Universidad ORT Uruguay

Facultad de Ingeniería

Documentación Obligatorio 1

Evidencia del diseño y especificación de la API

Diseño de Aplicaciones 2

Link al Repositorio: IngSoft-DA2/301178_231810_280070 (github.com)

Integrantes:

- Angelina Maverino - 280070
- Yliana Otero - 301178
- María Belén Rywaczuk - 231810

Tutores:

- Juan Ignacio Irabedra De Maio
- Juan Pablo Barrios García
- Ignacio Valle Dubé

Grupo N6A - 2024

ÍNDICE

Introducción

SmartHome API, es una plataforma centralizada para la gestión de dispositivos inteligentes en hogares. La plataforma está diseñada para soportar múltiples usuarios con distintos roles: administradores, dueños de empresa, y dueños de hogares, ofreciendo diferentes niveles de acceso y permisos. Estos endpoints incluyen funcionalidades para la creación y administración de dispositivos, usuarios y roles, así como la integración de dispositivos de diferentes fabricantes.

La colección de endpoints está organizada en cinco categorías clave:

1. Gestión de usuarios: Creación, modificación y eliminación de usuarios, y asignación de roles.
2. Gestión de dispositivos: Agregar, consumir y administrar dispositivos inteligentes conectados a la plataforma.
3. Gestión de hogares: Agregar, consumir y administrar hogares.
4. Gestión de empresas: Agregar, consumir y administrar empresas de dispositivos inteligentes.
5. Gestión de notificaciones: Enviar y consumir notificaciones asociadas a dispositivos.

Criterios seguidos para asegurar que la API cumple con REST

En esta sección se discuten las distintas características de SmartHome API por las cuales podemos decir que es una API REST.

Principios fundamentales

Para que una aplicación sea considerada una API REST, debe cumplir con los siguientes principios fundamentales:

- **Arquitectura Cliente-Servidor:** Debe haber una separación entre el cliente (quien consume la API) y el servidor (donde reside la API). Esto permite que los dos puedan evolucionar de forma independiente.

  Nuestra aplicación sigue una arquitectura Cliente-Servidor ya que:
  - El **cliente** y el **servidor** están separados, y se comunican mediante solicitudes HTTP.
  - El cliente solo interactúa con la API expuesta por el servidor, mediante los controladores, sin acceder directamente a la lógica de negocio o a los datos.
  - La lógica de negocio está encapsulada en servicios, y los datos se manejan a través de repositorios, asegurando que el cliente solo interactúe con la API de manera controlada y no con las capas internas de la aplicación.
- **Interfaz Uniforme:** Todos los recursos deben ser accedidos a través de una interfaz uniforme, generalmente HTTP, y deben tener identificadores únicos (URLs). Los métodos HTTP como GET, POST, PUT, DELETE deben ser usados para interactuar con los recursos:

  - Nuestra aplicación sigue el principio de **Interfaz Uniforme** gracias a la implementación de controladores que exponen la API de manera consistente y estandarizada.

- Los controladores definen las rutas y los métodos HTTP (como GET, POST, PUT, DELETE) que son utilizados para acceder a los diferentes recursos del sistema. Por ejemplo, todas las operaciones relacionadas con usuarios se gestionan a través de un controlador específico de UserController, mientras que los dispositivos, y demás recursos tienen sus propios controladores.

  La API expone acciones específicas a través de métodos HTTP estándar:
  GET: Para obtener información de un recurso.
  POST: Para crear un recurso nuevo.
  PUT: Para modificar o actualizar un recurso existente.
  DELETE: Para eliminar un recurso.
- Los controladores también se encargan de enviar respuestas uniformes al cliente. En caso de éxito, se envía un código de estado HTTP como 200 OK o 201 Created, y en caso de error, los controladores retornan códigos como 404 Not Found o 400 Bad Request, junto con mensajes detallados que describen el problema.

- **Stateless:** La API debe ser stateless, lo que significa que cada solicitud del cliente debe contener toda la información necesaria para que el servidor la entienda. El servidor no debe almacenar ninguna información sobre el estado del cliente entre solicitudes.
  Nuestra aplicación es Statless ya que:
  - **Evita el Almacenamiento de Sesiones en el Servidor:** En nuestra aplicación, utilizamos **tokens GUID** (Global Unique Identifier) como identificadores únicos para autenticar y rastrear solicitudes del cliente de manera segura y sin estado (stateless). Estos tokens se generan como valores únicos, imposibles de repetir, y se envían en cada solicitud para que el servidor pueda verificar la identidad del usuario sin necesidad de mantener una sesión.
  - Se envía toda la **información necesaria** (autenticación, parámetros) con cada solicitud.
- **Caché:** Las respuestas de la API deben ser marcadas como cacheables cuando sea posible, lo que permite a los clientes almacenar la respuesta y reutilizarla, reduciendo la carga del servidor.

  En el método GetDeviceTypes, utilizamos el atributo [ResponseCache(Duration = 3600, Location = ResponseCacheLocation.Client)]. Esto significa que la respuesta se almacenará en caché durante 1 hora y se podrá acceder rápidamente desde el lado del cliente. Si una solicitud se hace dentro de este período, se sirve la respuesta desde la caché, lo que reduce la latencia y mejora la eficiencia.

  Se eligió este endpoint porque se sabe que estos tipos no suelen cambiar constantemente.

  Se puede ver en Postman como las solicitudes a GetDeviceTypes, son cacheables, en la sección Cache-Control.

- **Sistema en capas:** La arquitectura puede estar compuesta por diferentes capas.
  En nuestra aplicación, la arquitectura de capas está organizada de manera que cada componente tiene una responsabilidad específica, facilitando la separación de preocupaciones y el cumplimiento de principios como SOLID.
    - **Capa de Presentación (Controladores/API)**

  Esta es la capa más externa de la aplicación, donde los usuarios interactúan con el sistema a través de solicitudes HTTP.
  No tiene conocimiento directo de cómo se implementan los datos o la lógica de negocio.
    - **Capa de Aplicación (Servicios)**

  La capa de servicios es responsable de la lógica de negocio de la aplicación.
  Los servicios interactúan únicamente con la interfaz de los repositorios (IRepository) para acceder a los datos, sin conocer cómo estos están almacenados.
    - **Capa de Persistencia (Repositorios)**

  La capa de persistencia es donde se gestiona el acceso a los datos. Utilizamos un repositorio genérico IRepository<T> que define las operaciones básicas de CRUD (Create, Read, Update, Delete) para las entidades.

  En nuestro caso, implementamos SqlRepository<T>, que se comunica directamente con la base de datos usando Entity Framework. El SmartHomeContext es pasado a este repositorio, asegurando que las consultas a la base de datos sigan las reglas establecidas por Entity Framework y que las migraciones reflejen los cambios en el modelo.

  En nuestra aplicación, hemos diseñado cada capa de manera que dependa de **abstracciones** (interfaces) y no de implementaciones concretas, lo cual es un principio clave para cumplir con el **Principio de Inversión de Dependencias (D) de SOLID**.

Información adicional

Recursos identificados por URIs: Los recursos están identificados por URIs (Uniform Resource Identifiers).

Ejemplo: api/v1/administrators

Representaciones de recursos: La representación de recursos se hace por medio del formato JSON.

Nuestra API REST sigue los estándares de las URI, garantizando que las rutas sean claras, consistentes y semánticamente correctas, lo que facilita su uso y comprensión. Algunos de los principales estándares de URI que seguimos son:

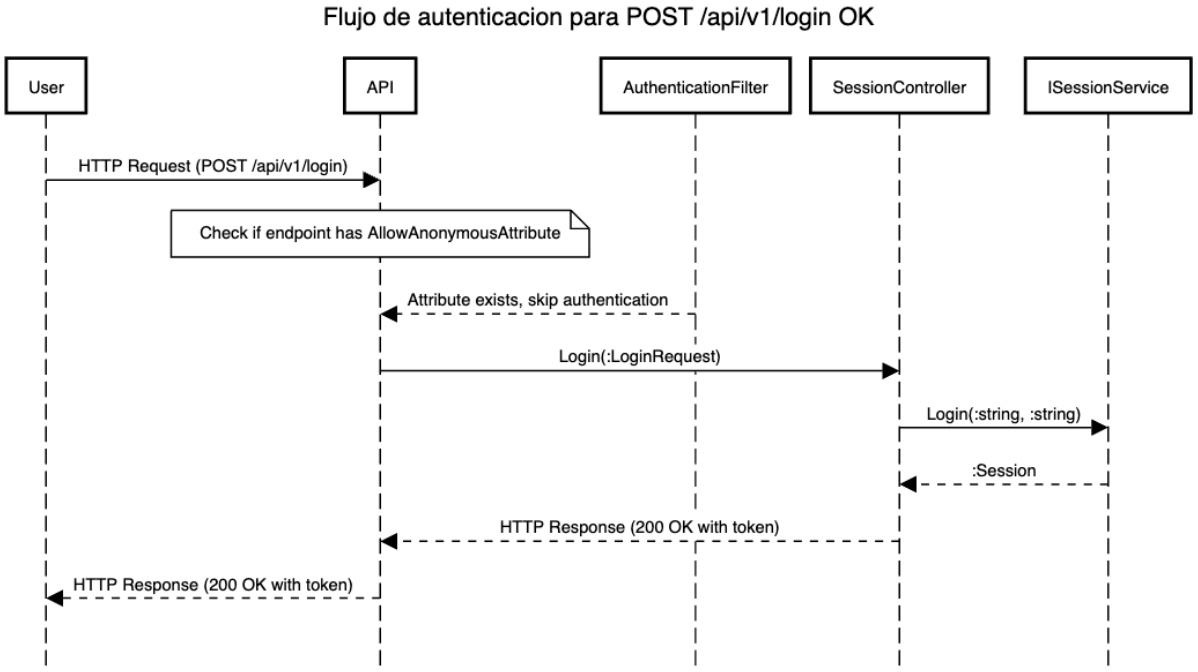
- **Uso de sustantivos:** En lugar de utilizar verbos, las URIs deben representar recursos mediante sustantivos. Por ejemplo, en nuestra API usamos /api/v1/home-owners para representar los propietarios de hogares. No usamos verbos como /getHomeOwners.
- **Formato en minúsculas:** Las URIs deben estar en minúsculas, lo que es más fácil de leer y menos propenso a errores. Por ejemplo, usamos /api/v1/device-types en lugar de /api/v1/DeviceTypes.
- **Identificación Única de Recursos:** Cada recurso tiene una URI única que lo representa, lo que sigue el principio de identificación clara de los recursos.
- **Versionado en la URI:** Implementamos el versionado a través del prefijo v1 en nuestras rutas, lo que permite gestionar actualizaciones futuras sin romper la compatibilidad con versiones anteriores: /api/v1/…
- **Profundidad mínima:** Nuestras URIs tratan de no exceder muchos niveles de profundidad para mantenerlas simples y comprensibles, como en /api/v1/device-types.

Descripción del mecanismo de autenticación de requests

En nuestra aplicación, la autenticación de **requests** se implementa con **tokens GUID**. Esto asegura que solo los usuarios autenticados puedan acceder a los recursos protegidos de la API.

Este mecanismo se implementa a través de un filtro de autenticación llamado AuthenticationFilter. Cuando se recibe una solicitud HTTP, el filtro se activa antes de que se ejecute la acción del controlador correspondiente. Además, la API maneja tres atributos para autenticacion: AllowAnonymous, RolesWithPermissions y RestrictToPrivilegedMembers

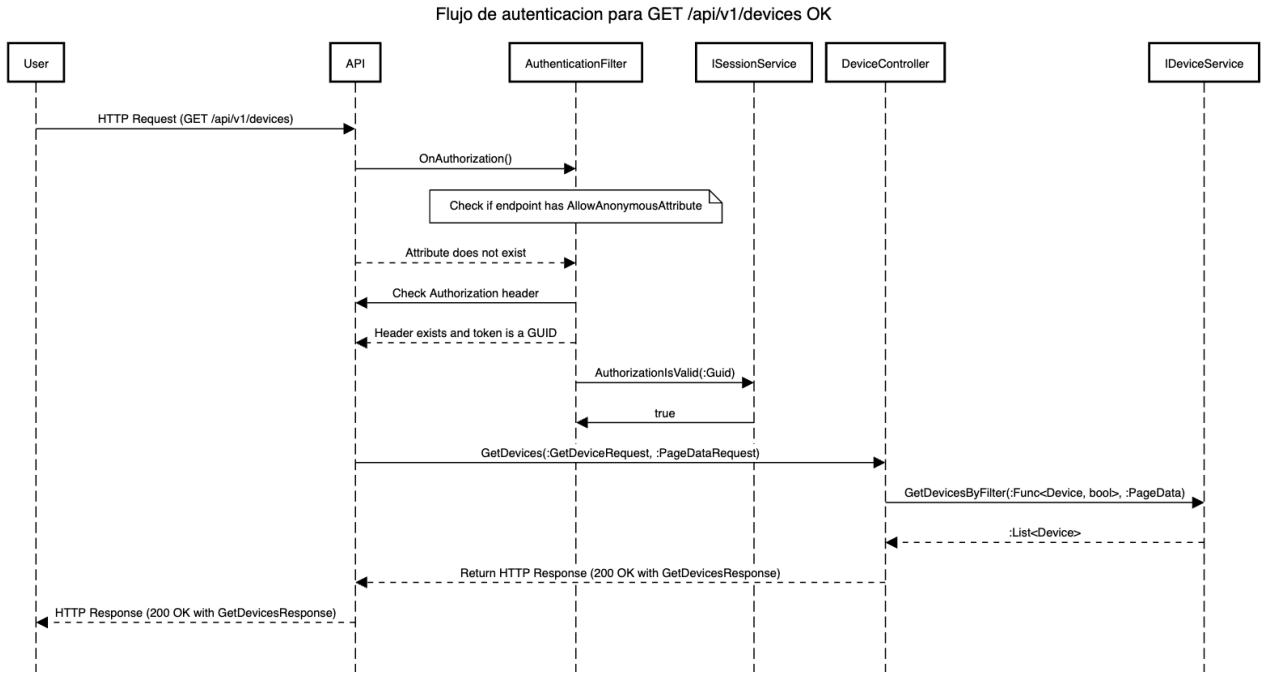
Si el método del controlador tiene el atributo AllowAnonymous, se permite el acceso sin restricciones. Es decir, el método es público y puede ser accedido sin necesidad de incluir headers de autorización. Este es el caso de los endpoints para login (POST /api/v1/login) y creación de cuenta para dueño de hogar (POST /api/v1/home-owners).



Notar que este endpoint permite iniciar sesión con correo electrónico y contraseña, y cuando estas credenciales son correctas, retorna un token GUID. Este token es el que se utilizará para autenticar los demás endpoints.

Si el método del controlador no tiene el atributo AllowAnonymous, el filtro de autenticación busca el header Authorization en la solicitud HTTP. Este header debe contener un token GUID de autenticación válido. Si el header no está presente o el token no es válido (no es un GUID, o no corresponde a ninguna sesión), se devuelve un error HTTP 401 (Unauthorized).
No se diagrama este caso en específico porque este chequeo se realiza en cualquiera de los siguientes casos que están diagramados.

En caso de que el método no tenga ningún atributo de autorización (AllowAnonymous, RolesWithPermissions o RestrictToPrivilegedMembers), el filtro es aplicado pero únicamente para validar que exista el Authorization header y que el token sea válido. Es decir, cualquier persona autenticada puede acceder al endpoint correspondiente. Este es el caso de los endpoints para listar dispositivos (GET /api/v1/devices, GET /api/v1/devices/{id}), para listar tipos de dispositivo soportados (GET /api/v1/devices/types), y para ver o enviar notificaciones (GET /api/v1/notifications, POST /api/v1/notifications).

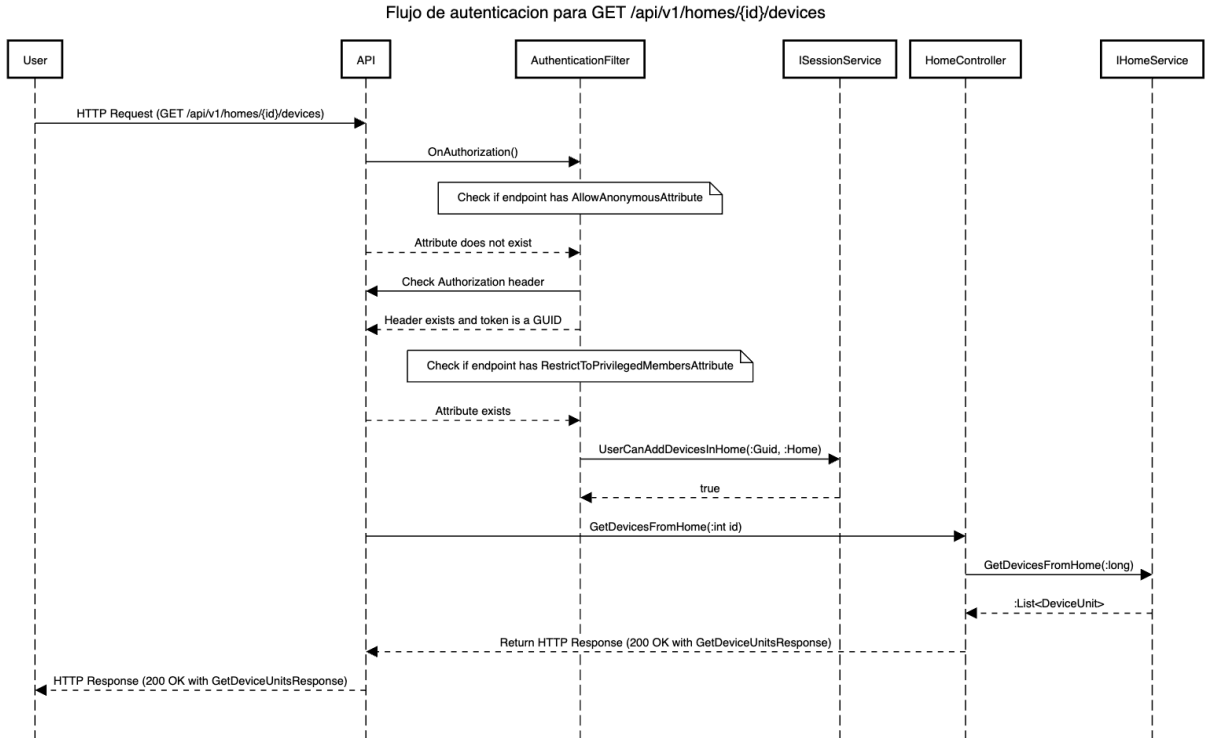


Si la request no tiene header de autorización, o si el token es invalido, se devuelve un error HTTP 401 (Unauthorized).

Otra posibilidad es que el método del controlador tenga el atributo RestrictToPrivilegedMembers. Este atributo representa los permisos que deben tener los miembros de un hogar para poder ejecutar la acción correspondiente, por lo recibe dos valores booleanos: uno determina si es necesario que el usuario tenga permiso para listar los dispositivos de un hogar, y el otro determina si es necesario que el usuario tenga permiso para agregar dispositivos al hogar.

Una vez que el filtro verificó que existe el Authorization header y el token es válido, procede a chequear si el usuario tiene los correspondientes permisos sobre el hogar.

Este es el caso de los endpoints para crear y listar dispositivos en un hogar (POST /api/v1/homes/{id}/devices, GET /api/v1/homes/{id}/devices)

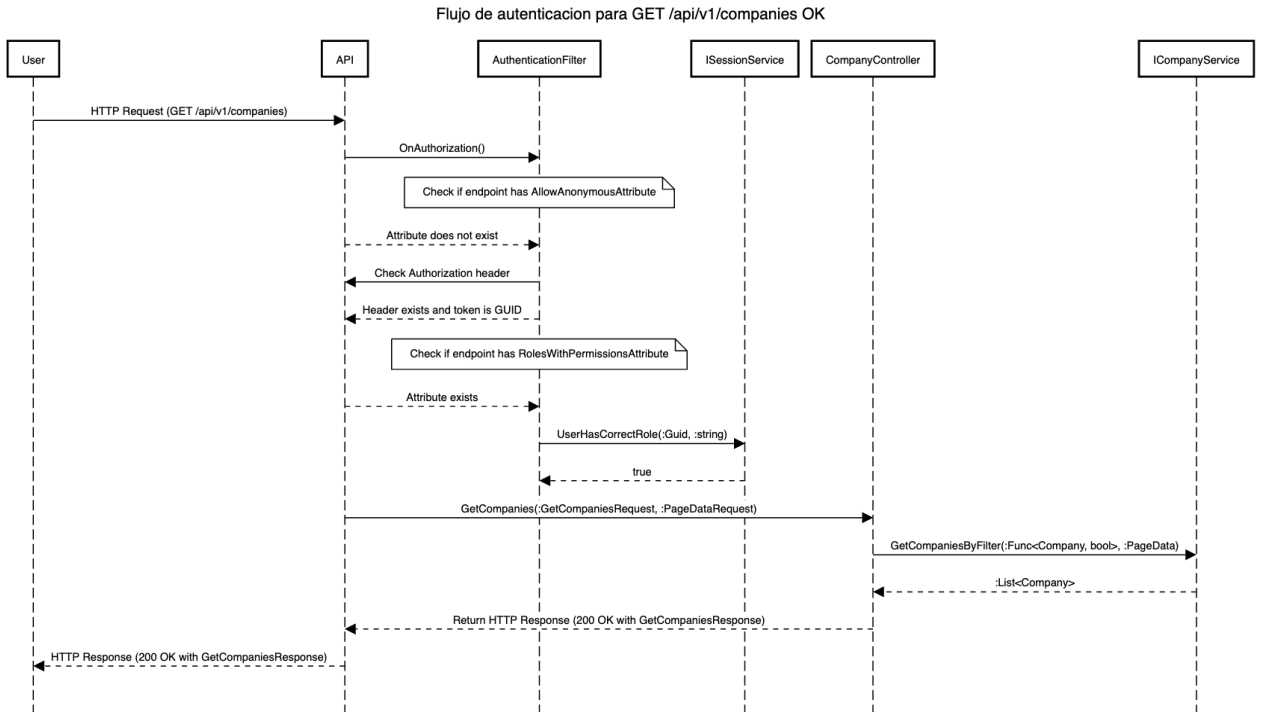


Si la request no tiene header de autorización, o si el token es invalido, se devuelve un error HTTP 401 (Unauthorized). Si el usuario no tiene permiso para ejecutar la acción en el hogar, se devuelve un error HTTP 403 (Forbidden).

Por último, el método del controlador podría tener el atributo RolesWithPermissions. Este atributo recibe una lista de roles (por ejemplo “Administrator”), los cuales tendrán acceso al endpoint. En este caso, si existe el Authorization header y el token es válido, el filtro verifica además que el usuario correspondiente a la sesión identificada por el token tenga alguno de los roles de la lista. Este es el caso de todos los demás endpoints.

Si la request no tiene header de autorización, o si el token es invalido, se devuelve un error HTTP 401 (Unauthorized). Si el usuario no tiene ninguno de los roles necesarios, se devuelve un error HTTP 403 (Forbidden).

Observación: puede hacer click en las imágenes para abrirlas en Google Drive y verlas con mayor tamaño.



Descripción general de códigos de error (1xx, 2xx, 4xx, 3xx, 5xx)

En nuestra API REST, los códigos de estado HTTP son respuestas estándar que indican el resultado de una solicitud realizada por el cliente. Estos códigos están divididos en cinco categorías principales, cada una representando un tipo diferente de respuesta:

1xx - Códigos Informativos
Indican que la solicitud ha sido recibida y el proceso sigue en curso.

En esta entrega no fueron utilizados.

2xx - Códigos de Éxito
Indican que la solicitud se ha procesado con éxito.

Se utilizan:

- 200 OK: La solicitud ha sido exitosa y el servidor devuelve la respuesta solicitada.
- 201 Created: Se ha creado un nuevo recurso con éxito, generalmente como resultado de una solicitud POST.

3xx - Códigos de Redirección
Indican que se requiere alguna acción adicional del cliente para completar la solicitud.

En esta entrega no fueron utilizados.

4xx - Errores del Cliente

Indican que hubo un problema con la solicitud realizada por el cliente.

Se utilizan:

- 400 Bad Request: La solicitud está mal formada o contiene datos incorrectos.
- 401 Unauthorized: La solicitud no tiene las credenciales necesarias o no tiene un token válido.
- 403 Forbidden: El cliente está autenticado, pero no tiene permisos para acceder al recurso solicitado.
- 404 Not Found: El recurso solicitado no existe o no se encuentra disponible en el servidor.
- 412 Precondition Failed: Cuando una o más de las condiciones predefinidas en los encabezados de la solicitud no se cumplieron.
- 415: Unsupported Media Type: cuando una solicitud requiere request body y no es provisto.

5xx - Errores del Servidor
Indican que el servidor encontró un problema al procesar la solicitud.

Se utilizan:

- 500 Internal Server Error: El servidor encontró un error inesperado que le impide procesar la solicitud.

Nota: Se utiliza el error 500 para excepciones no controladas. Esto lo configuramos utilizando un filtro para excepciones: CustomExceptionFilter. Toda excepción no controlada, o excepción de tipo NotImplementedException, es capturada por este filtro y retorna el error HTTP 500 Internal Server Error. Consideramos que en una próxima entrega sería buena idea implementar el resto de excepciones posibles a través de filtros de excepción.

Tabla de descripción de los resources de la API para requerimientos

A continuación se muestra la descripción de los recursos de la API, según la tabla enviada a los profesores para confirmar los endpoints necesarios acorde a los requerimientos funcionales. Puede también leer el swagger desde este link, y ver la documentación de SwaggerGen generada a partir del swagger desde este link.

Mantenimiento de cuentas de administrador

Administrator	DELETE	/api/v1/administrators/{id}	Autentication Header			200 Ok. 401 Unauthorized. 403 Forbidden. 404 Not Found. 500 internal server error.	200: Ok. 401: no se envia el token de autorizacion. 403 No se tienen los permisos necesarios. 500: Error en el servidor. 404 no existe el admin con ese id.
Administrator	POST	api/v1/administrators	Autentication Header	{ "Name": string, "Surname": string, "Email": string, "Password": string, "Photo": string }	{ "Name": string, "Surname": string, "Email": string, "Password": string, "Photo": string }	201 Created. 400 Invalid Input. 401 Unauthorized. 403 forbidden. 500 Internal Server Error. 409 Conflict	201: Administrador creado correctamete. 401: No se envia el token de autorizacion. 403: no se tienen los permisos suficientes. 500: Error en el servidor. 409 Ya existe un usuario con ese Email. 400: Se envio una request mal formada

Creación de cuentas para dueños de empresas de dispositivos inteligentes

CompanyOwner	POST	api/v1/companyowners	Autentication Header	{ "name": string, "surname": string, "email": string, "password": string }	{ "Name": string, "Surname": string, "Email": string, "Password": string, "Photo": string }	201 Created. 400 Invalid Input. 401 Unauthorized. 403 forbidden. 500 Internal Server Error. 409 Conflict	201: CompanyOwner creado correctamete. 401: No se envia el token de autorizacion. 403: no se tienen los permisos suficientes. 500: Error en el servidor. 409 Ya existe un usuario con ese Email. 400: Se envio una request mal formada

Listar todas las cuentas

Users	GET	api/v1/users	Autentication Header	{"FullName": string?, "Role": string?,"Page": int?, "PageSize": int? }	{ "name": string, "surname": string, fullName": string, "Roles": List<Roles>, "createdAt": timestamp }	200 ok. 401 unauthorized. 403 forbidden. 500 Internal Server Error, 404 Not found.	200: Ok. 401: No se envia el token de autorizacion. 403: no se tienen los permisos suficientes. 500: Error en el servidor. 404: lista de usuarios no encontrada

Listar todas las empresas

Company	GET	api/v1/companies	Autentication Header	{"Company": string?, "Owner": string?,"Page": int?, "PageSize": int? }	{ "name": string, "RUT": string, "LogoURL": string?, "FullName": string, "email": string}	200 ok. 401 unauthorized. 403 forbidden. 500 Internal Server Error	200: Ok. 401: No se envia el token de autorizacion. 403: no se tienen los permisos suficientes. 500: Error en el servidor

Manejo de notificaciones

Notifications	GET	api/v1/notifications		{ "HomeId": int?, "UserId": int? ,"createdAt": DateTime?, "Read": bool?, "Kind": string? }	{ "event": string, "device": { "kind": string, "name": string, "id": string }, "createdAt": DateTime, "Read": bool, "ReadAt": timestamp }	200 ok. 401 unauthorized. 403 forbidden. 500 Internal Server Error	200: Ok. 401: No se envia el token de autorizacion. 403: no se tienen los permisos suficientes. 500: Error en el servidor
Notifications	POST	api/v1/notifications		{ "Event": string, "HomeId": long, "HardwareId": long }	{ "Event": string, "HomeId": long, "HardwareId": long }	200 ok. 401 unauthorized. 403 forbidden. 500 Internal Server Error. 404 Not found	200: Ok. 401: No se envia el token de autorizacion. 403: no se tienen los permisos suficientes. 500: Error en el servidor. 404 notificacion no encontrada

Registrar cámaras de seguridad y Registrar sensores de ventanas

Devices	POST	api/v1/devices/window-sensors	Autentication Header	{ "name": string, "model": long, "description": string, "photoUrls": string[], "Functionalities": List<WindowSensorFunctionality>?, Company: long }	{ "name": string, "model": long, "description": string, "photoUrls": string[], "Functionalities": List<WindowSensorFunctionality>?, Company: long }	201 ok. 400 Bad Request. 401 unauthorized. 403 forbidden. 500 Internal Server Error. 404 Not Found	201: WindowSensor creado correctamete. 401: No se envia el token de autorizacion. 403: no se tienen los permisos suficientes. 500: Error en el servidor. 404 company not found
Devices	POST	api/v1/devices/security-cameras	Autentication Header	{ "name": string, "model": long, "description": string, "photoUrls": string[], "locationType": LocationType, "functionalities": []SecurityCameraFunctionalities , "Company": long}	{ "name": string, "model": long, "description": string, "photoUrls": string[], "locationType": LocationType, "functionalities": []SecurityCameraFunctionalities }	201 ok. 400 Bad Request. 401 unauthorized. 403 forbidden. 500 Internal Server Error. 404 Not Found	201: WindowSensor creado correctamete. 401: No se envia el token de autorizacion. 500: Error en el servidor. 404 Company not found

Autenticación mediante credenciales

Session	POST	/api/v1/login		{"email": string, "password": string }	{ "token": string }	. 404 Not Found.	200: Ok. 400: La request esta mal formada. 404: no existe usuario con ese email y contraseña. 500: Error en el servidor

Creación de una empresa

Company	POST	api/v1/companies	Autentication Header	{ "name": string, "logoUrl": string, "RUT": string, "ownerID": long }	{ "name": string, "logoUrl": string, "RUT": string, "ownerID": long }	201 ok. 400 Bad Request. 401 unauthorized. 403 forbidden. 500 Internal Server Error. 404 Not found	201: Company creada correctamete. 401: No se envia el token de autorizacion. 403: no se tienen los permisos suficientes. 500: Error en el servidor. 404: CompanyOwner no encontrado

Creación de cuenta para dueño de hogar

HomeOwner	POST	api/v1/home-owners		{ "name": string, "surname": string, "email": string, "password": string, "photo": string }	{ "name": string, "surname": string, "email": string, "password": string, "photo": string }	201 ok. 400 Bad Request. 500 Internal Server Error. 409 Conflict.	201: HomeOwner creado correctamete. 400 bad request. 500: Error en el servidor. 409 Ya existe un usuario con ese email

Listado de dispositivos

Devices	GET	api/v1/devices	Autentication Header	{"Name": string?, "Model": long?,"Company": string?,"Kind": string?,"Page": int?, "PageSize": int? }	{ "name": string, "model": long, "photoUrl": string?, "companyName": string? }	200 ok. 401 unauthorized. 500 Internal Server Error	200: Ok. 401: No se envia el token de autorizacion o se envia pero no corresponde a ningun usuario. 500: Error en el servidor

Listado de tipos de dispositivos soportados

Devices	GET	api/v1/device-types	Autentication Header		{ "supportedDevices": string[] }	200 ok. 401 unauthorized. 500 Internal Server Error	200: Ok. 401: No se envia el token de autorizacion o se envia pero no corresponde a ningun usuario. 500: Error en el servidor

Creación de hogares

Home	POST	api/v1/homes/	Autentication Header	{"OwnerId":long, "Latitude": double, "Longitude": double, "MaximumMembers": int, "DoorNumber": int, "Street": string }	{"Street":string, "Latitude": double "Longitude": double, "DoorNumber": int}	201 created. 400 bad request 401 unauthorized 403 forbidden 500 internal server error 404 Not found 412 precondition failed	201 Created. 400: La request esta mal formada. 401: no se envia el token de autorizacion. 403 No se tienen los permisos disponibles. 500: Error en el servidor. 404 no existe usuario y mail con esa contraseña. 412 s eencontro el usuario pero no tiene rol de home owner

Agregar miembros al hogar

Home	PATCH	api/v1/homes/{id}/members	Autentication Header	{ "UserEmail": string, "ReceivesNotifications": bool, "HasPermissionToListDevices": bool, "HasPermissionToAddADevice": bool}		200 ok. 401 unauthorized. 400 Bad Request. 403 forbidden. 500 internal server error 404 not found. 412 Precondition Failed. 409 conflict	200: Ok. 401: no se envia el token de autorizacion. 403 No se tienen los permisos disponibles. 500: Error en el servidor. 404 no existe el usuario y mail con esa contraseña. 403: no se tienen los permisos suficientes. 412: No se cumple la precondicion de que no se haya llegado al maximo de miembros del hogar. 409 member va existe
Home	PATCH	api/v1/homes/{id}/devices	Autentication Header	{ "DeviceId": long, "IsConnected": bool }		200 ok. 401 unauthorized. 400 Bad Request. 403 forbidden. 500 internal server error 404 not found	200: Ok. 401: no se envia el token de autorizacion. 403 No se tienen los permisos disponibles. 500: Error en el servidor 404 no se encuentra la casa

Listado de miembros de un hogar

Home	GET	api/v1/homes/{id}/members	Autentication Header		{ "HasPermissionToListDevices": bool, "HasPermissionToAddADevice": bool,"ReceivesNotifications": bool ,"Email": string,"Photo": string, "FullName": string }	200 Ok. 401 unauthorized. 400 Bad Request. 403 Forbidden. 404 Not Found. 500 internal server error.	200: Ok. 401: no se envia el token de autorizacion. 403 No se tienen los permisos necesarios. 500: Error en el servidor. 404 no existe el hogar.

Listado de dispositivos de un hogar

Home	GET	api/v1/homes/{id}/devices	Autentication Header		{ "HardwareId": Guid, "Device": string, "IsConnected": bool }	200 Ok. 401 unauthorized. 400 Bad Request. 403 Forbidden. 404 Not Found. 500 internal server error.	200: Ok. 401: no se envia el token de autorizacion. 403 No se tienen los permisos necesarios. 500: Error en el servidor. 404 no existe el hogar.

Configuración de notificaciones para miembros

Home	PATCH	api/v1/homes/{id}/members	Autentication Header	{ "Memberemail": string, "ReceivesNotifications": bool }		201 Created. 400 Bad Request. 401 unauthorized. 403 forbidden. 500 internal server error 404 Not found	201 Created. 400: La request esta mal formada. 401: no se envia el token de autorizacion. 403 No se tienen los permisos disponibles. 500: Error en el servidor. 404 home not found

Descripción de los endpoints por controllers y colección Postman

A continuación se describen los endpoints definidos en cada API Controller (y por ende en cada ruta base), y todas sus posibles respuestas, así como decisiones de diseño tomadas. Puede verlo también en la colección de Postman, la cual puede descargar desde este link.

AdministratorController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/administrators.

1. **POST /api/v1/administrators:** Este endpoint se utiliza para crear un nuevo administrador. El cuerpo de la solicitud debe contener los datos necesarios para crear un administrador, según lo especificado en el modelo *PostAdministratorRequest*.
    1. Si la creación es exitosa, devuelve un estado 201 (Created) con la respuesta *PostAdministratorResponse*.
    2. Si el administrador ya existe, devuelve un estado 409 (Conflict).
    3. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    4. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).
2. **DELETE /api/v1/administrators/{id}:** Este endpoint se utiliza para eliminar un administrador existente con el ID especificado.
    1. Si la eliminación es exitosa, devuelve un estado 200 (OK).
    2. Si el administrador no se encuentra, devuelve un estado 404 (Not Found).

Además, para todos los endpoints se cumple que:

- Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).
- Si no se incluye Authorization Header, o si se incluye pero el token incluido está mal formado o no corresponde a ninguna sesión, devuelve un estado 401 (Unauthorized).
- Si se incluye Authorization Header con un token válido, pero el usuario correspondiente no tiene suficientes permisos (rol *Administrator*), devuelve un estado 403 (Forbidden).

CompanyController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/companies

1. **GET /api/v1/companies:** Este endpoint se utiliza para obtener una lista de empresas. Los parámetros de consulta *GetCompaniesRequest* y *PageDataRequest* permiten filtrar y paginar los resultados, respectivamente.
    1. Si la solicitud es exitosa, devuelve un estado 200 (OK) con la respuesta *GetCompaniesResponse*.
    2. Si los query parameters de la solicitud están mal formados, devuelve un estado 400 (Bad Request).
2. **POST /api/v1/companies:** Este endpoint se utiliza para crear una nueva empresa. El cuerpo de la solicitud debe contener los datos necesarios para crear una empresa, según lo especificado en el modelo *PostCompanyRequest*.
    1. Si la creación es exitosa, devuelve un estado 201 (Created).
    2. Si no se encuentra un propietario de empresa con el id especificado y sin empresa asociada, devuelve un estado 404 (Not Found).
    3. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    4. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).

Además, para todos los endpoints se cumple que:

- Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).
- Si no se incluye Authorization Header, o si se incluye pero el token incluido está mal formado o no corresponde a ninguna sesión, devuelve un estado 401 (Unauthorized).
- Si se incluye Authorization Header con un token válido, pero el usuario correspondiente no tiene suficientes permisos (rol *Administrator* para el GET y *CompanyOwner* para el POST), devuelve un estado 403 (Forbidden).

CompanyOwnerController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/company-owners

1. **POST /api/v1/company-owners:** Este endpoint se utiliza para crear un nuevo propietario de empresa. El cuerpo de la solicitud debe contener los datos necesarios para crear un propietario de hogar, según lo especificado en el modelo *PostCompanyOwnerRequest*.
    1. Si la creación es exitosa, devuelve un estado 201 (Created) con la respuesta *PostHomeOwnerResponse*.
    2. Si el propietario de la casa ya existe, devuelve un estado 409 (Conflict) con un mensaje de error.
    3. Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).
    4. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    5. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).
    6. Si no se incluye Authorization Header, o si se incluye pero el token incluido está mal formado o no corresponde a ninguna sesión, devuelve un estado 401 (Unauthorized).
    7. Si se incluye Authorization Header con un token válido, pero el usuario correspondiente no tiene suficientes permisos (rol *Administrator*), devuelve un estado 403 (Forbidden).

DeviceController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/devices

1. **GET /api/v1/devices:** Este endpoint se utiliza para obtener una lista de dispositivos. Los parámetros de consulta *GetDeviceRequest* y *PageDataRequest* permiten filtrar y paginar los resultados, respectivamente.
    1. Si la solicitud es exitosa, devuelve un estado 200 (OK) con la respuesta *GetDevicesResponse*.
    2. Si los query parameters de la solicitud están mal formados, devuelve un estado 400 (Bad Request).
2. **GET /api/v1/devices/{id}:** Este endpoint se utiliza para obtener un dispositivo específico por su ID.
    1. Si el dispositivo se encuentra, devuelve un estado 200 (OK) con la respuesta *GetDeviceResponse*.
    2. Si el dispositivo no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
3. **GET /api/v1/devices/types:** Este endpoint se utiliza para obtener una lista de los tipos de dispositivos.
    1. Devuelve un estado 200 (OK) con la respuesta *GetDeviceTypesResponse*.

4. **POST /api/v1/devices/window-sensors:** Este endpoint se utiliza para crear un nuevo sensor de ventana. El cuerpo de la solicitud debe contener los datos necesarios para crear un sensor de ventana, según lo especificado en el modelo *PostWindowSensorRequest*.
    1. Si la creación es exitosa, devuelve un estado 201 (Created).
    2. Si la compañía no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The company was not found."
    3. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    4. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).
    5. Si se incluye Authorization Header con un token válido, pero el usuario correspondiente no tiene suficientes permisos (rol *CompanyOwner*), devuelve un estado 403 (Forbidden).
5. **POST /api/v1/devices/security-cameras:** Este endpoint se utiliza para crear una nueva cámara de seguridad. El cuerpo de la solicitud debe contener los datos necesarios para crear una cámara de seguridad, según lo especificado en el modelo *PostSecurityCameraRequest*.
    1. Si la creación es exitosa, devuelve un estado 201 (Created).
    2. Si la compañía no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The company was not found."
    3. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).
    4. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    5. Si se incluye Authorization Header con un token válido, pero el usuario correspondiente no tiene suficientes permisos (rol *CompanyOwner*), devuelve un estado 403 (Forbidden).

Además, para todos los endpoints se cumple que:

- Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).
- Si no se incluye Authorization Header, o si se incluye pero el token incluido está mal formado o no corresponde a ninguna sesión, devuelve un estado 401 (Unauthorized).

HomeController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/homes

1. **GET /api/v1/homes:** Este endpoint se utiliza para obtener una lista de hogares, permitiendo filtrar según el modelo *GetHomeRequest*.
    1. Si la solicitud es exitosa, devuelve un estado 200 (OK) con la respuesta *GetHomesResponse*.
    2. Si los query parameters de la solicitud están mal formados, devuelve un estado 400 (Bad Request).
2. **POST /api/v1/homes:** Este endpoint se utiliza para crear un nuevo hogar, y requiere un body con los datos especificados en *PostHomeRequest*.
    1. Si la creación es exitosa, devuelve un estado 201 (Created).
    2. Si el propietario del hogar no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The home owner was not found."
    3. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).

    4. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).
3. **GET /api/v1/homes/{id}:** Este endpoint se utiliza para obtener un hogar específico por su ID.
    1. Si el hogar se encuentra, devuelve un estado 200 (OK) con la respuesta *GetHomeResponse*.
    2. Si el hogar no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
4. **GET /api/v1/homes/{id}/members:** Este endpoint se utiliza para obtener los miembros de un hogar.
    1. Si la solicitud es exitosa, devuelve un estado 200 (OK) con la respuesta *GetMembersResponse*.
    2. Si el recurso no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
5. **POST /api/v1/homes/{id}/members:** Este endpoint se utiliza para agregar un miembro a un hogar, y requiere un body con los parametros especificados en *PostHomeMemberRequest*.
    1. Si la actualización es exitosa, devuelve un estado 200 (OK) con el mensaje "The home was updated successfully".
    2. Si el recurso no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
    3. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    4. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).
6. **PATCH /api/v1/homes/{id}/members:** Este endpoint se utiliza para cambiar el permiso de notificación de un miembro de un hogar, y requiere un body con los parametros especificados en *PatchDeviceRequest*.
    1. Si la actualización es exitosa, devuelve un estado 200 (OK) con el mensaje "The home was updated successfully".
    2. Si el recurso no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
    3. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    4. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).
7. **GET /api/v1/homes/{id}/devices:** Este endpoint se utiliza para obtener los dispositivos de un hogar.
    1. Si la solicitud es exitosa, devuelve un estado 200 (OK) con la respuesta *GetDeviceUnitsResponse*.
    2. Si el recurso no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
8. **POST /api/v1/homes/{id}/devices:** Este endpoint se utiliza para agregar dispositivos a un hogar, y requiere un body con los parámetros especificados en *PostHomeDevicesRequest*.
    1. Si la actualización es exitosa, devuelve un estado 200 (OK) con el mensaje "The home was updated successfully".
    2. Si el recurso no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
    3. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    4. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).

9. **PATCH /api/v1/homes/{id}/devices:** Este endpoint se utiliza para actualizar el estado de conexión de un dispositivo, y requiere un body con los parámetros especificados en *PatchHomeMemberRequest*.
    1. Si la actualización es exitosa, devuelve un estado 200 (OK) con el mensaje "The home was updated successfully".
    2. Si el recurso no se encuentra, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
    3. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
    4. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).

Además, para todos los endpoints se cumple que:

- Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).
- Si no se incluye Authorization Header, o si se incluye pero el token incluido está mal formado o no corresponde a ninguna sesión, devuelve un estado 401 (Unauthorized).
- Si se incluye Authorization Header con un token válido, pero el usuario correspondiente no tiene suficientes permisos (rol *Administrator* para los endpoints 1 y 3, rol HomeOwner para los endpoints 2, 4, 5, 6 y 9, y ser el owner del hogar o ser un miembro del hogar con permisos suficientes para los endpoints 7 y 8), devuelve un estado 403 (Forbidden).

HomeOwnerController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/home-owners

1. **POST /api/v1/home-owners:** Este endpoint se utiliza para crear un nuevo propietario de hogar. El cuerpo de la solicitud debe contener los datos necesarios para crear un propietario de hogar, según lo especificado en el modelo *PostHomeOwnerRequest*.
    1. Si la creación es exitosa, devuelve un estado 201 (Created) con la respuesta *PostHomeOwnerResponse*.
    2. Si el propietario de la casa ya existe, devuelve un estado 409 (Conflict) con un mensaje de error.
2. **PUT /api/v1/home-owners/{id}:** Este endpoint se utiliza para actualizar un propietario de hogar existente con el ID especificado. El cuerpo de la solicitud debe contener los datos necesarios para actualizar un propietario de hogar, según lo especificado en el modelo *PutHomeOwnerRequest*.
    1. Si la actualización es exitosa, devuelve un estado 200 (OK) con la respuesta *PostHomeOwnerResponse*.
    2. Si el propietario de la casa no se encuentra, devuelve un estado 404 (Not Found) con un mensaje de error.
    3. Si no se incluye Authorization Header, o si se incluye pero el token incluido está mal formado o no corresponde a ninguna sesión, devuelve un estado 401 (Unauthorized).
    4. Si se incluye Authorization Header con un token válido, pero el usuario correspondiente no tiene suficientes permisos (rol *HomeOwner*), devuelve un estado 403 (Forbidden).

Además, para todos los endpoints se cumple que:

- Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).

- Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
- Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).

NotificationController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/notifications

Observar que estos endpoints puede utilizarlos cualquier usuario autenticado. Esto se debe a que la letra no especifica quien genera o consume las notificaciones, por lo que para esta entrega decidimos dejarlo así.

1. **GET /api/v1/notifications:** Este endpoint se utiliza para obtener una lista de notificaciones basada en los filtros proporcionados en el cuerpo de la solicitud *GetNotificationsRequest*.
    1. Si la solicitud es exitosa, devuelve un estado 200 (OK) con la respuesta *GetNotificationsResponse*.
    2. Si no se encuentran notificaciones que coincidan con los filtros, devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."
    3. Si los query parameters de la solicitud están mal formados, devuelve un estado 400 (Bad Request).
2. **POST /api/v1/notifications:** Este endpoint se utiliza para enviar notificaciones a los miembros de un hogar que reciben notificaciones. El cuerpo de la solicitud debe contener los datos necesarios para crear una notificación, según lo especificado en el modelo *PostNotificationRequest*.
    1. Si la ejecución es exitosa, devuelve un estado 201 (Created). Observar que si el hogar no tiene ningún miembro con notificaciones activadas, o si el dispositivo está desconectado, se obtiene también este resultado pues no representa un error ya que simplemente no es necesario enviar las notificaciones.
    2. Si no se encuentra el recurso relacionado con la notificación (el hogar, o el dispositivo con el correspondiente hardware id en los dispositivos del hogar), devuelve un estado 404 (Not Found) con el mensaje "The requested resource was not found."

Además, para todos los endpoints se cumple que:

- Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).
- Si no se incluye Authorization Header, o si se incluye pero el token incluido está mal formado o no corresponde a ninguna sesión, devuelve un estado 401 (Unauthorized).

SessionController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/login

1. **POST /api/v1/login:** Este endpoint se utiliza para iniciar sesión en la aplicación. El cuerpo de la solicitud debe contener el correo electrónico y la contraseña del usuario, según lo especificado en el modelo *LoginRequest*.
    1. Si la autenticación es exitosa, devuelve un estado 200 (OK) con la respuesta *LoginResponse* con el token.
    2. Si no se encuentra el usuario con las credenciales proporcionadas, devuelve un estado 404 (Not Found) con un mensaje de error.
    3. Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).

4. Si el cuerpo de la solicitud está mal formado, devuelve un estado 400 (Bad Request).
5. Si el cuerpo de la solicitud está vacío, devuelve un estado 415 (Unsupported Media Type).

UserController

Aquí se encuentran los endpoints definidos en la ruta base /api/v1/users

1. **GET /api/v1/users:** Este endpoint se utiliza para obtener una lista de usuarios basada en los filtros proporcionados en el cuerpo de la solicitud *GetUsersRequest*.
    1. Si la solicitud es exitosa, devuelve un estado 200 (OK) con la respuesta *GetUsersResponse*.
    2. Si los query parameters de la solicitud están mal formados, devuelve un estado 400 (Bad Request).
    3. Si ocurre un error inesperado, devuelve un estado 500 (Internal Server Error).
    4. Si no se incluye Authorization Header, o si se incluye pero el token incluido está mal formado o no corresponde a ninguna sesión, devuelve un estado 401 (Unauthorized).
    5. Si se incluye Authorization Header con un token válido, pero el usuario correspondiente no tiene suficientes permisos (rol *Administrator*), devuelve un estado 403 (Forbidden).