

**Universidad ORT Uruguay**

**Facultad de Ingeniería**



*Obligatorio 1 Diseño de Aplicaciones 2*

Enzo Izquierdo (283145)

Manuel Graña (285727)

Martín Salaberry (294238)

[Repositorio](#)

2024

# Índice

<b>Introducción</b>	<b>4</b>
<b>Instalación</b>	<b>5</b>
Configuración de SQL Server (Docker)	5
Deploy en IIS	5
<b>Vista de Deployment</b>	<b>6</b>
Descripción de la Arquitectura física	6
Componentes de la Arquitectura	6
<b>Vista de Implementación</b>	<b>7</b>
Diagrama de componentes	7
Arquitectura lógica	7
Independencia de librerías	9
1. BusinessLogic	10
2. DataAccess	10
3. WebApi	10
Justificación de cómo se cumple con la independencia de librerías:	11
Posible Reuso de Componentes en otras aplicaciones	11
<b>Vista Lógica</b>	<b>11</b>
BusinessLogic	12
Diagrama de clases del dominio en BusinessLogic	12
Métricas y principios de paquetes	12
Herencia de Cámara con Device	12
Herencia en OwnedDevice	13
Manejo de notificaciones	14
DataAccess	15
Métricas y principios de paquetes	15
Tablas de la base de datos	16
Herencia de PaginatedRepositoryBase en Repositorios	16
WebApi	17
Métricas y principios	18
Utilización de Polimorfismo	19
Autenticación y Autorización en HomeConnect API	19
Autenticación con "Bearer + Guid":	19
Detalles adicionales:	20
Autorización basada en Permisos	20
Mejoras implementadas en la segunda entrega	20
Impacto de los cambios	21
Decisiones de diseño principales	22
Descripción del mecanismo de acceso a datos utilizado	22
Agrupación de tipos de usuario en User	23
Clase intermediaria entre Home y User	23
Descripción del manejo de excepciones	24
Clase intermediaria entre Device y Home	24
Extensibilidad de Importadores	25
Uso de Factory	26
Extensibilidad de Validadores	26
Criterio de asignación de responsabilidad	27

# Descripción del diseño

## Introducción

HomeConnect es una plataforma digital diseñada para centralizar y gestionar diferentes dispositivos inteligentes en el hogar, creando un entorno integrado y automatizado. Este sistema permite a los usuarios, con distintos roles, gestionar y controlar dispositivos de forma eficiente.

## Funcionalidades Implementadas

Se implementaron todas las funcionalidades previstas para esta entrega.

### 1er instancia

- Registro y gestión de usuarios
- Administración de dispositivos y cuentas de empresas
- Listado y filtrado de cuentas, empresas y dispositivos
- Creación y gestión de hogares, miembros y notificaciones
- Registro de dispositivos inteligentes como cámaras y sensores
- Autenticación de usuarios
- Configuración de notificaciones y permisos para miembros

### 2da instancia

- Implementación de interfaz de usuario
- Asignación de alias a hogares
- Cambiar nombre de dispositivos en un hogar
- Administradores y dueños de empresas pueden operar como dueños de hogar
- Al crear empresas, se define la lógica de validación de modelos que se ejecutará al registrar dispositivos. Nuevas lógicas pueden agregarse en tiempo de ejecución implementando una interfaz.
- Se define la lógica para importar dispositivos pudiendo agregarse nuevas lógicas en tiempo de ejecución.

- Soporta nuevos tipos de dispositivos (sensor de movimiento, lámpara inteligente), se lista el estado de las lámparas y sensores al listar los dispositivos del hogar.
- Agrupación de dispositivos por cuarto.

## Instalación

### Configuración de SQL Server (Docker)

La base de datos del proyecto corre en un contenedor de Docker, por lo que lo primero es configurarlo. Iremos al archivo “Aplicacion/docker-compose.yml” en el directorio raíz y lo podremos configurar. Hay que indicar la contraseña en **SA\_PASSWORD** y el puerto en **ports**. Por defecto son “*Passw1rd*” y “*1433*”, respectivamente.

Luego debemos configurar los connection strings para poder realizar la conexión a este. En la carpeta Aplicacion, en el archivo appsettings.json configuraremos el string para la versión de producción. Buscaremos la línea “Production”. Aquí podremos configurar la IP (Server), nombre de la base de datos (Database), User Id y Password, esta última debe ser la misma que utilizamos en el archivo “/docker-compose.yml”.

**Ubicación de los scripts SQL con datos de prueba:** Se incluyen los backups de una base de datos sin ningún dato adicional (solo con roles, permisos y cuenta de administrador) así como un backup de la base de datos con datos de prueba precargados.

**Cuenta de administrador por defecto (con la que se puede empezar a registrar dueños de empresa u otros administradores):**

- **Administrador:** *admin@admin.com* - contraseña: *Admin123@*

### Deploy en IIS

Para realizar el deploy de la aplicación, debemos tener IIS activado en Windows, con los módulos de *ASP.NET Core Module v2* y *URL Rewrite* debidamente instalados.

Teniendo esto en cuenta, debemos abrir IIS Manager y crear dos sitios: uno para el Frontend y otro para WebApi. Debemos indicar el puerto en el que queremos que corra tanto la WebApi como el frontend, y debemos indicar la ruta donde se encuentra cada aplicación. Para esto último, vamos a mover las carpetas de “*web-api*” y “*frontend*” que se encuentran en la carpeta **Aplicación/** a la ruta *C:/inetpub/homeconnect/* y vamos a indicar la ruta correspondiente a cada aplicación en cada caso.

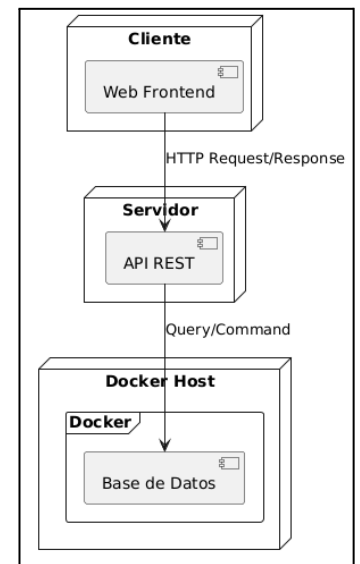
El puerto que se use para la WebApi debe ser configurado en el frontend, para ello se buscará la línea que contiene la siguiente configuración: **apiUrl:"http://localhost:8099"** en el archivo "C:\inetpub\homeconnect\frontend\chunk-2GDBKQSI.js y se modificará el puerto **8099** por el que queremos configurar. De esta forma, cambiando el puerto en un único lugar, ya lo habremos configurado para toda la aplicación.

## Vista de Deployment

### Descripción de la Arquitectura física

La arquitectura física de la aplicación se basa en tres elementos clave, diseñados para garantizar independencia, escalabilidad y un manejo eficiente de los recursos. En este sistema, la API REST actúa como el núcleo de la aplicación, gestionando las solicitudes y respuestas entre el cliente y la base de datos. Esta API está alojada en un servidor dedicado, proporcionando robustez y confiabilidad. Por su parte, la base de datos está desplegada en un contenedor Docker, lo que permite una administración eficiente de los recursos, mayor portabilidad y escalabilidad sencilla a medida que crecen las necesidades del sistema.

El front-end se implementa como una aplicación Angular de una sola página (SPA), diseñada para ofrecer una experiencia de usuario fluida e interactiva. Esta interfaz de usuario implementa todas las funcionalidades descritas en los requisitos del sistema, garantizando una integración total con la API REST. La implementación de Angular como framework permite un desarrollo modular y sostenible, con un enfoque en el rendimiento y la facilidad de mantenimiento.



### Componentes de la Arquitectura

#### 1. Cliente Web Frontend:

- El cliente web fue desarrollado como una aplicación Angular siguiendo un enfoque de aplicación de una sola página (SPA) e integrando PrimeNG (junto con la librería de utilidades CSS PrimeFlex) como biblioteca principal para la interfaz de usuario.

#### 2. API REST:

- La API REST es el intermediario entre el cliente y la base de datos. Implementa endpoints que permiten realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los datos.

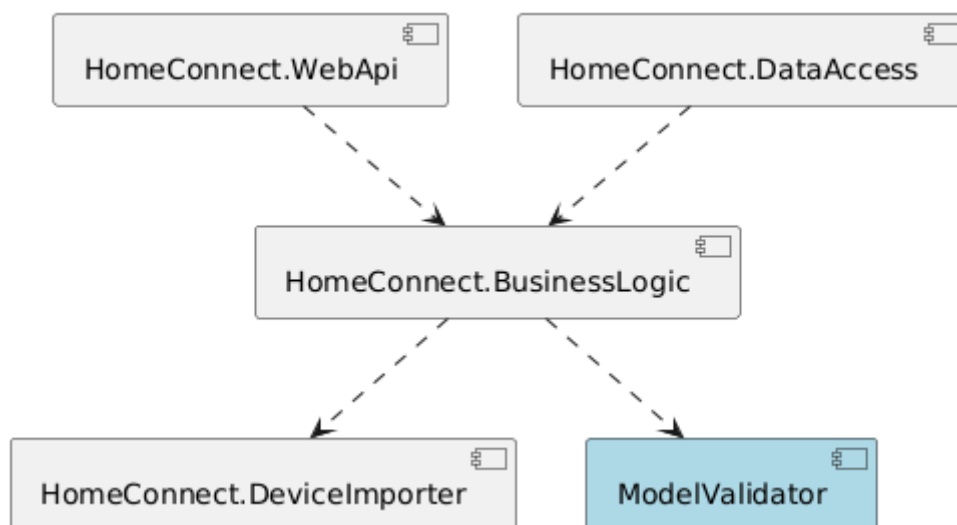
### 3. Base de Datos:

- La base de datos está alojada en un contenedor Docker, lo que facilita su gestión y escalabilidad. Este enfoque permite que la base de datos se ejecute en un entorno aislado, minimizando conflictos y asegurando la integridad de los datos.

## Vista de Implementación

Nuestra solución se organiza en 3 componentes principales: BusinessLogic, DataAccess, WebApi.

## Diagrama de componentes



## Arquitectura lógica

La solución está dividida en capas de manera que agrupen responsabilidades similares, siguiendo el modelo de **Clean Architecture**. En la **capa de presentación**, donde se manejan las interacciones con el usuario, tenemos a **WebApi** que contiene la API web, junto con sus controladores y los filtros (middleware).

Luego está la **capa de persistencia**, que maneja el acceso a la base de datos, donde tenemos a **DataAccess** que gestiona los repositorios y el contexto de la base de datos. Por último

tenemos la **capa de aplicación**, que gestiona la lógica de negocio y el dominio. En esta tenemos a **BusinessLogic** que es donde se encuentran la lógica negocio que abarca los servicios y clases del dominio. Por ultimo, se encuentran DeviceImporter y ModelValidator, paquetes que solo declaran interfaces (es decir son totalmente abstractos y estables. Distinguimos en el diagrama a ModelValidator ya que este contiene una dll que utilizamos en tiempo de ejecución, además de no ser desarrollado por nosotros sino que por terceros. Se cumple el principio de dependencias estables porque ambos son mas estables que BusinessLogic que dependen de el.

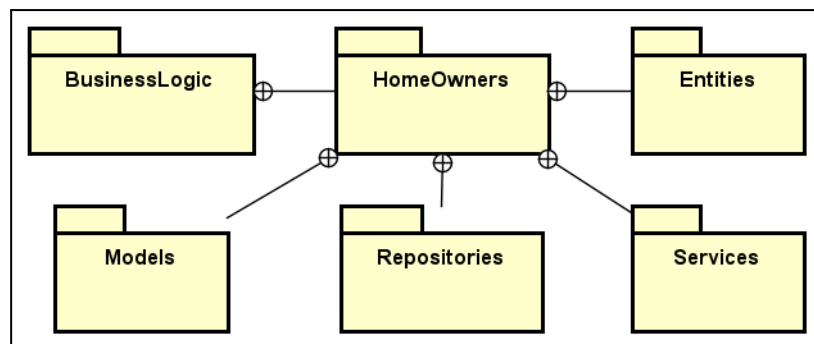
Decidimos no separar el dominio en una capa separada porque no nos aportaba ningún beneficio. El dominio solo va a ser usado por nuestra aplicación, no va a ser reusado por otras, además por la transitividad, aunque lo separemos, los proyectos que dependan de BusinessLogic, que depende del dominio, también dependen de este.

Nos decidimos entonces por organizar nuestras clases en **vertical slices**, agrupando en namespaces las clases que colaboren entre sí para cubrir la misma responsabilidad. En la siguiente tabla ilustramos esto.

Namespace	Responsabilidad
BusinessLogic.Admins	Eliminación de administradores. Obtención de lista de usuarios y empresas.
BusinessLogic.Auth	Manejo de autenticación y autorización.
BusinessLogic.BusinessOwners	Creación de empresas Creación de dispositivos Validación de dispositivos y asignación de validadores
BusinessLogic.Devices	Gestión y validación de datos de Device y Camera Gestión y validación de datos de las relaciones OwnedDevice, LampOwnedDevice y SensorOwnedDevice Importar dispositivos
BusinessLogic.Helpers	Cargar y gestionar implementaciones de interfaces usando reflection.
BusinessLogic.HomeOwners	Creación de Hogares. Creación de OwnedDevices. Gestión y listado de miembros de hogar. Gestión y listado de dispositivos de hogar. Gestión y listado de habitaciones de hogar.
BusinessLogic.Notifications	Creación y asignación de notificaciones Notificar a usuarios que tienen los permisos necesarios para recibir notificaciones

BusinessLogic.Roles	Declaración de las clases de roles y permisos, así como la interfaz del repositorio de roles
BusinessLogic.Users	Creación y validación de datos de User (Administradores, dueños de empresa y dueños de hogar).
DataAccess.Repositories	Implementación de los repositorios declarados como interfaces en los namespaces de BusinessLogic Configuración del contexto de la base de datos
WebApi.Filters	Aplicación de filtros para manejo de autenticación y autorización Manejo de excepciones en las solicitudes
WebApi.Controllers	Gestión de todos los endpoints, cada uno en un namespace llamado de acuerdo a su ruta correspondiente. Ej: WebApi.Controllers.Admin

A su vez, dentro de cada namespace están las interfaces que utilizan (si se cumple el caso que solo ellos la utilicen) tanto de servicios como de repositorios, los servicios que utilizan, las entidades del dominio y los modelos, Dtos que utilizamos para hacer que las funciones no reciban muchos parámetros. HomeOwners, por ejemplo, tiene los 4 posibles subpaquetes dentro.



En esta entrega agregamos puntualmente a BusinessLogic.Devices los namespaces Helpers, donde implementamos una fabrica utilizada por ImporterService para crear Dispositivos (llamando a BusinessOwnerService) según el tipo, e Importer, donde definimos la interfaz IDeviceImporter y el DTO DeviceArgs.

## Independencia de librerías

Para abordar el principio de **independencia de librerías** y minimizar el impacto de los cambios en los componentes físicos de la solución, hemos diseñado la arquitectura en módulos independientes, organizados en **assemblies** separados:



## 1. BusinessLogic

Este paquete contiene la lógica de negocio y las entidades del dominio, así como las **interfaces de los servicios y de los repositorios**. Aquí se define cómo deben comportarse los servicios y cómo interactúan con los repositorios. Nuestra manera de asignar responsabilidades a los servicios es la de asignar responsabilidades similares, no necesariamente uno por recurso, es por esto que los controladores de WebApi en ocasiones pueden llamar a servicios diferentes según el endpoint.

La razón por la cual mantenemos las **interfaces de los repositorios** dentro de BusinessLogic es porque esta capa también contiene las **clases del dominio**, lo cual significa que **DataAccess**, que maneja la persistencia de datos, siempre va a conocer este dominio. No tiene sentido mover estas interfaces a otro assembly, ya que siempre habrá una dependencia hacia BusinessLogic, dado que ambos manejan el dominio.

## 2. DataAccess

Este paquete es responsable de implementar los repositorios que interactúan directamente con la base de datos. Implementa las interfaces que se definen en **BusinessLogic**, permitiendo así un claro desacoplamiento entre la lógica de negocio y la lógica de persistencia.

Al mantener la implementación en este assembly independiente, permitimos que la lógica de acceso a datos pueda ser modificada o reemplazada sin afectar a otras capas del sistema.

## 3. WebApi

Aquí es donde se exponen los servicios a través de endpoints. La WebApi depende de las **interfaces de los servicios** que están en **BusinessLogic**. Usamos estas interfaces para hacer la inyección de dependencias y desacoplar la API de las implementaciones concretas de los servicios.

Si bien **WebApi** podría beneficiarse de que las interfaces de los servicios estuvieran en un assembly separado de BusinessLogic, no vemos una justificación clara para hacerlo.

Actualmente, no estamos considerando múltiples implementaciones de estos servicios, y la complejidad añadida de mover las interfaces no parece aportar beneficios significativos en este contexto.

## Justificación de cómo se cumple con la independencia de librerías:

- **Mínimo impacto de cambios:** Si es necesario modificar la implementación de un repositorio o servicio, podemos hacerlo dentro del assembly correspondiente (**DataAccess** o **BusinessLogic**), sin afectar a otros componentes. Los cambios en el código de la WebApi no impactan en la lógica de negocio o de acceso a datos, y viceversa, debido al uso de interfaces y a la clara separación de responsabilidades.
- **Inyección de Dependencias:** Aprovechamos la inyección de dependencias para desacoplar las implementaciones concretas de las interfaces, asegurando que cualquier cambio en una implementación no tenga un efecto en las demás capas del sistema.

## Posible Reuso de Componentes en otras aplicaciones

El diseño modular facilita el reuso de componentes. El paquete **HomeConnect.BusinessLogic**, ubicado en la capa de **aplicación**, es el más adecuado para reutilizar, ya que sigue el **principio de cierre común (Common Closure Principle)**, ya que en este se define tanto la lógica de negocio como el dominio, cuyos componentes generalmente cambian en conjunto.

En contraste, **HomeConnect.WebApi** (presentación) y **HomeConnect.DataAccess** (persistencia) son paquetes concretos que dependen de detalles específicos de infraestructura, lo que limita su reutilización.

Este diseño permite cambiar las capas de presentación y persistencia sin afectar la lógica de negocio, lo que mejora la sostenibilidad del sistema. **BusinessLogic** es más reutilizable y desacoplado de los detalles técnicos.

## Vista Lógica

La **Vista Lógica** describe la organización interna del software, enfocándose en cómo los módulos y componentes principales interactúan entre sí para satisfacer los requisitos funcionales.

Primero veremos las métricas de los 3 paquetes principales de nuestra aplicación, las cuales comentaremos más a detalle en cada paquete.

Paquete	Cohesión	Abstracción	Inestabilidad	Distancia de la secuencia principal
BusinessLogic	2.7	0.26	0.46	0.2
DataAccess	2.48	0.01	0.99	0
WebApi	1.2	0	1	0

## BusinessLogic

### Diagrama de clases del dominio en BusinessLogic

#### Métricas y principios de paquetes

Las métricas del paquete BusinessLogic reflejan un diseño sólido y extensible, alineándose con varios principios de diseño de paquetes. Su cohesión relacional de 2.7 demuestra el cumplimiento del **Common Closure Principle (CCP)**, al agrupar clases que cambian juntas, como las relacionadas con dispositivos o usuarios. Con una abstracción de 0.26, equilibra clases concretas e interfaces, cumpliendo el **Stable Abstractions Principle (SAP)** al facilitar extensibilidad sin complejidad innecesaria.

El paquete también respeta el **Common Reuse Principle (CRP)**, al organizar subpaquetes, donde las clases son utilizadas juntas de manera consistente, evitando dependencias innecesarias. Su inestabilidad de 0.46 refleja su papel como intermediario entre DataAccess y WebApi, alineándose con el **Stable Dependencies Principle (SDP)** al depender ellos dos de un paquete más estable, en este caso BusinessLogic. Estos valores, junto con una distancia de 0.2 de la secuencia principal, demuestran un diseño reutilizable conforme al **Release Reuse Equivalency Principle (REP)** y libre de ciclos según el **Acyclic Dependencies Principle (ADP)**.

#### Herencia de Cámara con Device

La clase Device fue elegida como una clase padre debido a que contiene todos los atributos comunes entre sensores y cámaras, mientras que los atributos específicos de las cámaras están incorporados en la clase Camera de la que extiende Device.

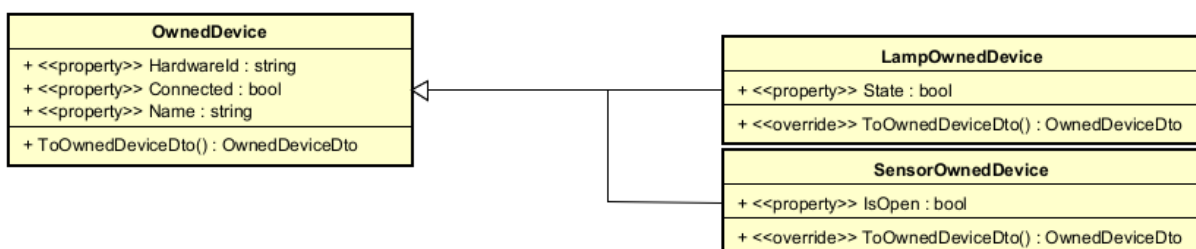
La estructura de herencia elegida garantiza que los dispositivos compartan características clave, mientras que los detalles específicos de cada tipo de dispositivo se manejan en las clases derivadas, promoviendo la reutilización de código y asegurando un diseño coherente, mantenible y extensible. Más adelante si los atributos de Sensor cambian, se podría

implementar Sensor como extensión de Device. De igual manera se podría implementar nuevos tipos de dispositivos como extensiones de Device en caso de necesitar más atributos o como parte de este, añadiendo su nombre a DeviceTypes, un enum que se utiliza para validar el tipo de dispositivo. Este mismo es el que utilizamos para mostrar los dispositivos soportados por el sistema.

## Herencia en OwnedDevice

Para manejar el nuevo requerimiento de que las lámparas y sensores muestren su estado, y anticipándonos a la posibilidad de que nuevos tipos puedan necesitar almacenar datos específicos, implementamos una jerarquía de herencia con OwnedDevice como clase base, y LampOwnedDevice y SensorOwnedDevice como clases derivadas.

Se creó la función ToOwnedDeviceDto, que devuelve un DTO capaz de contener todos los datos posibles de cualquier OwnedDevice. Los campos específicos de ciertos dispositivos (como el estado de lámparas y sensores) se definen como nulleables, ya que no son comunes a todos los tipos. Las clases hijas sobrescriben este método, asignando los valores específicos que gestionan al DTO correspondiente.



De este modo, si se necesita manejar un nuevo tipo de OwnedDevice, solo es necesario:

- Agregar los nuevos datos al DTO (lo cual sería necesario para devolverlos).
- Implementar una nueva clase hija que sobrescriba el método ToOwnedDeviceDto para incluir dichos datos.

Además, al encontrarse tanto OwnedDevice, como sus clases hijas y el DTO en el paquete BusinessLogic.Devices, se cumple el Principio de Clausura Común, ya que estos elementos cambiarán en conjunto.

Este enfoque facilita la extensión y mantiene el diseño organizado y escalable.

## Implementación de Factory

Además, debido a estos nuevos tipos implementamos, OwnedDeviceFactory, en esta clase se crea una instancia de OwnedDevice o de sus clases especializadas como LampOwnedDevice o SensorOwnedDevice, dependiendo del tipo de dispositivo. Esto facilita la creación de dispositivos sin necesidad de cambiar el código cliente cada vez que se agregue un nuevo tipo de dispositivo.

El uso de factories también favorece el principio de Responsabilidad Única (SRP) al delegar la creación de objetos a clases especializadas. Además, al utilizar el patrón Factory, se minimiza la duplicación de código y se facilita la extensión del sistema en el futuro.

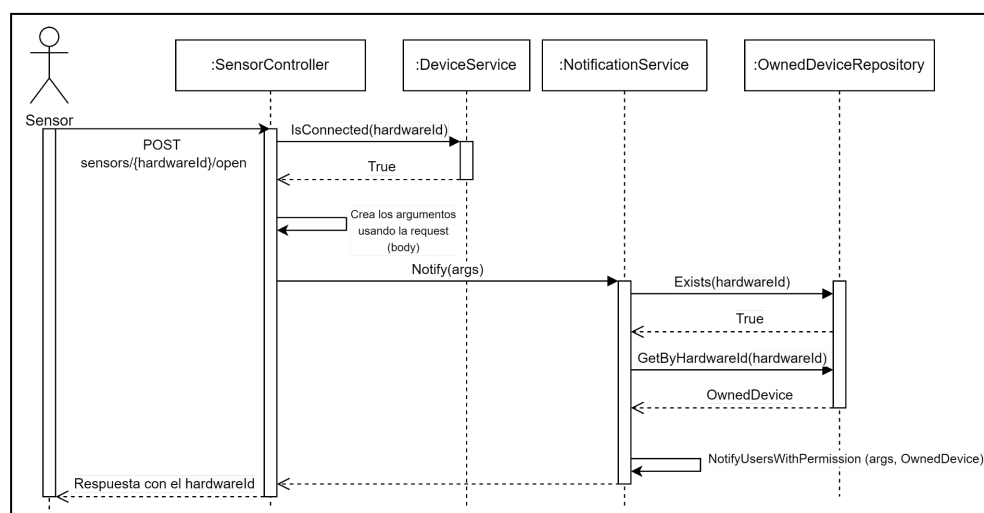
### Manejo de notificaciones

La estructura de notificaciones complementa la arquitectura de dispositivos. Gracias a la herencia proporcionada por Device, las notificaciones se pueden asociar con cualquier tipo de dispositivo (sea una cámara, un sensor, o cualquier futuro tipo de dispositivo que se añada al sistema), ya que todas heredan de la clase base Device. Esto significa que cuando se añaden nuevos tipos de dispositivos al sistema, no es necesario modificar la lógica de notificaciones, manteniendo así el principio de abierto/cerrado.

La clase Notification actúa como el punto central para el manejo de notificaciones, actuando como una asociación entre los dispositivos y usuarios del sistema, añadiendo atributos relativos a la notificación, las notificaciones se asocian a un OwnedDevice, clase que asocia dispositivos y hogares, de la que hablaremos mas adelante.

El proceso de notificación se inicia a través del método Notify del NotificationService. Este método verifica la existencia del dispositivo, obtiene los miembros que deben ser notificados basándose en sus permisos, y crea las notificaciones correspondientes.

Diagrama de secuencia centrado en el controlador, mostrando la comunicación entre los tres componentes para notificar a los usuarios:



## .DataAccess

.DataAccess se encarga de implementar las interfaces de repositorios que son declaradas en BusinessLogic, escondiendo en sí la complejidad del manejo de datos en la base de datos. Como no nos aporta mostrar un diagrama de clases de DataAccess, el siguiente diagrama de componentes ilustra lo que sucede con UserRepository.



Los **repositorios** en nuestro proyecto actúan como adaptadores entre los servicios de negocio y el acceso a la base de datos, aplicando el **patrón Adapter** para conectar interfaces que, de otro modo, serían incompatibles. Los repositorios implementan interfaces comunes que permiten que los servicios interactúen con los datos sin conocer los detalles específicos de cómo se acceden o manipulan en la base de datos. De este modo, se abstrae la complejidad del acceso a los datos, lo que facilita que las clases con interfaces diferentes trabajen juntas sin modificar su código original.

Esta implementación del patrón Adapter también cumple con el **principio de abierto/cerrado (OCP)**, ya que los servicios están abiertos a la extensión sin necesidad de modificar su lógica interna. Es posible cambiar la implementación de un repositorio (por ejemplo, pasar de una base de datos SQL a NoSQL) sin afectar la lógica de los servicios. Los repositorios permiten que los servicios interactúen con diferentes tipos de fuentes de datos, manteniendo el sistema flexible y extensible sin cambios disruptivos en el código de las clases que manejan la lógica de negocio.

Adicionalmente, como manejamos los permisos en la base de datos, estos se inicializan los en DataAccess, ya siendo asignados a roles, lo cual permite que la asignación de responsabilidades sea dinámica y no afecte a otras capas del sistema.

## Métricas y principios de paquetes

El paquete DataAccess muestra una alta inestabilidad de 0.99, característica esperada en su rol como capa adaptadora para interactuar con la base de datos, sin afectar directamente a las capas superiores. Vemos cómo se cumple el **Stable Dependencies Principle (SDP)** al

dependen de **BusinessLogic**, un paquete más estable, el cual define las interfaces que se implementan. Además está claro el cumplimiento de **Acyclic Dependencies Principle (ADP)**, ya que la dependencia con BusinessLogic es unidireccional, evitando ciclos.

Con una cohesión relacional de 2.48, el paquete **DataAccess** agrupa clases relacionadas con la persistencia, como el contexto y los repositorios, cumpliendo con los principios de **Release Reuse Equivalence (REP)**, **Common Reuse (CRP)** y **Common Closure (CCP)**. Estos principios se cumplen porque el paquete se puede liberar como una unidad coherente y reutilizable, agrupa clases que se utilizan juntas y asegura que los cambios en la persistencia afecten solo a este paquete, minimizando el impacto en otras capas del sistema.

Su abstracción es baja (0.01), lo cual es adecuado para su naturaleza concreta, alineándose con **Stable Abstractions Principle (SAP)**, ya que es inestable y concreta. Finalmente, la distancia de 0 a la secuencia principal refleja simplicidad y robustez, alineándose con los principios de estabilidad y abstracción en un diseño limpio y funcional.

### **Tablas de la base de datos**

Herencia de PaginatedRepositoryBase en Repositorios

En vistas de que la estructura de los algoritmos de paginación eran los mismos en las tres clases que necesitaban paginar (BusinessRepository, DeviceRepository, UserRepository) decidimos implementar el patrón Template Method con la finalidad de reutilizar código y reducir la complejidad.

Dividimos el algoritmo de paginación y filtrado en los siguientes pasos:

1. **GetQueryable():** Retorna el IQueryable donde se van a aplicar los filtros y la paginación posteriormente. Por ejemplo, en el caso que queremos obtener una lista de usuarios con paginación y filtrado, retornamos el propio DbSet de los usuarios. *Este método debe ser implementado por las clases que heredan.*
2. **ApplyFilters():** Aplica los filtros. Nuevamente, retorna un IQueryable luego de aplicarle los filtros necesarios. *Este método debe ser implementado por las clases que heredan.*
3. **Paginate():** Una vez obtenidos y filtrados los datos, este método se encarga de devolvernos la lista ya paginada con información el tamaño de página, la página

actual y el total de páginas en un record PagedData. *Este método ya contiene una implementación por defecto que es usada por todas las clases hijas.*

De esta manera, se expone el template method **GetAllPaged()** y las clases que heredan solo tienen que redefinir **GetQueryable()** y **ApplyFilters()**, sin necesidad de redefinir el orden del algoritmo ni el método **Paginate()**.

### Diagrama

Finalmente, es oportuno mencionar que nuestra aplicación utiliza un **modelo de base de datos conectado**. A su vez, hemos configurado el ciclo de vida de los repositorios como Scoped, es decir, cada instancia del contexto se crea al inicio de la solicitud y se destruye al final de la misma. Esto implica que las entidades que se recuperen desde la base de datos permanecen vinculadas al Context mientras la request está activa, ofreciendo así varias ventajas. Primero, permite que cualquier modificación, inserción o eliminación realizada en las entidades sea trackeada por EF Core, sin necesidad de realizar ninguna operación adicional. Además, el ciclo de vida Scoped es ideal en este escenario porque cada request HTTP recibe su propio DbContext, lo que evita compartir el contexto entre múltiples solicitudes. A diferencia de un ciclo de vida Transient, donde se crea una nueva instancia cada vez que se inyecta el servicio, el ciclo Scoped nos garantiza ser más eficientes (minimizar las interacciones con la base de datos) sin sacrificar control (nos permite tener un control más preciso sobre los cambios en las entidades dentro de esa solicitud).

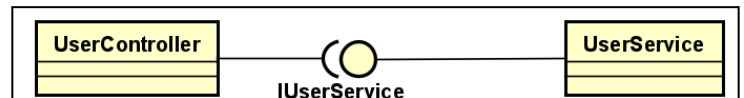
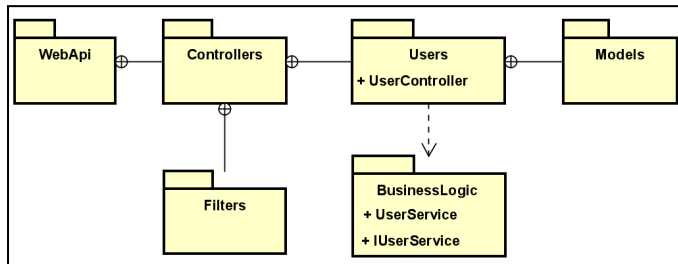
## WebApi

Por último tenemos el namespace WebApi, el cual tiene 2 sub-namespaces principales. **Controllers** contiene las clases de que implementan los endpoints que declaramos en el archivo de especificación de la API. Estos se comunican con los servicios a través de sus interfaces. Existe solo una relación entre clases de WebApi, CameraController y SensorController ya que extienden BaseDeviceController.

Por otro lado tenemos **Filters**, son 4. Estos se encargan de abstraer al resto de clases de: la verificación de la sesión para identificar a los usuarios, la verificación de permisos generales y de permisos relacionados a un hogar. Por último un filtro que se encarga de manejar las excepciones.



Al igual que con DataAccess, no vemos conveniente diagramar las clases de WebApi, ya que no se relacionan entre sí, por lo que usaremos de ejemplo UserController, que utiliza la interfaz brindada e implementada en BusinessLogic, la cual le es inyectada. En todos los controllers se cumple la misma regla, utilizan interfaces de servicios que son brindadas e implementadas por BusinessLogic, las cuales les son inyectadas.



## Métricas y principios

Las métricas de **WebApi** reflejan un diseño adecuado para su propósito actual. La cohesión relacional de 1.2, aunque ligeramente por debajo del límite ideal de 1.5, es razonable en este caso, ya que, a pesar de que la interacción entre las clases sea baja, los controladores están diseñados como puntos de entrada específicos cuya única responsabilidad es delegar llamadas a los servicios de **BusinessLogic**. Esto cumple con el **Release Reuse Equivalency Principle (REP)**, ya que su propósito común es ser un punto de entrada para las solicitudes HTTP y delegar las llamadas a la lógica de negocio, lo que a su vez permite que **WebApi** se libere y reutilice como una unidad coherente, cumpliendo con el **Common Reuse Principle (CRP)**. Además, cumple con el **Common Closure Principle (CCP)**, ya que los controladores cambian únicamente debido a cambios en las solicitudes y no por cambios en otras capas del sistema.

La **abstracción** de 0 también es adecuada, ya que **WebApi** se enfoca en la implementación concreta de los endpoints, lo que es congruente con el **Stable Abstractions Principle (SAP)**, pues no necesita ser abstracto para ser utilizado. La **inestabilidad** de 1 (máxima inestabilidad) no representa un problema significativo, ya que ningún paquete depende directamente de **WebApi**. Esto cumple con el **Acyclic Dependencies Principle (ADP)**, ya que no existen ciclos de dependencia, y con el **Stable Dependencies Principle (SDP)**, dado que **WebApi** depende de **BusinessLogic**, que es un paquete más estable. Aunque el frontend es el único consumidor actual, este solo es utilizado por nosotros, y cualquier cambio en los endpoints puede ser coordinado directamente.

Finalmente, se ve una distancia 0 de la secuencia principal indicando que el diseño está alineado con las recomendaciones del modelo de estabilidad y abstracción: los controladores no están abstraídos ni deben serlo, cumpliendo bien con su propósito.

Este depende de BusinessLogic, ya que en ese paquete están los servicios con los cuales los controladores se comunican, por lo tanto se cumple el principio de dependencias estables, ya que BusinessLogic es más estable que el.

## Utilización de Polimorfismo

En nuestro diseño, el **polimorfismo** se ha utilizado para mejorar la flexibilidad y capacidad de extensión del sistema, se puede ver especialmente en los controladores de **cámaras** y **sensores**, que heredan de una clase base llamada **BaseDeviceController**. Esta herencia permite que ambos controladores compartan funcionalidades comunes, como conectar y desconectar dispositivos, evitando la duplicación de código y promoviendo una estructura más limpia y organizada. Además, al definir las características comunes en un solo lugar, facilitamos la integración de nuevos tipos de dispositivos en el futuro; por ejemplo, si deseamos agregar un termostato, solo necesitaremos crear un nuevo controlador que herede de **BaseDeviceController**, y este ya tendrá el endpoint para conectarlo y desconectarlo. Este enfoque no solo mejora la mantenibilidad del sistema al garantizar un comportamiento coherente entre los dispositivos, sino que también sienta las bases para un crecimiento del sistema en adelante.

## Autenticación y Autorización en HomeConnect API

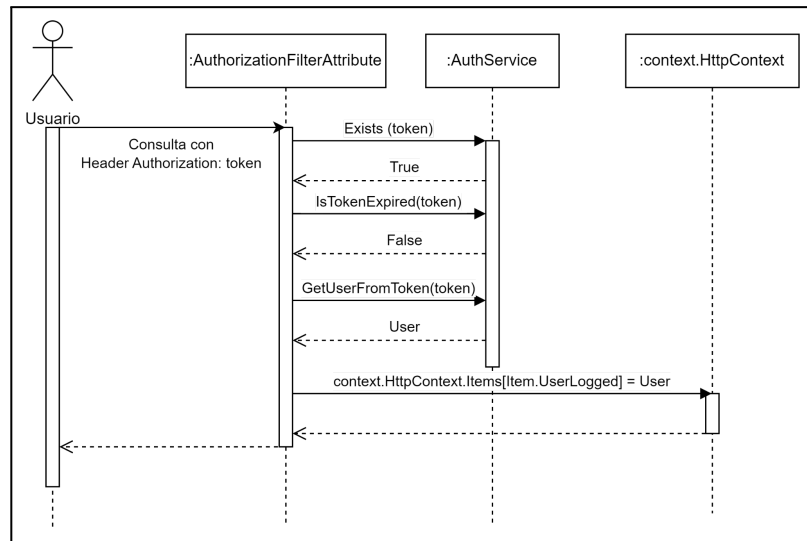
### Autenticación con "Bearer + Guid":

**Flujo:** Los usuarios inician sesión mediante un endpoint ([/auth](#)), proporcionando sus credenciales. Si son válidas, la API genera un token en formato "Bearer + Guid". Este token se guarda en la base de datos asociado al usuario y tiene una fecha de expiración.

**Verificación del Token:** Para cada solicitud protegida, el cliente envía el token en el encabezado HTTP ([Authorization: Bearer <Guid>](#)). La API, a través de un **middleware de autenticación**, revisa la base de datos para comprobar la validez del token y verifica si está expirado. Si el token es válido y no ha expirado, la solicitud continúa su ejecución.

## Detalles adicionales:

- El middleware es responsable de interceptar todas las solicitudes entrantes que necesiten autorización y realizar las verificaciones necesarias.
- TokenRepository contiene información sobre los tokens, incluyendo la fecha de expiración.
- Si el token ha expirado o no es válido, el middleware devuelve una respuesta de error indicando que la autenticación falló.



## Autorización basada en Permisos

- Protección de endpoints: Utilizamos AuthorizationFilter y HomeAuthorizationFilter, otros middlewares, para asegurar que solo los usuarios con los permisos necesarios puedan acceder a ciertos endpoints. Por ejemplo:
  - Solo los usuarios con el System Permission “create-business-owner” pueden crear cuentas de dueños de empresas.
  - Solo los dueños de hogares con el Home Permission “get-devices” pueden listar dispositivos en este.

## Mejoras implementadas en la segunda entrega

### Filtros pasados como argumentos (FilterArgs)

En esta mejora, se refactorizó el manejo de parámetros en varias funciones, especialmente en métodos que requieren múltiples filtros y parámetros, como GetBusinesses o GetUsers. En lugar de pasar varios argumentos a las funciones, se introdujo FilterArgs, que agrupa todos los parámetros necesarios. Encapsulando los parámetros de filtrado y paginado, se facilita la extensión del sistema y se mejora la mantenibilidad al agregar o modificar filtros sin afectar la firma de las funciones.

## Reubicar lógica en los controladores

Se refactorizaron controladores para eliminar la lógica de negocio existente en ellos, trasladandola a los servicios correspondientes. Dejando a los controladores con su responsabilidad de delegar las llamadas a la API hacia los servicios. Esto hace que el código sea más comprensible, pues cada clase tiene una única responsabilidad (cumple el principio SRP).

## Reubicar lógica en repositorios

Refactorizamos los repositorios para eliminar toda lógica de negocio en ellos, asegurando que estos se mantengan limpios y enfocados únicamente en su responsabilidad (operaciones CRUD). Esto mejora significativamente el proyecto, siguiendo el principio de responsabilidad única (SRP).

## Mejoras en Notification Controller

En NotificationController, movimos la lógica de validación del formato de la fecha a NotificationService para un mejor manejo de responsabilidades y para dejar el controller lo más limpio posible. Además, a la hora de obtener las notificaciones teníamos el problema de tener que marcarlas como leídas, por lo que teníamos dos llamadas a servicios: en una, obteníamos las notificaciones, y en otra, las marcamos como leídas. Arreglamos esto haciendo que Notification implemente ICloneable, permitiendonos clonar las notificaciones. De este modo, retornamos una copia de las notificaciones y podemos marcarlas como leídas en una única llamada al servicio.

## Impacto de los cambios

Nombre Resumido	Impacto
Alias para Hogares	Mínimo. Se agregó un atributo NickName a Home.
Renombrar Dispositivos	Mínimo. Se agregó un atributo Name a la clase de asociación entre Device y Home, OwnedDevice. Esto permite que el nombre sea único para cada conexión.
Roles Múltiples	Mínimo. Se reemplazó el atributo de rol por una lista de roles, y cómo nos manejamos con permisos, podemos conocer si el usuario tiene el permiso necesario iterando sobre los mismos.
Agrupación de Dispositivos por cuarto	Se implementó una nueva clase de dominio Room que tiene la responsabilidad de agrupar dispositivos de un hogar. Realizamos las validaciones de atributos a nivel de las clases del dominio, por lo que también agregamos dicha responsabilidad a esta clase. Para configurar la relación One to Many con Home, agregamos una lista de Rooms como atributo en Home.

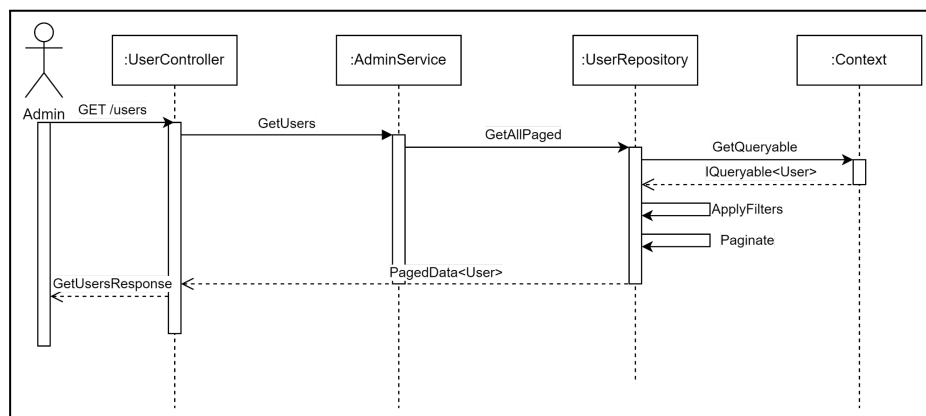
Validación Dinámica de Modelos	Se agregó validador a las empresas, el cual se obtiene consultando a ValidatorService. Al agregar un dispositivo este verifica si es válido.
Importación Dinámica de Dispositivos	Se obtiene una lista de DeviceArgs usando la función declarada en la interfaz y se llama a los métodos de BusinessOwnerService creados anteriormente para crear dispositivos o camaras (debido a la diferencia en sus datos).
Soporte para Nuevos Tipos de Dispositivos	Se agrega su tipo al enum de DeviceTypes. Se agrega una clase que hereda de OwnedDevice en caso de que se tengan que manejar datos sobre su estado. <a href="#">Explicado mas adelante.</a>

## Decisiones de diseño principales

### Descripción del mecanismo de acceso a datos utilizado

- **Controladores:** son el punto de entrada para las request en nuestra web API. Los controladores manejan las solicitudes HTTP y retornan respuestas HTTP. Llamamos a servicios para acceder a la business logic.
- **Servicios:** encapsulan la business logic de la aplicación. Coordinan tareas y delegan trabajo a repositorios.
- **Repositorios:** abstraen la lógica de acceso a la base de datos, brindando métodos para interactuar con esta.
- **Context:** la clase Context es parte de Entity Framework, la cual es una ORM (Object Relational Mapper) que permite que trabajemos con una base de datos utilizando objetos.
- **Entidades:** son el dominio, mapean clases a tablas en la base de datos.

Se realiza una solicitud a la API web y es recibida por un controlador. El controlador llama a un método en un servicio, pasando los parámetros necesarios. El servicio utiliza un repositorio para obtener datos de la base de datos. El repositorio utiliza el Contexto para consultar la base de datos y mapear los resultados a las entidades. El repositorio devuelve los datos al servicio. Luego, el servicio puede aplicar lógica adicional sobre los datos. El servicio pasa a devolver el resultado al controlador. El controlador empaqueta el resultado en una respuesta HTTP y lo envía finalmente al cliente.

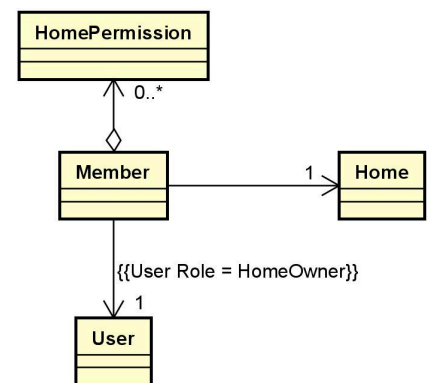


## Agrupación de tipos de usuario en User

Decidimos unificar los tres tipos de usuarios registrados en la plataforma (Administrador, Dueño de empresa y Dueño de hogar) en una sola clase, ya que comparten prácticamente todos sus atributos, excepto la foto de perfil, la cual optamos por hacerla un atributo nullable. No consideramos necesario crear una clase separada para HomeOwner solo por ese pequeño detalle, ya que agregar complejidad adicional al modelo por un atributo no aportaría un beneficio significativo. Sin embargo, esta decisión de consolidar las clases nos llevó a un reto con respecto a la gestión de **roles y permisos**. Dado que todos los usuarios ahora pertenecen a la misma clase, no podemos asignarles permisos directamente. En lugar de ello, implementamos un sistema de roles donde cada usuario tiene una lista de roles (Administrador, Dueño de empresa o Dueño de hogar). Cada rol está asociado con una lista de permisos, que se configura al momento de crear la base de datos. En cada endpoint de la aplicación, verificamos si el usuario tiene los permisos necesarios en alguno de sus roles para realizar la acción solicitada, asegurando así un control de acceso flexible y seguro sin necesidad de clases adicionales para cada tipo de usuario, permitiéndonos en un futuro agregar mas tipos de usuario sin tener que cambiar la clase User.

## Clase intermediaria entre Home y User

Otros problemas que se nos presentaron fueron: cómo hacer posible que un usuario pertenezca a varios hogares y el cómo manejar los permisos que tiene un usuario sobre un hogar específico, pues cada hogar tiene miembros, y estos pueden tener o no ser capaces de añadir dispositivos, obtener la lista de dispositivos del hogar y recibir notificaciones.



En nuestra solución, un hogar tiene una lista de miembros (Member). El rol de la clase Member es vincular a un usuario con un hogar, definiendo también qué tipo de permisos tiene en este, contando con una lista de HomePermission, clase encargada de almacenar los permisos específicos que un miembro puede tener.

Al separar los permisos en HomePermission pudimos diferenciar bien a los permisos de un hogar del resto de permisos existentes en nuestro sistema. También ganamos flexibilidad pues diferentes miembros tienen permisos distintos sin que estos estén vinculados directamente a User.

Al introducir Member como intermediario entre Home y User, fomentamos la separación entre la información relacionada con los usuarios y la información específica de los hogares, reduciendo el acoplamiento entre estas entidades. Además le permitimos a un usuario “ser” Member de más de un hogar, con sus permisos correspondientes, solucionado el problema inicial.

## Descripción del manejo de excepciones

En el paquete **WebApi**, implementamos un filtro llamado *ExceptionHandler*, cuya finalidad es interceptar todas las excepciones no manejadas y comunicarlas al usuario final. Este filtro nos proporciona una manera centralizada de gestionar las excepciones, asignándoles códigos de error HTTP correspondientes.

El *ExceptionHandler* utiliza un diccionario que mapea las excepciones soportadas a sus respectivos códigos de error HTTP. En caso de que una excepción no se encuentre en este diccionario, se retorna un código de error genérico 500 (Internal Server Error).

Esta implementación garantiza una respuesta consistente y controlada ante errores inesperados, mejorando la experiencia del usuario y la mantenibilidad del sistema.

La única excepción a esto es cuando se envían tipos de datos incorrectos en el cuerpo de una solicitud (por ejemplo, un número en lugar de un bool), se generan excepciones que no están siendo controladas. Aunque tenemos un filtro de excepciones (*ExceptionHandler*) que asigna códigos de error HTTP a las excepciones definidas, este no captura las excepciones relacionadas con tipos de datos inválidos, lo que provoca respuestas inesperadas en herramientas como Postman.

Dado que la API está destinada a ser usada por desarrolladores, consideramos que las excepciones detalladas proporcionadas automáticamente en estos casos son útiles para identificar y corregir errores en las solicitudes. A futuro, una solución posible sería aceptar todos los datos como strings nulleables y parsearlos internamente, lo que daría más control y evitaría estos errores de tipo no manejados.

## Clase intermediaria entre Device y Home

Una de las funcionalidades que se nos pidió fue la asociación de dispositivos a hogares. La solución más simple hubiera sido agregar una lista de dispositivos directamente en la entidad

**Hogar.** Sin embargo, esto presentaba el problema de no poder asociar el mismo dispositivo varias veces al mismo hogar, algo que podría ser necesario en algunos casos. Para resolver esto, optamos por crear una clase intermedia llamada **Own Device**, que tiene referencias tanto al **Hogar** como al **Dispositivo**, un estado de conexión y un ID único. Esta estructura nos permitió cumplir con el requisito de tener múltiples asociaciones entre dispositivos y hogares, manteniendo la flexibilidad necesaria.

Esta decisión refuerza varios principios clave. Primero, el **Principio de Responsabilidad Única (SRP)** se respeta al separar las responsabilidades: el **Hogar** no gestiona directamente las asociaciones con dispositivos, sino que delega esta tarea a la clase **OwnedDevice**, que se encarga de gestionar las conexiones y el estado de los dispositivos. Además, siguiendo el **Principio de Abierto/Cerrado (OCP)**, esta estructura es fácilmente extensible. Si en el futuro la empresa quisiera agregar más características a la asociación entre dispositivos y hogares (como historial de conexiones o tipos de dispositivos), podemos extender **OwnedDevice** sin modificar la clase **Hogar**.

## Extensibilidad de Importadores

Para agregar un nuevo importador, es necesario crear una clase que implemente la interfaz `IDeviceImporter`. Esta clase la creamos en un proyecto aparte para no brindarle acceso a nuestro código a quienes vayan a implementar importadores nuevos. Esta interfaz define dos métodos:

- `List<DeviceArgs> ImportDevices(Dictionary<string, string> parameters)`
- `List<string> GetParams()`

Cada importador puede requerir distintos datos, por lo que la función `GetParams()` debe devolver los nombres de los parámetros necesarios para su funcionamiento. Actualmente, todos los parámetros se definen como `string`, lo que implica que es responsabilidad del desarrollador realizar el parsing adecuado en su implementación.

En este contexto, el método `ImportDevices()` recibe un diccionario clave-valor donde:

- La clave es el nombre del parámetro.
- El valor es el dato asociado.

Este método debe devolver una lista de `DeviceArgs`, que contiene todos los parámetros para los tipos de dispositivos actualmente disponibles. Los parámetros que no sean comunes a



todos los dispositivos deben definirse como nulleables. Esto permite que en el frontend se genere dinámicamente una lista de campos de texto basada en los valores obtenidos de `GetParams()`.

Adicionalmente, todos los archivos desde los que se desee importar se almacenan en la carpeta `ImportFiles`, permitiendo al sistema usarlos en tiempo de ejecución usando reflection.

Por lo tanto, el desarrollador, al obtener el nombre del archivo, debe usar la ruta:

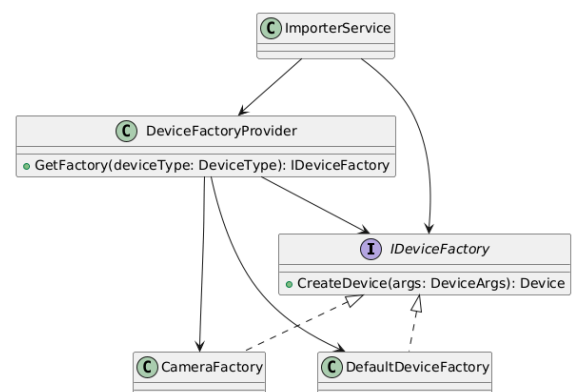
```
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "ImportFiles", [nombre del archivo])
```

Así mismo, para agregar el importador creado, debe compilarlo y poner el dll obtenido en la carpeta “`Importers`”.

Este enfoque asegura extensibilidad y una integración sencilla para nuevos tipos de importadores.

### Uso de Factory

Se crea la clase `IDeviceFactory`, la cual define un método para crear dispositivos. Luego creamos las dos implementaciones concretas de esta interfaz, `DefaultDeviceFactory` y `CameraFactory`, así, pudiendo crear camaras o dispositivos dado que recibimos un `DeviceArgs` que tiene los datos obtenidos de la importacion. Se implementa una tercera clase, `DeviceFactoryProvider`, que dado un tipo de dispositivo nos devuelve la factory correspondiente. Se delegan las responsabilidades crear dispositivos según el tipo a estas clases.



## Extensibilidad de Validadores

A diferencia de con los importadores, la interfaz que utilizamos para los validadores fue brindada por terceros, por lo que solo contamos con su dll compilado. En esta se define la función `EsValido`, que dado un model retorna un booleano. Para trabajar con esto, al igual que con los importadores, utilizamos reflection, permitiéndonos crear nuevas implementaciones y utilizarlas en tiempo de ejecución. Para agregarlo, se debe compilar la implementación y agregar el dll obtenido a la carpeta “`Validators`”.

Al momento de crear una nueva empresa, dinámicamente desplegamos la lista de validadores, dando la opción al usuario de elegir uno. Al elegirlo, se le asigna el id del validador a la empresa creada, pudiendo cambiarlo más adelante.

Para no brindarle acceso a nuestro código a quienes creen implementaciones, hicimos un proyecto aparte donde está la dll.

## Criterio de asignación de responsabilidad

En la asignación de responsabilidades, seguimos el **principio de responsabilidad única (SRP)** para asegurar que cada componente del sistema tenga una única razón para cambiar. Los **controladores** se encargan de delegar las llamadas a la API hacia los servicios, limitándose a gestionar las solicitudes HTTP sin involucrarse en la lógica de negocio. Los **servicios**, por su parte, interactúan con los repositorios y aplican la lógica necesaria sobre los datos, siendo responsables de coordinar las operaciones del sistema.

Los **repositorios** se encargan de implementar las operaciones **CRUD** (Crear, Leer, Actualizar y Borrar) y abstraen los detalles del acceso a la base de datos, cumpliendo con el patrón de repositorio al proporcionar un acceso centralizado a los datos sin que los servicios necesiten conocer los detalles de cómo se gestionan estos datos. Además, los **filtros** actúan como middleware, delegando en sí el control de aspectos transversales como la autenticación, la autorización y la gestión de excepciones, asegurando que el flujo principal de la lógica de negocio en los servicios permanezca limpio y enfocado en las reglas específicas de la aplicación

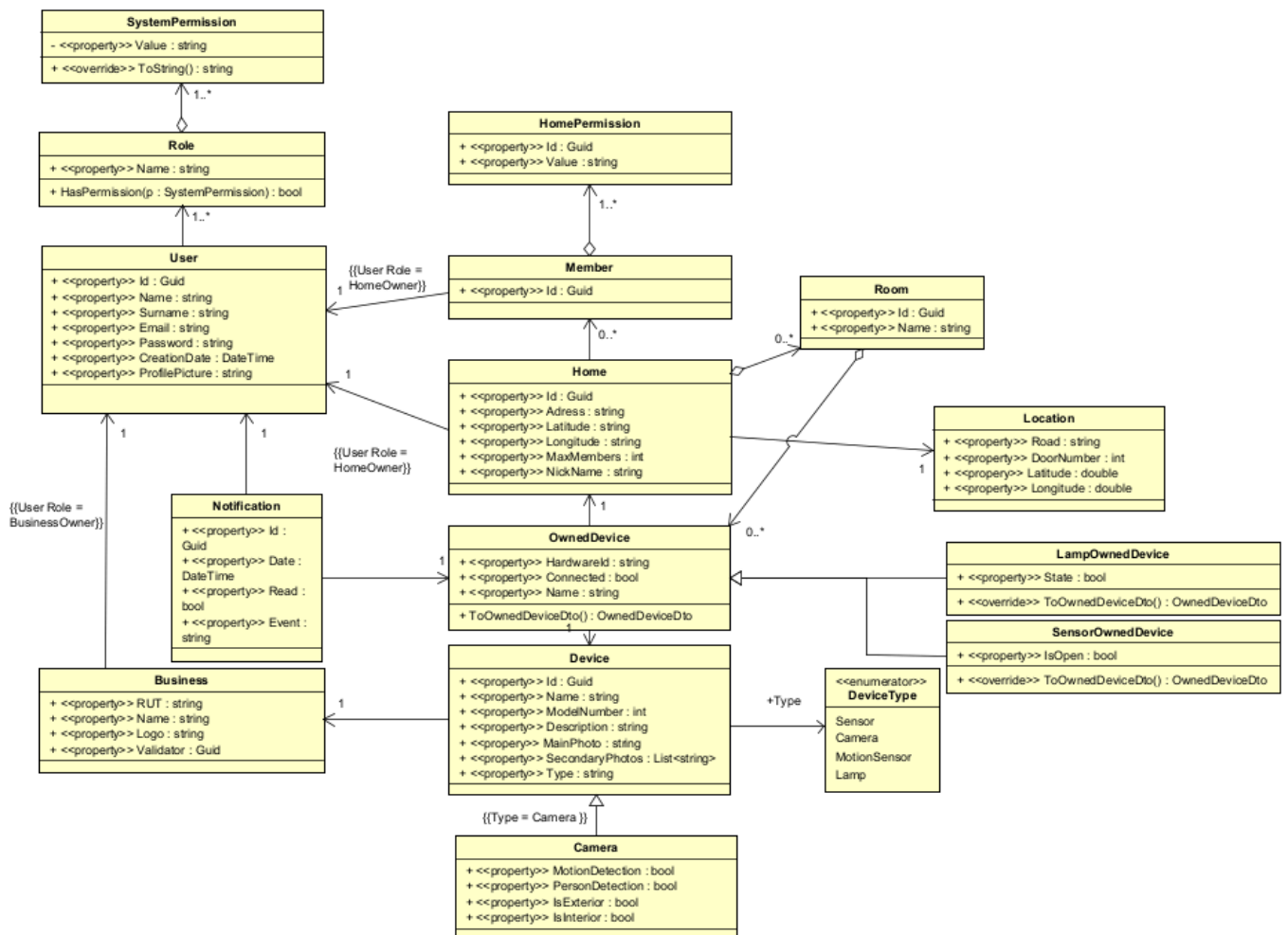
## Anexo

### Índice

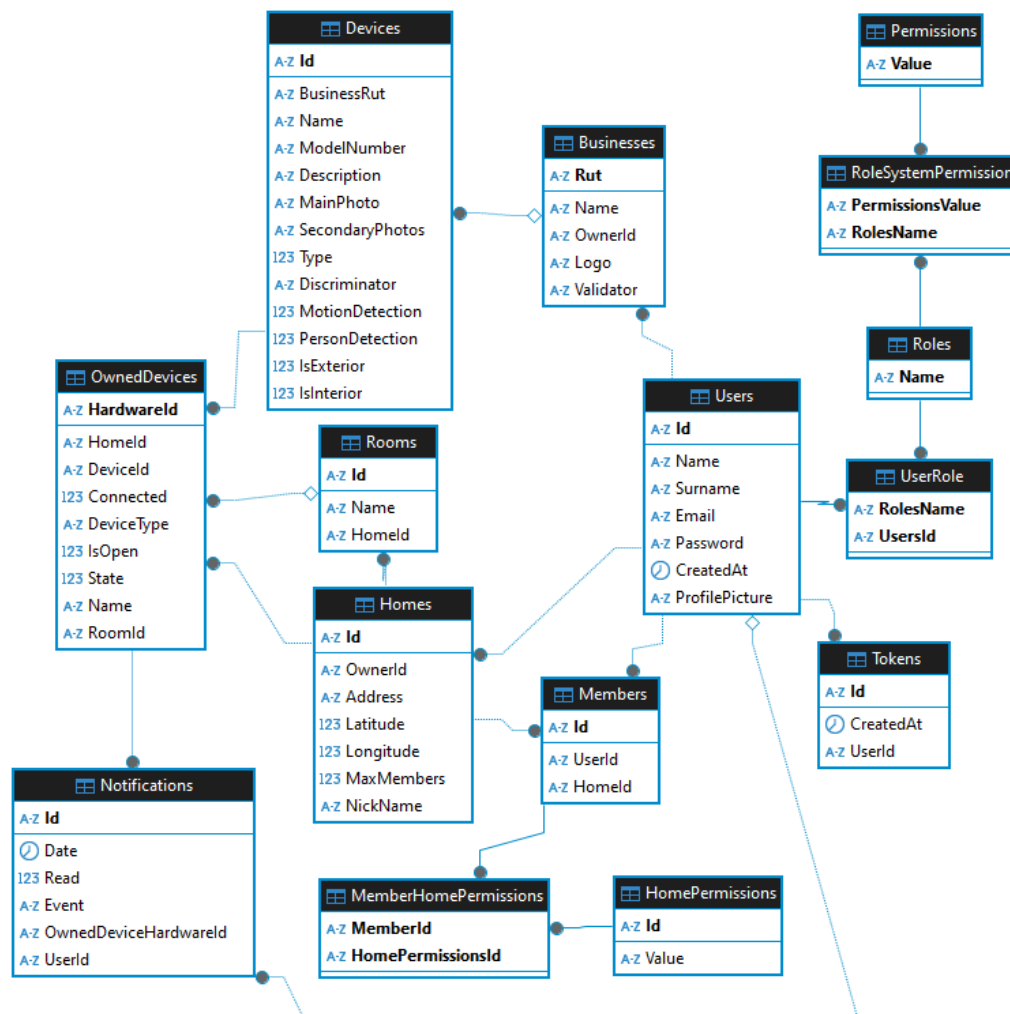
<b>Anexo</b>	<b>27</b>
Índice	27
Diagrama de clases del dominio en BusinessLogic	28
Tablas de la base de datos	29
Herencia de PaginatedRepositoryBase en Repositorios	30
Referencias	30
Frontend	30
Organización de la aplicación	31
Manejo de errores	33
Manejo de permisos y rutas	33
Cobertura de los casos de prueba	34
Especificación de la API	35
Nuevos Endpoints	35
26) Listar hogares	35
27) Obtener hogar	36
28) Modificar alias de hogar	37
29) Asignar rol de dueño de hogar	38
30) Crear notificación de detección de movimiento de sensor de movimiento	39
31) Registrar sensor de movimiento	40
32) Registrar lámpara	41
33) Encender lámpara	43
34) Apagar lámpara	43
35) Obtener validadores	44
36) Obtener importadores	45
37) Importar dispositivos	45
38) Obtener archivos a importar	47
39) Modificar nombre de dispositivo	47
40) Obtener permisos de usuario sobre el hogar	49
41) Configurar validador para empresa	49
42) Mover dispositivo de cuarto	51
43) Crear cuarto	52
44) Asociar dispositivo a un cuarto	53
45) Obtener cuartos de un hogar	54
46) Obtener dispositivos asociados a un hogar	55
47) Obtener dispositivos de una empresa	57
48) Obtener datos de una cámara	58

49) Obtener empresas de un dueño de empresa	59
Endpoints Modificados	61
20,23) Conectar dispositivo	61
20,23) Desconectar dispositivo	61
18) Listar dispositivos del hogar	62
10) Autenticación	63
15) Agregar miembros al hogar	64
12) Listar dispositivos	66

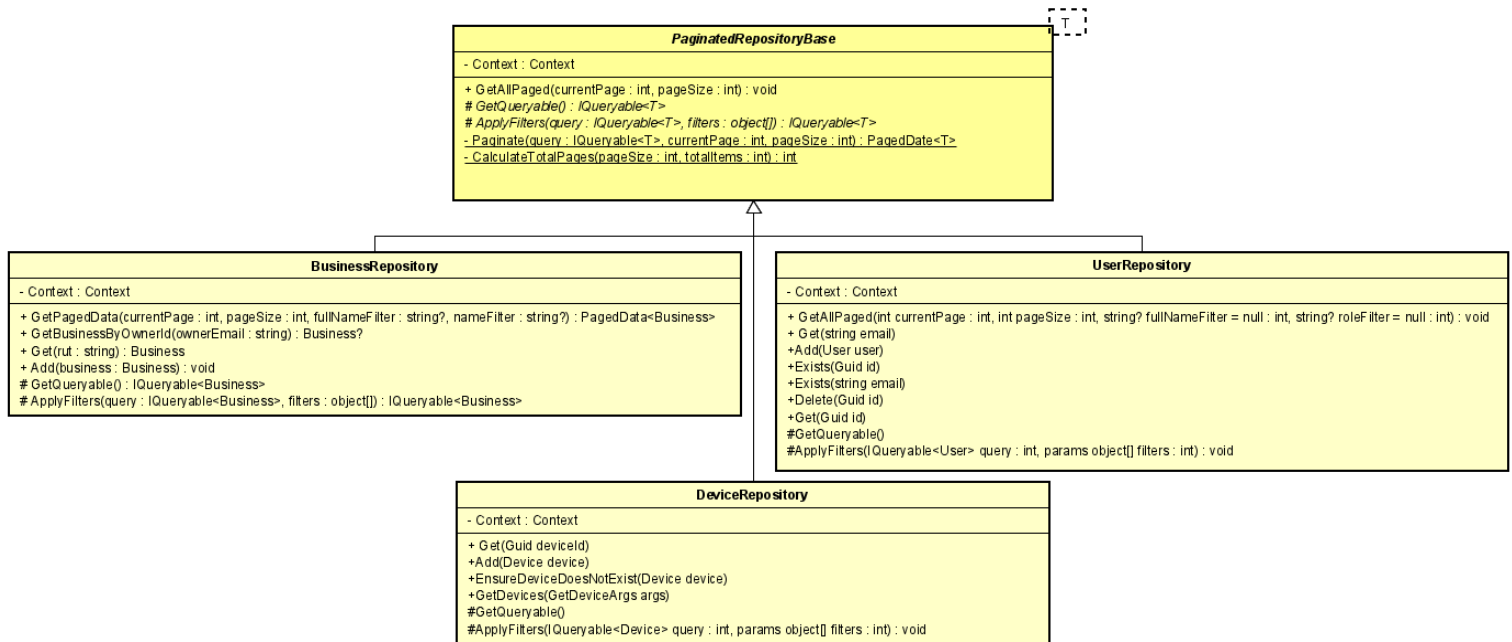
## Diagrama de clases del dominio en BusinessLogic



## Tablas de la base de datos



## Herencia de PaginatedRepositoryBase en Repositorios



## Referencias

- Biblioteca para la interfaz de usuario: [PrimeNG - Angular UI Component Library](#)
- Biblioteca CSS para la interfaz de usuario: [PrimeFlex - Utility-First CSS Library](#)
- DeviceArgs:

```

public string Name { get; set; } = null;
public string ModelNumber { get; set; } = null;
public string Description { get; set; } = null;
public string MainPhoto { get; set; } = null;
List<string> SecondaryPhotos { get; set; } = null;
string Type { get; set; } = null;
bool? MotionDetection { get; set; } = null;
bool? PersonDetection { get; set; } = null;
bool? IsExterior { get; set; } = false;
bool? IsInterior { get; set; } = true;
    
```

## Frontend

El frontend se implementa como una aplicación Angular de una sola página (SPA). Esta interfaz de usuario implementa todas las funcionalidades descritas en los requisitos del sistema, garantizando una integración total con la API REST.

## Organización de la aplicación

La arquitectura utilizada permite un desarrollo modular y sostenible. Dentro de la carpeta `app/`, creamos los siguientes módulos:

- **root**

La responsabilidad de este módulo es la de agrupar aquellos componentes que queremos mostrar en toda la página y configurar el routing de la aplicación.

Podríamos haber hecho esto directamente en el `AppModule`, sin embargo, quisimos que dicho módulo sea lo más genérico posible, facilitando la reutilización del `RootComponent` quisiéramos tener una organización distinta de la aplicación en el futuro, por ejemplo, si quisiéramos que algunas páginas no tuvieran la sidebar. Además, separamos mejor las responsabilidades, ya que dejamos que el `AppComponent` se encargue de la configuración global de la aplicación (por ejemplo, proveer el `HttpClient` y otros providers, configurar el `not found`, etc.)

- **notifications**

Define "notifications-page" en la ruta `/notifications`. En la misma se muestra el business component "notifications-message" que es quién se encarga de hacer las llamadas al endpoint de notificaciones y mostrarlas (agrupa una pequeña porción del negocio).

- **homes**

Agrupar las páginas: `homes-page (/homes)`, que se encarga de mostrar la tabla de hogares y el formulario para agregar un hogar, y `home-page (/homes/:id)`, que se encarga de mostrar la información de un hogar y agrupar los formularios para agregar miembros y agregar cuartos.

Los formularios, como en el resto de casos, se definieron dentro del módulo, ya que consideramos que esta parte de código no será reutilizada fuera de dicho módulo, al tratarse de formularios que queremos tener en un solo lugar (tenerlos en varias pages complejizaría la aplicación sin un beneficio añadido).

Las tablas de hogares, de miembros, y de cuartos, son business components. La comunicación con los services se hace dentro de estos business components, así como el manejo de errores, por lo que la responsabilidad de las pages es únicamente la de agrupar business components y/o components y configurar la organización de las mismas (por ejemplo, colocamos las tablas dentro de componentes app-panel para organizar mejor la página brindando una mejor experiencia al usuario).

- **devices**

Define la página devices-page en /devices. En esta página se muestran los dos business components de devices-table y device-types-table y se organizan en paneles.

- **businesses**

Agrupar las páginas de business-page (/businesses/:rut) y businesses-page (/businesses) y declara los formularios para crear una empresa, crear un dispositivo, cambiar validador e importar dispositivos.

- **auth**

Agrupar los formularios de login y registro. Define una única auth page donde, de momento, solo tenemos el router-outlet, pero decidimos dejarla para seguir con el mismo nivel de modularidad que en los demás casos. Si en el futuro quisiéramos agregar algo en común a estos formularios, o agregar un componente, como podría ser un footer o un header, se podría perfectamente modificar esta página sin un mayor impacto.

- **admins**

Agrupar los formularios de crear administrador y crear dueño de empresa, y define la página admins-page (/admins).

En resumen, esta organización presenta las siguientes ventajas:

- **Separación de responsabilidades:** Los módulos encapsulan su lógica de ruteo, formularios y páginas, lo que permite gestionar cambios específicos sin afectar otras partes de la aplicación.



- **Business Components:** Se encargan de la lógica de comunicación con servicios y manejo de errores, dejando a las páginas la tarea de organizar y presentar estos componentes.
- **Consistencia:** La modularidad aplicada incluso en casos simples, como en auth-page, asegura una estructura uniforme que facilita la extensión y mantenibilidad del proyecto.

Por otro lado, tenemos una carpeta con components/ donde encapsulamos componentes genéricos para construir el resto de elementos de la aplicación. En los mismos, configuramos el template de los componentes del UI Kit que utilizamos, PrimeNG. Utilizar estos componentes, y no directamente los provistos por PrimeNG, nos garantiza que en el futuro podamos cambiar esta librería sin afectar a toda la aplicación ya que el cambio se propagaría únicamente a esta carpeta. Por otro lado, en la carpeta business-components/, como ya mencionamos, contamos con componentes que se encargan de agrupar los componentes genéricos y agregarles la lógica de negocio, como puede ser comunicarse con los services, que luego se comunican con la REST API.

## Manejo de errores

Las respuestas de error de la API REST están diseñadas para ser claras e interpretables por el usuario final. Por lo tanto, cuando el servidor devuelve un error, mostramos directamente el mensaje proporcionado por la API.

En caso de errores del cliente, como fallos en la conexión, mostramos un mensaje genérico y predeterminado para ocultar las complejidades técnicas al usuario y mejorar la experiencia: "Something went wrong, please try again later."

## Manejo de permisos y rutas

Cuando el usuario inicia sesión, el endpoint auth devuelve un diccionario "Roles", donde la clave corresponde al rol y el valor es un array con los permisos que tiene dicho rol.

El acceso a las rutas está controlado por dos guards de autenticación, que verifican si el usuario está o no autenticado, y por un guard de permisos, que verifica que el usuario tiene el permiso necesario para acceder a la ruta.

Cuando el usuario se autentica, guardamos en el localStorage una entrada con el rol actual del usuario (un string, por ejemplo HomeOwner) y un diccionario que mapea los roles que tiene el usuario con los permisos correspondientes a cada rol.

Si el usuario tiene dos roles, los puede alternar, para lo que se hace una llamada al AuthService para alternar el rol actual del usuario (se modifica el valor en el local storage).










Luego, cuando el usuario intenta acceder a una ruta, se verifica en el PermissionsGuard que los permisos necesarios para acceder a la ruta se incluyan en los permisos del rol actual. Para el manejo de rutas a nivel visual, creamos un servicio RouteService donde filtramos las rutas a las que el usuario puede acceder en base a sus permisos, y el sidebar se vale de este servicio para mostrar las rutas correspondientes.

Manejar esto a nivel de permisos, en lugar de a nivel de roles estrictamente, nos garantiza que si el día de mañana los permisos cambian en el back, el impacto en el frontend va a ser mínimo promoviendo una mayor mantenibilidad y flexibilidad.

## Cobertura de los casos de prueba

- Informe de Cobertura de Código

El porcentaje de cobertura alcanzó el 97% cuando revisamos los *Coverage Results* en Rider. Las líneas que están sin cubrir refieren mayormente a lo relacionado con los modelos (getters, setters) y a las clases/atributos creadas para hacer uso de las convenciones de Entity Framework Core.

▼  Total	97%	105/3699
▼  HomeConnect.DeviceImporter	100%	0/32
>  DeviceImporter.Models	100%	0/32
▼  HomeConnect.WebApi	99%	16/1264
▼  HomeConnect.WebApi	99%	16/1264
>  Filters	100%	0/258
>  Controllers	98%	16/1006
▼  HomeConnect.DataAccess	99%	6/461
>  HomeConnect.DataAccess	99%	6/461

▼ HomeConnect.BusinessLogic	96%	83/1942
▼ {} BusinessLogic	96%	83/1942
> 🚀 FilterArgs	100%	0/18
> 🚀 PagedData<T>	100%	0/9
> 🚀 Pagination	100%	0/6
> {} Users	100%	0/183
> {} Admins	100%	0/55
> {} Notifications	99%	1/163
> {} BusinessOwners	99%	1/274
> {} Auth	99%	1/98
> {} Roles.Entities	97%	1/37
> {} HomeOwners	97%	18/659
> {} Devices	97%	11/390
> {} Helpers	0%	50/50

Tuvimos dificultades para probar `AssemblyInterfaceLoader`, por lo que decidimos no hacer pruebas unitarias y probar la clase funcionalmente. Somos conscientes que esto no reemplaza a los test unitarios.

▼ {} Helpers	0%	50/50
▼ 🚀 AssemblyInterfaceLoader<TInterface>	0%	50/50
🚀 CreateDirectoryInfo(string)	0%	7/7
> 🚀 GetImplementationsList(string)	0%	17/17
> 🚀 GetImplementationByName(string,string,params object[])	0%	9/9
🚀 GetImplementationIdByName(string,string)	0%	3/3
🚀 GetImplementationByIndex(int,params object[])	0%	4/4
> 🚀 GetImplementationById(Nullable<Guid>,string)	0%	9/9
🚀 AssemblyInterfaceLoader()	0%	1/1

## Especificación de la API

### Nuevos Endpoints

#### 26) Listar hogares

Header: Authorization

**GET** /homes

Obtiene una lista de todos los hogares del usuario autenticado, ya sea donde es dueño o miembro.

### Respuestas:

- 200 OK
  - Descripción: la lista de hogares fue obtenida exitosamente.
  - Cuerpo de la respuesta (Response Body):

```
{
  "homes": [
    {
      "id": string,
      "name": string | null,
      "address": string,
      "latitude": double,
      "longitude": double,
      "maxMembers": int
    }
  ]
}
```

- 400 Bad Request
  - Descripción: solicitud no válida.
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente.
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene permisos para listar los hogares.
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.

## 27) Obtener hogar

Header: Authorization

**GET** /homes/{homesId}

Obtiene los detalles de un hogar específico registrado del usuario autenticado.

### Respuestas:

- 200 OK
  - Descripción: los detalles del hogar fueron obtenidos exitosamente.

- Cuerpo de la respuesta (Response Body):

```
{
  "id": string,
  "name": string | null,
  "owner": {
    "id": string,
    "name": string,
    "surname": string
  },
  "address": string,
  "latitude": double,
  "longitude": double,
  "maxMembers": int
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, el formato del homeId es incorrecto.
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente.
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene permisos para obtener este hogar.
- 404 Not Found
  - Descripción: no se encontró el hogar indicado.
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.

## 28) Modificar alias de hogar

Header: Authorization

**PATCH** /homes/{homeId}/name

Modifica el alias de un hogar específico.

**Cuerpo de la solicitud (Request Body):**

- newName (requerido)
  - Descripción: El nuevo alias del hogar.
  - Tipo: Cadena de texto.

- Ejemplo: newName="Mi Casa Principal"

### Respuestas:

- 200 OK

- Descripción: el alias del hogar fue actualizado exitosamente.
- Cuerpo de la respuesta (Response Body):

```
{  
  "homeId": string  
}
```

- 400 Bad Request

- Descripción: solicitud no válida, faltan campos requeridos o el formato del request body es incorrecto.

- 401 Unauthorized

- Descripción: el token de autorización es inválido o está ausente.

- 403 Forbidden

- Descripción: el usuario autenticado no tiene permisos para modificar el alias del hogar.

- 404 Not Found

- Descripción: no se encontró el hogar indicado.

- 500 Internal Server Error

- Descripción: ocurrió un error inesperado en el servidor.

## 29) Asignar rol de dueño de hogar

Header: Authorization

**PATCH** /users/home\_owner\_role

Asigna el rol de dueño de hogar al usuario.

### Respuestas:

- 200 OK

- Descripción: el rol de dueño de hogar fue asignado exitosamente al usuario.
- Cuerpo de la respuesta (Response Body):

```
{
  "id": string,
  "roles": [
    { roleName: string[] }
  ]
}
```

- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente.
- 409 Conflict
  - Descripción: el usuario ya tiene el rol de dueño de hogar asignado.
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.

### 30) Crear notificación de detección de movimiento de sensor de movimiento

Header: Ninguno

**POST** /motion\_sensors/{hardwareId}/movement-detected

Crea una notificación informando que el sensor de movimiento detectó movimiento.

**Cuerpo de la solicitud (Request Body):** Ninguno

**Respuestas:**

- 200 OK
  - Descripción: la notificación se creó y envió correctamente.
  - Cuerpo de la respuesta (Response Body):

```
{
  "hardwareId" = string
}
```

- 400 Bad Request
  - Descripción: el dispositivo está desconectado
- 404 Not Found
  - Descripción: no se encontró el dispositivo indicado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

### 31) Registrar sensor de movimiento

Header: Authorization

**POST** /motion\_sensors

Registra un sensor de ventana.

#### Cuerpo de la solicitud (Request Body):

- name (requerido):
  - Descripción: Nombre del sensor de ventana.
  - Tipo: Cadena de texto.
  - Ejemplo: name=Sensor Ventana Principal
- modelNumber (requerido):
  - Descripción: Número de modelo del sensor.
  - Tipo: Entero.
  - Ejemplo: modelNumber=5678
- description (requerido):
  - Descripción: Descripción del sensor de ventana.
  - Tipo: Cadena de texto.
  - Ejemplo: description=Sensor de ventana gran angular
- mainPhoto (requerido):
  - Descripción: Imagen principal del sensor de ventana
  - Tipo: Cadena de texto
  - Ejemplo: mainPhoto=https://ejemplo.com/foto.jpg
- secondaryPhotos (opcional):
  - Descripción: Lista de URLs de imágenes utilizadas para comercializar el sensor de ventana.
  - Tipo: Array de cadenas de texto.
  - Ejemplo: secondaryPhotos=https://ejemplo1.com/foto.jpg,  
<https://ejemplo2.com/foto2.jpg>

#### Respuestas:

- 200 OK
  - Descripción: el sensor de movimiento fue registrado exitosamente



- Cuerpo de la respuesta (Response Body):

```
{  
  "id": string  
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, faltan campos requeridos o el formato del request body es incorrecto
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene los permisos necesarios para registrar el sensor
- 409 Conflict
  - Descripción: ya existe un sensor con el mismo número de modelo
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

## 32) Registrar lámpara

Header: Authorization

**POST** /lamps

Registra una lámpara.

**Cuerpo de la solicitud (Request Body):**

- name (requerido):
  - Descripción: Nombre de la lámpara.
  - Tipo: Cadena de texto.
  - Ejemplo: name=Lámpara Ventana Principal
- modelNumber (requerido):
  - Descripción: Número de modelo de la lámpara.
  - Tipo: Entero.
  - Ejemplo: modelNumber=5678
- description (requerido):

- Descripción: Descripción de la lámpara.
- Tipo: Cadena de texto.
- Ejemplo: description=Lampara LED 500 lumens
- mainPhoto (requerido):
  - Descripción: Imagen principal de la lámpara
  - Tipo: Cadena de texto
  - Ejemplo: mainPhoto=https://ejemplo.com/foto.jpg
- secondaryPhotos (opcional):
  - Descripción: Lista de URLs de imágenes utilizadas para comercializar la lámpara.
  - Tipo: Array de cadenas de texto.
  - Ejemplo: secondaryPhotos=https://ejemplo1.com/foto.jpg,  
<https://ejemplo2.com/foto2.jpg>

### Respuestas:

- 200 OK
  - Descripción: el sensor de movimiento fue registrado exitosamente
  - Cuerpo de la respuesta (Response Body):
 

```
{
  "id": string
}
```
- 400 Bad Request
  - Descripción: solicitud no válida, faltan campos requeridos o el formato del request body es incorrecto
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene los permisos necesarios para registrar el sensor
- 409 Conflict
  - Descripción: ya existe un sensor con el mismo número de modelo
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

### 33) Encender lámpara

Header: Ninguno

**POST** /lamps/{hardwareId}/turn\_on

Enciende la lámpara y genera una notificación en el caso de que estuviera apagada.

**Cuerpo de la solicitud (Request Body):** Ninguno

**Respuestas:**

- 200 OK
  - Descripción: la lámpara se encendió y la notificación se creó y envió correctamente.
  - Cuerpo de la respuesta (Response Body):

```
{
  "hardwareId": string
}
```

- 400 Bad Request
  - Descripción: el dispositivo está desconectado
- 404 Not Found
  - Descripción: no se encontró el dispositivo indicado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

### 34) Apagar lámpara

Header: Ninguno

**POST** /lamps/{hardwareId}/turn\_off

Apaga la lámpara y genera una notificación en el caso de que estuviera encendida.

**Cuerpo de la solicitud (Request Body):** Ninguno

**Respuestas:**

- 200 OK
  - Descripción: la lámpara se apagó y la notificación se creó y envió correctamente.

- Cuerpo de la respuesta (Response Body):

```
{  
  "hardwareId": string  
}
```

- 400 Bad Request
  - Descripción: el dispositivo está desconectado
- 404 Not Found
  - Descripción: no se encontró el dispositivo indicado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

### 35) Obtener validadores

Header: Authorization

GET /device\_validators

**Descripción:** Obtiene una lista de validadores disponibles.

**Respuestas:**

- 200 OK
  - Descripción: se retornó la lista de validadores disponibles.
  - Cuerpo de la respuesta (Response Body):

```
{  
  "validators": [  
    string  
  ]  
}
```

- 401 Unauthorized
  - Descripción: El token de autorización es inválido o está ausente.
- 500 Internal Server Error
  - Descripción: Ocurrió un error inesperado en el servidor.

### 36) Obtener importadores

Header: Authorization

GET /device\_importers

**Descripción:** Obtiene una lista de importadores disponibles.

**Respuestas:**

- 200 OK
  - Descripción: se retornó la lista de importadores disponibles con sus parametros.
  - Cuerpo de la respuesta (Response Body):

```
{
  "importers": [
    {
      "name": "string",
      "parameters": [
        "string"
      ]
    }
  ]
}
```

- 401 Unauthorized
  - Descripción: El token de autorización es inválido o está ausente.
- 500 Internal Server Error
  - Descripción: Ocurrió un error inesperado en el servidor.

### 37) Importar dispositivos

Header: Authorization

POST /devices

**Descripción:** Importa una lista de dispositivos desde un archivo, usando un importador específico.

### Cuerpo de la solicitud (Request Body):

- importerName(requerido):
  - Descripción: Nombre del importador a usar.
  - Tipo: Cadena de texto.
  - Ejemplo: importerName = JsonImporter
- route(requerido):
  - Descripción: Nombre del archivo a importar.
  - Tipo: Cadena de texto.
  - Ejemplo: route = archivo.json
- parameters(requerido):
  - Descripción: Parámetros adicionales como clave-valor.
  - Tipo: Diccionario de claves y valores, donde tanto las claves como los valores son cadenas de texto.
  - Ejemplo:
  -

```
{
  "key1": "value1",
  "key2": "value2"
}
```

○

### Respuestas:

- 200 OK
  - Descripción: se importaron todos los dispositivos correctamente. Se devuelve una lista con los modelos de los dispositivos.
  - Cuerpo de la respuesta (Response Body):

```
{
  "importedDevices": [
    string
  ]
}
```

- 401 Unauthorized

- Descripción: El token de autorización es inválido o está ausente.
- 500 Internal Server Error
  - Descripción: Ocurrió un error inesperado en el servidor.

### 38) Obtener archivos a importar

Header: Authorization

GET /device\_import\_files

**Descripción:** Obtiene una lista de los archivos disponibles para importar.

**Respuestas:**

- 200 OK
  - Descripción: se retornó la lista de archivos a importar disponibles.
  - Cuerpo de la respuesta (Response Body):

```
{
  "importFiles": [
    "file1.csv",
    "file2.json"
  ]
}
```

- 401 Unauthorized
  - Descripción: El token de autorización es inválido o está ausente.
- 500 Internal Server Error
  - Descripción: Ocurrió un error inesperado en el servidor.

### 39) Modificar nombre de dispositivo

Header: Authorization

PATCH /devices/{hardwareId}/name

**Descripción:** Modifica el nombre del dispositivo.

**Parámetros de ruta:**

- hardwareId (requerido)
  - Descripción: identificador del dispositivo que se desea modificar
  - Tipo: GUID

#### **Cuerpo de la solicitud (Request Body):**

- newName (requerido)
  - Descripción: nuevo nombre que se le asignará al dispositivo
  - Tipo: Cadena de texto

#### **Respuestas:**

- 200 OK
  - Descripción: se retornó la lista de archivos a importar disponibles.
  - Cuerpo de la respuesta (Response Body):

```
{
  "deviceId": string
}
```

- 401 Unauthorized
  - Descripción: El token de autorización es inválido o está ausente.
- 403 Forbidden
  - Descripción: El usuario autenticado no tiene permisos para modificar el nombre del dispositivo especificado.
- 404 Not Found
  - Descripción: No se encontró ningún dispositivo con el hardwareId especificado.
- 500 Internal Server Error
  - Descripción: Ocurrió un error inesperado en el servidor.

## **40) Obtener permisos de usuario sobre el hogar**

Header: Authorization

**GET** /homes/{homesId}/permissions



**Descripción:** Obtiene la lista de permisos que tiene el usuario autenticado sobre el hogar determinado

**Respuestas:**

- 200 OK
  - Descripción: los permisos del usuario sobre el hogar fueron obtenidos exitosamente.
  - Cuerpo de la respuesta (Response Body):

```
{
  "homeId": string,
  "permissions": [string]
}
```

- 401 Unauthorized
  - Descripción: El token de autorización es inválido o está ausente.
- 403 Forbidden
  - Descripción: El usuario autenticado no es miembro de este hogar.
- 404 Not Found
  - Descripción: No se encontró el hogar indicado.
- 500 Internal Server Error
  - Descripción: Ocurrió un error inesperado en el servidor.

## 41) Configurar validador para empresa

Header: Authorization

**PATCH** /businesses/{businessId}/validator

Ejemplo:

**PATCH** <https://api.homeconnect.com/businesses/12345/validator>

Configura el validador asignado a una empresa.

**Cuerpo de la solicitud (Request Body):**

- validator (requerido):
  - Descripción: Nombre del validador a utilizar por la empresa. Dejar campo vacío en caso de no querer un validador.

- Tipo: string
- Ejemplo: validator=""
- Ejemplo: validator="SixCharacters3Letters3NumbersModelValidator"

### Respuestas:

- 200 OK
  - Descripción: el validador se estableció correctamente.
  - Cuerpo de la respuesta (Response Body):

```
{
  "businessRut": GUID,
  "validator": string
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, faltan campos requeridos, el validador no existe o el formato del request body es incorrecto
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene los permisos necesarios para modificar la configuración de notificaciones
- 404 Not Found
  - Descripción: no se encontró la empresa indicada
- 409 Conflict
  - Descripción: la empresa indicada no le pertenece al usuario logeado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.

## 42) Mover dispositivo de cuarto

Header: Authorization

**PATCH** /devices/{deviceId}/room

Actualiza el cuarto asociado a un dispositivo

### Parámetros de ruta:

- deviceId (requerido):
  - Descripción: Identificador del dispositivo que se desea mover de cuarto
  - Tipo: string
  - Ejemplo: deviceId="a6b9c9f2-3aec-47a6-94eb-52e4df9f8e75"

### Cuerpo de la solicitud (Request Body):

- targetRoomId (requerido):
  - Descripción: Identificador del cuarto al que se desea mover el dispositivo
  - Tipo: string
  - Ejemplo: targetRoomId="a6b9c9f2-3aec-47a6-94eb-52e4df9f8e75"

### Respuestas:

- 200 OK
  - Descripción: el dispositivo se movió de cuarto correctamente.
  - Cuerpo de la respuesta (Response Body):

```
{  
  "targetRoomId": string,  
  "deviceId": string  
}
```
- 400 Bad Request
  - Descripción: solicitud no válida, faltan campos requeridos, el dispositivo no pertenece al cuarto de origen o el formato del request body es incorrecto
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene los permisos necesarios para mover el dispositivo de cuarto
- 404 Not Found

- Descripción: no se encontró el dispositivo
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.

### 43) Crear cuarto

Header: Authorization

**POST** /homes/{homesId}/rooms

Crea un nuevo cuarto asociado a un hogar específico

#### Parámetros de ruta:

- homesId (requerido):
  - Descripción: Identificador del hogar en el que se desea crear el cuarto
  - Tipo: string
  - Ejemplo: homesId="a6b9c9f2-3aec-47a6-94eb-52e4df9f8e75"

#### Cuerpo de la solicitud (Request Body):

- name (requerido):
  - Descripción: Nombre del cuarto
  - Tipo: string
  - Ejemplo: name="Sala de estar"

#### Respuestas:

- 200 OK
  - Descripción: el cuarto se creó exitosamente.
  - Cuerpo de la respuesta (Response Body):

```
{  
  "roomId": string  
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, faltan campos requeridos o el formato del request body es incorrecto

- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene los permisos necesarios para crear un cuarto en este hogar
- 404 Not Found
  - Descripción: no se encontró el hogar especificado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.

## 44) Asociar dispositivo a un cuarto

Header: Authorization

**POST** /rooms/{roomId}/devices

Asocia un dispositivo a un cuarto específico

### Parámetros de ruta:

- roomId (requerido):
  - Descripción: Identificador del cuarto al que se quieren asociar los dispositivos
  - Tipo: string
  - Ejemplo: roomId="a6b9c9f2-3aec-47a6-94eb-52e4df9f8e75"

### Cuerpo de la solicitud (Request Body):

- deviceId (requerido):
  - Descripción: Identificador del dispositivo a asociar
  - Tipo: string
  - Ejemplo: deviceId="a6b9c9f2-3aec-47a6-94eb-52e4df9f8e75"

### Respuestas:

- 200 OK
  - Descripción: el dispositivo fue asociado exitosamente al cuarto
  - Cuerpo de la respuesta (Response Body):

```
{
  "roomId": string,
  "deviceId": string
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, faltan campos requeridos o el formato del request body es incorrecto
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene los permisos necesarios para asociar un dispositivo a un cuarto en este hogar
- 404 Not Found
  - Descripción: no se encontró el cuarto especificado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.

## 45) Obtener cuartos de un hogar

Header: Authorization

**GET** /homes/{homesId}/rooms

Obtiene una lista de los cuartos asociados a un hogar específico

### Parámetros de ruta:

- homesId (requerido):
  - Descripción: Identificador del hogar cuyos cuartos se desean obtener
  - Tipo: string
  - Ejemplo: homesId="a6b9c9f2-3aec-47a6-94eb-52e4df9f8e75"

### Respuestas:

- 200 OK
  - Descripción: Se obtuvo la lista de cuartos asociados al hogar exitosamente

- Cuerpo de la respuesta (Response Body):

```
{
  "rooms": [
    {
      "id": string,
      "name": string,
      "homeId": string,
      "ownedDevicesId": [string]
    }
  ]
}
```

- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no es miembro del hogar indicado o no tiene permiso para listar los cuartos
- 404 Not Found
  - Descripción: no se encontró el hogar especificado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.

## 46) Obtener dispositivos asociados a un hogar

Header: Authorization

**GET** /homes/{homeId}/devices

Obtiene una lista de los dispositivos asociados a un hogar específico.

### Parámetros de ruta:

- homeId (requerido):
  - Descripción: Identificador del hogar cuyos dispositivos se desean obtener
  - Tipo: string
  - Ejemplo: homeId="a6b9c9f2-3aec-47a6-94eb-52e4df9f8e75"

### Respuestas:

- 200 OK
  - Descripción: Se obtuvo la lista de dispositivos asociados al hogar exitosamente
  - Cuerpo de la respuesta (Response Body):

```
{
  "devices": [
    {
      "id": null,
      "hardwareId": "e38a02fb-98c7-4839-8cf7-c4edfea9a82a",
      "name": "Handmade Soft Cheese",
      "businessName": "Cristal",
      "type": "Sensor",
      "modelName": "861",
      "mainPhoto":
        "https://i.imgur.com/N9CqyET_d.webp?maxwidth=520&shape=thumb&fidelity=high",
      "secondaryPhotos": [
        "http://placeimg.com/640/480",

        "https://i.imgur.com/N9CqyET_d.webp?maxwidth=520&shape=thumb&fidelity=high"
      ],
      "state": null,
      "isOpen": false,
      "roomId": "0b8b5e44-98af-4fdf-8b87-c54b59ed06f9"
    }
  ]
}
```

- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no es miembro del hogar indicado o no tiene permiso para listar los dispositivos
- 404 Not Found
  - Descripción: no se encontró el hogar especificado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor.



## 47) Obtener dispositivos de una empresa

GET `/businesses/{businessId}/devices`

Header: Authorization

Ejemplo:

GET `https://api.homeconnect.com//businesses/123/devices`

Obtiene una lista paginada de los dispositivos registrados por una empresa.

### Parámetros de Consulta (Query Parameters):

- **page** (opcional):
  - Descripción: Especifica el número de la página de resultados que se desea obtener. Utilizado para la paginación.
  - Tipo: Entero.
  - Valor predeterminado: 1.
  - Ejemplo: `page=2`
- **pageSize** (opcional):
  - Descripción: Especifica el número de resultados por página.
  - Tipo: Entero.
  - Valor predeterminado: 10.
  - Ejemplo: `pageSize= 50`

### Respuestas:

- **200 OK**
  - Descripción: la lista de dispositivos fue obtenida exitosamente
  - Cuerpo de la respuesta (Response Body):

```
{
  "devices": [
    {
      "id": string,
      "name": string,
      "businessName": string,
      "type": string,
      "modelNumber": int,
      "photo": string,
      "description": string
    }
  ]
}
```

```
    ],  
    "pagination": {  
      "page": int,  
      "pageSize": int,  
      "totalPages": int  
    }  
  }  
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, el formato de algún parámetro es incorrecto o la combinación de parámetros no es válida
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

## 48) Obtener datos de una cámara

GET /cameras/{cameraId}

Header: Authorization

Ejemplo:

GET <https://api.homeconnect.com/cameras/1234>

Obtiene una lista paginada de los dispositivos registrados.

**Parámetros de Consulta: Ninguno**

**Respuestas:**

- 200 OK
  - Descripción: la información de la cámara fue obtenida exitosamente
  - Cuerpo de la respuesta (Response Body):

```
{  
  "id": "string",  
  "name": "string",  
  "description": "string",  
  "modelNumber": "string",  
  "mainPhoto": "string",  
  "secondaryPhotos": [  
    "string"  
  ]  
}
```

```
],  
  "motionDetection": "bool",  
  "personDetection": "bool",  
  "exterior": "bool",  
  "interior": "bool"  
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, el formato de algún parámetro es incorrecto, el dispositivo no es una cámara, no existe o la combinación de parámetros no es válida
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

## 49) Obtener empresas de un dueño de empresa

GET [/users/{userId}/businesses](#)

Header: Authorization

Ejemplo:

GET <https://api.homeconnect.com/users/1234/businesses>

Obtiene una lista paginada de las empresas registrados por un dueño de empresa.

### Parámetros de Consulta (Query Parameters):

- page (opcional):
  - Descripción: Especifica el número de la página de resultados que se desea obtener. Utilizado para la paginación.
  - Tipo: Entero.
  - Valor predeterminado: 1.
  - Ejemplo: page=2
- pageSize (opcional):
  - Descripción: Especifica el número de resultados por página.
  - Tipo: Entero.
  - Valor predeterminado: 10.
  - Ejemplo: pageSize= 50

## Respuestas:

- 200 OK
  - Descripción: la lista de empresas fue obtenida exitosamente
  - Cuerpo de la respuesta (Response Body):

```
{
  "businesses": [
    {
      "name": "string",
      "ownerName": "string",
      "ownerSurname": "string",
      "ownerEmail": "string",
      "rut": "string",
      "logo": "string"
    }
  ],
  "pagination": {
    "page": "int",
    "pageSize": "int",
    "totalPages": "int"
  }
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, el formato de algún parámetro es incorrecto o la combinación de parámetros no es válida
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

## Endpoints Modificados

### 20,23) Conectar dispositivo

Header: Ninguno

**POST** /devices/{hardwareId}/turn\_on

Ejemplo:

**POST** [https://api.homeconnect.com/devices/{hardwareId}/turn\\_on](https://api.homeconnect.com/devices/{hardwareId}/turn_on)

Cambia el estado de conexión de un dispositivo a conectado.

**Cuerpo de la solicitud (Request Body):** Ninguno

**Respuestas:**

- 200 OK
  - Descripción: el estado de conexión de un dispositivo fue actualizado exitosamente
  - Cuerpo de la respuesta (Response Body):

```
{
  "connected": bool,
  "hardwareId": string,
}
```

- 404 Not Found
  - Descripción: no se encontró el dispositivo indicado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

## 20,23) Desconectar dispositivo

Header: Ninguno

**POST** /devices/{hardwareId}/turn\_off

Ejemplo:

**POST** [https://api.homeconnect.com/devices/{hardwareId}/turn\\_off](https://api.homeconnect.com/devices/{hardwareId}/turn_off)

Cambia el estado de conexión de un dispositivo a desconectado.

**Cuerpo de la solicitud (Request Body):** Ninguno

**Respuestas:**

- 200 OK
  - Descripción: el estado de conexión de un dispositivo fue actualizado exitosamente
  - Cuerpo de la respuesta (Response Body):

```
{
  "connected": bool,
  "hardwareId": string,
}
```

- 404 Not Found
  - Descripción: no se encontró el dispositivo indicado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

## 18) Listar dispositivos del hogar

Header: Authorization

**GET** /homes/{homeId}/devices

Ejemplo:

**GET** <https://api.homeconnect.com/homes/12345/devices>

Obtiene una lista de los dispositivos asociados a un hogar.

### Respuestas:

- 200 OK
  - Descripción: la lista de dispositivos asociados al hogar fue obtenida exitosamente
  - Cuerpo de la respuesta (Response Body):

Los campos state e isOpen se usan para manejar el estado de lámparas y sensores de ventana respectivamente. En otros dispositivos estos campos no están visibles.

```
{
  "devices": [
    {
      "hardwareId": string,
      "name": string,
      "businessName": string,
      "type": string,
      "modelNumber": int,
      "mainPhoto": string,
      "secondaryPhotos": [string],
      "state": bool,
      "isOpen": bool
    }
  ]
}
```

- 400 Bad Request

- Descripción: solicitud no válida, el formato del homeId es incorrecto
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene los permisos necesarios para listar los dispositivos del hogar
- 404 Not Found
  - Descripción: no se encontró el hogar indicado
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

## 10) Autenticación

### POST /auth

Autentica a un usuario mediante correo y contraseña.

Cuerpo de la solicitud (Request Body):

- email (requerido):
  - Descripción: Correo electrónico del usuario que se está autenticando.
  - Tipo: Cadena de texto.
  - Ejemplo: email=johndoe@example.com
- password (requerido):
  - Descripción: Contraseña del usuario.
  - Tipo: Cadena de texto.
  - Ejemplo: password=JohnDoe#2001

### Respuestas:

- 200 OK
  - Descripción: autenticación exitosa, el usuario ha sido autenticado correctamente
  - Cuerpo de la respuesta (Response Body):

```
{
  "token": string,
```

```
"roles": [
  { roleName: string[] }
]
```

- 400 Bad Request
  - Descripción: solicitud no válida, faltan campos requeridos o el formato del request body es incorrecto
- 401 Unauthorized
  - Descripción: el correo electrónico o la contraseña son incorrectos
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor

## 15) Agregar miembros al hogar

Header: Authorization

**POST** /homes/{homesId}/members

Ejemplo:

**POST** <https://api.homeconnect.com/homes/12345/members>

Agrega un home owner existente a un hogar.

**Cuerpo de la solicitud (Request Body):**

- email (requerido):
  - Descripción: Email del dueño de hogar a agregar.
  - Tipo: string
  - Ejemplo: email=john.doe@gmail.com
- canAddDevices (opcional, por defecto *false*):
  - Descripción: Permiso para agregar dispositivos al hogar.
  - Tipo: bool
  - Ejemplo: canAddDevices=true
- canListDevices (opcional, por defecto *false*):
  - Descripción: Permiso para listar los dispositivos del hogar.
  - Tipo: bool
  - Ejemplo: canListDevices=false
- canNameDevices (opcional, por defecto *false*):
  - Descripción: Permiso para nombrar los dispositivos del hogar.



- Tipo: bool
- Ejemplo: canNameDevices=true

### Respuestas:

- 200 OK
  - Descripción: el miembro fue agregado exitosamente al hogar
  - Cuerpo de la respuesta (Response Body):

```
{
  "homeId": string,
  "memberId": string
}
```
- 400 Bad Request
  - Descripción: solicitud no válida, faltan campos requeridos o el formato del request body es incorrecto
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 403 Forbidden
  - Descripción: el usuario autenticado no tiene los permisos necesarios para agregar miembros al hogar
- 404 Not Found
  - Descripción: no se encontró el hogar o el dueño de hogar indicado
- 409 Conflict
  - Descripción: el miembro ya está agregado al hogar o la casa está llena
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servido

## 12) Listar dispositivos

GET /devices

Header: Authorization

Ejemplo:

GET <https://api.homeconnect.com/devices>

Obtiene una lista paginada de los dispositivos registrados.

### **Parámetros de Consulta (Query Parameters):**

- name (opcional):
  - Descripción: Filtra por nombre de dispositivo.
  - Tipo: Cadena de texto.
  - Ejemplo: name=example\_cam1
- type (opcional):
  - Descripción: Filtra por tipo de dispositivo.
  - Tipo: Cadena de texto.
  - Valores posibles: type=Camera
- model (opcional):
  - Descripción: Filtra por número de modelo.
  - Tipo: Entero.
  - Ejemplo: model=123
- businessName (opcional):
  - Descripción: Filtra por nombre de la empresa.
  - Tipo: Cadena de texto
  - Ejemplo: businessName=ExampleBusiness
- page (opcional):
  - Descripción: Especifica el número de la página de resultados que se desea obtener. Utilizado para la paginación.
  - Tipo: Entero.
  - Valor predeterminado: 1.
  - Ejemplo: page=2
- pageSize (opcional):
  - Descripción: Especifica el número de resultados por página.
  - Tipo: Entero.
  - Valor predeterminado: 10.
  - Ejemplo: pageSize= 50

### **Respuestas:**

- 200 OK
  - Descripción: la lista de dispositivos fue obtenida exitosamente
  - Cuerpo de la respuesta (Response Body):

```
{
  "devices": [
    {
      "id": string,
      "name": string,
      "businessName": string,
      "type": string,
      "modelName": int,
      "photo": string,
      "description": string
    }
  ],
  "pagination": {
    "page": int,
    "pageSize": int,
    "totalPages": int
  }
}
```

- 400 Bad Request
  - Descripción: solicitud no válida, el formato de algún parámetro es incorrecto o la combinación de parámetros no es válida
- 401 Unauthorized
  - Descripción: el token de autorización es inválido o está ausente
- 500 Internal Server Error
  - Descripción: ocurrió un error inesperado en el servidor