



## Emitto

API de envíos de correos electrónicos masivos. Permite enviar notificaciones, boletines u otras comunicaciones de forma sencilla a múltiples destinatarios.

## Acerca del proyecto

The screenshot shows the API documentation for the 'Enviar correo electrónico' endpoint. The left sidebar contains a search bar and a navigation menu with sections: Introduction, Auth (Iniciar sesión, Estado de sesión, /auth/logout), Mail (Enviar correo electrónico), SecretKey, and Users. The main content area is titled 'Enviar correo electrónico' and includes a 'Headers' section with 'x-key-emitto' (string, required) and a description 'Clave secreta necesaria para autorización'. The 'Body' section is labeled 'application/json' and describes the 'Payload para enviar un correo electrónico'. It lists fields: 'from' (string, required, Example), 'subjectEmail' (string, required, Example), 'sendTo' (array of strings, required, Example), 'message' (string, required, Example), and 'attachments' (array of objects, with a 'Show Child Attributes' button). The 'Responses' section lists status codes: 201 (Correo enviado correctamente), 400 (Datos inválidos), 403 (No se encontró o es inválido el header personalizado "x-key-emitto"), and 500 (Error interno del servidor). On the right, a 'Test Request' panel shows a cURL command and the resulting JSON response: { "success": true, "statusCode": 201, "message": "Correo enviado correctamente" }. At the bottom left, there is a link to 'Open API Client' and a note 'Powered by Scalar'.

Este proyecto es una API RESTful para el envío de correos electrónicos masivos, diseñada para facilitar la comunicación eficiente y automatizada con múltiples usuarios o clientes. Es ideal para organizaciones, instituciones educativas, sistemas administrativos y aplicaciones web que requieren funcionalidades de notificación o difusión por correo electrónico.

## Funcionalidades Principales

Este servicio API está diseñado para gestionar el envío automatizado de correos masivos y validaciones seguras mediante claves secretas. Las funcionalidades disponibles actualmente son:

### Autenticación

- Login de usuarios (admin) con JWT.

- Soporte para estrategias de autenticación con `passport-local` y `passport-jwt`.
- Generación y verificación de tokens seguros.

## Envío de Correos

- Envío de correos individuales o masivos utilizando `nodemailer` y `@nestjs-modules/mailer`.
- Integración con colas `Bull` para envío asíncrono y controlado.
- Visualización del estado de las tareas de envío mediante Bull Board - QueueDash.

## Gestión de Secret Keys

- CRUD completo de claves (crear, obtener, actualizar y eliminar).
- Uso de claves secretas en cabeceras HTTP ( `x-key-emitto` ) para validar el origen autorizado de las solicitudes de envío de correos.

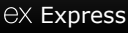




## Otras funcionalidades

- Documentación interactiva-moderna generada con openapi ( `/openapi` ).
- Monitor de colas ( `/queuedash` ) para visualizar trabajos activos, pendientes y fallidos.
- Manejo de errores estructurado y validación de datos con `class-validator`.



# Tecnologías

Este proyecto utiliza un stack moderno basado en **Node.js** y **NestJS**, junto con herramientas para colas, correos, autenticación y documentación.





## Backend Principal

-  **Express** Adaptador HTTP para NestJS.
-  **NestJS** Framework progresivo para construir APIs escalables en Node.js.
-  **Sequelize** ORM para PostgreSQL.
-  **Sequelize-TypeScript** Integración de Sequelize con TypeScript.
-  **PostgreSQL** Sistema de base de datos relacional.






## Envío de Correos

-  **NestJS-Mailer**  
Módulo de NestJS para enviar correos electrónicos.
-  **Nodemailer** Biblioteca para el envío de correos a través de SMTP.



## Tareas y Procesamiento Asíncrono

-  **Bull** Sistema de colas basado en Redis.
-  **Bull Board** Panel visual para monitorear colas Bull.
-  **QueueDash** Panel visual para monitorear colas Bull.
-  **Redis** Almacenamiento clave-valor para procesamiento asíncrono.




## Autenticación y Seguridad

-  **Passport** Middleware de autenticación.
-  **NestJS-Passport** Integración de Passport con NestJS.
-  **NestJS-JWT** JWT con soporte para NestJS.
-  **bcrypt**  **bcryptjs** Encriptación de contraseñas.








## Validación y Transformación

-  **class-validator** Validación de DTOs y clases.
-  **class-transformer** Transformación de objetos y clases.

## Documentación

-  **NestJS-Swagger** Generador Swagger para NestJS.
-  **OpenAPI-Client** Cliente generado para APIs OpenAPI.
-  **swagger-ui-express** UI para documentación de APIs Swagger.

## Utilidades y Desarrollo

-  **dotenv** Manejo de variables de entorno.
-  **morgan** Logger HTTP.
-  **uuid** Generador de identificadores únicos.
-  **moment** Manipulación de fechas y horas.
-  **cookie-parser** Middleware para parsear cookies.
-  **reflect-metadata** Decoradores y metadatos requeridos por TypeScript/NestJS.
-  **rxjs** Librería reactiva para manejo de flujos asíncronos.

# API de Envío de Correos Electrónicos

## Requisitos Previos

- [Node.js](#) (v22.14.0 recomendado)
- [npm](#) (10.9.2 viene con Node.js)
- [Gmail](#) Cuenta de servicio de correo

# Instalación

## 1. Clona el repositorio

```
git clone https://github.com/ANHELLLOS/api-emitto.git
```

## 2. Install NPM packages

```
npm i
```

## 3. Configura tus .env .development .production

```
NODE_ENV=      # development or production
```

```
# DATA BASE
```

```
DB_PORT=5432
```

```
DB_HOST=127.0.0.1
```

```
DB_NAME=
```

```
DB_USER=
```

```
DB_PASSWORD=
```

```
DB_SSL=false
```

```
# SERVER
```

```
PORT=4000
```

```
# SMTP GOOGLE
```

```
MAIL_HOST=smtp.gmail.com
```

```
MAIL_PORT=587
```

```
MAIL_FROM=
```

```
MAIL_USER=
```

```
MAIL_PASS=
```

```
# FACTORY INFO
```

```
EMITTO_URL= # url client
```

```
EMITTO_EMAIL= # info support
```

```
# JSON WEB TOKEN
```

```
JWT_TIME=      # 60s, 30m, 30d
JWT_SECRET= # secret key

# REDIS

REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=
WORKER_CONCURRENCY=5 # Jobs concurrentes por worker
MAX_WORKERS=8 # Máximo número de workers

# ORIGINS DOMAINS

CORS_ORIGIN=
```

#### 4. Base datos Postgres - Redis

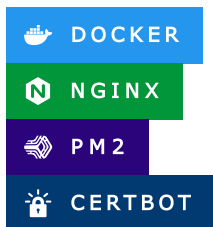
```
docker run -d \
--name postgres-alvanra \
-e POSTGRES_USER={tu_user} \
-e POSTGRES_PASSWORD={tu_pass} \
-e POSTGRES_DB={tu_db_name} \
-p 5432:5432 \
-v postgres_data:/var/lib/postgresql/data \
postgres:latest
```

```
docker run -d \
--name emitto-redis \
-p 127.0.0.1:6379:6379 \
redis:latest \
redis-server --requirepass your_pass
```

#### 5. Ejecutar server - local

```
npm run start:dev
```

## Despliegue en Producción



# Arquitectura de Despliegue

El despliegue de la aplicación se realiza sobre una **VPS** bajo una arquitectura basada en contenedores, lo que facilita la escalabilidad, el aislamiento y la portabilidad de los servicios.

## Tecnologías empleadas

- **Docker**: Contenerización de los servicios (frontend, backend, base de datos, etc.).
- **NGINX**: Actúa como **proxy inverso**, redirigiendo el tráfico HTTP/HTTPS a los contenedores adecuados.
- **Certbot + Let's Encrypt**: Para la emisión y renovación automática de certificados SSL, garantizando una conexión segura mediante HTTPS.
- **PM2**: Utilizado para la gestión de procesos Node.js fuera de los contenedores (si aplica), proporcionando reinicio automático, monitoreo y administración de logs.

## Flujo general

1. El dominio apunta a la IP pública de la VPS.
2. **NGINX** escucha en los puertos 80 (HTTP) y 443 (HTTPS).
3. Las solicitudes entrantes son redirigidas hacia los contenedores Docker correspondientes (por ejemplo: `/api` al backend, `/` al frontend).
4. **Certbot** genera y renueva los certificados SSL automáticamente.
5. Si se ejecutan servicios auxiliares fuera de Docker, **PM2** los gestiona como demonios.

## Ventajas

- Seguridad mediante HTTPS (SSL/TLS).
- Alta disponibilidad gracias a la gestión de procesos con PM2.
- Separación de responsabilidades mediante Docker.
- Fácil mantenimiento y escalabilidad.

---

## 1. Crear red de docker

Crea una red de Docker para que todos los contenedores se comuniquen entre sí de forma eficiente y segura. Esto facilita la conexión entre servicios como la base de datos, backend y frontend, sin necesidad de exponer todos los puertos al host.

```
docker network create emitto-net
```

## 2. Crear contenedore de Postgres

```
docker run -d \  
  --name postgres-alvanra \  
  --network emitto-net \  
  -e POSTGRES_USER={tu_user} \  
  -e POSTGRES_PASSWORD={tu_pass} \  
  -e POSTGRES_DB={tu_db_name} \  
  -p 5434:5432 \  
  -v postgres_data:/var/lib/postgresql/data \  
  postgres:latest
```

**NOTA:** La bandera `-v postgres_data:/var/lib/postgresql/data` asegura que los datos de su base de datos persistan incluso si el contenedor se detiene o se elimina.

## 3. Crear contenedore de Redis

```
docker run -d \  
  --name emitto-redis \  
  --network emitto-net \  
  -p 127.0.0.1:6379:6379 \  
  redis:latest \  
  redis-server --requirepass your_pass
```

## 3. Leventar API Emitto

### 1. Clona el repositorio

```
git clone https://github.com/ANHELL0S/api-emitto.git
```

### 2. Ejecuta la API NestJS con PM2 y compila para producción

- Paso 1: Compila el proyecto NestJS

```
npm run build
```

- Paso 2: Ejecuta la API con PM2

```
pm2 start dist/main.js --name api-emitto
```

- Paso 3: Reinicia la API con entorno de producción

```
NODE_ENV=production pm2 restart api-emitto --update-env
```

## 4. Configuración de NGINX y Certbot para despliegue seguro

### 1. Instalar NGINX

- Instalar NGINX:

```
sudo apt-get install nginx
```

- Iniciar el servicio NGINX:

```
sudo systemctl start nginx
```

- Verificar el estado del servicio NGINX:

```
sudo systemctl status nginx
```

- Habilitar el inicio automático (opcional):

```
sudo systemctl enable nginx
```

### 2. Instalar Certbot

```
sudo apt update
```

```
sudo apt install certbot python3-certbot-nginx -y
```



### 3. Crear certificado SSL - Certbot

```
sudo certbot --nginx -d tudominio.com -d www.tudominio.com
```

### 4. Configurar NGINX como Proxy Inverso

Ejemplo básico de configuración de NGINX para redirigir tráfico backend (NestJS API):

Primero crea un archivo (ej. emitto-api) en la ruta /etc/nginx/sites-available/

```
upstream emitto_server {
    least_conn;
    server localhost:4000; // donde corre la api
    keepalive 32; # Conexiones persistentes para mejor performance
}

server {
    if ($host = tu_dominio.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;
    server_name tu_dominio.com;

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name tu_dominio.com;

    # Límites para archivos grandes (agregar esto)
    client_max_body_size 20M; # Permite archivos hasta 20MB
    client_body_buffer_size 128k;
    client_body_timeout 300s;
    proxy_request_buffering off; # Importante para archivos grandes

    ssl_certificate /etc/letsencrypt/live/tu_dominio.com/fullchain.pem; #
    ssl_certificate_key /etc/letsencrypt/live/tu_dominio.com/privkey.pem;
```

```

# Configuración de SSL
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers off;
ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";

location / {
    proxy_pass http://emitto_server;

    # Timeouts extendidos (agregar esto)
    proxy_connect_timeout 600s;
    proxy_send_timeout 600s;
    proxy_read_timeout 600s;
    send_timeout 600s;

    # Headers existentes
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_cache_bypass $http_upgrade;

    # Buffer settings (agregar esto)
    proxy_buffers 8 16k;
    proxy_buffer_size 32k;
}
}

```

## 5. Crear certificados TLS

## 6. Crear enlace simbólico

Para habilitar la configuración del sitio emitto-api en NGINX, se debe crear un enlace simbólico desde el archivo de configuración ubicado en sites-available hacia sites-enabled. Esto permite que NGINX reconozca y cargue dicha configuración.

Ejecuta el siguiente comando con permisos de administrador:

```
sudo ln -s /etc/nginx/sites-available/emitto-api /etc/nginx/sites-enabled
```

## 7. Recargar NGINX para aplicar los cambios

```
sudo systemctl reload nginx
```

## 8. Fix redireccionar tráfico a API

- Si tienes problemas con nginx no redirige el tráfico elimina la conf por defecto de nginx:

```
sudo rm /etc/nginx/sites-enabled/default  
sudo systemctl reload nginx
```

# Contribución

Las contribuciones son lo que hace de la comunidad de código abierto un lugar increíble para aprender, inspirarse y crear. Cualquier contribución que hagas es **muy apreciada**.

Si tienes alguna sugerencia para mejorar esto, por favor, bifurca el repositorio y crea una solicitud de incorporación de cambios. También puedes abrir una incidencia con la etiqueta "mejora".

¡No olvides darle una estrella al proyecto! ¡Gracias de nuevo!

1. Fork el proyecto
2. Crea tu rama de funciones( `git checkout -b feature/AmazingFeature` )
3. Confirma tus cambios( `git commit -m 'Add some AmazingFeature'` )
4. Push la rama( `git push origin feature/AmazingFeature` )
5. Abrir una PR

# License

Distribuido bajo la licencia del proyecto. Ver `LICENSE.txt` para más información.