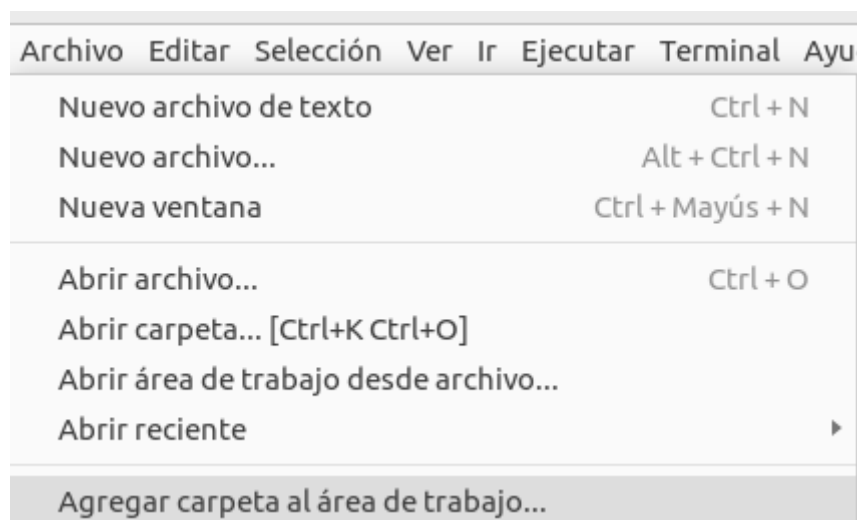
**`npm install`**

```
root@usuario:/home/usuario# cd formulariodecontactoemailjs/
root@usuario:/home/usuario/formulariodecontactoemailjs# npm install
```

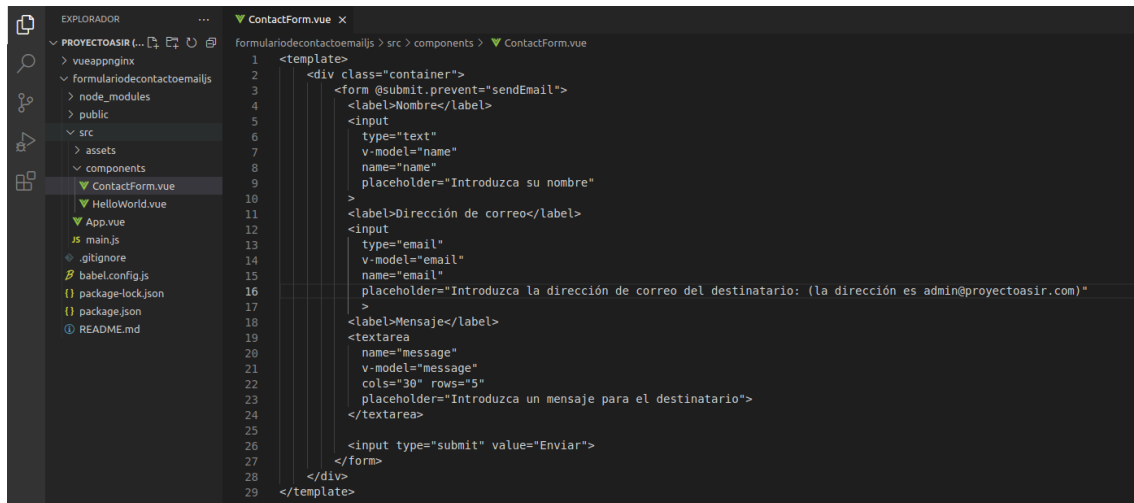
Esperamos: mientras se instalan los paquetes:

```
root@usuario:/home/usuario/formulariodecontactoemailjs# npm install
( [REDACTED] ) :: idealTree: timing idealTree Completed in 3287ms
```

Mientras, aprovecharemos para agregar la carpeta del formulario, al editor `vscode`:



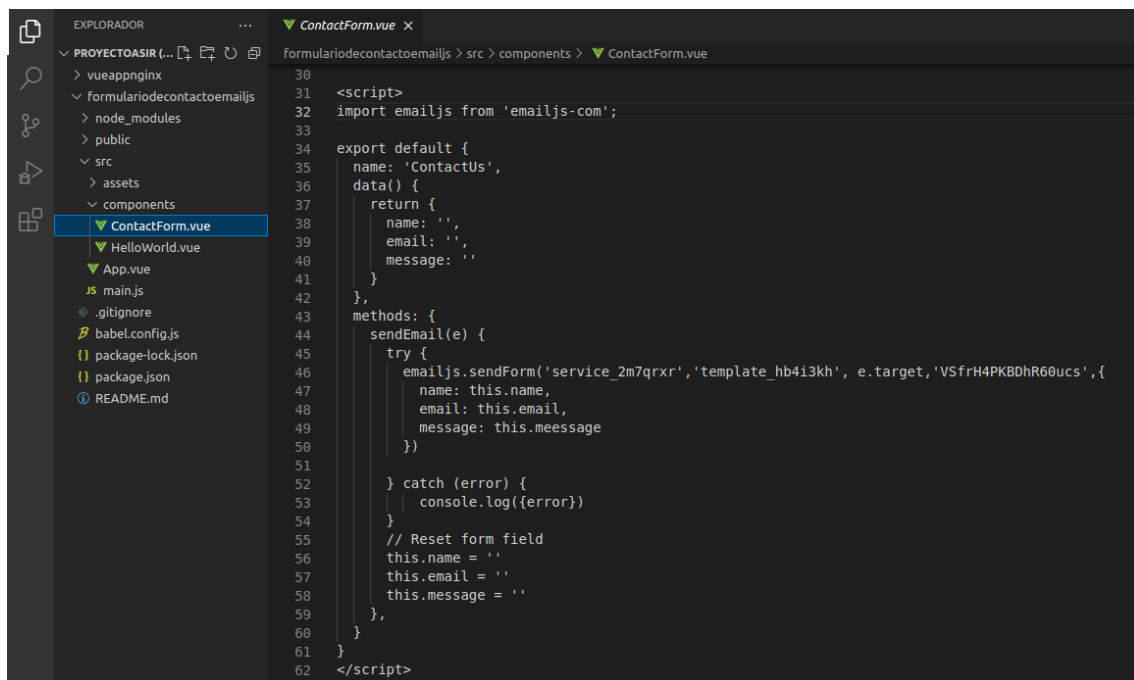
Una vez agregado el directorio al vscode, pasamos a implementar el código del formulario:



```
1 <template>
2   <div class="container">
3     <form @submit.prevent="sendEmail">
4       <label>Nombre</label>
5       <input
6         type="text"
7         v-model="name"
8         name="name"
9         placeholder="Introduzca su nombre"
10      >
11     <label>Dirección de correo</label>
12     <input
13       type="email"
14       v-model="email"
15       name="email"
16       placeholder="Introduzca la dirección de correo del destinatario: (la dirección es admin@proyectoasir.com)"
17     >
18     <label>Mensaje</label>
19     <textarea
20       name="message"
21       v-model="message"
22       cols="30" rows="5"
23       placeholder="Introduzca un mensaje para el destinatario">
24     </textarea>
25     <input type="submit" value="Enviar">
26   </form>
27 </div>
28 </template>
```

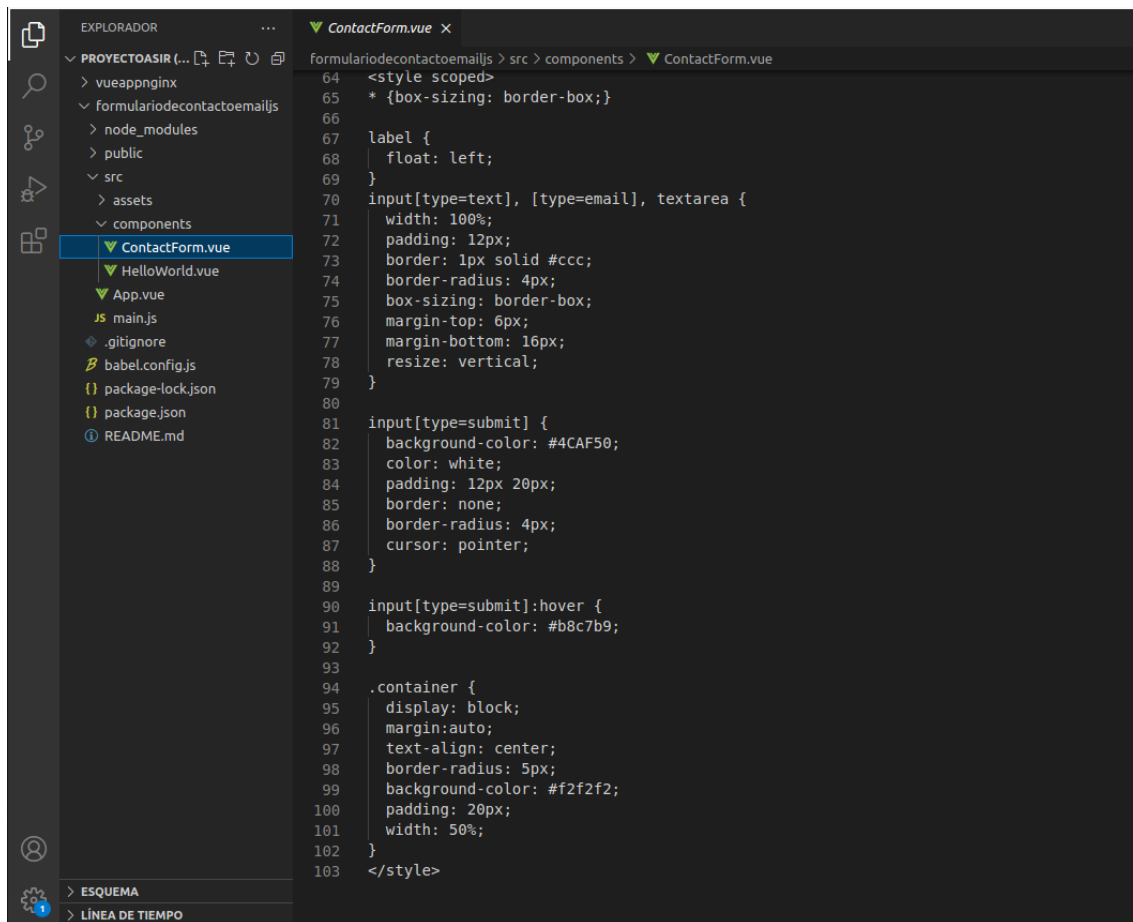
En el código del archivo ContactForm.vue, vemos una plantilla de un formulario genérico con tres campos. Campos que, he configurado para adaptarlo con la plantilla configurada en la web de emailjs.

En esta segunda parte del código perteneciente al formulario, tenemos un script en el cual se importa la librería de emailjs por un lado. Por otro tenemos funciones, entre las cuales se implementan los datos del propio formulario, devolviéndolos por pantalla en forma de return. Luego tenemos otra función, la cual nos permite enviar correos mediante la vinculación de la plantilla configurada en la web dentro de la función. Cuando digo vinculación, me refiero a que la función emailjs.sendform() llame a la plantilla del formulario, especificando el id del servicio de correos y el id de la plantilla dentro de la misma función. Por último, añadimos una función de log de errores por si nos sale algo mal y resetearemos los campos del formulario, cuando enviemos los datos a través del mismo.



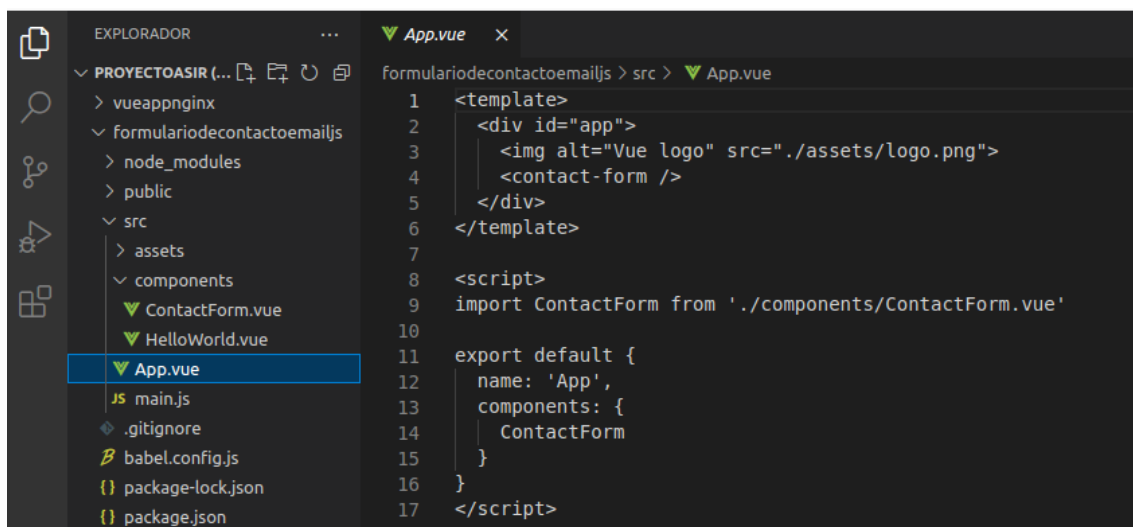
```
30
31 <script>
32 import emailjs from 'emailjs-com';
33
34 export default {
35   name: 'ContactUs',
36   data() {
37     return {
38       name: '',
39       email: '',
40       message: ''
41     }
42   },
43   methods: {
44     sendEmail(e) {
45       try {
46         emailjs.sendForm('service_2m7qrxr', 'template_hb4i3kh', e.target, 'VSfrH4PKBdHR60ucs', {
47           name: this.name,
48           email: this.email,
49           message: this.meessage
50         })
51       } catch (error) {
52         console.log({error})
53       }
54       // Reset form field
55       this.name = ''
56       this.email = ''
57       this.message = ''
58     },
59   },
60 }
61
62 </script>
```

En esta tercera y última parte del formulario, encontramos entradas de estilo para el mismo. Entradas de estilo para cada campo donde encontramos márgenes, tipos de bordes, colores, anchura, padding etc.



```
64 <style scoped>
65 * {box-sizing: border-box;}
66
67 label {
68   float: left;
69 }
70 input[type=text], [type=email], textarea {
71   width: 100%;
72   padding: 12px;
73   border: 1px solid #ccc;
74   border-radius: 4px;
75   box-sizing: border-box;
76   margin-top: 6px;
77   margin-bottom: 16px;
78   resize: vertical;
79 }
80
81 input[type=submit] {
82   background-color: #4CAF50;
83   color: white;
84   padding: 12px 20px;
85   border: none;
86   border-radius: 4px;
87   cursor: pointer;
88 }
89
90 input[type=submit]:hover {
91   background-color: #b8c7b9;
92 }
93
94 .container {
95   display: block;
96   margin: auto;
97   text-align: center;
98   border-radius: 5px;
99   background-color: #f2f2f2;
100   padding: 20px;
101   width: 50%;
102 }
103 </style>
```

Este será nuestro segundo y último archivo de código para el formulario. Aquí encontraremos lo que aparecerá en el portal de bienvenida de nuestra app vue. De hecho, lo que vamos a hacer a través del código script y la función, es llamar al archivo del formulario definido también en la etiqueta <template>



```
1 <template>
2   <div id="app">
3     
4     <contact-form />
5   </div>
6 </template>
7
8 <script>
9   import ContactForm from './components/ContactForm.vue'
10
11   export default {
12     name: 'App',
13     components: {
14       ContactForm
15     }
16   }
17 </script>
```