

Manual de usuario de las API's expuestas por (GCAE).

ANDRES FELIPE ESCOBAR LOPEZ

DIRECTOR:

DARÍO ENRIQUE SOTO DURAN

CODIRECTOR:

AIXA EILEEN VILLAMIZAR JAIMES



Tecnológico de Antioquia - Institución Universitaria
Ingeniería en Software
Medellín, Colombia.
2024

OBJETIVO DEL DOCUMENTO

El objetivo de este documento es instruir al usuario el uso de las API's expuestas por GCAE, dando una explicación detallada del JSON requerido por el generador de código y el uso adecuado de la plataforma.

TABLA DE CONTENIDO

| | |
|--------------------------------|---|
| OBJETIVO DEL DOCUMENTO..... | 2 |
| TABLA DE CONTENIDO..... | 3 |
| TABLA DE FIGURAS | 4 |
| 1. DOCUMENTACIÓN SWAGGER | 5 |

TABLA DE FIGURAS

| | |
|--|----|
| Ilustración 1 Api-versión | 5 |
| Ilustración 2 Api Doc-swagger | 6 |
| Ilustración 3 Ejemplo GCAE.spec.json..... | 7 |
| Ilustración 4 GitHub GCAE.spec.json..... | 7 |
| Ilustración 5 JWT Login..... | 8 |
| Ilustración 6 Login JWT Token..... | 8 |
| Ilustración 7 Encabezado Autorización JWT Bearer | 8 |
| Ilustración 8 Recursos del API autenticación..... | 10 |
| Ilustración 9 Recursos Autenticación..... | 10 |
| Ilustración 10 Recurso /api/v1/auth/signin..... | 10 |
| Ilustración 11 Recurso autenticación..... | 11 |
| Ilustración 12 Ejecutar petición | 11 |
| Ilustración 13 HTTP 201 Respuesta signin..... | 11 |
| Ilustración 14 signin HTTP 400 Solicitud con cuerpo de entrada malo | 12 |
| Ilustración 15 signin HTTP 401 Usuario no autorizado | 13 |
| Ilustración 16 signin HTTP 500 Error interno en el servidor | 13 |
| Ilustración 17 Recurso /api/v1/auth/login | 14 |
| Ilustración 18 HTTP 200 Respuesta login..... | 14 |
| Ilustración 19 login HTTP 400 Solicitud con cuerpo de entrada malo..... | 15 |
| Ilustración 20 login HTTP 401 Usuario no autorizado | 15 |
| Ilustración 21 login HTTP 500 Error interno en el servidor | 15 |
| Ilustración 22 Generate App Api..... | 15 |
| Ilustración 23 Recursos generador de aplicaciones..... | 16 |
| Ilustración 24 Recurso /api/v1/generateApp | 16 |
| Ilustración 25 HTTP 201 Respuesta generateApp | 20 |
| Ilustración 26 generateApp HTTP 400 Solicitud con cuerpo de entrada malo | 21 |
| Ilustración 27 generateApp HTTP 401 Usuario no autorizado | 21 |
| Ilustración 28 generateApp HTTP 500 Error interno en el servidor..... | 21 |
| Ilustración 29 Recurso /api/v1/getApp..... | 21 |
| Ilustración 30 Parámetro Path appName | 22 |
| Ilustración 31 HTTP 200 Respuesta getApp..... | 22 |
| Ilustración 32 getApp HTTP 400 Solicitud con cuerpo de entrada malo..... | 22 |
| Ilustración 33 getApp HTTP 401 Usuario no autorizado | 23 |
| Ilustración 34 getApp HTTP 500 Error interno en el servidor | 23 |
| Ilustración 35 Postman Api..... | 23 |

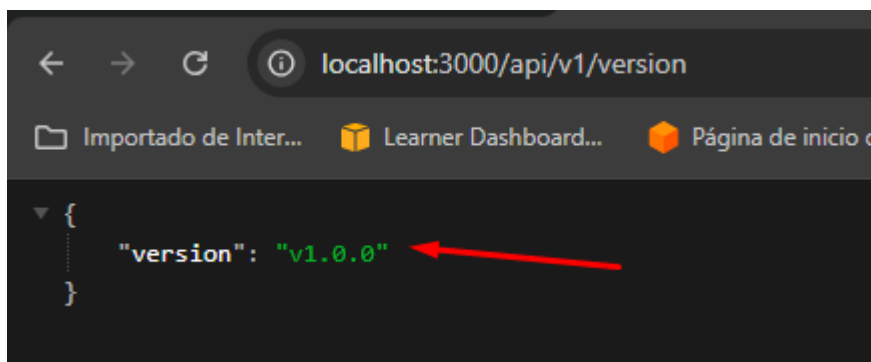
1. DOCUMENTACIÓN SWAGGER

GCAE expone una interfaz de documentación para visualizar de manera más clara cada API REST expuesta, mostrando las entidades requeridas y los diferentes tipos de respuesta según los códigos HTTP que corresponda.

A continuación, se muestra la entrada principal a la documentación y como verificar si la aplicación está corriendo de manera correcta y con la versión adecuada.

Para verificar la versión corriendo de GCAE, ingrese la siguiente URL ***<http://localhost:3000/api/v1/version>*** en cualquier navegador y ejecute enter o enviar:

Ilustración 1 Api-versión



Debe entregar una respuesta de esta manera

```
{ "version": "v1.0.0" }
```

La documentación de Swagger para GCAE se puede visualizar en la siguiente ruta, “*para casos de ejemplos de este documento el host es local para el ambiente productivo cambiar el host al host correspondiente al servidor donde se aloja la aplicación*”. Ingrese la siguiente URL <http://localhost:3000/api/v1/api-docs/#/> se debe visualizar lo siguiente:

Ilustración 2 Api Doc-swagger

localhost:3000/api/v1/api-docs/#/

Swagger

REST - Swagger v1.0.0 - dilo OAS 3.0

GCAE REST API with Swagger doc:

- [Download PDF application user manual](#)
- [Download JSON from the sample application for MONGO](#)
- [Download JSON from the sample application for POSTGRES](#)
- [Contact the developer](#)

Authorize

GCAE API software that automates the process of generating base code with standard architectures within the framework of SOLID principles focused on the Back-end.

Generate App

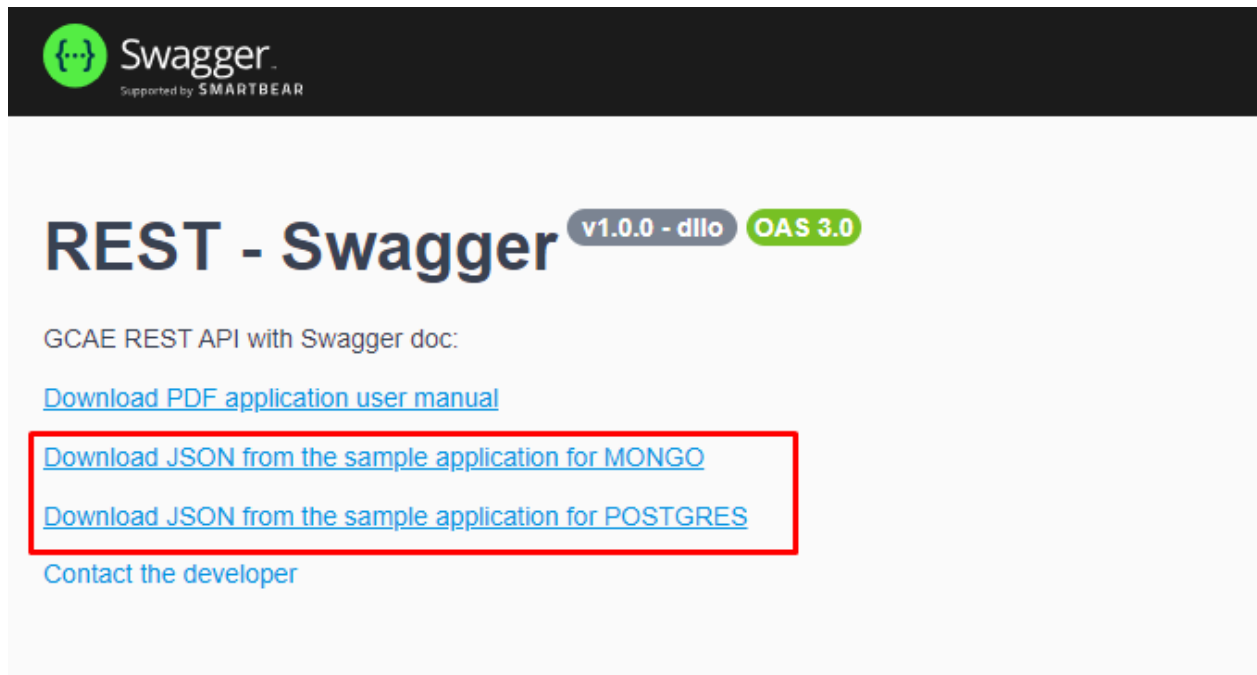
Auth

Schemas

- AppModel >
- UserModel >
- AuthModel >
- LoginModel >
- DefaultException >
- ResponseMessageModel >

NOTA: El archivo ***GCAE_MONGO.spec.json*** o ***GCAE_POSTGRES.spec.json***, es el JSON de ejemplo con cada atributo requerido para generar una aplicación desde la plataforma. Puede ser descargado desde el siguiente enlace

Ilustración 3 Ejemplo GCAE.spec.json



The image shows the Swagger REST API page for GCAE. The header features the Swagger logo and the text "Supported by SMARTBEAR". The main heading is "REST - Swagger" with two version badges: "v1.0.0 - d11o" and "OAS 3.0". Below the heading, it says "GCAE REST API with Swagger doc:". There are four links: "Download PDF application user manual", "Download JSON from the sample application for MONGO", "Download JSON from the sample application for POSTGRES", and "Contact the developer". The last two links are enclosed in a red rectangular box.

REST - Swagger

v1.0.0 - d11o OAS 3.0

GCAE REST API with Swagger doc:

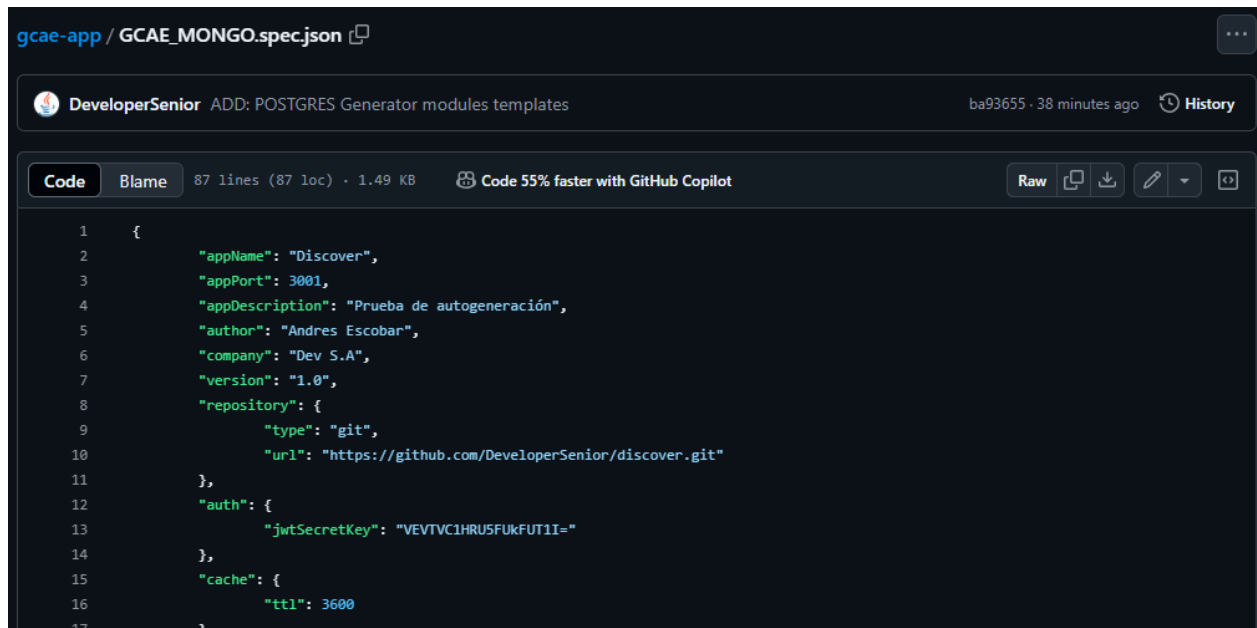
[Download PDF application user manual](#)

[Download JSON from the sample application for MONGO](#)

[Download JSON from the sample application for POSTGRES](#)

[Contact the developer](#)

Ilustración 4 GitHub GCAE.spec.json



The image shows a GitHub repository view for the file "gcae-app / GCAE_MONGO.spec.json". The repository is owned by "DeveloperSenior" and has a commit "ADD: POSTGRES Generator modules templates" from "ba93655 · 38 minutes ago". The file is 87 lines (87 loc) and 1.49 KB. It includes a GitHub Copilot badge. The code is a JSON object with the following structure:

```
1  {
2      "appName": "Discover",
3      "appPort": 3001,
4      "appDescription": "Prueba de autogeneración",
5      "author": "Andres Escobar",
6      "company": "Dev S.A",
7      "version": "1.0",
8      "repository": {
9          "type": "git",
10         "url": "https://github.com/DeveloperSenior/discover.git"
11     },
12     "auth": {
13         "jwtSecretKey": "VEVTVC1HRUSFUKFUT1I="
14     },
15     "cache": {
16         "ttl": 3600
17     }
18 }
```

Lo siguiente explica cada opción y botón que se visualiza en la interfaz:

Ilustración 5 JWT Login



Ilustración 6 Login JWT Token

A dialog box titled "Available authorizations" with a close button (X) in the top right corner. The main content area is titled "Authorization (http, Bearer)". Below this, it says "Enter the token without the Bearer: prefix, e.g. 'abcde12345'." followed by "Value:". There is a text input field below "Value:". A red arrow points to the input field. At the bottom of the dialog, there are two buttons: "Authorize" (green text) and "Close" (grey text).

Este botón permite incluir el token JWT que entrega el login a GCAE el cual permite consumir las API's de manera segura.

Al darle al boton **Authorize**, lo que se hace es crear una Cabecera de autenticación tipo Bearer

Ilustración 7 Encabezado Autorización JWT Bearer

Available authorizations



Authorization (http, Bearer)

Authorized

Enter the token without the **Bearer:** prefix, e.g. "abcde12345".

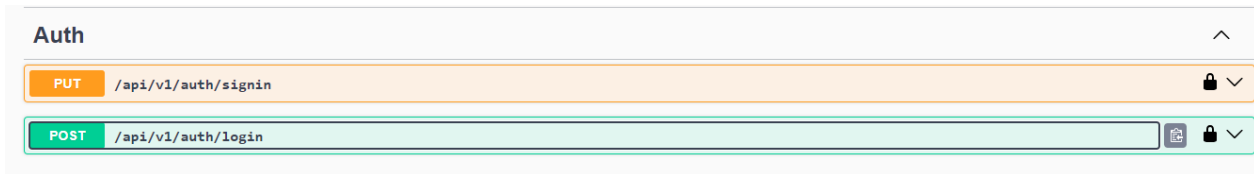
Value: *****

Logout

Close

El primer API es:

Ilustración 8 Recursos del API autenticación



Este punto de integración permite registrar e ingresar usuarios al sistema, autoriza el consumo de las API's expuestas por GCAE en el punto de integración **"Generate App"**

Ilustración 9 Recursos Autenticación



Ilustración 10 Recurso /api/v1/auth/signin



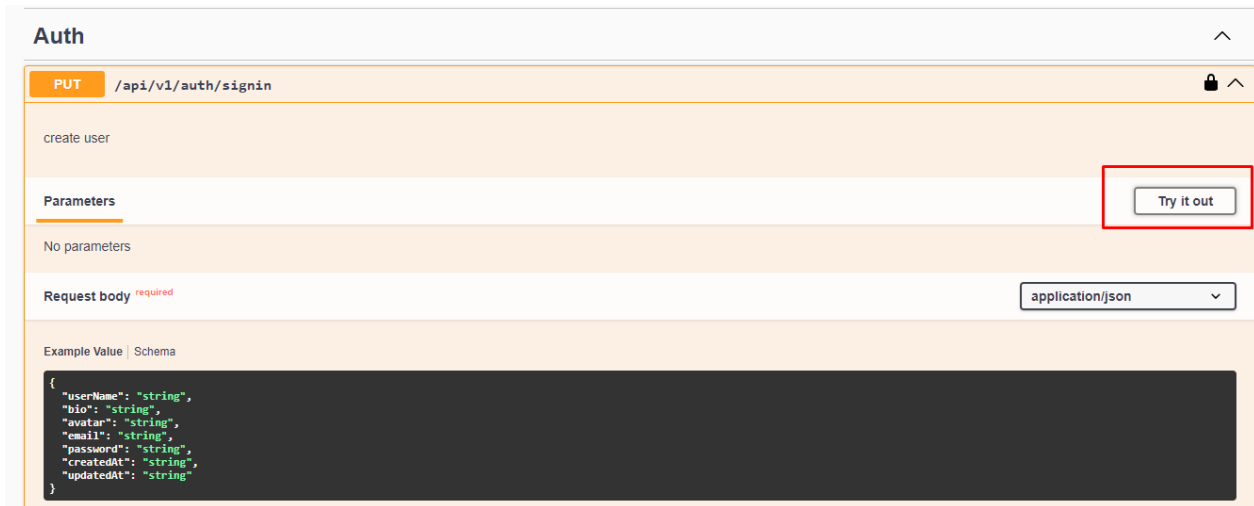
El primer recurso es tipo HTTP PUT ***/api/v1/auth/signin***, permite registrar un usuario nuevo con el siguiente JSON en el cuerpo de la petición:

```
{
  "userName": "string",
  "bio": "string",
  "avatar": "string",
  "email": "string",
  "password": "string"
}
```

- **userName:** Nombre de usuario con el que se va a identificar en la aplicación, campo tipo cadena de texto.
- **bio:** Descripción corta de la biografía del usuario que se está registrando, campo tipo cadena de texto.
- **avatar:** Link o URL de la foto o el avatar del usuario que se está registrando, campo tipo URL.
- **email:** Correo electrónico del usuario que se registra, este campo permite el ingreso a la plataforma por el API de login, campo tipo email.
- **password:** Contraseña que se utiliza para ingresar a la plataforma, campo tipo cadena de texto, se almacena encriptado en la base de datos.

Este JSON permite registrar un usuario nuevo por medio del recurso ***/api/v1/auth/signin***, utilizando el botón ***Try it out***.

Ilustración 11 Recurso autenticación



Se debe colocar el JSON con cada atributo diligenciado y click al boton **Execute**

Ilustración 12 Ejecutar petición



La respuesta exitosa se vería algo así:

Ilustración 13 HTTP 201 Respuesta signin

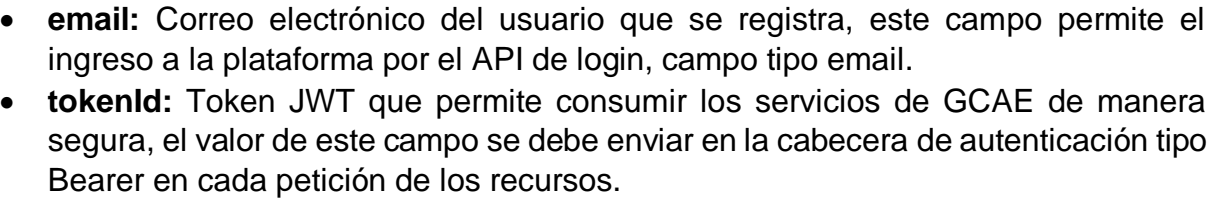


Ilustración 14 signin HTTP 400 Solicitud con cuerpo de entrada malo

400 bad request error body

Media type
application/json

Example Value | Schema

```
[
  {
    "message": "string"
  }
]
```

Ilustración 15 signin HTTP 401 Usuario no autorizado

401 Unauthorized

Media type
application/json

Example Value | Schema

```
{
  "code": "string",
  "type": "string",
  "message": "string",
  "exception": "string"
}
```

Ilustración 16 signin HTTP 500 Error interno en el servidor

500 internal server error

Media type
application/json

Example Value | Schema

```
{
  "code": "string",
  "type": "string",
  "message": "string",
  "exception": "string"
}
```

Auth

Ilustración 19 login HTTP 400 Solicitud con cuerpo de entrada malo

400 bad request error body

Media type
application/json

Example Value | Schema

```
{
  "message": "string"
}
```

Ilustración 20 login HTTP 401 Usuario no autorizado

401 Unauthorized

Media type
application/json

Example Value | Schema

```
{
  "code": "string",
  "type": "string",
  "message": "string",
  "exception": "string"
}
```

Ilustración 21 login HTTP 500 Error interno en el servidor

500 internal server error

Media type
application/json

Example Value | Schema

```
{
  "code": "string",
  "type": "string",
  "message": "string",
  "exception": "string"
}
```

El segundo API es la generación de aplicaciones

Ilustración 22 Generate App Api

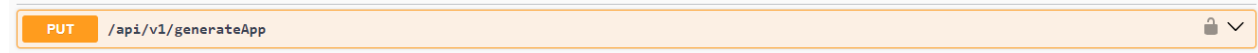
Generate App

| | | |
|-----|--------------------------|---|
| PUT | /api/v1/generateApp | 🔒 |
| GET | /api/v1/getApp/{appName} | 🔒 |
| GET | /api/v1/getDemo | 🔒 |

Este punto de integración permite crear Apps y Descargar aplicaciones que ya fueron creadas en el sistema.

Generate App

Ilustración 24 Recurso `/api/v1/generateApp`



El primer recurso es tipo HTTP PUT **`/api/v1/generateApp`**, permite crear una aplicación nueva con el siguiente JSON en el cuerpo de la petición:

```
{
  "appName": "string",
  "appPort": number,
  "appDescription": "string",
  "author": "string",
  "company": "string",
  "version": "string",
  "repository": {
    "type": "string",
    "url": "string"
  },
  "auth": {
    "jwtSecretKey": "string"
  },
  "cache": {
    "ttl": number
  },
  "dataBase": {
    "type": "string",
    "serviceName": "string",
    "host": "string",
    "protocol": "string",
    "port": number,
    "user": "string",
    "pass": "string"
  },
  "entities": [
    {
      "name": "string",
      "description": "string",
      "fields": [

```



```

        "name": "string",
        "type": "string",
        "items": {
            "type": "string",
            "ref": "string"
        },
        "pk": boolean,
        "required": boolean
    }
}
]
}

```

- **appName:** Nombre de la aplicación, es un valor único y se va a utilizar para volver a descargar la aplicación por el recurso `/api/v1/getApp/{appName}`, campo tipo cadena de texto.
- **appPort:** Define el puerto TCP por el cual escuchan los servicios la aplicación a generar, campo tipo numérico.
- **appDescription:** Descripción funcional de la aplicación a generar, campo tipo cadena de texto.
- **author:** Define el nombre del dueño de la aplicación a generar, campo tipo cadena de texto.
- **company:** Define el nombre de la compañía que respalda la aplicación a generar, campo tipo cadena de texto.
- **version:** Define la versión inicial de la aplicación a generar, campo tipo cadena de texto.
- **repository:** Es un objeto que define la configuración del repositorio de versión de código para la aplicación a generar, contiene lo siguientes atributos:
 - **type:** Define el tipo de repositorio (git | svn | etc) de la aplicación a generar, es un campo tipo cadena de texto.
 - **url:** Define la URL del servicio de control de versiones de la aplicación a generar, es un campo tipo URL.
- **auth:** Es un objeto que define la configuración del token JWT que va a utilizar la aplicación a generar, contiene los siguientes atributos:
 - **jwtSecretKey:** Define la clave secreta para la llave privada que requiere JWT, esta cadena debe estar encriptada en base 64. Es una cadena (codificada en UTF-8), un búfer, un objeto o un KeyObject

que contiene el secreto para los algoritmos HMAC o la clave privada codificada en PEM para RSA y ECDSA. En el caso de una clave privada con frase de contraseña, { key, passphrase } se puede utilizar un objeto (según la documentación de cifrado); en este caso, asegúrese de pasar la opción. Al firmar con algoritmos RSA, la longitud mínima del módulo es 2048, excepto cuando la opción allowInsecureKeySizes se establece en verdadero. Las claves privadas por debajo de este tamaño se rechazarán con un error. Ver la documentación <https://github.com/auth0/node-jsonwebtoken>

- **cache:** Es un objeto que define la configuración del cache que se va a utilizar en la aplicación a generar, contiene los siguientes atributos:
 - **ttl:** Define la cantidad de tiempo que los datos pueden existir en el cache antes de ser descartados, (predeterminado: 0) el ttl estándar como número en segundos para cada elemento de caché generado. 0= ilimitado, campo tipo numérico.
- **dataBase:** Es un objeto que define la configuración de la conexión a la base de datos que va a utilizar la aplicación a generar, contiene los siguientes atributos:
 - **type:** Define el tipo de base de datos, solo está permitido el valor "MONGO" y "POSTGRES", es un campo tipo cadena de texto.
 - **serviceName:** Define el nombre del servicio o de la base de datos, es un campo tipo cadena de texto.
 - **host:** Define el DNS o IP del host donde está alojado el servicio de la base de datos, NOTA: si es requerido incluir el puerto en la cadena del host es permitido, **ej: localhost:27017**, es un campo tipo cadena de texto.
 - **port:** Define el puerto de escucha de la base de datos, es un campo tipo numérico.
 - **protocol:** Define el protocolo de conexión a la base de datos MONGO, **ej: mongodb+srv o mongodb**, es un campo tipo cadena de texto.
 - **user:** Define el usuario de conexión a la base de datos, campo tipo cadena de texto.
 - **pass:** Define la contraseña del usuario de conexión a la base de datos, debe estar encriptada en base 64.

- **entities:** Es una lista de objetos que define la configuración de las entidades que va a contener la aplicación a generar, estas entidades son las que van a tener los procesos de negocio y todo el flujo modular de la aplicación, se va a generar para cada una un C.R.U.D, contiene objetos entidades con los siguientes atributos:

- **name:** Define el nombre de la entidad a generar, Los nombres deben de ser descriptivos y no muy largos, con la primera letra de cada palabra que lo forma en mayúsculas. Por lo general se usan sustantivos o adjetivos. Ej: Persona, Usuario, FacturaVenta, Compra, Venta, Articulo.
- **description:** Descripción de manera natural del comportamiento o el objetivo de la entidad a generar.
- **fields:** Es un objeto que define los atributos o características que va a contener la entidad a generar, contiene los siguientes atributos:

1. **name:** Define el nombre del atributo de la entidad a generar, tener en cuenta las buenas prácticas de nombramiento, es un campo tipo cadena de texto.

NOTA: Al nombrar atributos, se recomienda seguir las siguientes buenas prácticas:

Elegir nombres descriptivos: Los nombres deben ser claros y consistentes, y describir qué hace el código.

Seguir normas de nomenclatura: Es importante seguir las convenciones y normas de nomenclatura establecidas.

Evitar palabras y símbolos reservados: No se deben usar palabras y símbolos reservados.

Utilizar etiquetas únicas: Las etiquetas deben ser significativas y únicas.

Revisar los nombres: Es importante revisar los nombres y etiquetas.

Utilizar un nombre corto y claro: Los nombres deben ser cortos y claros para facilitar su mapeo.

Utilizar el mismo nombre que se mostrará en los formularios: Es recomendable utilizar el mismo nombre que se mostrará en los formularios.

Utilizar una longitud de 2 a 4 palabras o entre 8 y 20 caracteres: La longitud recomendada es de 2 a 4 palabras o entre 8 y 20 caracteres.

2. **type:** Define el tipo de dato que almacena el atributo, solo está permitido los siguientes valores (**String** / **Array** / **Object** / **Number** / **Date** / **Relation**), es un campo tipo cadena de texto
3. **Ítems:** Solo aplica si el tipo ingresado es **Array** o **Relation**, es un objeto que define las características de la lista o la relación (solo POSTGRES), contiene los siguientes atributos:

type: Define el tipo de dato del valor de cada item de la lista, solo está permitido los siguientes valores (**String** / **Object** / **Number** / **Date**), es un campo tipo cadena de texto

ref: Solo aplica al tipo **Object**, hace referencia al Nombre del Objeto (Solo MONGO) o a la entidad que se relaciona o hace clave foránea con el campo (Solo para POSTGRES) y debe existir como una entidad de la aplicación, es un campo tipo cadena de texto.

4. **pk:** Define si el atributo de la entidad es clave o identificador únicos, es un campo tipo booleano. (Predeterminado false)
5. **required:** Define si el atributo de la entidad es obligatorio, es un campo tipo booleano. (Predeterminado false)

Este recurso descarga un archivo en formato Zip, la respuesta exitosa se vería algo así:

Ilustración 25 HTTP 201 Respuesta generateApp

| Server response | |
|-----------------|---|
| Code | Details |
| 201 | <p>Response body</p> <p>Download file</p> <p>Response headers</p> <pre> access-control-allow-origin: * connection: keep-alive content-disposition: attachment; filename=mobil-api.zip content-length: 71218 content-security-policy: default-src 'self';base-uri 'self';font-src 'self' https: data:;form-action 'self';frame-ancestors 'self';img-src 'self' data:;object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self' https: 'unsafe-inline';upgrade-insecure-requests content-type: application/octet-stream cross-origin-opener-policy: same-origin cross-origin-resource-policy: same-origin date: Tue,17 Sep 2024 13:37:12 GMT etag: W/"11632-m3cy8NBKJDD0WwVtW8PfRzLRs" keep-alive: timeout=5 origin-agent-cluster: ?1 referrer-policy: no-referrer strict-transport-security: max-age=15552000; includeSubDomains x-content-type-options: nosniff x-dns-prefetch-control: off x-download-options: noopen x-frame-options: SAMEORIGIN x-permitted-cross-domain-policies: none x-xss-protection: 0 </pre> |

Ilustración 26 generateApp HTTP 400 Solicitud con cuerpo de entrada malo

400 bad request error body

Media type
application/json

Example Value | Schema

```
[
  {
    "message": "string"
  }
]
```

Ilustración 27 generateApp HTTP 401 Usuario no autorizado

401 Unauthorized

Media type
application/json

Example Value | Schema

```
{
  "code": "string",
  "type": "string",
  "message": "string",
  "exception": "string"
}
```

Ilustración 28 generateApp HTTP 500 Error interno en el servidor

500 internal server error

Media type
application/json

Example Value | Schema

```
{
  "code": "string",
  "type": "string",
  "message": "string",
  "exception": "string"
}
```

Generate App

Ilustración 29 Recurso /api/v1/getApp

GET /api/v1/getApp/{appName}

📄 🔒 ▼

El segundo recurso es tipo HTTP GET ***/api/v1/getApp/{appName}***, permite descargar una aplicación existente no contiene cuerpo de petición, y debe ir el nombre de la aplicación a buscar en el parámetro Path ***{appName}***:

Ilustración 30 Parámetro Path appName

The screenshot shows a REST client interface. At the top, the method is 'GET' and the URL is '/api/v1/getApp/{appName}'. Below the URL bar, the text 'Get the existing application' is displayed. A 'Parameters' tab is active, showing a table with two columns: 'Name' and 'Description'. There is one parameter named 'appName' with a red asterisk indicating it is required. Its description is 'App name to get.' and its type is '(path)'. The value 'Discover' is entered in the input field. A red arrow points from the '(path)' label to the input field. At the bottom, there are 'Execute' and 'Clear' buttons. A 'Cancel' button is also visible in the top right corner of the parameters section.

Este recurso descarga un archivo en formato Zip , la respuesta exitosa se vería algo así:

Ilustración 31 HTTP 200 Respuesta getApp

The screenshot shows the 'Server response' section of a REST client. The status code is '200' and the message is 'Undocumented'. The 'Response body' is a 'Download file' link. The 'Response headers' are displayed in a dark box with the following content:

```
access-control-allow-origin: *
connection: keep-alive
content-disposition: attachment; filename=mobil-api.zip
content-length: 71218
content-security-policy: default-src 'self';base-uri 'self';font-src 'self' https: data:;form-action 'self';frame-ancestors 'self';img-src 'self' data;object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self' https: 'unsafe-inline';upgrade-insecure-requests
content-type: application/octet-stream
cross-origin-opener-policy: same-origin
cross-origin-resource-policy: same-origin
date: Tue, 17 Sep 2024 13:48:48 GMT
etag: W/"11632-m3cy6NBKJDD0Mv6VwV8PFRzLRs"
keep-alive: timeout=5
origin-agent-cluster: ?1
referrer-policy: no-referrer
strict-transport-security: max-age=15552000; includeSubDomains
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-download-options: noopen
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-xss-protection: 0
```

Ilustración 32 getApp HTTP 400 Solicitud con cuerpo de entrada malo

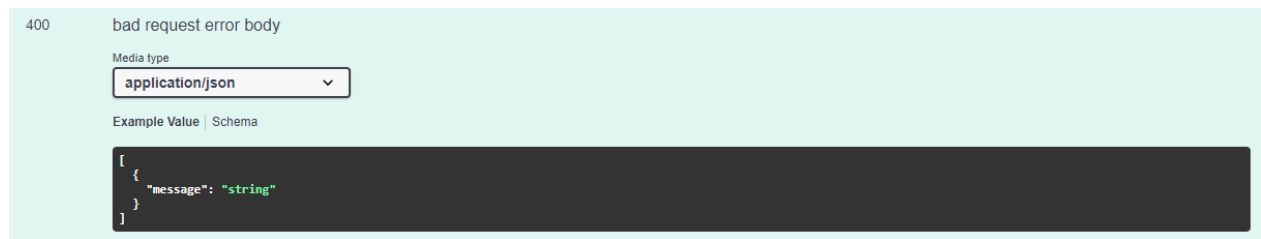


Ilustración 33 getApp HTTP 401 Usuario no autorizado

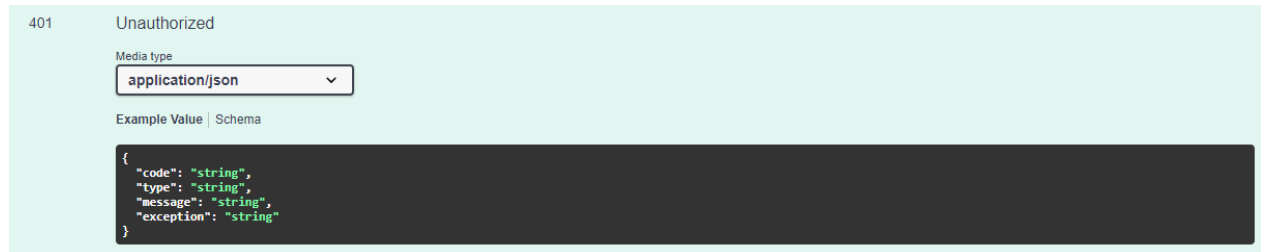
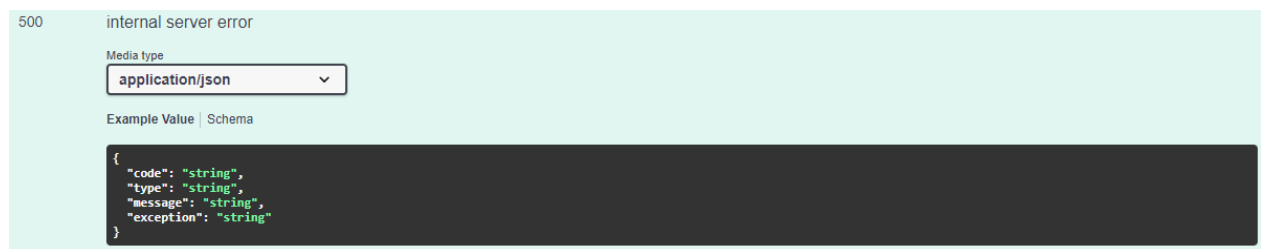


Ilustración 34 getApp HTTP 500 Error interno en el servidor



Otra forma de utilizar las API's es importando el archivo [gcae-api.postman_collection.json](#) en la herramienta [Postman](#)

Ilustración 35 Postman Api

