

# Universidad ORT Uruguay

## Facultad de Ingeniería

### Obligatorio 2 - Diseño de aplicaciones 2

Grupo: M6A

### **Descripción del diseño**

<https://github.com/IngSoft-DA2-2023-2/228352-262843-291192.git>

#### **Autores**

Franca Chechi - 291192

Rafael Nuñez - 262843

Juan Andres Tejera - 228352

#### **Tutores**

Francisco Bouza

Juan Irabedra

Santiago Tonarelli

# Abstract

Este documento fue escrito con el objetivo de demostrar la aplicación de buenas prácticas en cuanto al diseño de la solución. Se detalla la estructura del sistema organizado con el modelo 4+1. Se generaron diagramas UML para que se pueda leer y entender el documento con mayor facilidad. Se puede no solo encontrar la explicación del uso de herencias y interfaces, sino también los errores conocidos y documentación sobre la implementación de reflection.

# Índice

<b>Descripción general del trabajo.....</b>	<b>4</b>
<b>Costo de instalación.....</b>	<b>4</b>
<b>Errores conocidos.....</b>	<b>5</b>
<b>Diagrama general de paquetes.....</b>	<b>5</b>
<b>Diagrama de clases.....</b>	<b>7</b>
IDataAccess.....	7
DataAccess.....	7
ILogic.....	7
Logic.....	8
Models.....	8
Api.....	9
Domain.....	10
IImporter.....	10
JsonImporter y XmlImporter.....	11
<b>Reflection.....</b>	<b>11</b>
<b>Justificación y explicación del diseño.....</b>	<b>16</b>
Usuarios.....	16
Reportes.....	17
Categorías.....	17
Invitaciones.....	18
Métricas a nivel de paquetes.....	18
<b>Modelo de tablas de la estructura de la base de datos.....</b>	<b>20</b>
<b>Diagramas de interacción.....</b>	<b>20</b>
<b>Diagrama de componentes y conectores.....</b>	<b>22</b>
<b>Mecanismos utilizados para permitir la extensibilidad.....</b>	<b>22</b>
Reflection	
Como mencionamos anteriormente, para permitir la integración de nuevos importadores desarrollados por terceros, se ha definido la interfaz IImporter. Esta interfaz establece un contrato que todo importador debe cumplir:.....	22
Extensibilidad en la interfaz gráfica.....	23
Extensibilidad en roles del sistema.....	23
<b>Resumen de las mejoras al diseño.....</b>	<b>23</b>
<b>Anexo.....</b>	<b>25</b>
Cobertura de código:.....	25
Documentación de la API.....	26
Descripción general del trabajo.....	4
Costo de instalación.....	4
Errores conocidos.....	5
Diagrama general de paquetes.....	5
Diagrama de clases.....	7
IDataAccess.....	7
DataAccess.....	7

ILogic.....	7
Logic.....	8
Models.....	8
Api.....	9
Domain.....	10
Importer.....	10
JsonImporter y XmlImporter.....	11
Reflection.....	11
Justificación y explicación del diseño.....	16
Usuarios.....	16
Reportes.....	17
Categorías.....	17
Invitaciones.....	18
Métricas a nivel de paquetes.....	18
Modelo de tablas de la estructura de la base de datos.....	20
Diagramas de interacción.....	21
Diagrama de componentes y conectores.....	22
Mecanismos utilizados para permitir la extensibilidad.....	22
Reflection	
Como mencionamos anteriormente, para permitir la integración de nuevos importadores desarrollados por terceros, se ha definido la interfaz IImporter. Esta interfaz establece un contrato que todo importador debe cumplir:.....	22
Extensibilidad en la interfaz gráfica.....	23
Extensibilidad en roles del sistema.....	23
Resumen de las mejoras al diseño.....	23
Anexo.....	25
Cobertura de código:.....	25
Documentación de la API.....	26

## Descripción general del trabajo

Se implementó una nueva versión de un sistema para administrar edificios con diferentes tipos de usuario que cuenta con interfaz de usuario.

En el alcance del proyecto, se incluyeron funcionalidades que permiten al usuario dependiendo del rol:

1. Gestionar invitaciones a encargados y administradores de empresas constructoras
2. Dar de alta y baja a otros tipos de usuarios
3. Generar reportes
4. Gestionar edificios
5. Crear solicitudes y asignarlas a una persona de mantenimiento
6. Atender solicitudes guardando el tiempo que se demoró y su costo
7. Crear categorías para las solicitudes con la posibilidad de tener categorías vinculadas entre ellas de forma categoría padre - categoría hija
8. Creacion y gestion de empresas constructoras
9. Importar edificios dinámicamente

Se implementó utilizando las siguientes herramientas:

1. C# para el desarrollo
2. MSTest para las pruebas unitarias
3. Visual Studio y Visual Studio Code como IDE
4. Microsoft SQL Server Express 2017
5. Postman
6. NET Core SDK 8.0/ ASP.NET Core 8.0(C#)
7. Entity Framework Core 8.0
8. Herramientas para modelado de UML
9. Angular

## Costo de instalación

En la carpeta “Aplicación” se encuentran los ejecutables y librerías requeridas.

Strings de conexión a modificar:

- Se encuentran dentro de Frontend, en el archivo appsettings.json
- El formato para el caso que se utilice Windows como sistema operativo es:  

```
"Server=.\{instancia del servidor de SQL};
Database=BuildingManagerDB;Trusted_Connection=True;MultipleActiveResultSets=True"
```

En la carpeta “Base de datos” se encuentran los backups de la base de datos.

Credenciales de los usuarios iniciales de la aplicación:

<b><i>Rol</i></b>	<b><i>Email</i></b>	<b><i>Clave</i></b>
Administrador	admin@admin.com	admin

Encargado	manager2@gmail.com	manager
Administrador de empresa constructora	ccadmin@gmail.com	ccadmin
Mantenimiento	jp@gmail.com	password

## Errores conocidos

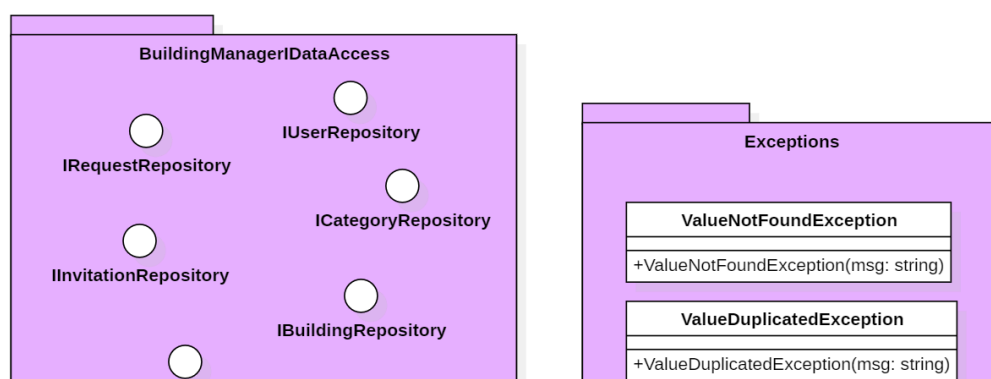
Habría que limitar la cantidad de elementos enviados en las listas de requests con `pageSize` y `page`, pero como no fue visto en clase no fue implementado. Esto hubiese servido para cuando hay muchos datos en la base de datos y así no bloquearla. Por ejemplo al traer la lista de encargados podríamos haber usado paginación.

Debido a que nuestro backend se desarrolló enteramente en inglés, el fronted, cuya UI esta en español tomas esos errores del back y los muestra, en varios casos fue posible hacer una traducción del error pero han quedado algunos otros casos en lo que no se ha hecho, probablemente lo más optimo para este tipo de casos es manejarnos con lang files.

## Diagrama general de paquetes

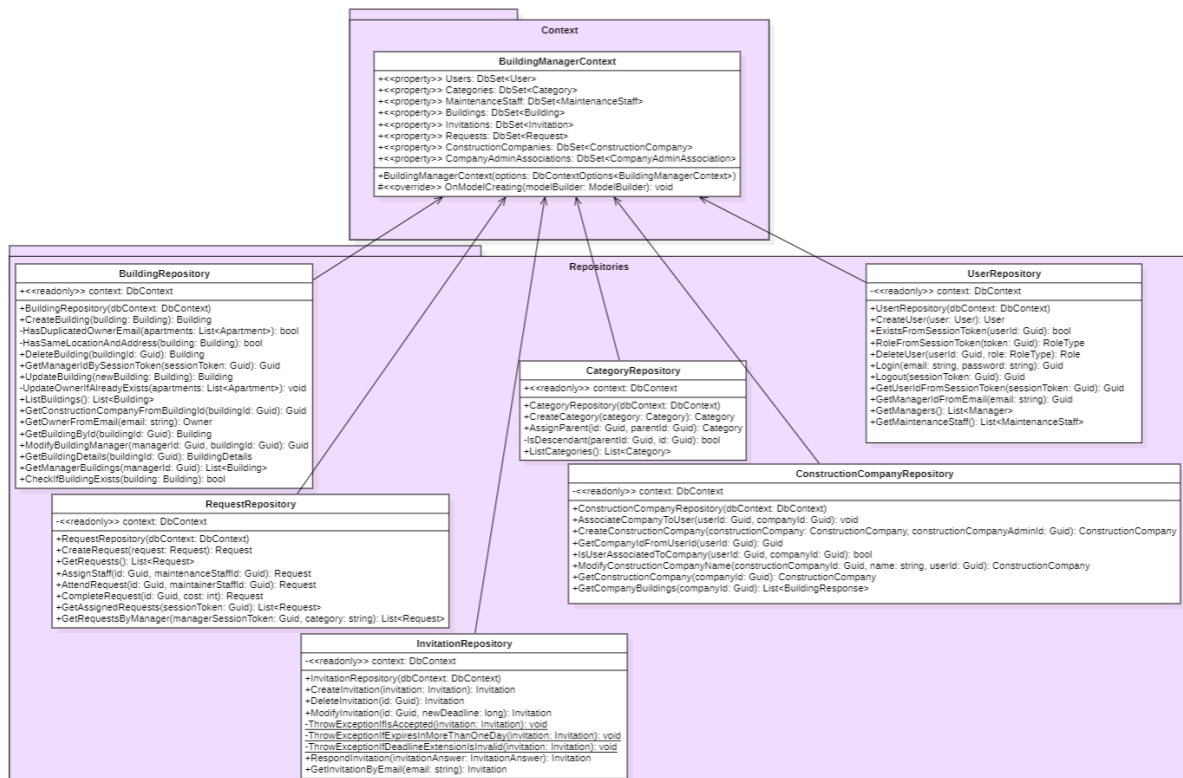
Hicimos uso de interfaces a nivel de la capa lógica y la capa de `dataAccess` aplicando DIP (dependency inversion principle) para que módulos de alto nivel no dependan de otros de bajo nivel. Esta decisión fue tomada con conciencia de forma que la api no dependa directamente de la lógica y la lógica no dependa directamente del `dataAccess`. Esto trae muchos beneficios, entre ellos, la posibilidad de cambiar fácilmente la implementación de las clases concretas sin afectar la implementación de la api. Es decir, en un futuro si necesitamos cambiar la implementación de la base de datos, por ejemplo cambiar de una base de datos a otra, esto no impactaría en absoluto en la capa lógica.

A su vez tenemos un paquete `BuildingManagerServiceFactory` para la implementación de inyección de dependencias que facilita el bajo acoplamiento entre módulos de diferente nivel.



## DataAccess

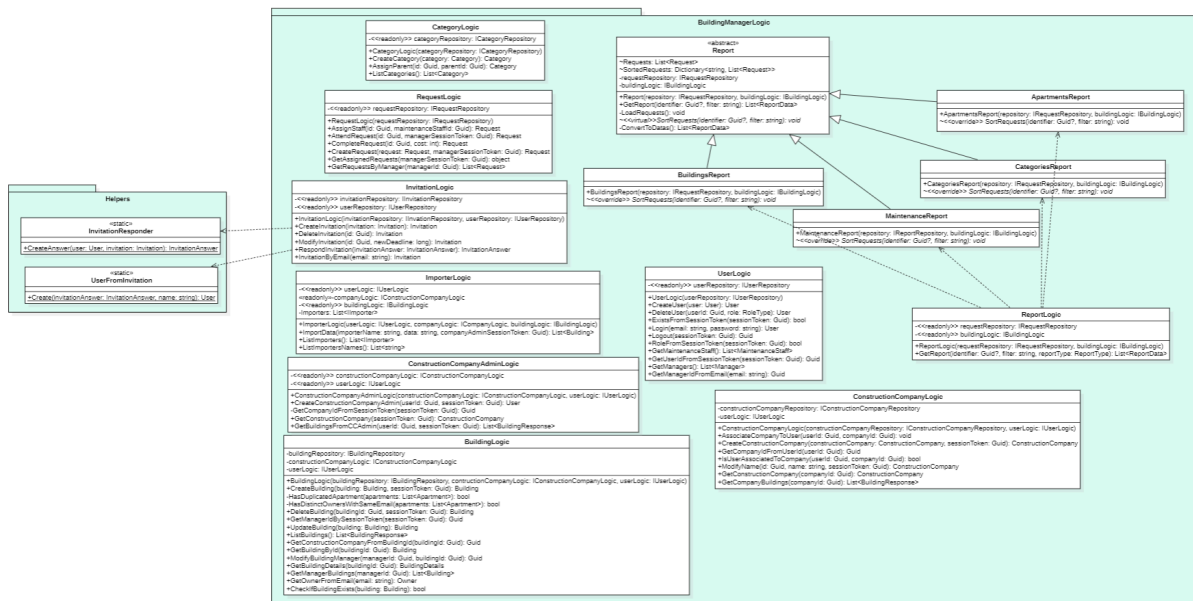
En este paquete se encuentran todos los repositorios responsables de comunicarse con la base de datos y el contexto. Dentro del namespace Repositories tenemos el BuildingRepository encargado de manejar los edificios, CategoryRepository encargado de las categorías, InvitationRepository que maneja las invitaciones pudiendo eliminar, crear y modificar invitaciones, RequestRepository que se encarga de las solicitudes, UserRepository que maneja a los usuarios con la entidad User y los sessionTokens y ConstructionCompanyRepository que se encarga de las empresas constructoras y su vínculo con los administradores de empresas constructoras.



## Logic

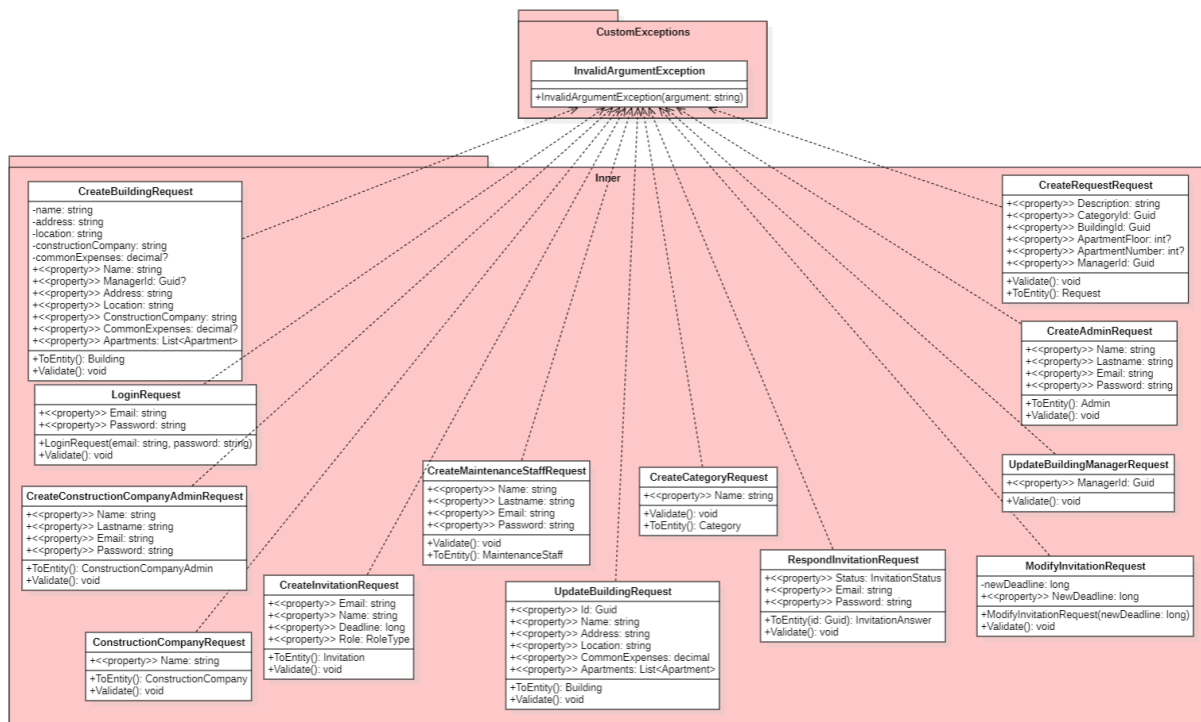
Se encuentran las implementaciones concretas de la lógica, lo referente a las reglas de negocio y a las interacciones entre clases. Además, implementamos los filtros de autenticación y de custom exceptions.





## Models

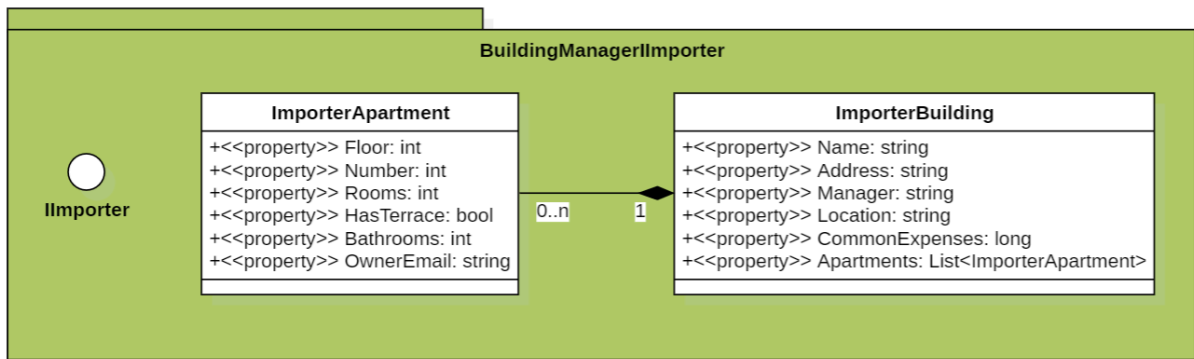
Se encuentran los modelos de respuestas y requests que utilizan los controllers.





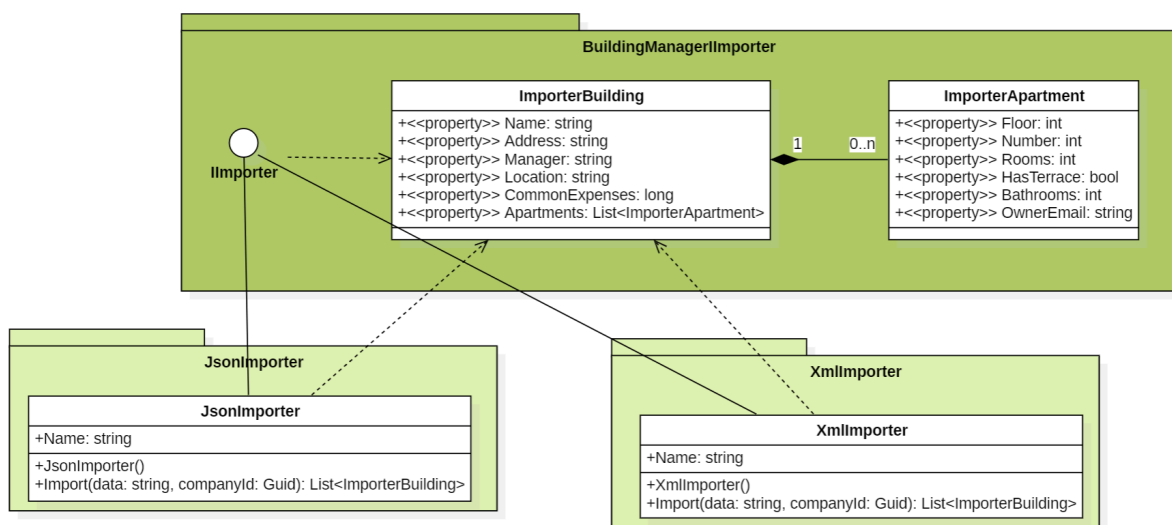


de lógica. De esta forma se crea una dependencia de la lógica a este, pero esa ya existía porque `ImporterLogic` conoce a `Importer`.



## JsonImporter y XmlImporter

En estos paquetes se encuentran las clases importadores que implementan la interfaz de `IImporter`.



## Reflection

Se implementó reflexión para la importación de edificios con el propósito de poder agregar nuevos importadores al sistema en tiempo de ejecución sin necesidad de recompilar la aplicación.

Los nuevos importadores realizados por terceros deben implementar la interfaz **IImporter** para que puedan ser utilizados en la aplicación.

```
public interface IImporter
{
    List<ImporterBuilding> Import(string data, Guid companyId);
    string Name { get; }
}
```

Es decir, el nuevo importador debe tener un nombre único y debe implementar el método Import que recibe el contenido del archivo con edificios en formato de string, y el id de la empresa constructora. Este método debe convertir el string en una lista de ImporterBuilding. Pensamos dos caminos para este método, el primero que convirtiera la data a una lista de la entidad Building y el segundo que convirtiera a una lista de una clase ImporterBuilding que se encuentra en el mismo paquete que la interfaz IImporter. El problema con la primera opción y por lo cual nos decidimos por la segunda opción es que al hacer esto generamos una dependencia de los importadores desarrollados por terceros con nuestro dominio, de forma que en un futuro no solo tendríamos que mantener nuestro código sino también el de los terceros. No solo dependerían del dominio sino también de la lógica de buildings, company y user en principio, ya que por ejemplo en el json dado en aulas se recibe el mail del encargado y propietarios en vez de sus ids como utiliza Building. Por esta razón la data debe ser convertida a la siguiente clase:

```
public class ImporterBuilding
{
    public string Name { get; set; }
    public string Address { get; set; }
    public string Manager { get; set; }
    public string Location { get; set; }
    public long CommonExpenses { get; set; }
    public List<ImporterApartment> Apartments { get; set; }
}
public class ImporterApartment
{
    public int Floor { get; set; }
    public int Number { get; set; }
    public int Rooms { get; set; }
    public bool HasTerrace { get; set; }
    public int Bathrooms { get; set; }
    public string OwnerEmail { get; set; }
}
```

Luego en la lógica esta clase es convertida en la entidad Building. Por último la lógica le pide a IBuildingLogic que realice la creación de cada uno de los edificios, chequeando antes que cada uno de ellos puedan ser creados y a su vez que no hayan edificios con el mismo nombre o dirección en la data.

**Para el JsonImporter la estructura del archivo debe ser la siguiente:**

```
{
  "edificios": [
    {
      "nombre": string,
      "direccion": {
        "calle_principal": string,
        "numero_puerta": int,
        "calle_secundaria": string
      },
      "encargado": string o null,
      "gps": {
        "latitud": int,
        "longitud": int
      },
      "gastos_comunes": int,
      "departamentos": [
        {
          "piso": int,
          "numero_puerta": int,
          "habitaciones": int,
          "conTerraza": bool,
          "baños": int,
          "propietarioEmail": string
        }
      ]
    }
  ]
}
```

Por ejemplo:

```
{
  "edificios": [
    {
      "nombre": "Las torres",
      "direccion": {
        "calle_principal": "Av. 6 de Diciembre",
        "numero_puerta": 3030,
        "calle_secundaria": "Av. Eloy Alfaro"
      },
      "encargado": "laura@gmail.com",
      "gps": {
        "latitud": -0.176,
        "longitud": -78.48
      },
      "gastos_comunes": 5000,
      "departamentos": [
        {
          "piso": 1,
          "numero_puerta": 101,
```

```

        "habitaciones": 3,
        "conTerraza": false,
        "baños": 2,
        "propietarioEmail": "juan.perez@gmail.com"
    }
}
]
}
]
}

```

**Para el XmlImporter la estructura del archivo debe ser la siguiente:**

```

<XmlImporterBuilding>
  <edificios>
    <XmlBuilding>
      <nombre>string</nombre>
      <direccion>
        <calle_principal>string</calle_principal>
        <numero_puerta>int</numero_puerta>
        <calle_secundaria>string</calle_secundaria>
      </direccion>
      <encargado>string</encargado>
      <gps>
        <latitud>int</latitud>
        <longitud>int</longitud>
      </gps>
      <gastos_comunes>int</gastos_comunes>
      <departamentos>
        <XmlApartment>
          <piso>int</piso>
          <numero_puerta>int</numero_puerta>
          <habitaciones>int</habitaciones>
          <conTerraza>bool</conTerraza>
          <baños>int</baños>
          <propietarioEmail>string</propietarioEmail>
        </XmlApartment>
      </departamentos>
    </XmlBuilding>
  </edificios>
</XmlImporterBuilding>

```

Por ejemplo:

```

<XmlImporterBuilding>
  <edificios>
    <XmlBuilding>
      <nombre>Edificio1</nombre>
      <direccion>
        <calle_principal>Av. 6 de Diciembre</calle_principal>
        <numero_puerta>3030</numero_puerta>

```

```

        <calle_secundaria>Av. Eloy Alfaro</calle_secundaria>
    </direccion>
    <encargado>laura@gmail.com</encargado>
    <gps>
        <latitud>-0.1769</latitud>
        <longitud>-78.489</longitud>
    </gps>
    <gastos_comunes>12345</gastos_comunes>
    <departamentos>
        <XmlApartment>
            <piso>1</piso>
            <numero_puerta>101</numero_puerta>
            <habitaciones>3</habitaciones>
            <conTerraza>>false</conTerraza>
            <baños>2</baños>
            <propietarioEmail>juan.perez@gmail.com</propietarioEmail>
        </XmlApartment>
    </departamentos>
</XmlBuilding>
</edificios>
</XmlImporterBuilding>

```

A su vez, la interfaz gráfica no sufre cambios al agregar o eliminar importadores ya que para mostrar los nombres de importadores existentes se los pide al backend, que a su vez este levanta los .dll ubicados en la carpeta Importers.

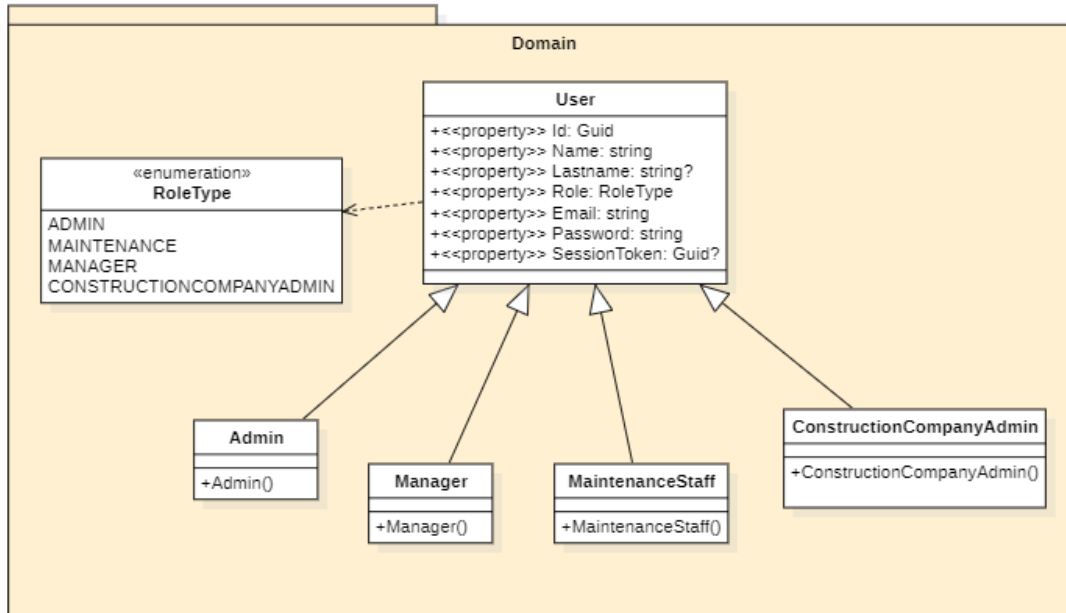
Tomamos la decisión de que se envíe a través de un endpoint la data del archivo en vez de que este se encuentre en el servidor puesto que nos parece más realista que el usuario no tenga que hacer nada en el servidor para poder importar edificios.



## Justificación y explicación del diseño

### Usuarios

Para esta segunda entrega se pidió un nuevo tipo de usuario ConstructionCompanyAdmin.

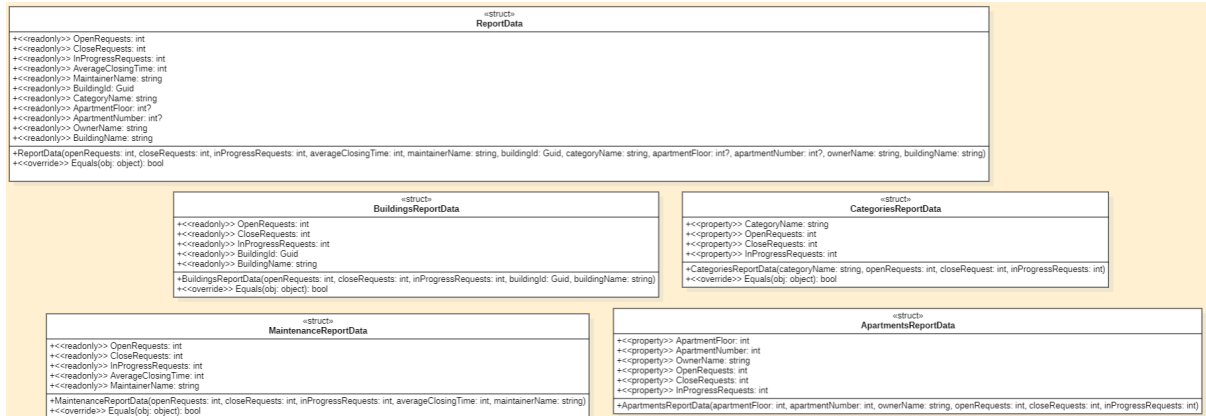


Al haber hecho herencia de usuarios en la entrega 1, agregar un nuevo rol no nos impactó en el resto de las capas. Para este nuevo usuario que no tiene apellido, en su constructor se setea este en un string vacío. A diferencia de los otros tipos de usuarios, para este nuevo rol se implementó una lógica aparte. Esto se realizó a partir del requerimiento “un admin de una empresa constructora puede crear otros usuarios admins para gestionar su empresa constructora”. Esta nueva lógica conoce las interfaces IUserLogic y IConstructionCompanyLogic, y al crear este tipo de usuario el método CreateConstructionCompanyAdmin recibe el nuevo usuario y el sessionToken del usuario creador. Luego, se le pide el id del usuario creador a partir del sessionToken a la IUserLogic, y el id de la empresa constructora de este usuario a IConstructionCompanyLogic, esto es para separar responsabilidades. Luego el IUserLogic crea el usuario agregándolo a la base de datos en la tabla Users implementada como TPH(Table per hierarchy) con su discriminante correspondiente. Por último IConstructionCompanyLogic le asocia la empresa constructora del usuario creador al nuevo usuario. Para el vínculo entre la empresa constructora y los usuarios de tipo constructionCompanyAdmin se utiliza una tercera tabla que asocia el id del usuario con el id de la empresa constructora a través de FKs. Otra forma de realizar este vínculo podría haber sido un campo en la tabla de Users para el id de la empresa constructora ya que es una relación 1 a n, pero esto generaría que todo el resto de tipos de usuarios tengan este campo en null y crearía chequeos innecesarios.

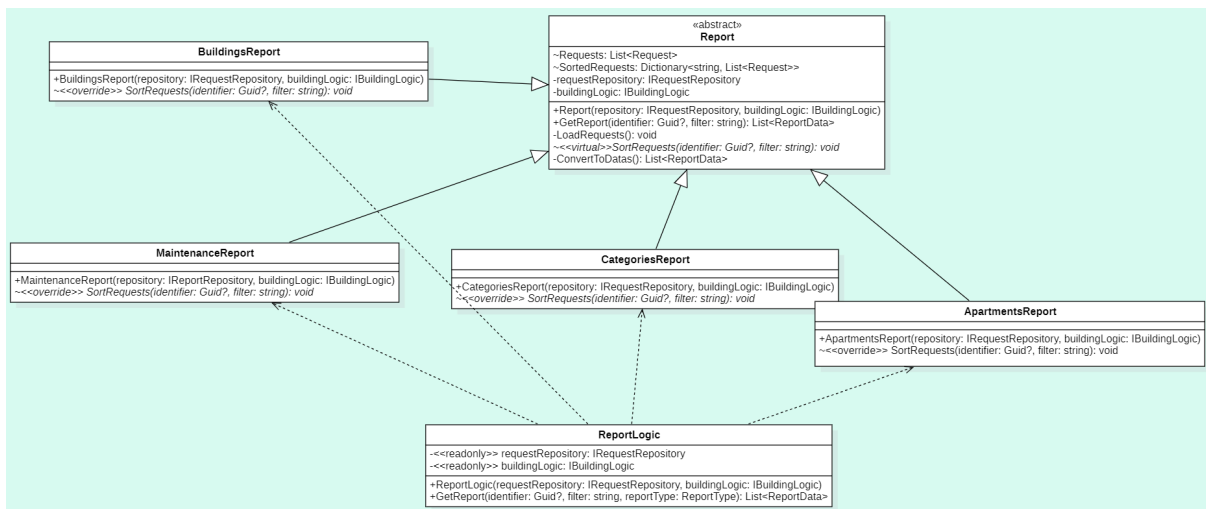
## Reportes

Un nuevo requerimiento es un nuevo reporte llamado ApartmentsReport.

Para su implementación se creó un nuevo struct ApartmentsReportData siguiendo las decisiones y diseño implementados en la entrega anterior y se agregaron ownerName, apartmentFloor y apartmentNumber al struct ReportData, el cual tiene todos los atributos del resto de reportDatas.

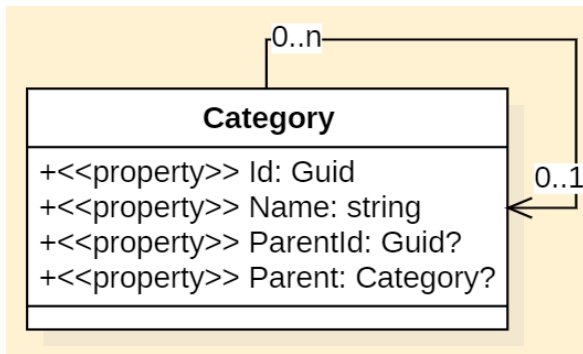


A nivel de la capa lógica se implementó una nueva clase ApartmentsReport siguiendo el template method pattern realizado en la primera entrega.



## Categorías

Para el nuevo requerimiento que pide que las categorías pueden involucrarse con una categoría padre se agregó a la clase categoría una property llamada Parent de tipo Category. Pensamos en utilizar el patrón composite pero nos pareció innecesario ya que pensando en el futuro no vemos nuevas posibilidades para realizar una herencia.



A nivel de base de datos la tabla Categories tiene el parentId como FK.

Para la lógica de asociar una categoría a otra realizamos un chequeo de que no se pueda hacer una asociación circular. Por ejemplo tenemos las siguientes categorías vinculadas entre sí:

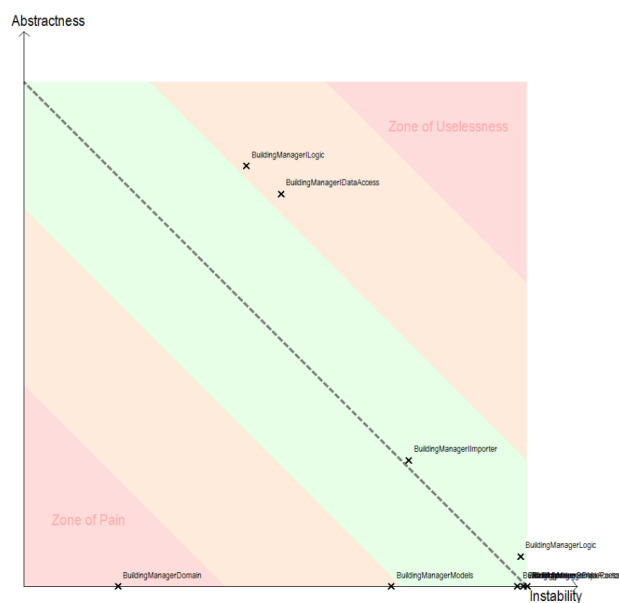
- Categoría A
  - Categoría B
    - Categoría C

No se puede asociar a la categoría C como padre de la categoría A.

## Invitaciones

En esta segunda entrega se puede, además de invitar a futuros encargados, invitar a futuros administradores de empresas constructoras. Para esto se modificó el modelo de entrada de `CreateInvitationRequest` para que tenga un rol del futuro usuario. A nivel de lógica realizamos una verificación para que ese rol sea de tipo encargado o administrador de empresas constructora únicamente.

## Métricas a nivel de paquetes



Assemblies	# lines of code	# IL instruction	# Types	# Abstract Types	# lines of comment	% Comment	% Coverage	Afferent Coupling	Efferent Coupling	Relational Cohesion	Instability	Abstractness	Distance
BuildingManagerDomain v1.0.0.0	355	2181	27	0	0	0	-	100	23	0.85	0.19	0	0.57
BuildingManagerImporter v1.0.0.0	25	98	4	1	0	0	-	4	13	0.75	0.76	0.25	0.01
BuildingManagerLogic v1.0.0.0	2	11	12	10	0	0	-	34	27	0.08	0.44	0.83	0.2
BuildingManagerDataAccess v1.0.0.0	2	14	9	7	0	0	-	22	23	0.11	0.51	0.78	0.2
BuildingManagerLogic v1.0.0.0	398	2745	17	1	0	0	-	1	77	0.71	0.99	0.06	0.03
BuildingManagerDataAccess v1.0.0.0	3432	42356	57	0	32	0.92	-	1	127	0.98	0.99	0	0.01
BuildingManagerServiceFactory v1.0.0.0	20	81	1	0	0	0	-	1	51	1	0.98	0	0.01
BuildingManagerModels v1.0.0.0	874	5420	48	0	0	0	-	17	46	0.42	0.73	0	0.19
BuildingManagerApi v1.0.0.0	161	1169	21	0	5	3.01	-	0	153	0.86	1	0	0
JsonImporter v1.0.0.0	55	344	6	0	0	0	-	0	23	1.67	1	0	0
XmlImporter v1.0.0.0	57	365	6	0	0	0	-	0	29	1.67	1	0	0

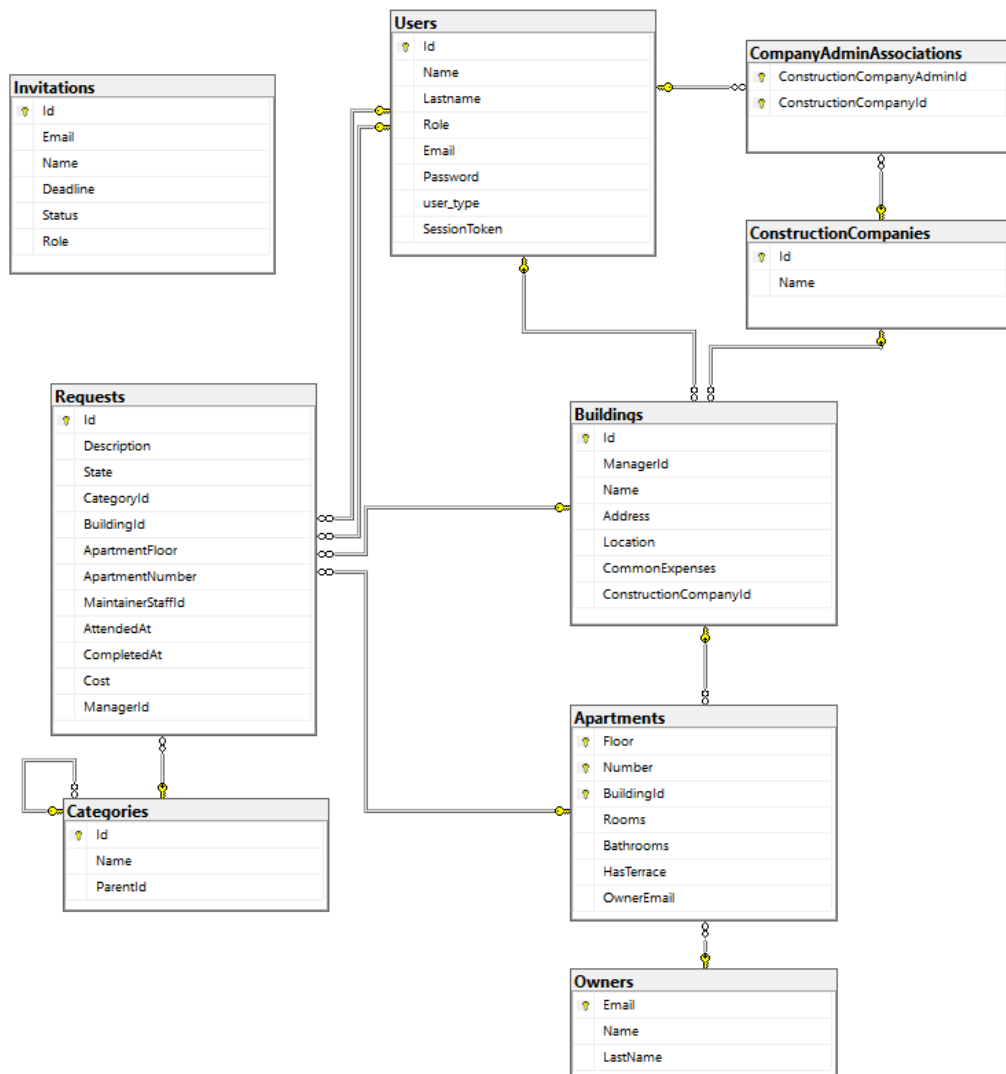
Se cumple **ADP** ya que no hay ciclos entre paquetes.

No se cumple **SDP** porque BuildingManagerLogic(inestabilidad = 0.44) depende de BuildingManagerImporter(inestabilidad = 0.76), para que se cumpla este principio deberían todos los paquetes depender de paquetes más estables que ellos.

No se cumple **SAP** porque el paquete de dominio tiene una distancia mayor a 0.30. Esto se debe a que no tiene ninguna clase abstracta y muchos otros paquetes dependen de él.

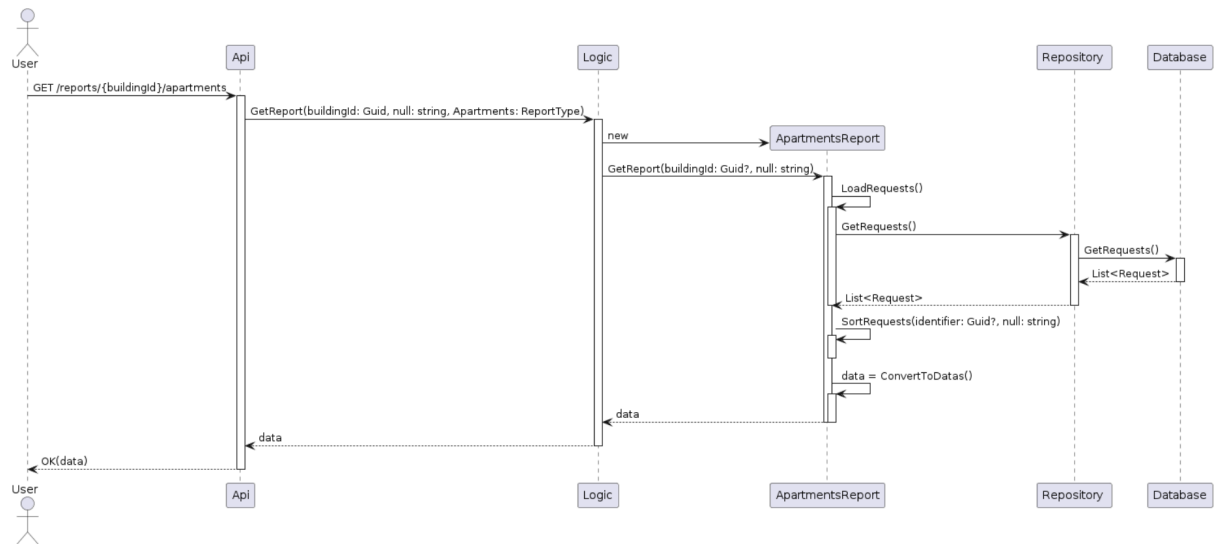
No se cumple la cohesión relacional ya que este número debería estar entre 1.5 y 4. El problema con la mayoría de los paquetes es que son demasiadas clases que no tienen relación entre ellas. Por ejemplo, muchas clases de lógica tienen relación con otras pero en vez de conocer su implementación, conocen sus interfaces, preferimos esto a tener una buena cohesión relacional.

## Modelo de tablas de la estructura de la base de datos



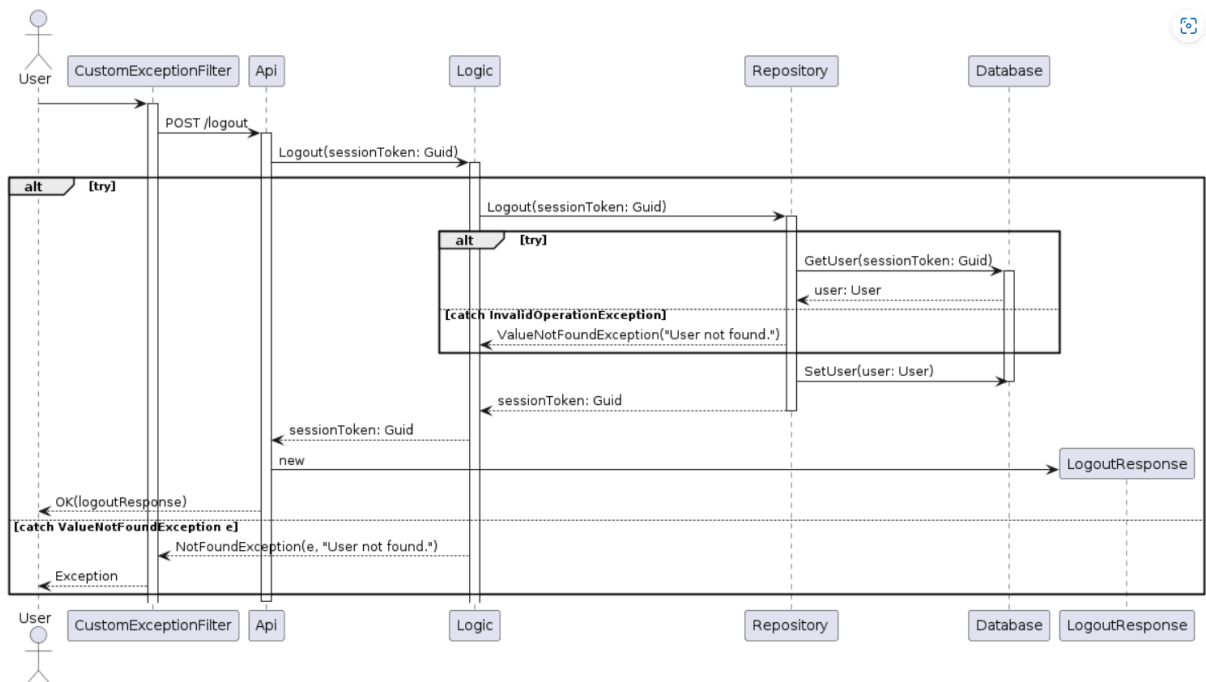
# Diagramas de interacción

## Reporte de apartamentos

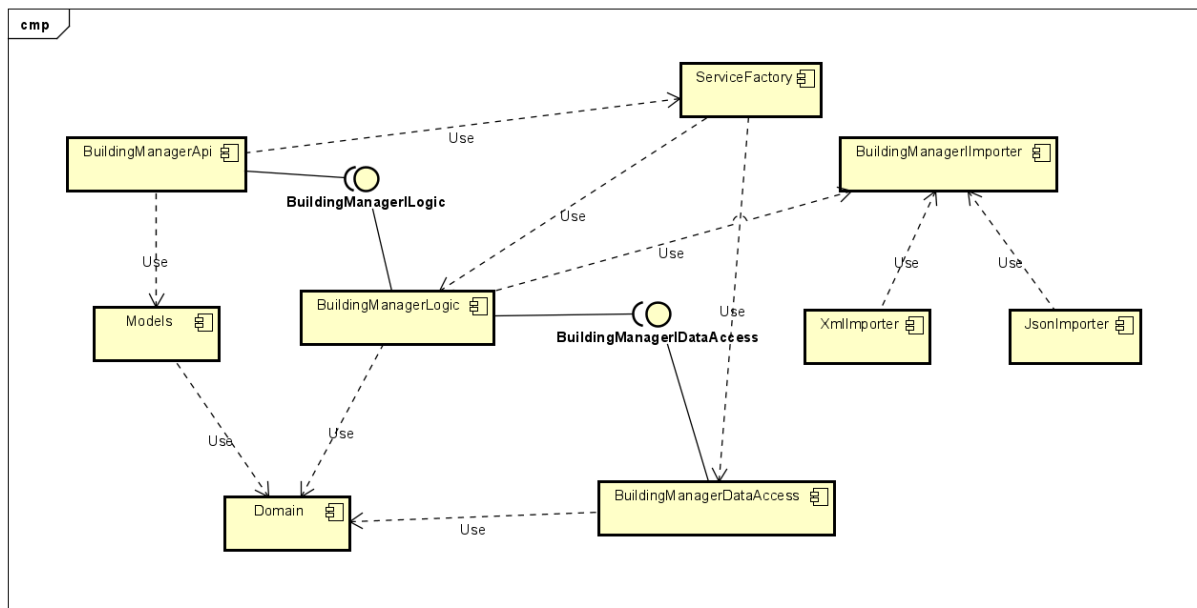


## Logout

No sabemos como suele ser la notación de excepciones y filtros



## Diagrama de componentes y conectores



## Mecanismos utilizados para permitir la extensibilidad

A continuación, se explican algunos mecanismos utilizados en el obligatorio para permitir la extensibilidad:

### Reflection

Como mencionamos anteriormente, para permitir la integración de nuevos importadores desarrollados por terceros, se ha definido la interfaz `IImporter`. Esta interfaz establece un contrato que todo importador debe cumplir:

```
public interface IImporter
{
    List<ImporterBuilding> Import(string data, Guid companyId);
    string Name { get; }
}
```

Los importadores deben implementar el método `Import`, que convierte los datos proporcionados en una lista de `ImporterBuilding`. Esta separación asegura que los importadores no dependan directamente de las entidades del dominio principal, evitando acoplamientos innecesarios y facilitando el mantenimiento y la integración.

Además, es extensible a la carga de componentes .dll (Assembly) ya que se puede agregar nuevos importadores sin necesidad de interrumpir la ejecución del programa.

## Extensibilidad en la interfaz gráfica

Volviendo sobre la importación de datos, la interfaz gráfica es completamente agnóstica respecto a los importadores específicos, si se quiere agregar un nuevo importador, no se requiere realizar ningún cambio sobre la UI, ya que en la lista de opciones de los importadores disponibles, al momento de agregar el nuevo importador a la carpeta del servidor, se mostrará en la lista de la UI esa nueva opción, sin tener que agregar algún valor hardcodeado para que se pueda seleccionar; además, la funcionalidad de carga de archivo permite la carga en cualquier formato, por lo que si se tiene el archivo en el formato correcto que acepta el importador, todo funcionará sin inconvenientes

## Extensibilidad en roles del sistema

La inclusión del nuevo rol `ConstructionCompanyAdmin` se facilitó gracias a la herencia de usuarios implementada en la primera entrega.

La clase padre `User` tiene como propiedad `Role` que es de tipo `RoleType`, y a su vez `RoleType` es un enum; quiere decir que si queremos agregar un nuevo rol de usuario, deberíamos agregar el nuevo rol en este enum, y crear la clase específica heredando de la clase `User`, lo cual permite agregar todos los roles que queramos y clases hijas que hereden de `User` con esos roles específicos sin impactar en otras funcionalidades.

## Resumen de las mejoras al diseño

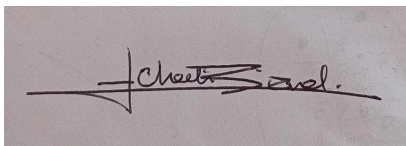
En este segundo obligatorio realizamos una mejor separación de responsabilidades. Por ejemplo, previamente existía repetición de código en diferentes lógicas y repositorio para la transformación de `sessionToken` a `userId`. Para corregir esto generamos el método `getUserIdFromSessionToken` en la lógica del usuario ya que es una responsabilidad propia de esta, y las otras lógicas que necesitan el `userId` conocen al `IUserLogic` a través de inyección de dependencias y llaman a este método. Esto no solo lo hicimos con este método sino con varios, de forma que algunas lógicas conocen a otras interfaces de lógicas, de forma de disminuir la repetición de código y la cohesión de responsabilidades de clases.



## DECLARACIÓN DE AUTORÍA

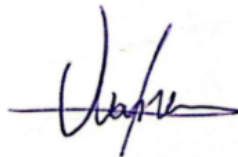
La declaración de autoría debe constar del siguiente texto: Nosotros, Franca Chechi, Anthony Núñez y Juan Tejera , declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizamos Diseño de Aplicaciones 2;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Franca Chechi

10/06/2024



Juan Tejera  
10/06/2024



Rafael Núñez

10/06/2024

# Anexo

## Cobertura de código:

Hierarchy	Covered (Blocks)	Not Covered (Blocks)	Covered (Lines)	Partially Covered (Lines)	Not Covered (Lines)
fchiq_COMPUDEFRANCA_2024-06-10.19_31_37.coverage	20527	8161	13697	348	15103
buildingmanagerapitest.dll	2971	39	2050	28	21
buildingmanagermodels.dll	2151	141	1149	68	66
buildingmanagerlogic.dll	4	0	4	0	0
buildingmanagerdomaintest.dll	1330	85	1450	178	36
buildingmanagerapi.dll	257	138	232	0	98
buildingmanagerimportertest.dll	77	0	77	0	0
buildingmanagerimporter.dll	24	2	24	0	3
buildingmanagerdomain.dll	789	25	440	48	16
buildingmanagerdataaccess.dll	5	0	2	0	0
buildingmanagerlogictest.dll	5364	54	3125	5	51
buildingmanagerlogic.dll	923	24	698	6	18
buildingmanagerdataaccesstest.dll	4265	55	3774	13	42
buildingmanagerdataaccess.dll	2367	7598	672	2	14752

En el caso de los paquetes de tests todas sus líneas van a aparecer como not covered.

En el paquete buildingManagerModels, las líneas no cubiertas son las de los equals:

```
0 references | Rafael, 3 days ago | 1 author, 2 changes
public override bool Equals(object obj)
{
    if (obj == null || GetType() != obj.GetType())
        return false;

    var other = (BuildingDetailsResponse)obj;
    return Id == other.Id &&
        Name == other.Name &&
        Address == other.Address &&
        Location == other.Location &&
        CommonExpenses == other.CommonExpenses &&
        ManagerId == other.ManagerId &&
        Manager == other.Manager &&
        ConstructionCompanyId == other.ConstructionCompanyId &&
        ConstructionCompany == other.ConstructionCompany &&
        Apartments == other.Apartments;
}
```

En el caso del paquete buildingManagerApi, las líneas no cubiertas son de los filtros y de program:

Hierarchy	Covered (Blocks)	Not Covered (Blocks)	Covered (Lines)	Partially Covered (Lines)	Not Covered (Lines)
fchiq_COMPUDEFRANCA_2024-06-10.19_31_37.coverage	20527	8161	13697	348	15103
buildingmanagerapitest.dll	2971	39	2050	28	21
buildingmanagermodels.dll	2151	141	1149	68	66
buildingmanagerlogic.dll	4	0	4	0	0
buildingmanagerdomaintest.dll	1330	85	1450	178	36
buildingmanagerapi.dll	257	138	232	0	98
BuildingManagerApi	0	36	0	0	27
Program	0	29	0	0	22
Program.<>c	0	7	0	0	5
BuildingManagerApi.Filters	0	102	0	0	71
BuildingManagerApi.Controllers	257	0	232	0	0

En el caso de buildingManagerImporter las líneas no cubiertas son de la custom exception

Hierarchy	Covered (Blocks)	Not Covered (Blocks)	Covered (Lines)	Partially Covered (Lines)	Not Covered (Lines)
fchiq_COMPUDEFRANCA_2024-06-10.19_31_37.coverage	20527	8161	13697	348	15103
buildingmanagerapitest.dll	2971	39	2050	28	21
buildingmanagermodels.dll	2151	141	1149	68	66
buildingmanagerlogic.dll	4	0	4	0	0
buildingmanagerdomaintest.dll	1330	85	1450	178	36
buildingmanagerapi.dll	257	138	232	0	98
buildingmanagerimportertest.dll	77	0	77	0	0
buildingmanagerimporter.dll	24	2	24	0	3
BuildingManagerImporter	24	0	24	0	0
BuildingManagerImporter.Exceptions	0	2	0	0	3

En el paquete buildingManagerDomain las líneas no cubiertas son por algunos equals y setters.

Hierarchy	Covered (Blocks)	Not Covered (Blocks)	Covered (Lines)	Partially Covered (Lines)	Not Covered (Lines)	
Invitation	21	1	13	5	0	
InvitationAnswer	26	1	10	5	0	
ListBuildingData	22	0	13	0	0	
ListManagersResponse	64	4	25	5	2	
get_Managers()	1	0	1	0	0	
set_Managers(System.Collections.Generic.List<BuildingManag	1	0	1	0	0	
ListManagersResponse(System.Collections.Generic.List<Build	22	0	13	0	0	
Equals(object)	40	4	10	5	2	
MaintenanceRequestData	33	0	33	0	0	

En el caso de buildingManagerDataAccess la mayoría de las líneas no cubiertas son por las migraciones y el contexto.

## Documentación de la API

La documentación de la API se encuentra en el siguiente enlace:

<https://docs.google.com/spreadsheets/d/19AEeouc6IU4U9RW2dG2NUOUUu55BoAP6Jr10FQElyoc/edit?usp=sharing>

También se agrega dentro de la carpeta de entrega, en el mismo directorio del presente documento con el nombre de “Documentación de la api 2”.