Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de Aplicaciones 2 - Obligatorio 1

Evidencia del diseño y especificación de la API

https://github.com/ORT-DA2/OBL-Asadurian-Reyes.git

Diego Asadurian 198874 - Hernán Reyes 235861

Tutor: Gabriel Piffaretti, Nicolas Blanco, Daniel Acevedo

07 de Octubre de 2021

Evidencia del diseño y especificación de la API **3**
Estructura 3
Mecanismo de autenticación 4
Descripción de los códigos devueltos 6

Resource de la API **7**
Sesión 7
Login 7
Cerrar Sesión 7
Usuario 8
Crear Usuario 8
Obtener Usuarios 8
Obtener Usuario 9
Proyecto 9
Crear Proyecto 10
Obtener proyectos 10
Obtener los Bugs de todos los proyectos 10
Obtener un Proyecto 11
Obtener bugs de un proyecto 11
Eliminar un proyecto 11
Modificar Proyecto 12
Bug 13
Crear Bug 13
Obtener bugs 13
Obtener un bug 14
Eliminar un bug 14
Modificar un bug 15
Developer 15
Obtener un bug asociado a un desarrollador 16
Agregar un desarrollador a un proyecto 16
Eliminar un desarrollador de un proyecto 17
Obtener cantidad de bugs resueltos por un desarrollador 17
Actualizar el estado de un bug con un desarrollador 17
Tester 17
Obtener bugs asociados a un tester 18
Agregar un tester a un proyecto 18
Eliminar un tester de un proyecto 19
Imports 19
Crear import de bugs 19

Evidencia del diseño y especificación de la API

El objetivo de este documento es especificar distintos criterios de diseño que se utilizaron para el desarrollo de nuestra aplicación API.

Se detalla la estructura, los manejos de modelos, inyección de dependencia, mecanismo de autenticación, manejo de excepciones entre otras.

Nuestra estructura utilizada es REST donde la misma debe de cumplir con los 6 principios principales. La misma está basada en una arquitectura HTTP a la hora de comunicarse con el cliente/servidor.

Estructura

- Modelos

Decidimos que cada entidad del dominio tiene sus diferentes modelos en nuestra api (EntryModel, OutModel, UpdateModel).

El modelo Entry se utiliza para especificar que dicho modelo va a ingresar al sistema, por otro lado el modelo Out, se utiliza para enviarle al usuario su solicitud. También el modelo Update para poder manejar qué datos son editables y cuáles no.

Algunos modelos a parte de los mencionados también tienen un modelo extra que es para brindarle al usuario una determinada información, como es el ejemplo del modelo ProjectReportModel, el cual únicamente lo utilizamos para enviarle al usuario el reporte del proyecto.

El propósito de dichos modelos fue limitar la información de estos mismos ya que la creación de un objeto puede no ser igual al retorno del mismo, por ejemplo.

- Clase base

Implementamos una clase base denominada ApiBaseController que hereda de ControllerBase. La misma se implementó para que todos nuestros controller hereden de ella.

De esta manera solo debemos especificar que todos los controllers son [ApiController] en una única clase y no en cada controller. Es así que no estamos repitiendo código.

De esta misma manera se le asignó nuestro filtro de excepciones a dicha clase base para que a todos nuestros controller se le aplique dicho filtro.

- Filters Exception

Por otro lado utilizamos un filter de excepción (ExceptionFilter), el cual lo denominamos en la clase ApiBaseController ya que es desde la cual todos los controller heredan de ella, con el fin de evitar duplicar código. Este filter lo que hace es obtener las excepciones de los métodos que son ejecutadas en los controllers y devolverle a nuestro cliente el status code con un mensaje de error correspondiente.

Mecanismo de autenticación

Para autenticar a los distintos usuarios se creó un filtro, el mismo se ejecuta antes de que la request lleguen a los controllers.
El funcionamiento de nuestra autenticación es el siguiente:

El usuario solicita loggearse mediante en request (POST) enviando en su body el email y la contraseña.

El sistema valida que la combinación de email y contraseña existan en la base de datos. Es decir exista el usuario y haya puesto correctamente la contraseña.

En caso de que alguno de los datos sean incorrectos se devuelve un código de estado 400 y un mensaje de lo sucedido, en este caso, email y/o contraseña invalido.

Por otro lado en caso de que el email exista y se haya puesto correctamente la contraseña el sistema genera un token, dicho token es generado por un identificador y a continuación un guid generado.

El identificador que generamos es el rol del usuario. Para dar un ejemplo un token de los 3 tipos de usuarios se visualizarán de la siguiente manera.

- Administrador-866e12c3-7656-43c3-bf19-01141ce8cb82
- Tester-7a24fd81-1698-4be0-a431-18dea61c70e6
- Desarrollador-58721e37-df41-45a6-9504-5bb2f33b9efd

De esta manera podemos identificar qué permisos tiene dicho token.

Luego de haberse logueado correctamente el sistema le devolverá el token generado y el cliente podrá y deberá utilizarlo en cada request a realizar.
Dicho Token debe ser colocado en el Header con la key “Authorization”.

En caso de que el token no tenga permiso para realizar dicha operación se le devolverá un código de error 403 Forbidden.

En caso de no haber ingresado token o el usuario no está logueado en el sistema se devolverá 401 Unauthorized.

- Filters Authorization

Para la autenticación comentada previamente utilizamos el AuthorizationFilter el cual recibe por parámetro un string el cual es el rol al cual se quiere autorizar o no, con esto evitamos la duplicación de código.

Este filtro se encarga de tomar el token desde el Header y validarlo, se encapsula la lógica en un solo lugar y este se ejecuta antes de llegar a los métodos del controller. Por lo que si el token no es válido o inexistente no se accede a los métodos del controller que lo implementen.

Otro detalle a tener en cuenta es que generamos una clase estática “Authorization”, la misma tiene los distintos permisos que puede tener un endpoint.
Es decir, nuestros token de autorización están conformados con un identificador como especificamos anteriormente (nuestro rol) y luego un guid generado. En esta clase se identificaron las posibles combinaciones de autorizaciones (un método puede ser accedido por dos tipos de roles, por ejemplo) por tanto se definieron las constantes necesarias para realizar las distintas autorizaciones.

```
public const string AllAutorization =
    Rol.administrator + "," + Rol.developer + "," + Rol.tester;

public const string Administrator =
    Rol.administrator;

public const string Developer =
    Rol.administrator;

public const string Tester =
    Rol.administrator;
```

Viendo nuestro código podemos ver la constante “AllAutorization” la cual tiene los identificadores de todos los roles. Pudiendo ser consumido en nuestro filtro de la siguiente manera.

```
[HttpGet]
[AuthorizationFilter(Autorization.AllAutorization)]
1 reference
public IActionResult GetAllBugs()
{
```

Descripción de los códigos devueltos

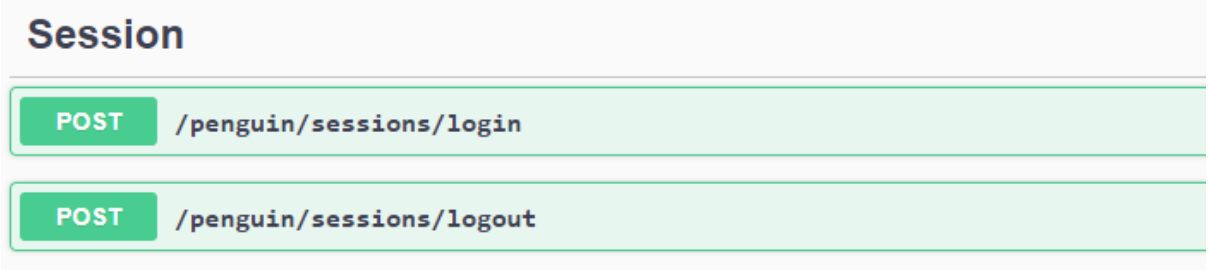
Los retornos que manejamos en nuestra API contienen mensajes de éxito o error, esto va de la mano con los Status Code que brindamos para indicarle al usuario más información sobre la petición que realizó el cliente al servidor.

Los StatusCode que manejamos en nuestra API fueron.

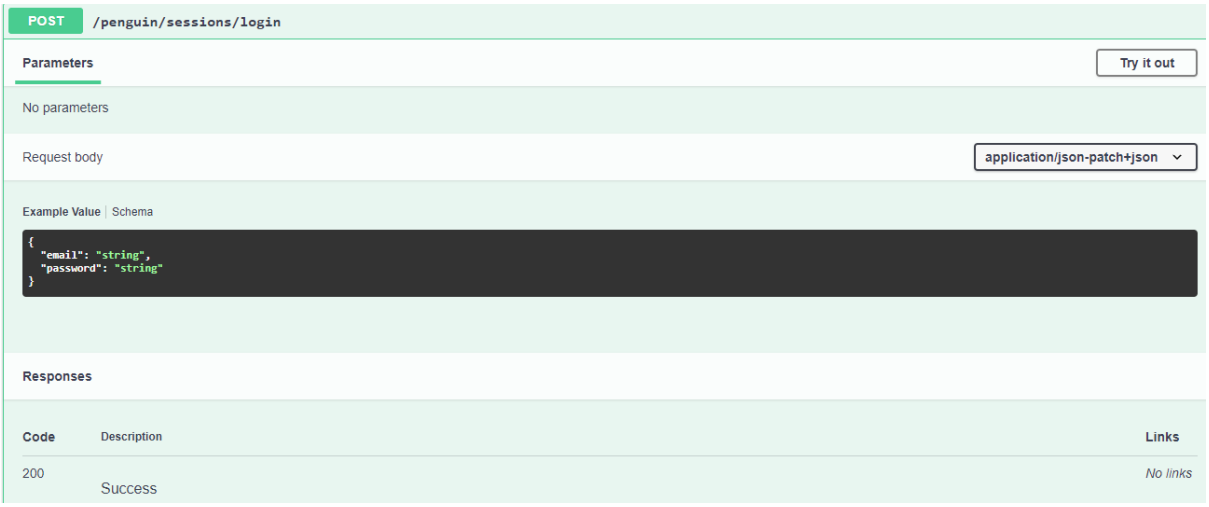
Status Code	Representación
200 Ok	La solicitud fue realizada correctamente y la respuesta incluye datos.
201 Created	Utilizada en solicitud POST, realizada correctamente.
204 NoContent	Solicitud realizada correctamente. Utilizada para métodos Update y Delete.
404 Error	Manejo de distintos errores en el sistema.
403	Identificar que no tenemos permisos
401	Identificar que no estamos logueados

Resource de la API

Sesión



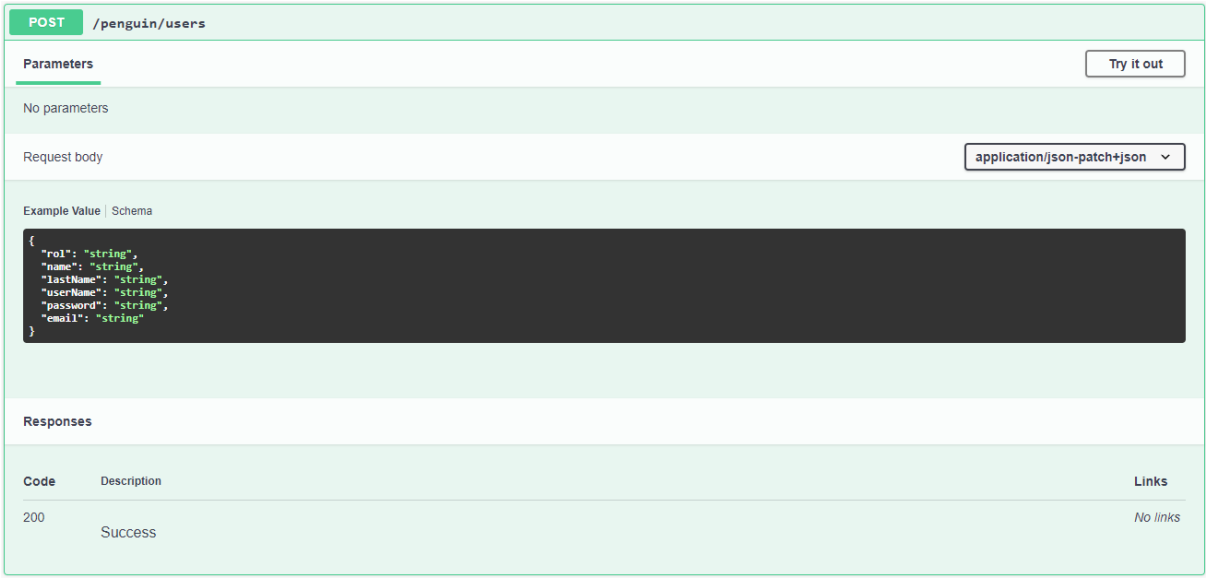
Login



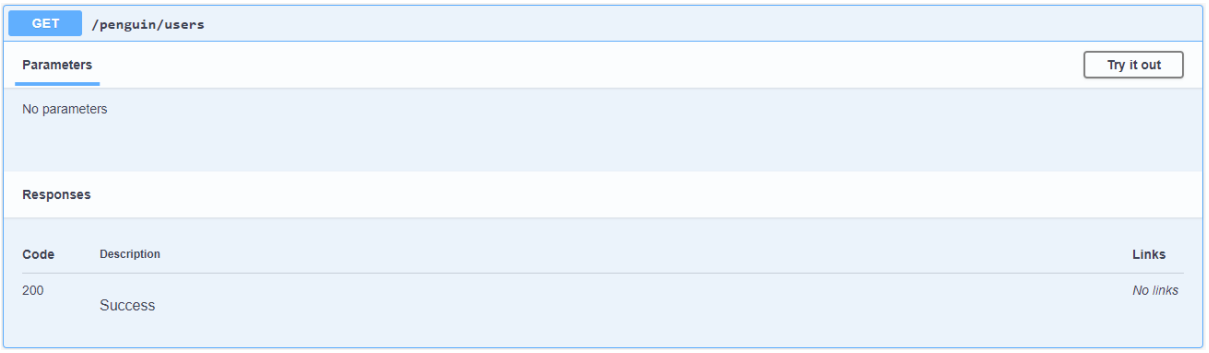
Cerrar Sesión

Usuario

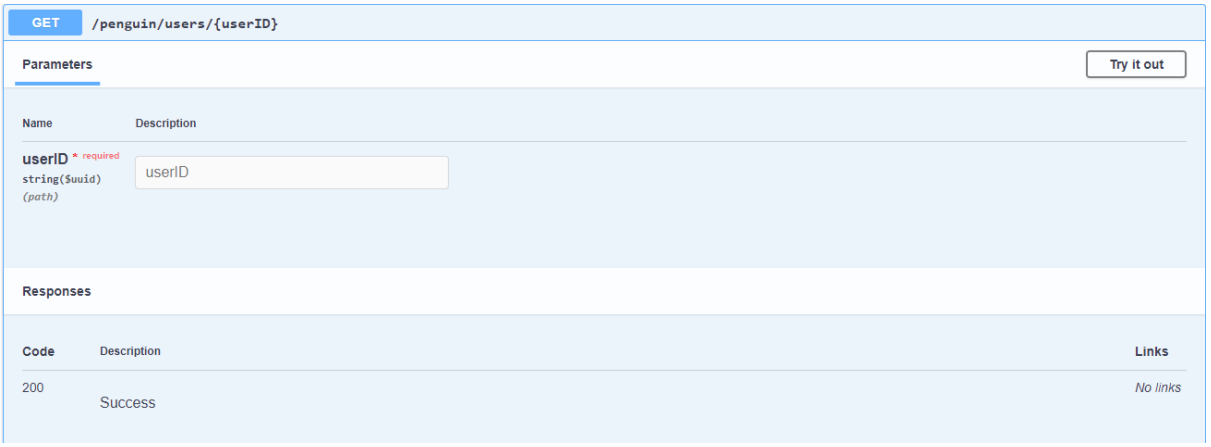
Crear Usuario



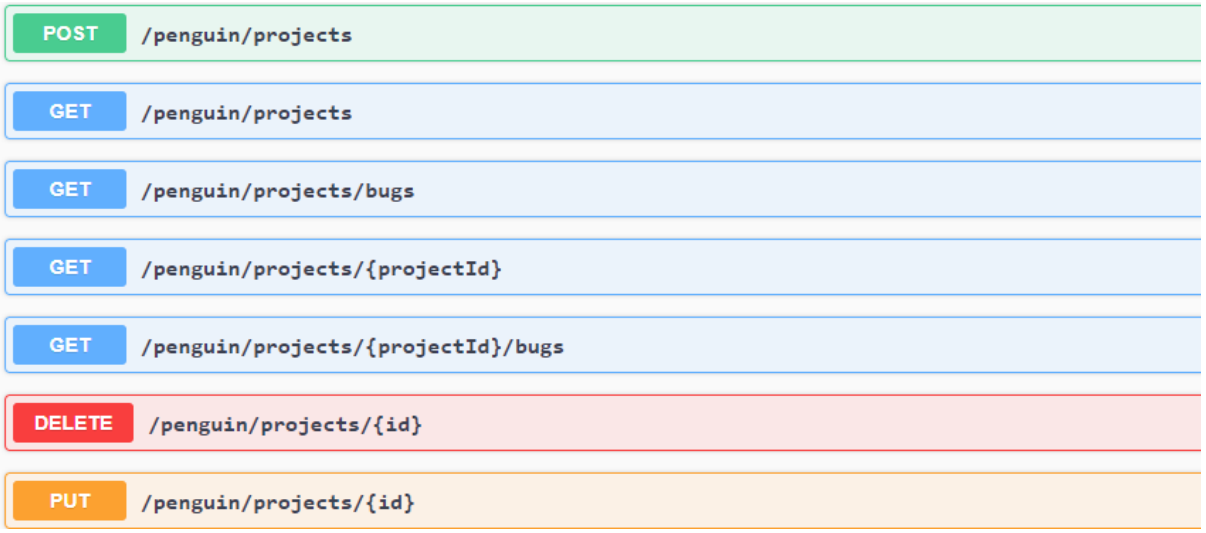
Obtener Usuarios



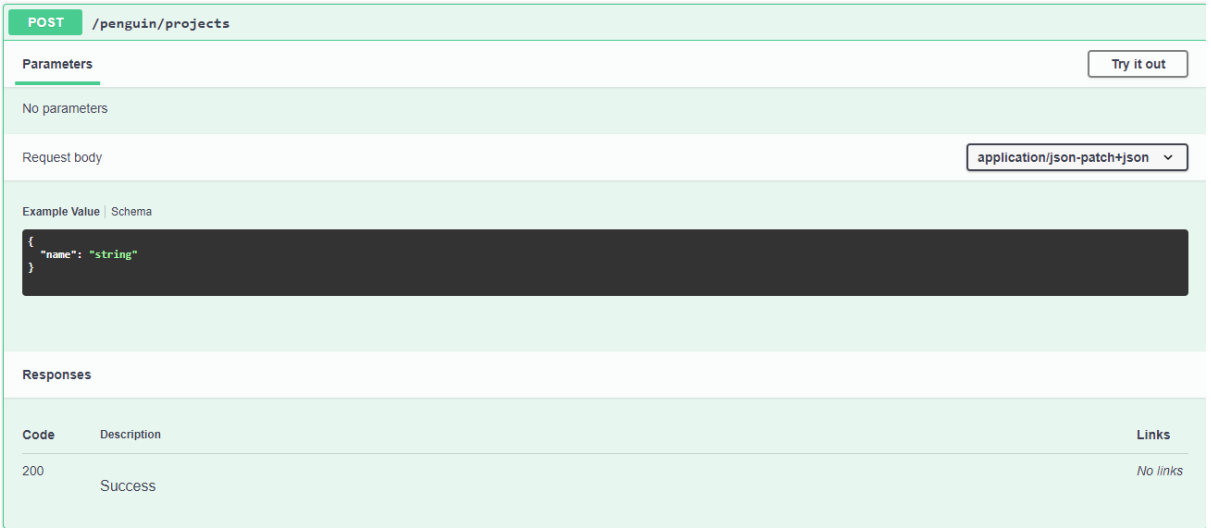
Obtener Usuario



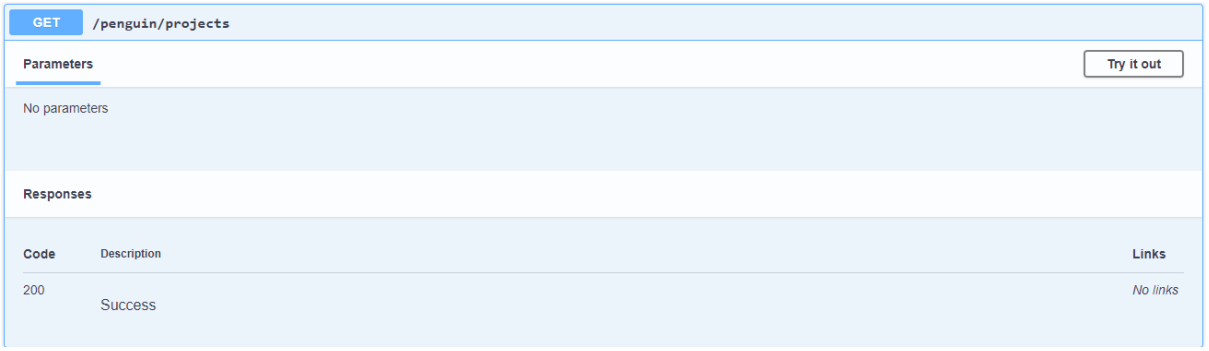
Proyecto



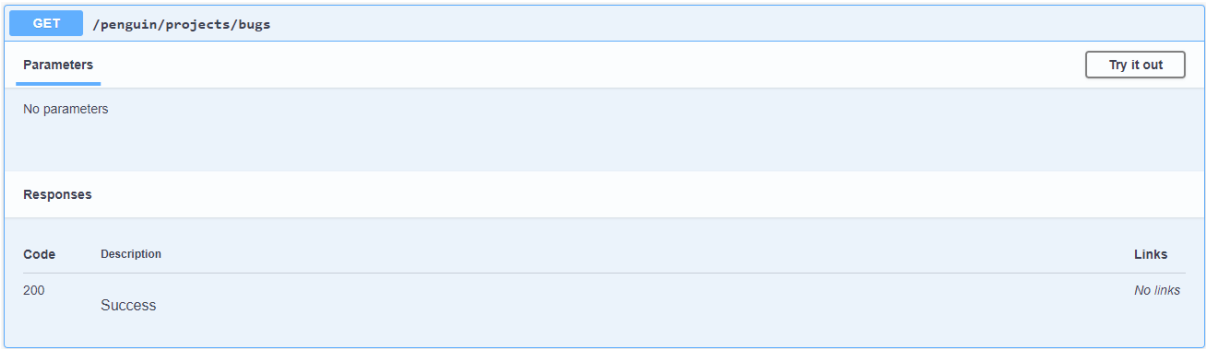
Crear Proyecto



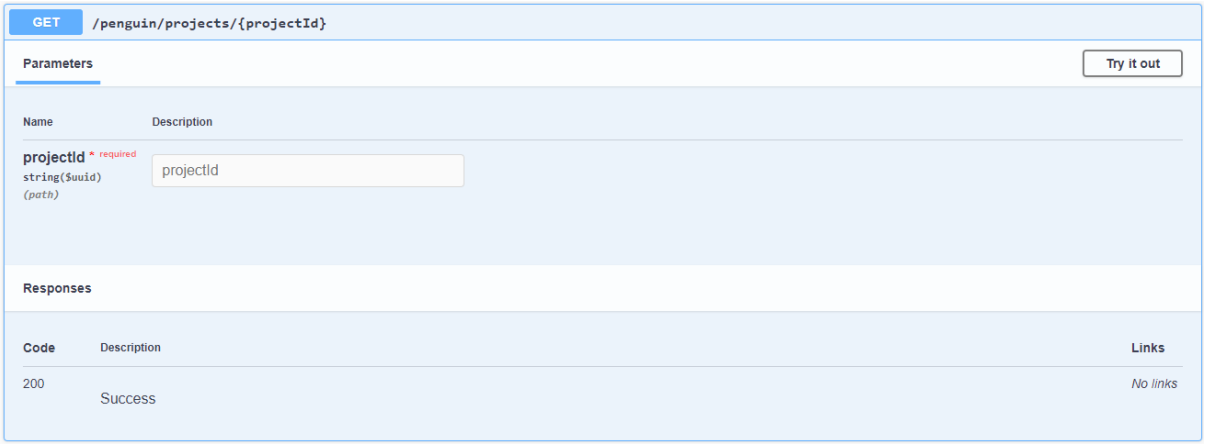
Obtener proyectos



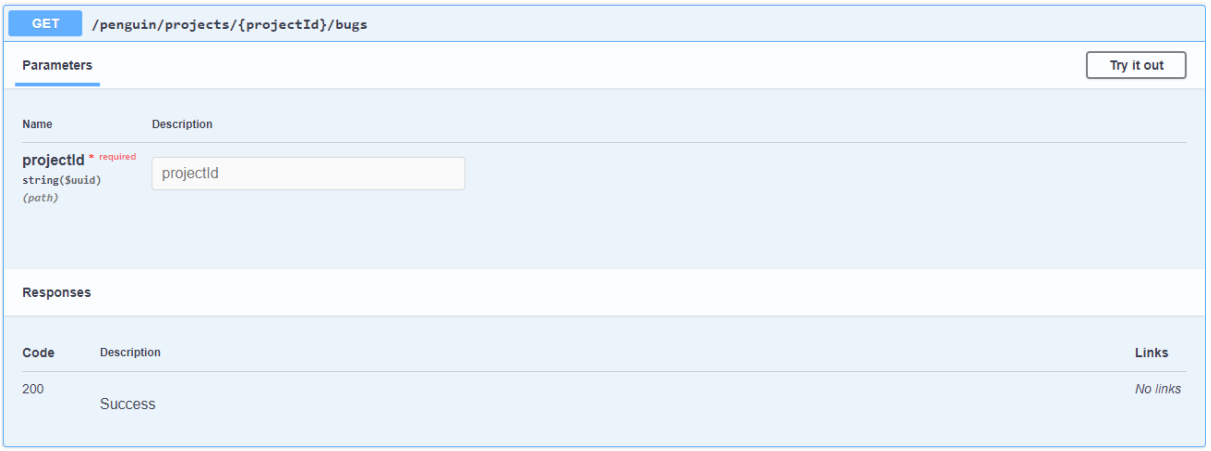
Obtener los Bugs de todos los proyectos



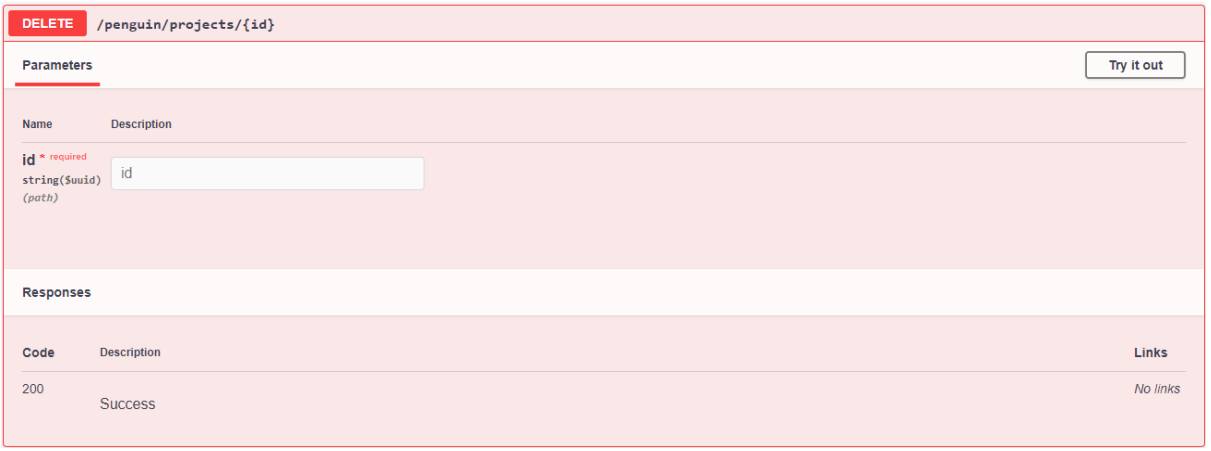
Obtener un Proyecto



Obtener bugs de un proyecto

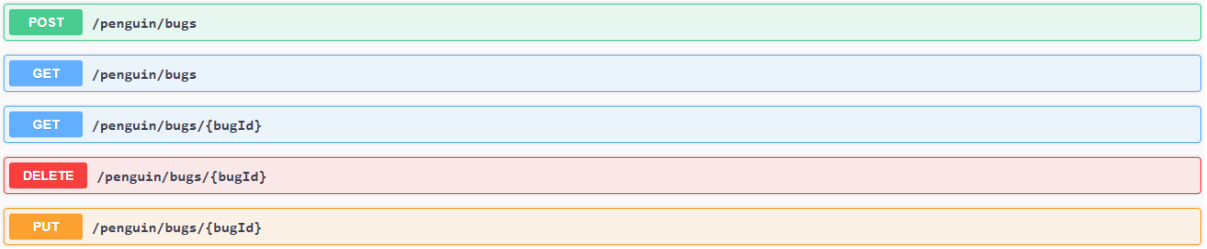


Eliminar un proyecto

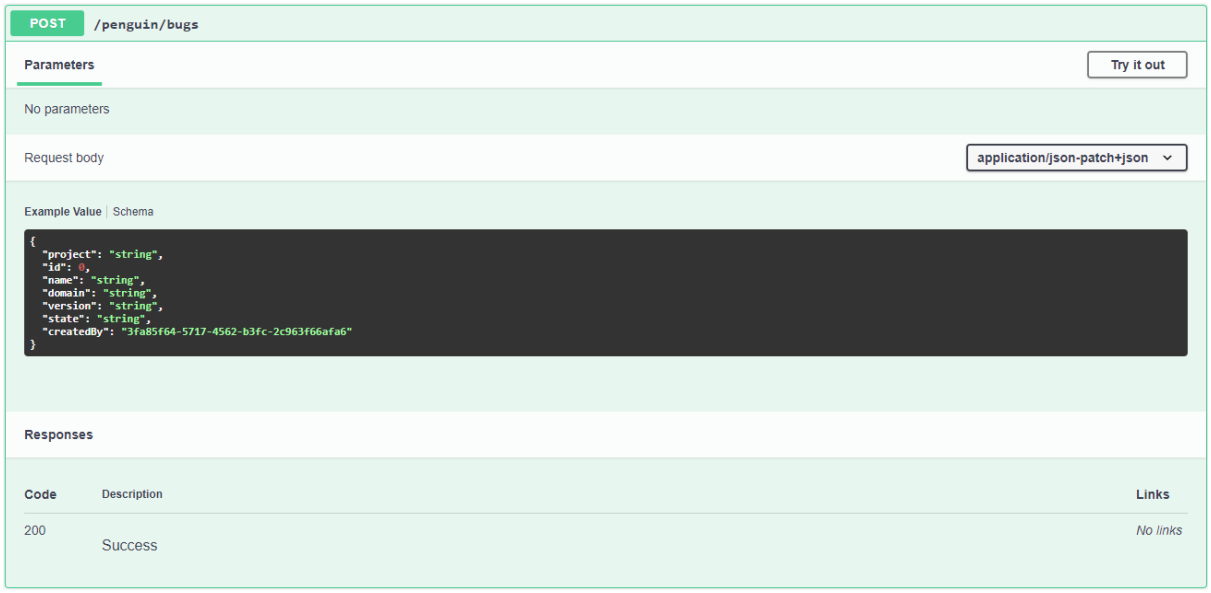


Modificar Proyecto

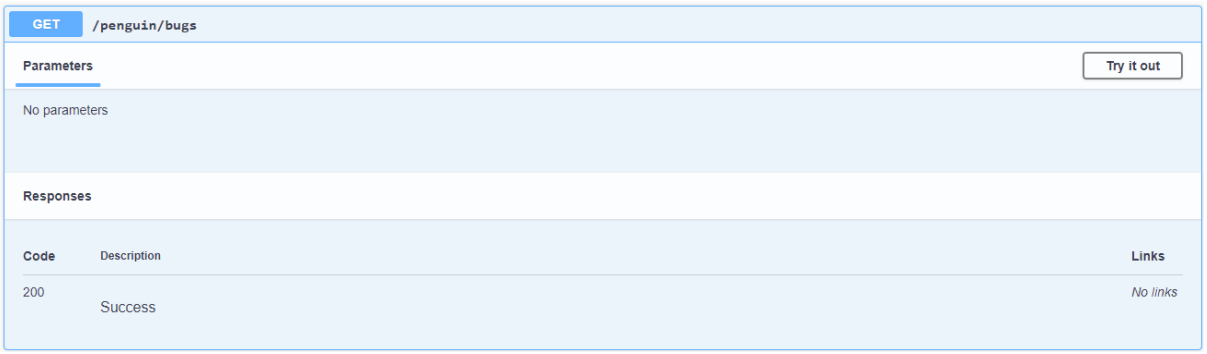
Bug



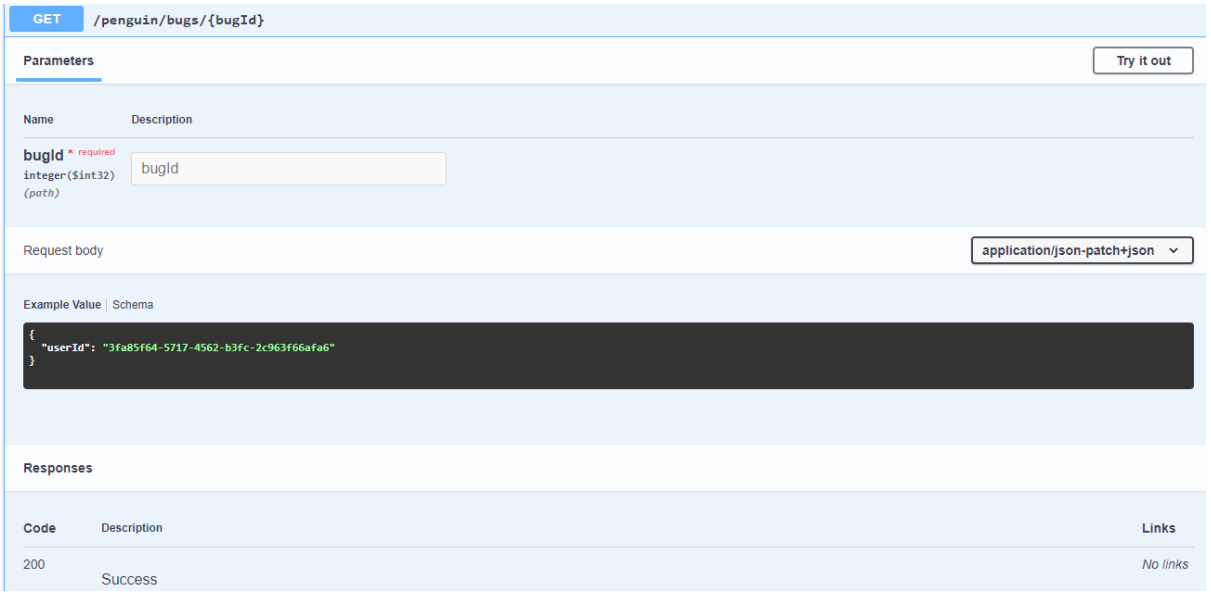
Crear Bug



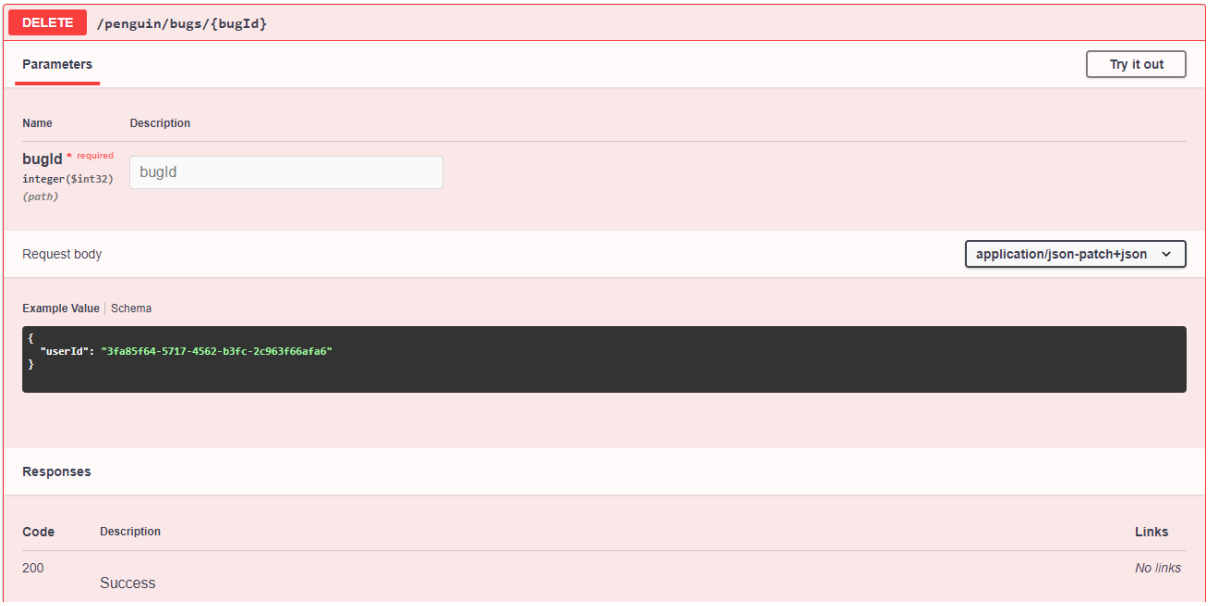
Obtener bugs



Obtener un bug



Eliminar un bug



Modificar un bug

Developer

GET /penguin/developers/{idDeveloper}/bugs

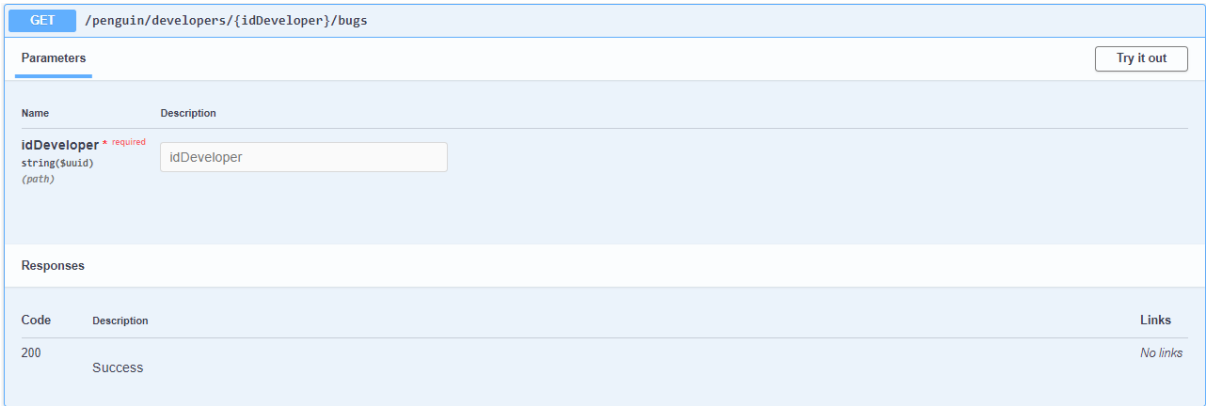
POST /penguin/developers/{idDeveloper}/project/{idProject}

DELETE /penguin/developers/{idDeveloper}/project/{idProject}

GET /penguin/developers/{idDeveloper}/countBugs

PUT /penguin/developers/{developerId}/bugState

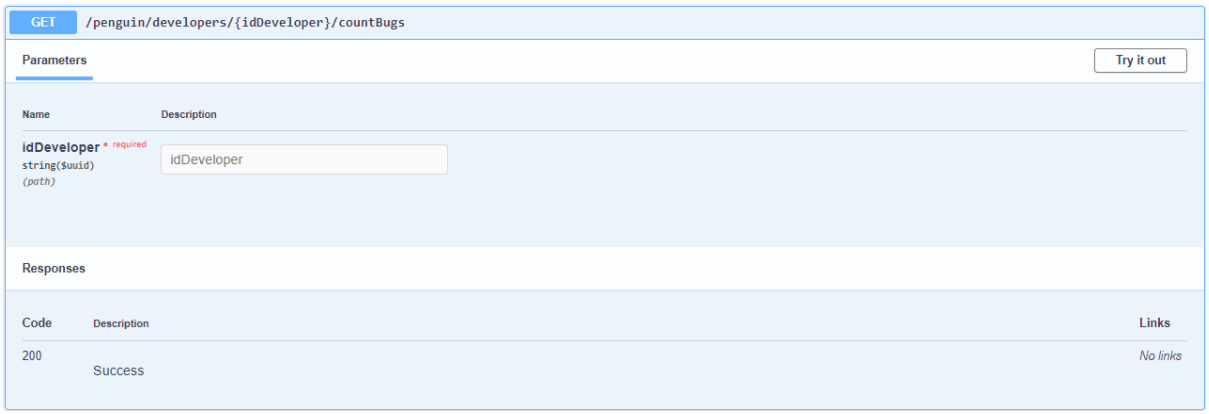
Obtener un bug asociado a un desarrollador



Agregar un desarrollador a un proyecto

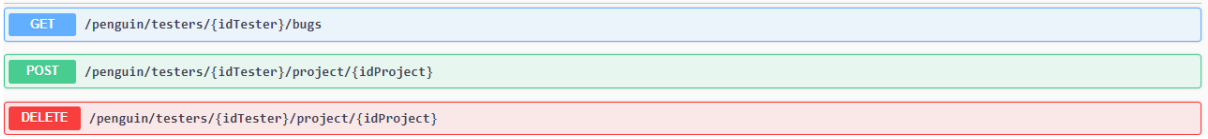
Eliminar un desarrollador de un proyecto

Obtener cantidad de bugs resueltos por un desarrollador

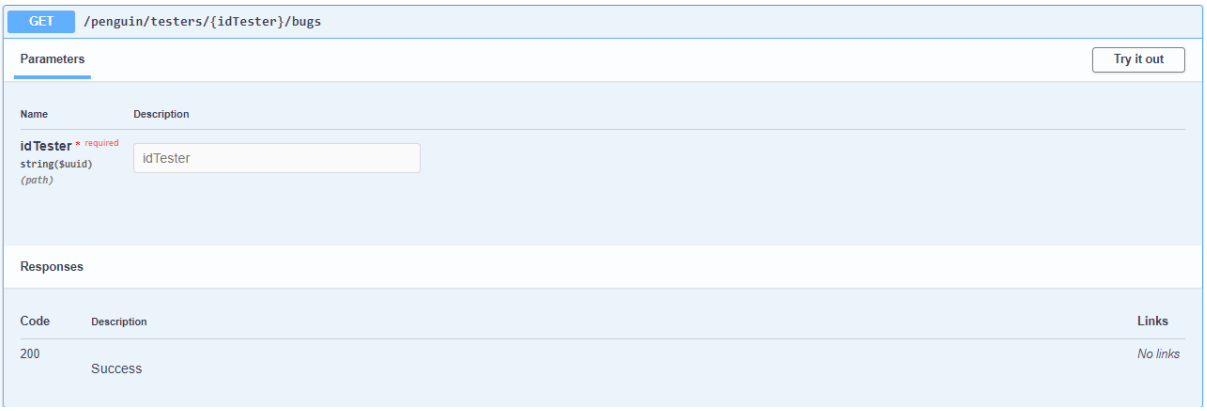


Actualizar el estado de un bug con un desarrollador

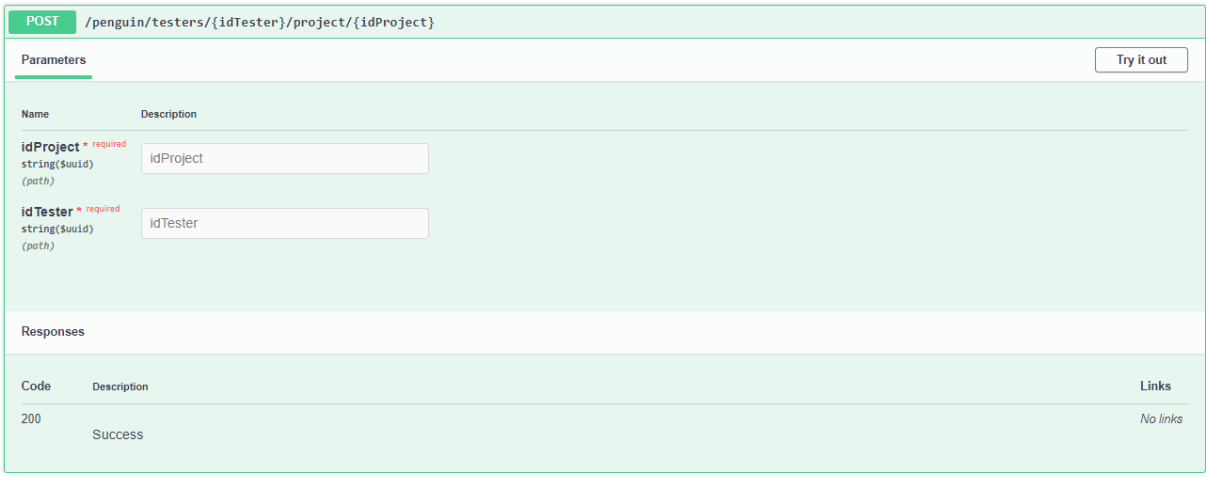
Tester



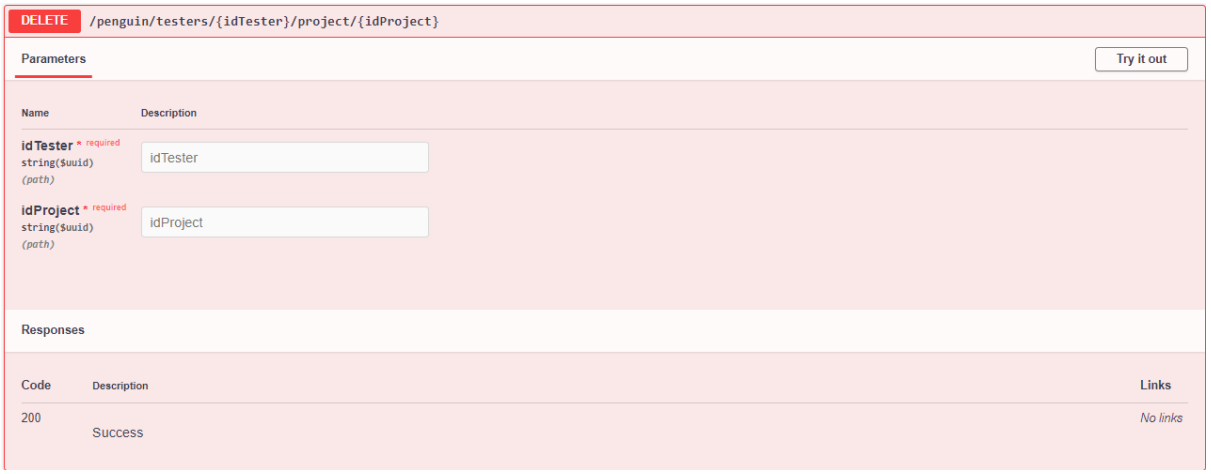
Obtener bugs asociados a un tester



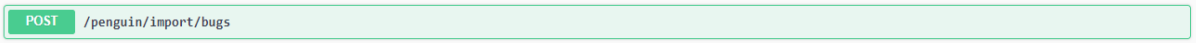
Agregar un tester a un proyecto



Eliminar un tester de un proyecto



Imports



Crear import de bugs