Universidad ORT Uruguay

Facultad de Ingeniería

Evidencia del diseño y especificación de la API para primer obligatorio de Diseño de Aplicaciones 2

Emiliano Yozzi - 230710

Franco Thomasset - 239611

Tutores:

Juan Irabedra,Santiago Tonarelli, Francisco Bouza

2022

Enlace al repositorio:ORT-DA2/239611-230710 (github.com)

Declaración de autoría:

Nosotros, Franco Thomasset y Emiliano Yozzi, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

• La obra fue producida en su totalidad mientras realizábamos Diseño de aplicaciones 2;

• Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;

• Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;

• En la obra, hemos acusado recibo de las ayudas recibidas;

• Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;

• Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes

Abstract:

El objetivo de este trabajo es el de representar un Sistema de blogs que permita la creación y gestión de contenido escrito en forma de artículos. La plataforma permite a los usuarios registrarse, crear perfiles y publicar artículos (de manera privada o pública) de forma cronológica que pueden ser comentados por otros usuarios.

Además el sistema ofrece funcionalidades como notificaciones de nuevos comentarios.

Se pueden crear, eliminar y modificar usuarios con roles de Blogger o Administrador.

A alto nivel el sistema está compuesto por una API Rest que permite interacción con nuestro Backend y Bases de Datos relacional.

Utilizamos las siguientes tecnologías:

- Microsoft Visual Studio 2022 Enterprise
- Microsoft SQL Server Express 2017
- Astah UML
- Entity Framework Core 6
- Postman
- ASP.NET Core 6.0

Índice

Características REST de nuestra API

- Utiliza la arquitectura cliente-servidor: los clientes envían solicitudes HTTP al servidor y reciben respuestas en formato JSON.

- Utiliza el protocolo HTTP: utiliza los métodos HTTP estándar, como GET, POST, PUT y DELETE, para interactuar con los recursos.

- Identificación de recursos: utiliza la URL para identificar los recursos, por ejemplo en nuestro UserController, "/api/users" para obtener todos los usuarios, "/api/users/{username}" para obtener un usuario específico y "/api/users/{username}/notifications" para obtener las notificaciones de un usuario.

- Respuestas en formato estándar: responde con un cuerpo en formato JSON que se puede procesar fácilmente por los clientes.

- Utilización de modelos, cada uno de los recursos tienen representaciones para las peticiones y las respuestas(modelos in y out), estas últimas características nos permiten la Interfaz Uniforme

- Sistema de capas: Como podemos ver en la documentación de nuestro diseño, en nuestro diagrama de componentes podemos ver que se tiene un sistema de capas, donde cada componente no pueda ver ninguna otra capa, menos la inmediatamente posterior, por lo tanto nuestra API no depende de ninguna forma de la base de datos

- Stateless: Toda la información necesaria se envía en la petición, el Servidor no toma ventaja de otra información guardada previamente

- Cacheable: El cliente puede guardarse la información necesaria para no tener que hacer una nueva request al servidor

Endpoints de nuestra API

POST: No se puede crear el mismo recurso más de una vez

GET: Es idempotente

PUT: No es idempotente

DELETE: No se si es idempotente

Códigos HTTP:

200 - OK

400 - Bad request

401- Unauthorized

404 - Not Found

500 - Internal Server Error

Mecanismo de autenticación de requests

Se utiliza un parámetro “Authorization” en el Header que tiene un token que se genera al hacer login

Descripción de los recursos de la API

User:

URL Base: https://localhost:7186/api/users

Resource: User

Endpoints:

Post

Se encarga de la creación de un usuario

Responses:

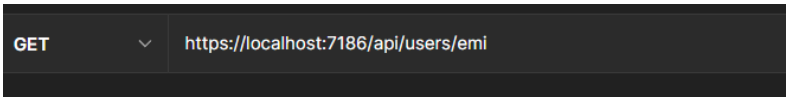
200:ok

400 Bad Request: Username already exists

500: Unexpected error: Invalid Username

500: Unexpected error:Invalid email

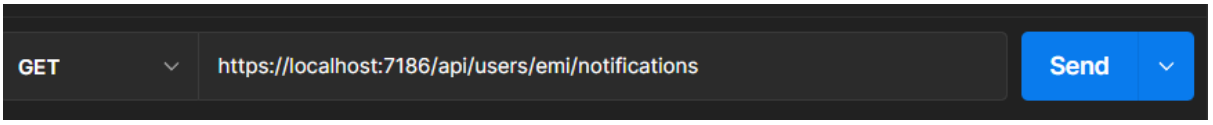
GET



Se encarga de traer un User por username

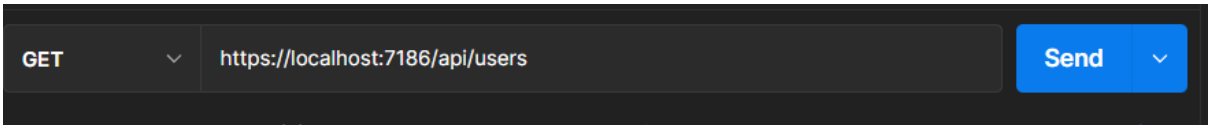
GET

Se encarga de traer las notificaciones

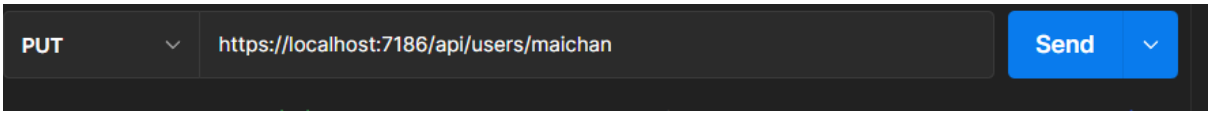


GET

se encarga de traer todos los usuarios si somos el admin

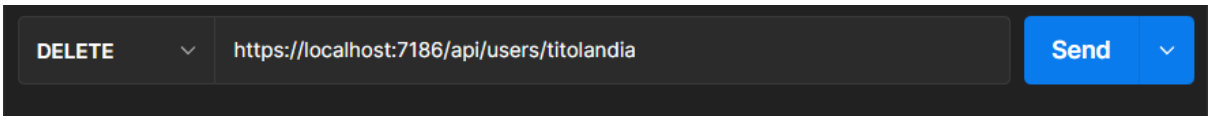


PUT



Le pasamos un username y nos permite modificar un usuario.

DELETE



Le pasamos un username y lo podemos borrar si estamos logueados con el usuario admin

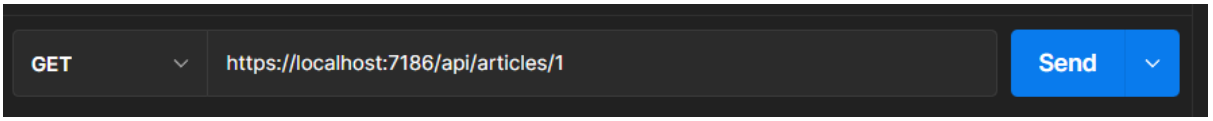
Articles:

URL Base: https://localhost:7186/api/articles

Resource: Article

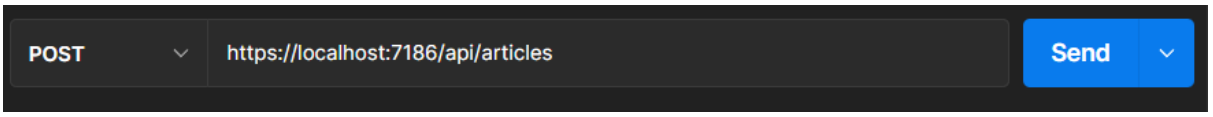
Endpoints:

GET



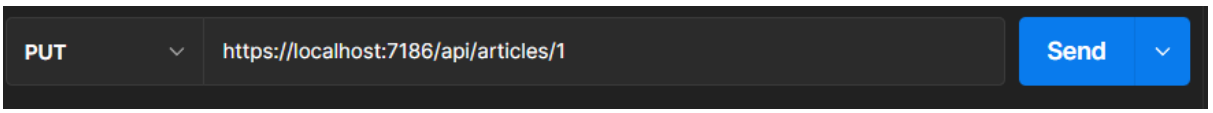
Le pasamos el id del artículo en la ruta y nos la trae.

POST



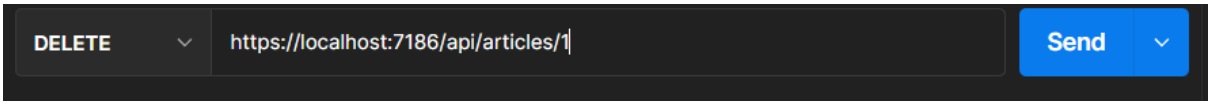
le pasamos en el Body titulo, texto, visibilidad, imagen, template y el nombre del usuario que está creando el artículo

PUT



Le pasamos en la ruta el id del artículo a ser modificado y luego en el body las modificaciones correspondientes.

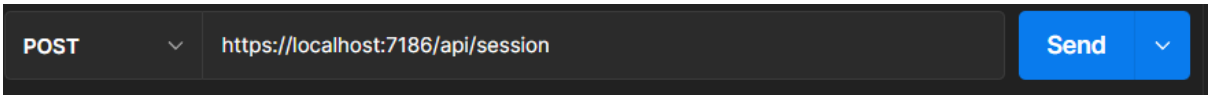
DELETE



Logueados como el admin le pasamos el id del artículo que queremos borrar.

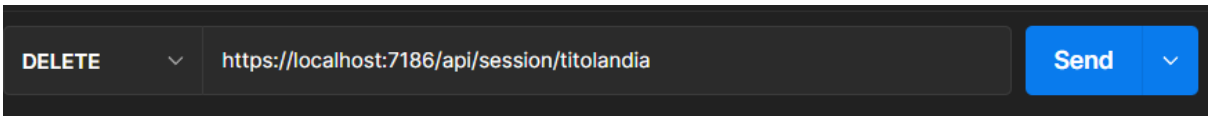
Session

POST



Con esto podemos iniciar sesión

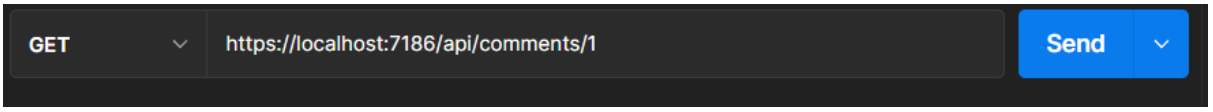
DELETE



Con esto podemos cerrar sesión.

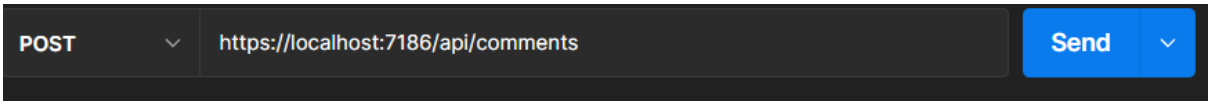
Comment

GET



Nos permite traer un comentario por ID

POST



Nos permite comentar