Universidad ORT Uruguay

Facultad de Ingeniería

Primer obligatorio

Diseño de Aplicaciones II

Entregado como requisito para la obtención del título de

Licenciatura en Sistemas

Guillermo Diotti - 285578

Juan Peyrot – 275076

Nicolás Toscano - 220264

Profesores:

Daniel Acevedo

Pablo Geymonat

https://github.com/IngSoft-DA2/285578-220264-275076

2024

Índice

Criterios seguidos para asegurar que la API cumple con los criterios REST

REST, o *Representational State Transfer*, es un estilo arquitectónico que define una serie de principios y restricciones para el diseño y construcción de APIs. Como equipo, nos encargamos de cumplir varias de estas características, las cuales aseguran una interacción eficiente y comprensible entre clientes y servidores. A continuación, se detallan los criterios clave más relevantes:

Interfaz Uniforme

Uno de los pilares de REST es la definición de una interfaz uniforme para acceder a los recursos de la API. Esta interfaz se estructura en torno a los verbos HTTP fundamentales: GET, POST, PUT y DELETE. En nuestra solución, nos encargamos de que cada recurso tenga un controlador específico, en el cual se implementan estos verbos de manera adecuada conjuntamente de su respectiva URI para satisfacer las diferentes necesidades de interacción. Esto asegura que los usuarios de la API puedan manipular los recursos de manera consistente y predecible, utilizando los métodos estándar para operaciones como la obtención, creación, actualización y eliminación de datos (CRUD).

Basado en Recursos (URIs)

Otro principio clave de REST es que la API debe estar basada en recursos, los cuales son identificados de manera única a través de URIs (*Uniform Resource Identifiers*). En nuestra solución, los URIs se definen de forma clara y concisa, utilizando nombres de recursos en plural, con sustantivos, y limitando la profundidad de los niveles a tres o menos (incluyendo el propio recurso). Esto facilita la comprensión y el uso de la API por parte de los desarrolladores, ya que permite una navegación sencilla y un acceso intuitivo a los recursos. Seguir un enfoque estándar y coherente en la definición de los URIs es fundamental para que los consumidores de la API puedan interactuar con ella de forma clara y eficiente.

Stateless (Sin Estado)

La arquitectura REST exige que las interacciones entre el cliente y el servidor sean stateless, es decir, sin estado. Esto significa que cada petición que hace el cliente debe contener toda la información necesaria para que el servidor pueda procesarla, sin depender de datos almacenados en interacciones previas. En nuestra implementación, esto se traduce en que la API no mantiene la sesión del usuario entre solicitudes. Por lo tanto, cada vez que un cliente interactúa con el servidor, debe proporcionar su propia autenticación en los headers de la request. Esta característica garantiza que las peticiones sean independientes y que el sistema sea más escalable y resistente, y ayuda a que, entre otras cosas, a que el código de los servicios sea mas sencillo y mantenible, pues se les inhibe de la responsabilidad de mantener contexto.

Protocolo Cliente-Servidor

Otro de los principios fundamentales de REST es la separación entre cliente y servidor. El cliente no necesita preocuparse por cómo el servidor gestiona el almacenamiento o procesamiento de los datos, mientras que el servidor no se ocupa del manejo de la interfaz de usuario ni del estado del cliente. En nuestro caso, esta separación es evidente: la API puede ser consumida por cualquier tipo de cliente, como Postman, Insomnia u otras aplicaciones web o clientes HTTP, y su único propósito es procesar las solicitudes que recibe y devolver la respuesta adecuada, sin importar qué cliente la esté utilizando. Este enfoque garantiza una mayor flexibilidad, permitiendo que diferentes clientes interactúen con la API sin necesidad de modificar su estructura.

Mecanismo de autenticación de requests

AuthenticationFilterAttribute

La clase *AuthenticationFilterAttribute* tiene como función principal verificar que las solicitudes a la API incluyan un token de autenticación válido en el encabezado (header) de "Authorization". Este proceso es esencial para asegurar que solo los usuarios autenticados puedan acceder a los recursos protegidos de la API.

Cuando se recibe una solicitud, la clase examina el token presente en el encabezado de la autorización. Este token debe estar asociado a un usuario registrado en el sistema, con una sesión activa (es decir, que tenga un token asignado). Si el token es válido y pertenece a un usuario, este será autorizado para continuar y acceder a los recursos que requieren autenticación.

Si, por el contrario, el token está ausente, es incorrecto, o no se asocia con ningún usuario, la API responderá con un código de estado HTTP 401 (Unauthorized). Esto indica que el usuario no está autenticado y, por lo tanto, no tiene acceso a los recursos solicitados.

AuthorizationFilterAttribute

La clase *AuthorizationFilterAttribute* es una subclase de *AuthenticationFilterAttribute*, lo que significa que hereda su funcionalidad, pero además añade una capa adicional de control. Esta clase introduce un atributo para inspección, que representa el código de permiso específico requerido para llevar a cabo ciertas operaciones en la API.

Cuando se ejecuta este filtro, primero se llama al método *OnAuthorization* de la clase padre para validar la autenticación del usuario y verificar que el token es válido. Una vez que se confirma la autenticación, la clase *AuthorizationFilterAttribute* se encarga de verificar si el usuario cuenta con el permiso necesario para realizar la operación solicitada. Este permiso se especifica en el atributo *Code*, el cual se compara con la lista de permisos del rol el usuario.

Si el usuario tiene el permiso adecuado (es decir, si su lista de permisos contiene el código requerido), será autorizado para realizar la operación y podrá proceder. Si no cuenta con

dicho permiso, la API devolverá un código de estado HTTP 403 (Forbidden), lo que indica que, aunque el usuario está autenticado, no tiene los permisos necesarios para realizar la operación solicitada.

NonAuthenticationFilterAttribute

A vistas de que existen requerimientos funcionales que solicitan que los usuarios *no autenticados* puedan crear una cuenta y loguearse, no podíamos usar los filtros anteriores para este propósito, pues estas requests no contienen datos de autorización, debido a que los usuarios no tienen una token válida para realizar operaciones al momento, lo que les imposibilitaría usar estas funcionalidades. Dejar estas funciones sin ningún filtro aplicado tampoco era opción, ya que en ese caso también podrían accederlas los usuarios autenticados, lo que contradecería el requerimiento.
A modo de solución, decidimos implementar un filtro custom siguiendo la documentación oficial de Microsoft como guía (https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-8.0), y el resultado fue implementar un filtro donde se excluyera el acceso a las funcionalidades listadas previamente a aquellos usuarios que hagan una petición con un header de autorización válido (es decir, usuarios autenticados).
A modo de síntesis, podemos decir que este filtro aplica la lógica inversa que los anteriores mencionados, ya que se encarga que solo los usuarios no autenticados puedan usar las funcionalidades de login y registro a la plataforma.
La justificación de la elección de un filtro personalizado para esta problemática, también se basa pensando de cara al futuro, ya que en el frontend de la aplicación las sesiones de usuario si deben mantenerse, y esto impediría que usuarios actualmente logueados en esta parte de la aplicación puedan registrarse o iniciar sesión cuando ya tienen una sesión activa, obligándolos a desloguearse para acceder a estas funciones (es decir, pasar a ser *usuarios no autenticados*).

Descripción general de códigos de estado

Códigos 1xx y 3xx: Respuestas Informativas y Redirecciones

Los códigos de estado de la familia 1xx corresponden a respuestas informativas. Estos códigos indican que el servidor ha recibido y está procesando la solicitud, pero aún no ha terminado de dar una respuesta final. En el caso de Homify, estos códigos no son particularmente relevantes, ya que son más comunes en otro tipo de situaciones.

Por su parte, los códigos de la familia 3xx son utilizados para indicar redirecciones. Estos códigos informan al cliente que la solicitud necesita ser redirigida a una nueva URL para ser completada. En Homify, aunque el framework puede manejar este tipo de códigos, no tienen un rol destacado dentro del sistema actual.

Códigos 2xx: Solicitudes Exitosas

Los códigos de estado 2xx son utilizados cuando una solicitud se ha completado correctamente. Dentro de nuestra solución, el código más común dentro de esta familia es el **200 (OK)**, que indica que la solicitud ha sido procesada con éxito. Esto ocurre, por ejemplo, cuando se realiza una operación como obtener datos o actualizar un recurso existente y todo transcurre sin problemas.

Códigos 4xx: Errores del Cliente

Para manejar los distintos códigos de error, implementamos un *ExceptionFilter*, el cual se ocupa de atrapar las excepciones no controladas por el sistema, y devolver códigos de error adecuados para aquellas operaciones no exitosas o conflictivas.

Los códigos de estado de la familia 4xx indican errores causados por el cliente. En nuestro caso, se utilizan varios códigos de este tipo para reflejar diferentes situaciones en las que la solicitud del cliente no puede ser procesada por diversos motivos:

- **400 (Bad Request):** Este código se utiliza cuando el servidor no puede entender la solicitud del cliente debido a un problema en su formato. Esto puede ocurrir cuando el cuerpo de la solicitud está mal estructurado o contiene datos incorrectos. En Homify, este código es lanzado cuando se presentan excepciones personalizadas como:
    - **ArgsNullException:** Cuando un argumento obligatorio es nulo.
    - **InvalidFormatException:** Cuando un argumento no tiene el formato esperado.
    - **NullRequestException:** Cuando la solicitud es completamente nula o vacía.
    - **ArgumentException**: Cuando uno de los argumentos proporcionados no es válido.
    - **InvalidOperationException**: Cuando el estado actual de una entidad no permite hacer ciertas operaciones sobre ella (ej. cuando se quieren agregar más miembros a una casa cuya capacidad está llena).
- **401 (Unauthorized):** Este código indica que el cliente no está autenticado y, por lo tanto, no tiene permiso para acceder al recurso solicitado. Esto ocurre cuando el filtro de autenticación no encuentra un token válido en el encabezado de la solicitud o cuando el token no está asociado a ningún usuario.
- **403 (Forbidden):** A diferencia del 401, el código 403 significa que el cliente está autenticado, pero no tiene los permisos necesarios para acceder al recurso solicitado. En Homify, se muestra cuando el usuario no cuenta con el código de permiso adecuado en su lista de permisos.
- **405 (Method Not Allowed):** Este código es lanzado por el propio framework cuando se intenta utilizar un método HTTP que no está permitido para un recurso específico. Por ejemplo, si se intenta realizar un **PUT** en un recurso que solo admite **GET** o **DELETE**.
- **409 (Conflict):** El código 409 se utiliza cuando hay un conflicto que impide la ejecución de la solicitud. En nuestra implementación, esto ocurre cuando se intenta insertar datos duplicados. La excepción **DuplicatedDataException** lanza este código, ya que los datos son válidos, pero generan un conflicto con los ya existentes.

Códigos 5xx: Errores del Servidor

Los códigos de la familia 5xx indican que ha ocurrido un error en el servidor, en lugar de un problema causado por la solicitud del cliente.

- **500 (Internal Server Error):** Este código indica un error interno en el servidor que no fue anticipado ni manejado. En nuestra solución, este código se muestra cuando ocurre una excepción no controlada, es decir, cuando el sistema no tiene una respuesta predefinida para un error que ha surgido durante el procesamiento de una solicitud.

Justificación

El uso de estos códigos de estado en Homify sigue las convenciones estándar de HTTP, lo que permite informar de manera clara y precisa al cliente sobre el resultado de sus solicitudes. Cada código está diseñado para proporcionar al cliente una indicación específica de si su solicitud fue exitosa, si hubo errores en su formato, si no está autorizado para acceder a un recurso o si ocurrió un problema inesperado en el servidor. Esta implementación asegura que los desarrolladores que consuman la API puedan comprender fácilmente lo que está ocurriendo y tomar las acciones necesarias para corregir o ajustar sus solicitudes.

Descripción de los resources de la API:

Recurso	Acción	URI	Headers	Body	Response	Status Code	Justificación
admins	POST	/admins	Authorization: token	{ “name”: string, “surname”: string, “email”: string, “password”: string }	{“email”: string, “id”: string}	201 Created 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	Un administrador puede crear otros administradores.
admins	DELETE	/admins/{adminId}	Authorization: token	-	{“message”: string}	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	Un administrador puede eliminar otros administradores.
company owners	POST	/company-owners	Authorization: token	{ “name”: string, “surname”: string, “email”: string, “password”: string }	{“email”: string, “id”: string}	201 Created 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	Un administrador puede crear cuentas para dueños de empresas de dispositivos.
admins	GET	/admins/accounts?fullName={fullName}&role={role}&limit={limit}&off	Authorization: token	-	[{ “name”: string, “surname”:	200 Ok 400 Bad Request	Un administrador puede listar todas las cuentas de la

		set={offset}			string, “fullName”: string, “role”: “string”, “createdAt”: “string” }]	401 Unauthorized 403 Forbidden 500 Internal Server Error	plataforma. Se debe aplicar paginacion y filtros por nombre completo y rol. En las querys, page es el numero de pagina, mientras que size es la cantidad de resultados a mostrar (por defecto son 10, por lo que este parametro es opcional).
companies	GET	/companies?company={company}&owner={owner}&limit={limit}&offset={offset}	Authorization: token	-	[{ “companyName”: string, “ownerFullName”: string, “ownerEmail”: string, “companyRUT”: string }]	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	Un administrador puede listar todas las empresas de la plataforma. Se debe aplicar paginacion y filtros por nombre de empresa y nombre completo de dueño. En las querys, page es el numero de pagina, mientras que size es la cantidad de resultados a mostrar (por defecto son 10, por lo que este parametro es opcional).

companies	POST	/companies	Authorization: token	{
				“name”: string,
				“logo”: string,
				“RUT”: string
				}
					{
					“name”: string,
					“id”: string,
					}
						201 Created
						400 Bad Request
						401 Unauthorized
						403 Forbidden
						500 Internal Server Error
							Un dueño de empresa, cuya cuenta esté marcada como incompleta, puede crear y asociar una única empresa a su cuenta.

devices	POST	/devices/cameras	Authorization: token	{
				“name”: string,
				“model”: string,
				“isExterior”: boolean,
				“isInterior”: boolean,
				“description”: string,
				“photos”: string[],
				“movementDetection”: boolean,
				“peopleDetection”: boolean
				}
					{
					“name”: string,
					“id”: string,
					}
						201 Created
						400 Bad Request
						401 Unauthorized
						403 Forbidden
						500 Internal Server Error
							Los dueños de empresas con cuentas completas pueden registrar diversos modelos de cámaras inteligentes en la plataforma

devices	POST	/devices/sensors	Authorization: token	{
				“name”: string,
				“model”: string,
				“description”: string,
				“photos”: string[],
				}
					{
					“name”: string,
					“id”: string,
					}
						201 Created
						400 Bad Request
						401 Unauthorized
						403 Forbidden
						500 Internal Server Error
							Los dueños de empresas con cuentas completas pueden registrar diferentes modelos de sensores inteligentes en la plataforma.

devices	PUT	/devices/{hardwareId}/activate	Authorization: token	-	{ “id”: string, “isActive”: boolean }	201 Created 400 Bad Request 401	Cualquier miembro perteneciente a la casa puede activar un dispositivo

						Unauthorized 403 Forbidden 500 Internal Server Error	(encenderlo).

sessions	POST	/sessions	-	{
				“email”: string,
				“password”: string
				}
					{“sessionToken” : string}	201 Created
						400 Bad Request
						500 Internal Server Error
							La plataforma permite a los usuarios autenticarse utilizando credenciales válidas para acceder a su cuenta. Esto creara una session identificada con un token, la cual expirara tras pasados 60 minutos, y para renovarla y poder seguir usando el sistema, se debe hacer un nuevo login.

homeowners	POST	/homeowners	-	{“name”: string, “surname”: string “profilePicture”: string, “email”: string, “password”: string}	{“email”: string, “id”: string}	201 Created 400 Bad Request 500 Internal Server Error	Una persona puede crear una cuenta en la plataforma como dueño de hogar.
devices	GET	/devices?deviceName ={device}&model={mo del}company={compan	Authorizati on: token	-	[{ “name”: string, “model”: string,	200 Ok 400 Bad Request	Una persona autenticada puede visualizar una lista

		y}&type={type}&limit={limit}&offset={offset}			“photo”: string, “companyName”: string }]	401 Unauthorized 403 Forbidden 500 Internal Server Error	de los dispositivos registrados en la plataforma. En las querys, page es el numero de pagina, mientras que size es la cantidad de resultados a mostrar (por defecto son 10, por lo que este parametro es opcional).
devices	GET	/devices/supported	Authorization: token	-	[{ “name”: string, }]	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	Una persona autenticada puede consultar una lista de los tipos de dispositivos que la plataforma admite

homes	POST	/homes	Authorization: token	{
				“address”: {“street”: string, “number”: number},
				“location”: {“lat”: number, “lon”: number},
				maxMembers: number
				}
					{“id”: string}	201 Created
						400 Bad Request
						401 Unauthorized
						403 Forbidden
						500 Internal Server Error
							Una persona autenticada puede crear múltiples hogares en la plataforma.

homes	PUT	/homes/{homeId}/mem	Authorizati	{“userEmail”: string}	{“newMembers”	200 Ok	El dueño del hogar

		bers	on: token		: string[]}	400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	puede agregar cuentas existentes en la plataforma a unirse al hogar como miembros.
homes	PUT	/homes/{homeId}/devices	Authorization: token	{“deviceId”: string}	{“hardwareId”: string, “deviceId”: string, “connected”: boolean}	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	El dueño del hogar o cualquier miembro con el permiso adecuado puede asociar dispositivos, previamente registrados en la plataforma, a cualquiera de los hogares bajo su control.
homes	PUT	/homes/{homeId}/{memberId}	Authorization: token	{“canAddDevices”: boolean, “canListDevices”: “boolean”}	{“userId”: string, “permissions”: string[]}	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	El dueño del hogar puede modificar los permisos que los miembros del hogar tienen sobre el mismo.
homes	GET	/homes/{homeId}/members	Authorization: token	-	[{ “fullName”: string, “email”: string, “photo”: string, “permissions”:	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden	El dueño del hogar puede ver una lista completa de los miembros asociados a su hogar.

					string[], mustBeNotified: boolean }]	500 Internal Server Error	
homes	GET	/homes/{homeId}/devices	Authorization: token	-	[{ “name”: string, “model”: string, “photo”: string, “connected”: boolean }]	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	El dueño del hogar o cualquier miembro con el permiso adecuado puede ver todos los dispositivos instalados en su hogar.

homes	PUT	/homes/{homeId}/notifications	Authorization: token	{ “homeUserId”: string }	{
					“membersToNotify”: string[]
					}
						200 Ok
						400 Bad Request
						401 Unauthorized
						403 Forbidden
						500 Internal Server Error
							El dueño del hogar puede seleccionar qué miembros del hogar recibirán notificaciones de los dispositivos.

notifications	POST	/notifications/person-detected	Authorization: token	{“hardwareId”: string, “date”: string, “personDetectedId”: string? }	{ “event”: string, “deviceId”: string, “hardwareId”: string, “date”: string, “personDetectedId”: string, isRead: bool }[]	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	Crear una notificación para los miembros configurados del hogar, cuando se detecte a una persona. Retorna un arreglo con todas las notificaciones creadas (una por cada usuario

							notificable).
notifications	POST	/notifications/movement-detected	Authorization: token	{“deviceId”: string, “hardwareId”: string, “date”: string, “action”: string? }	{“id”: string, “isRead”: bool, “event”: string, “deviceId”: string, “hardwareId”: string, “date”: string, “action”: string?}[]	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	Crear una notificación para los miembros configurados del hogar, cuando se detecte movimiento. Retorna un arreglo con todas las notificaciones creadas (una por cada usuario notificable).
notifications	POST	/notifications/window-movement	Authorization: token	{“deviceId”: string, “hardwareId”: string, “date”: string, “action”: string? }	{“id”: string, “isRead”: bool, “event”: string, “deviceId”: string, “hardwareId”: string, “date”: string, “action”: string?}[]	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden 500 Internal Server Error	Crear una notificación para los miembros configurados del hogar, cuando se abre o cierra una ventana. Retorna un arreglo con todas las notificaciones creadas (una por cada usuario notificable).
notifications	GET	/notifications?eventTriggered={eventTriggered}&date={date}&read={read}	Authorization: token	-	[{ “event”: string, “fromDevice”: string, “date”: string, “isRead”:	200 Ok 400 Bad Request 401 Unauthorized 403 Forbidden	Un miembro del hogar puede ver las notificaciones que le han generado.

					boolean, “id”: string }]	500 Internal Server Error	
notifications	PUT	/notifications/{notificationId}	Authorization: token	-	{ “id”: string, “isRead”: boolean }	200 Ok 401 Unauthorized 403 Forbidden 500 Internal Server Error	Un miembro de un hogar puede consumir una notificación que se le ha generado.