Universidad ORT Uruguay

Obligatorio 2

Diseño de Aplicaciones 2

Docente: Gabriel Piffaretti

Agustín Ferrari 240503

Ivan Monjardin 239850

Francisco Rossi 219401

ORT-DA2/240503_219401_ (github.com)

2021

Índice:

Evidencia del diseño y especificación de la API

1. Descripción General:

Para la creación de los endpoints en el proyecto del WebApi se siguió los criterios de REST, por ejemplo no usar verbos en la URI dado que cada acción ya tiene un verbo que indica que tipo de resultado se espera al ejecutar el endpoint.

Pudimos implementar todas las funcionalidades requeridas generando un endpoint asociado para poder ejecutar el endpoint desde el frontend.

Descripción y justificación de diseño de la API:

Criterios REST

Creamos una interfaz uniforme, basando nuestros endpoints, en los diferentes recursos que cuenta el sistema en forma de objetos de dominio del sistema.
Para todos los casos que se necesite mandar un body o recibir objetos, está indicado como se debe parser (JSON) para mapearlo en el sistema a una entidad concreta.
No se acopla a la tecnología del código .NET para que si se desea cambiar de implementación las URIs ya estén preparadas para aplicarse en otras diferentes.

Para continuar con criterios de REST, no se maneja el concepto de estados, esto indica que toda la información solicitada por un request, debe estar contenida en sí misma, por diferentes mecanismos (body, header, query, etc.) y también habilita que dos llamadas consecutivas se puedan realizar sin problemas.

Se simuló la relación entre cliente servidor, porque siempre se trabajó desde localhost, pero el sistema debería soportar realizar una comunicación entre cliente y servidor, cambiando algunas configuraciones.

Se utilizaron los principales métodos HTTP, Post, Put, Patch, Get y Delete, para indicar que tipo de acción realiza el endpoint, evitando agregar verbos a las URIs. También se siguió la recomendación de utilizar los recursos en plural en toda acción.

Mecanismos de autorización

Para resolver este problema de permisos dentro del sistema, utilizamos básicamente dos herramientas que provee .NET sin recurrir a JWT que no tuvimos tiempo de analizarlo.

La primera fue GUID, el cuál genera un número random y es utilizado en la industria para crear un token de sesión al usuario. Nos pareció chequear además que no pueda ser repetible este número, por lo que le concatenamos como string al número generado, la fecha actual del sistema. Esto nos da una certeza de que no se va a repetir el token dentro del sistema.

La segunda, fue IAuthorizationFilter, una interfaz que provee AspNetCore para poder aplicar un filtro en cada request que se realice si el filtro está activado. En nuestro caso se utilizó para confirmar que el cliente realizando la request, tenga permisos para la misma, según su rol.

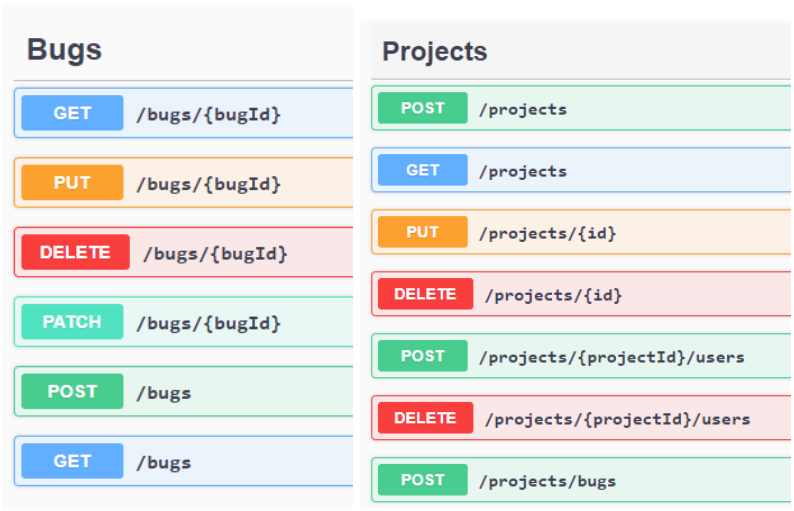
Códigos de estado

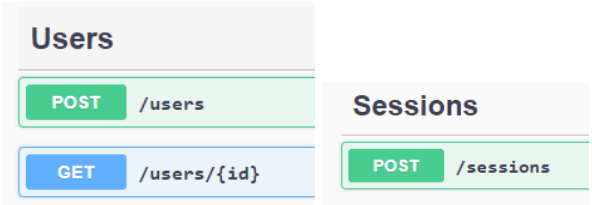
Nuestro sistema maneja distintos códigos de estado según la situación de cada acción.
Los código se manejaron en los filtros, que son los encargados de asegurarse que el usuario tenga el rol correspondiente, pero también de cachear las excepciones de algún método en específico. Para no preguntar por cada excepción del dominio, por ejemplo que el nombre de proyecto sea válido, el mail del usuario, etc. se creó una clase padre que abarca todos los casos pero sigue mostrando los mensajes específicos.
En este filtro se devuelve un 500 por defecto si hubo un error y el sistema no lo supo manejar.
409 se devuelve en el caso de que se quiera crear un Proyecto, pero el nuevo nombre ya se encuentra en la base de datos. Si la sintaxis que el cliente envía no es correcta se muestra un 400, por ejemplo si faltan campos de algún objeto en el JSON.
El último código que maneja este filtro es 403, indicando que el servidor entendió pero se rehúsa a realizar la acción. Creemos que este código debió ser 409 pero no nos dió el tiempo de cambiarlo.

El filtro de autorización, maneja dos códigos, 401 y 403, uno para indicar que no tiene permisos, mostrando como mensaje los roles autorizados, y el otro para indicar que no tiene permisos, y no se puede autenticar.

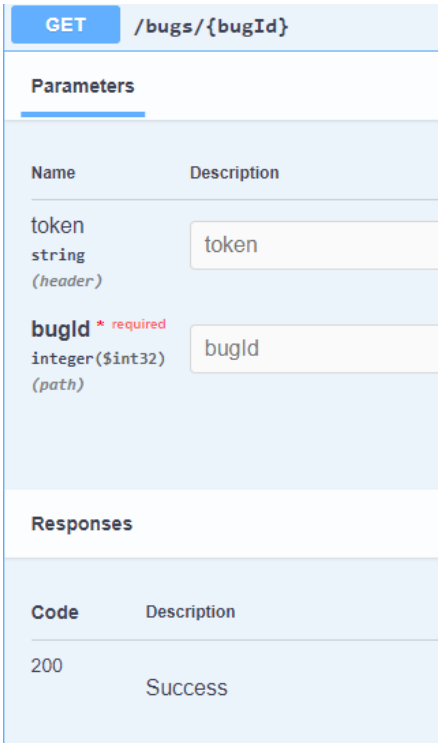
Resources

Se presentará por recurso los distintos endpoints generados, luego se detallará una breve descripción de los mismo.

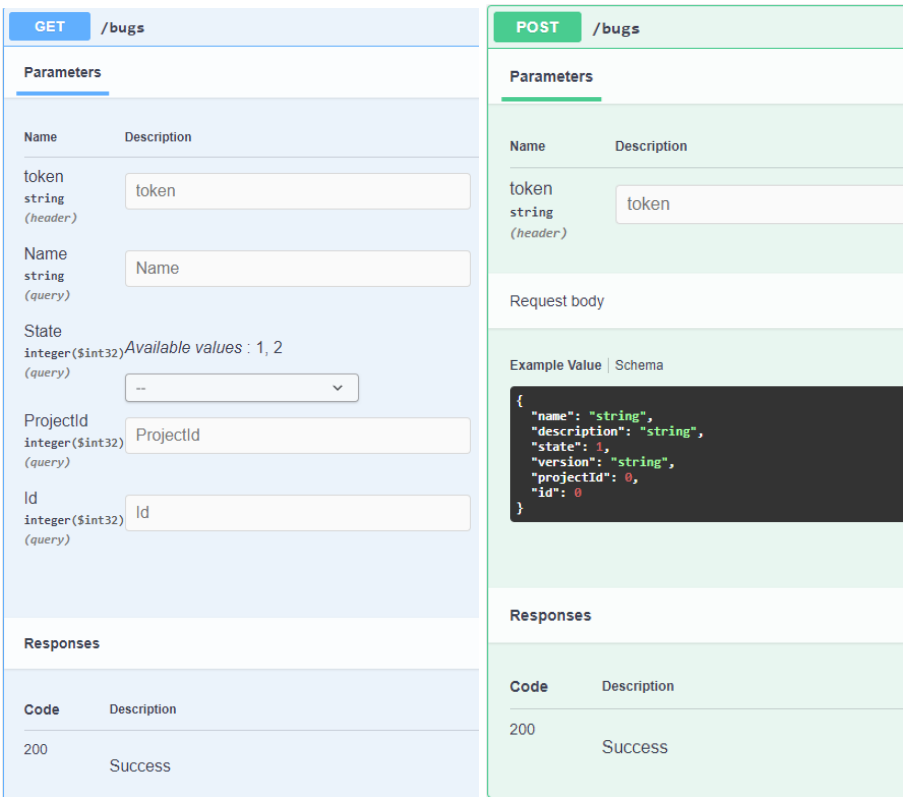




Bugs



Se puede observar que para ambos se utiliza la misma URI, algo que es recomendable para REST, sin embargo uno recibe un un body porque es el que va a agregar un bug y ambos indican un bug en particular con el id para traer o actualizar. Se puede distinguir por el verbo que está delante. Luego Delete y Patch son iguales que el Get, pero con el verbo, se puede distinguir la acción. El Patch solo actualiza el estado de Activo a Done.

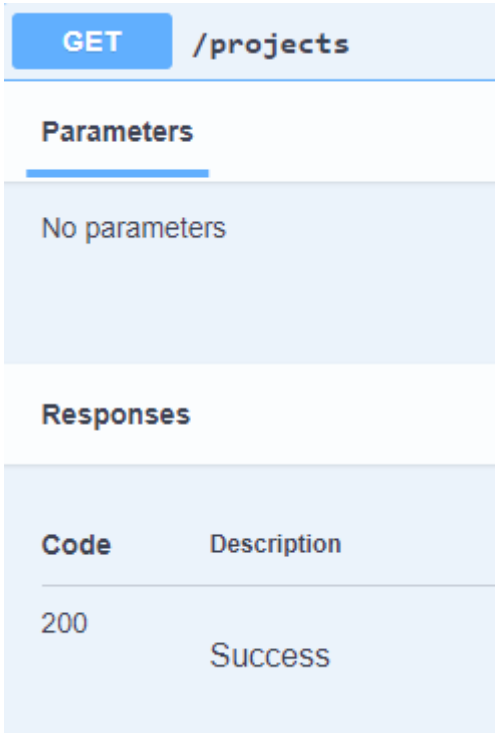


Estos endpoints también utilizan la misma URI entre sí, pero se les envía diferentes parámetros, el Get sirve para traer todos los bugs, y si se quiere se puede añadir filtros de búsqueda. El post, se debe de incluir un JSON correcto para poder agregar un objeto bug a la base de datos.

Se puede observar que todas las acciones del bug requieren un token para validar el rol del usuario logueado, para cada acción.

Projects

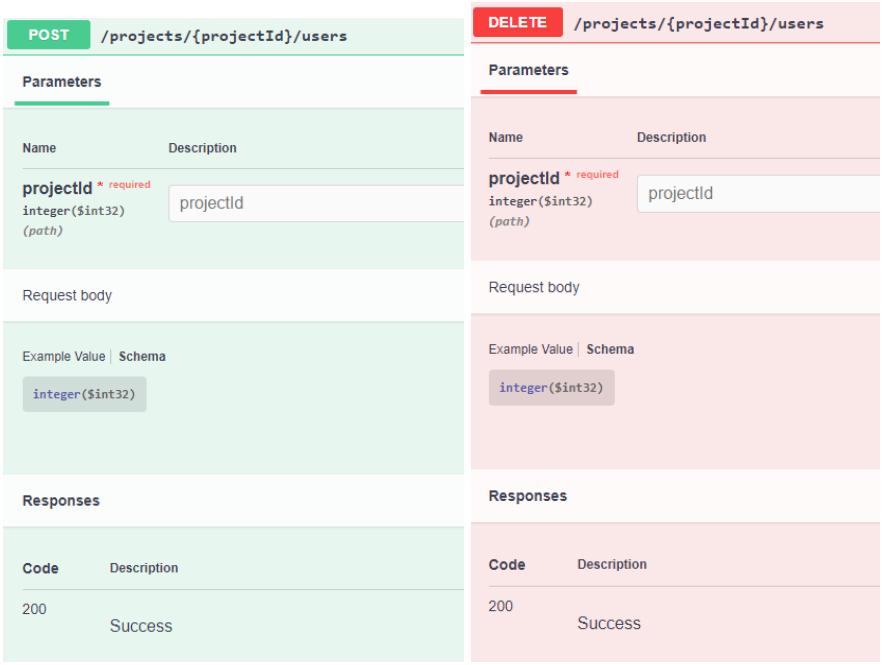
Este recurso es el que contiene más relaciones con los otros objetos del dominio. Por lo que se necesitó diferentes URIs, para realizar acciones más específicas.



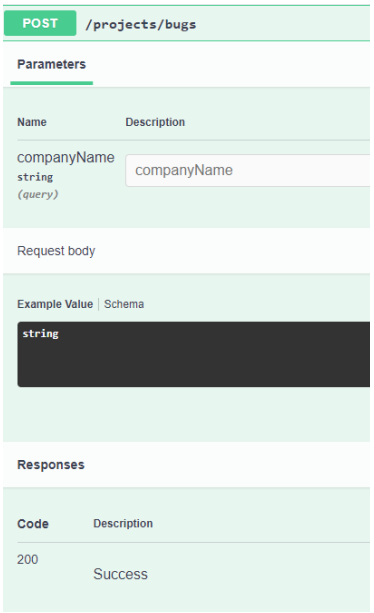
Para crear un proyecto, se puede hacer solo ingresando un nombre.

El get no precisa ningún parámetro, trae todos los proyectos del sistema con la cantidad de bugs en cada uno.

Para actualizar o borrar un proyecto específico (id) se utiliza la misma URI, pero el Put, actualiza el nombre del proyecto en la base de datos.



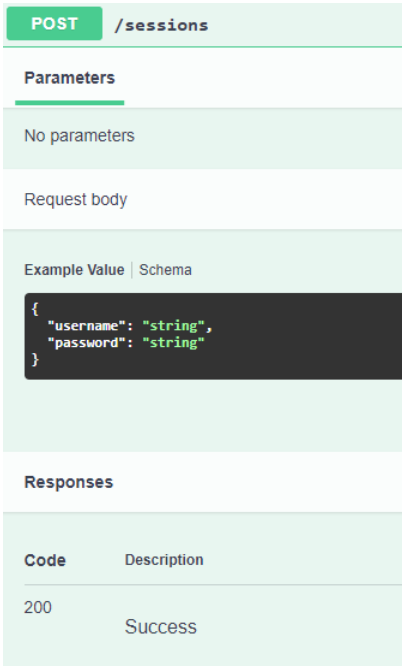
Estas acciones son para agregar o borrar un usuario a un proyecto en particular. Para eso, decidimos respetar la recomendación de no sobrepasar los 3 niveles de la URI y pasar un integer en el body para indicar el UserId a agregar en el proyecto.



Esta URI no es compartida, dado que creamos una sola, para crear proyectos en el sistema, ya sea por xml, o texto con cierto formato.

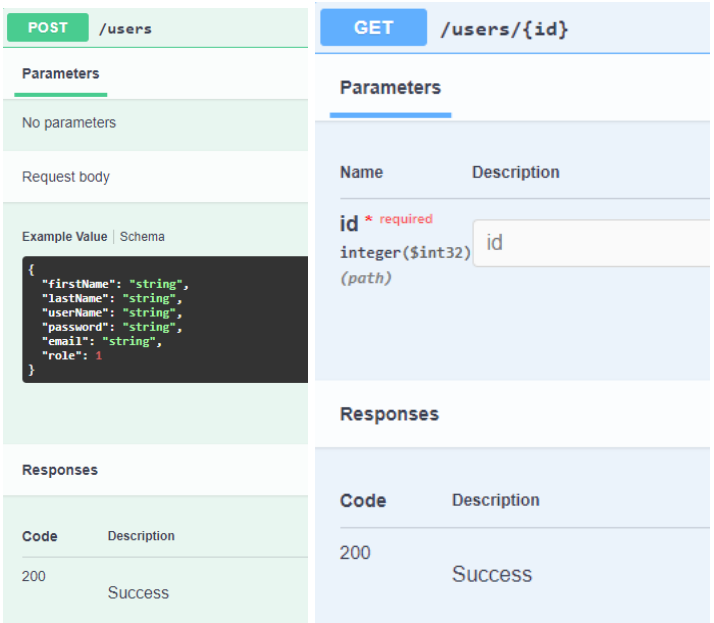
Aunque no se muestre, todos ellos requieren un token en el header, para validar que estas acciones solo las puede realizar un Admin.

Sessions



Se utiliza este endpoint para subir un inició de sesión al sistema, que determinará si el usuario puede acceder al sistema, con un username y password. Se carga un token en la base de datos si las credenciales son correctas (no está implementado el chequeo en este obligatorio), con la finalidad de autenticar su rol para diferentes acciones, mientras el usuario haya ingresado sesión.

Users



El único rol que debe agregar un usuario es el Admin (token solicitado por header), donde se ingresará por medio del body, un JSON, para mapearlo a una entidad del sistema.
Aunque no se pedía, nos pareció interesante poder obtener un user por su Id, pero estas dos acciones solo tiene permisos de realizarlo, el rol de Admin.