Universidad ORT Uruguay

Facultad de Ingeniería

Evidencia del diseño y especificación de la API.

Ignacio Loureiro - 191659
Malvina Jaume - 151281

Entregado como requisito de la materia
Diseño de Aplicaciones 2

14 de mayo de 2020

Índice general

1. Criterios seguidos

El siguiente documento ha sido realizado para describir el diseño realizado para solución de la implementación del API REST descripta en la letra del obligatorio entregado.

1.1. Descripción de criterios REST

Para el desarrollo de esta aplicación se tuvieron en cuenta las buenas prácticas aprendidas, los principios REST.

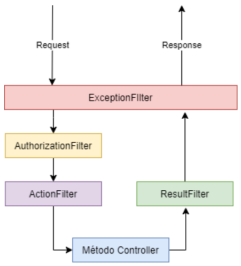
- Separamos los recursos lógicos En esta instancia identificamos 4:
  - Admins (Manejo de administradores)
  - Requests (Manejo de solicitudes)
  - Sessions (Manejo de sesiones)
  - Types (Manejo de diferentes tipos de solicitudes)
- Para cada recurso le definimos las acciones necesarias que deben de realizar usando métodos HTTP. NOTA: En la parte del informe swagger si puede ver la definición de los métodos usados.
- Endpoints en plural
- Generamos documentación con herramienta para que quede entendible por todos
- Utilizamos filtros, para la autenticación.

1.2. Descripción del mecanismo de autenticación de requets

Al tener la necesidad de realizar una autenticación o autorización de los request nos tenemos que asegurar de que el cliente que realiza esta solicitud tenga los permisos necesarios, para esto lo que se decidió aplicar en el solución fue la implementación de un filtro de seguridad en la WebApi.

ASP.NET Web Api proporciona una opción de extensibilidad para la realización del filtro. Los filtros nos permiten la ejecución de código antes o después de un procesamiento de una solicitud HTTP.

El orden de ejecución seria el siguiente:

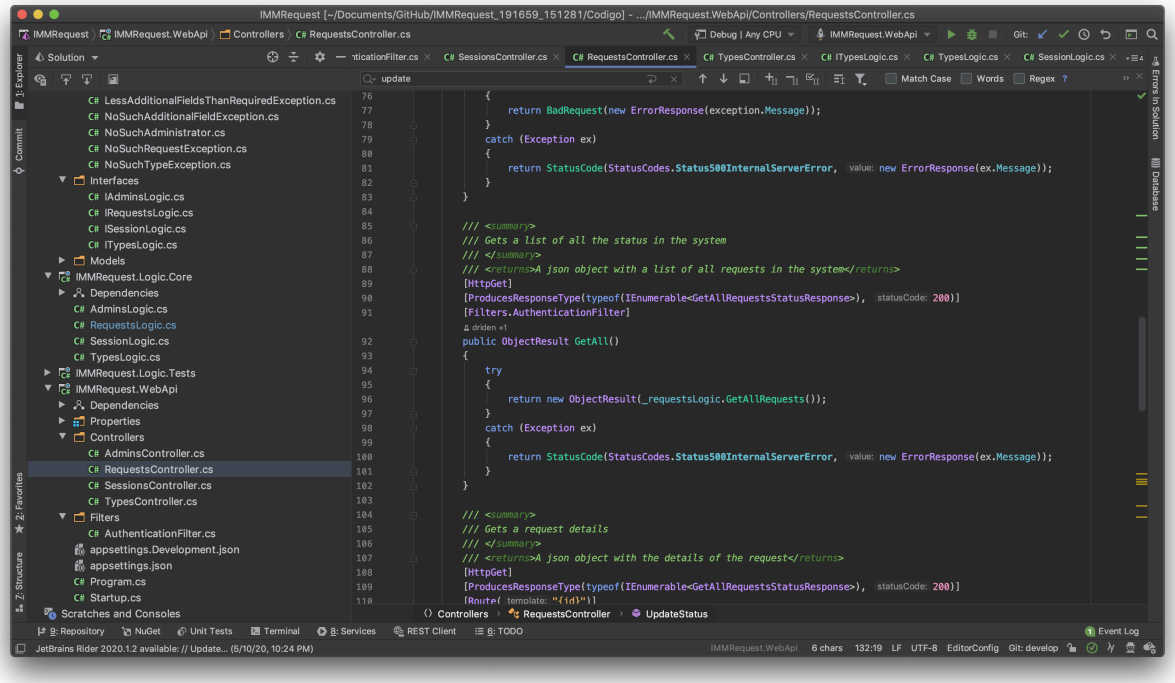


Cada usuario al loguearse generará un token aleatorio. Esta token será el usado para validar que el usuario relacionado a dicho toekn tiene los permisos necesarios para realizar el request.

A grandes rasgos los pasos para la implementación fue:

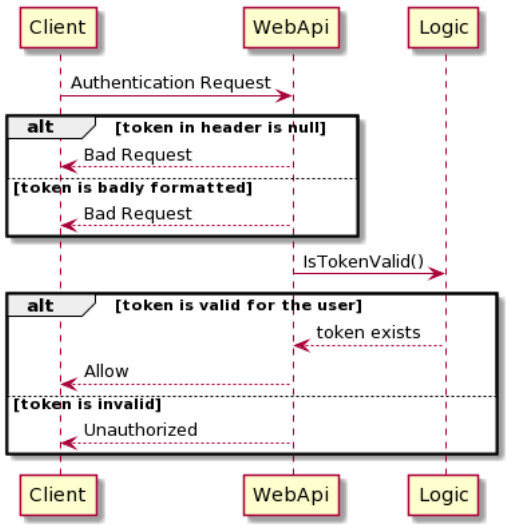
- Dentro de WebApi creamos una clase que representa al filtro **Authentication-Filter** la cual extiende de **Attribute** e implementa la interfaz **IActionFilter**. Esta clase es la responsable de la validación del encabezado que contenga un identificador. Validará que el token es el correcto.
- Se crea una clase **SessionsController** también en el paquete WebApi. Esta clase es la responsable de la comunicación con la lógica.
- Dentro del paquete Logic implementamos la clase que hace posible todo lo necesario para la acción de login, la validación y registro del token en la base de datos.

Para que el filtro se aplique en la request, lo único que es necesario tener es tener el tag en el request que se quiere aplicar. En la siguiente imagen mostramos como se ultiza en la request GetAll colocamos [Filters.AuthenticationFilter]



En resumen, antes de realizar el cada request se va a validar que el token generado previamente en el login, sea válido.

A continuación presentamos el diagrama de secuencia de login que genera el token.



1.3. Descripción general de códigos de error

Para esta instancia definimos los códigos de error:

- 200: Elemento creado o modificado con éxito.
- 400 : Falla alguna validación en el body del request.

- 404 : Falla cuando se intenta borrar o actualizar un recurso inexistente
- 500 : Falla en el servidor

2. Descripción de los resources de la API.

2.1. URL base.

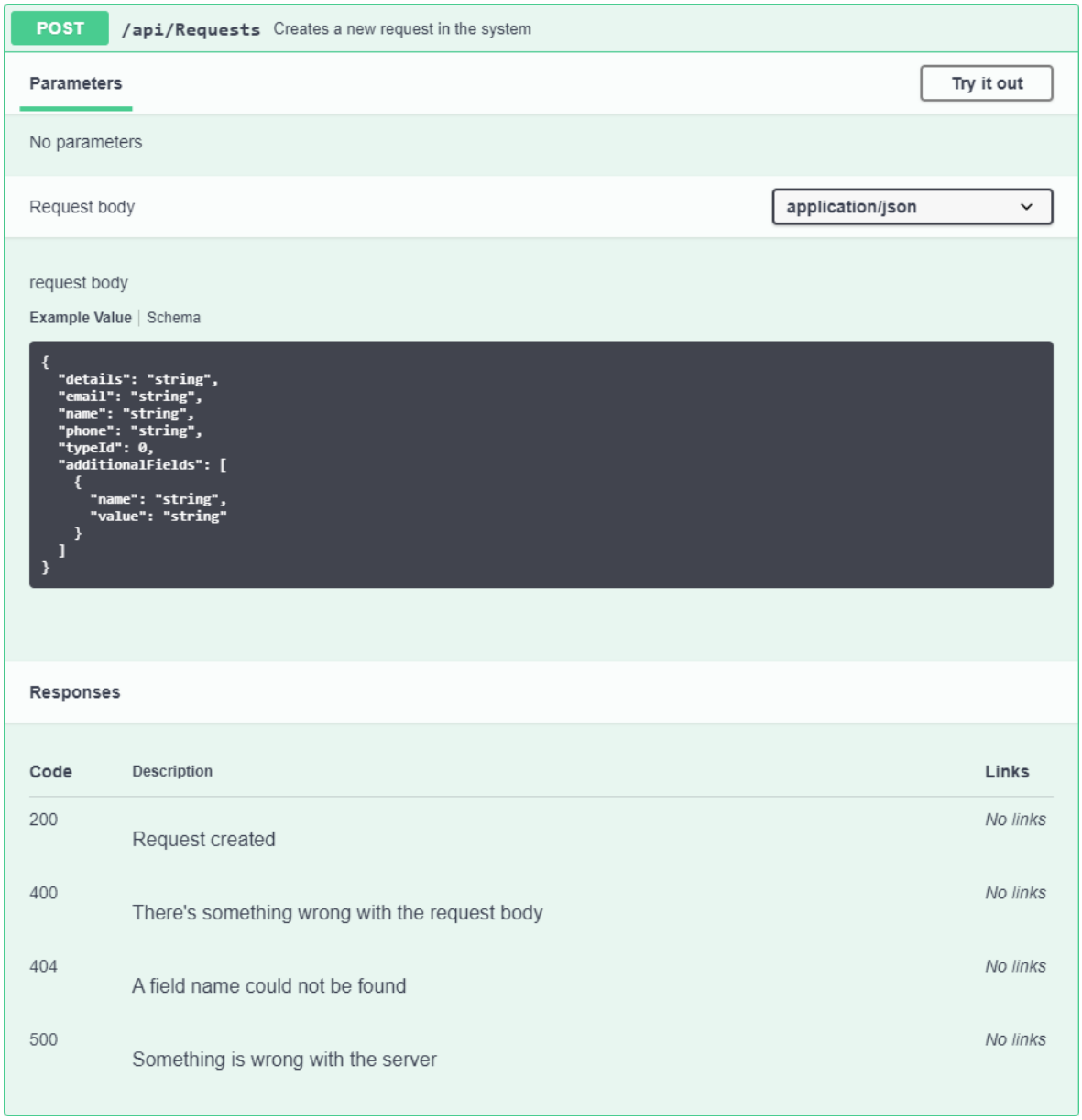
La url base de nuestra aplicación está formada por el hostname, en nuestro caso, con un desarrollo local **http://localhost:5000** y cada recurso está prefijado por la palabra "api". Por lo tanto la url base de nuestra WebApi, corriendo localmente es **http://localhost:5000/api**

La api fue documentada a medida que se desarrollaba utilizando swagger. para utilizar swagger tuvimos que:

- descargar el paquete de swashbuckle, una implementación de Swagger para .NET Core https://www.nuget.org/packages/Swashbuckle.AspNetCore/
- configurar el startup, siguiendo la guia de Microsoft https://www.nuget.org/packages/Swashbuc
- para poder generar metadata de la Api es necesario decorar los métodos con sintaxis conocida por Swagger.

```
/// <summary>
/// Creates a new request in the system
/// </summary>
/// <param name="request">request body</param>
/// <response code="200">Request created</response>
/// <response code="400">There's something wrong with the request body</response>
/// <response code="404">A field name could not be found</response>
/// <response code="500">Something is wrong with the server</response>
[HttpPost]
public ActionResult CreateRequest([FromBody] CreateRequest request)
```

En la imagen anterior se puede ver como está marcado nuestro endpoint, lo que ocasiona que Swagger pueda presentar la metadata de nuestra operación en el navegador.



Una vez levantada la app se puede ingresar en localhost:5000/swagger/index.html