Universidad ORT Uruguay

Facultad de Ingeniería en Sistemas

Obligatorio 1 - Diseño de Aplicaciones 2

Entregado como requisito del curso de la materia

Sebastian Escuder - 242739

Bruno Odella - 231665

Tutores: Francisco Bouza, Juan Irabedra y Santiago Tonarelli

2024

Link al repositorio de GitHub:https://github.com/IngSoft-DA2-2023-2/242739-231665.git

1. Introducción

1a. Descripción del sistema y propósito

El sistema está diseñado para facilitar la gestión y operación de varios edificios, centralizando el control y la supervisión de las instalaciones en una sola plataforma. Este sistema es especialmente útil para administradores de propiedades, personal de mantenimiento, y personal administrativo que necesitan una herramienta eficiente para coordinar actividades, mantenimiento, y comunicaciones con los residentes o inquilinos.

1b. Algunas aclaraciones

Como equipo de trabajo, acordamos programar todo el código del backend en ingles, además de usar Git usando la metodología de GitFlow para las features, y hacer merges con develop hasta la finalización, para luego hacer a master.

2. Diseño de la Aplicación

2a. Arquitectura general

El sistema desarrollado es una **Web API RESTful** implementada utilizando **ASP.NET Core**.

Componentes del sistema:

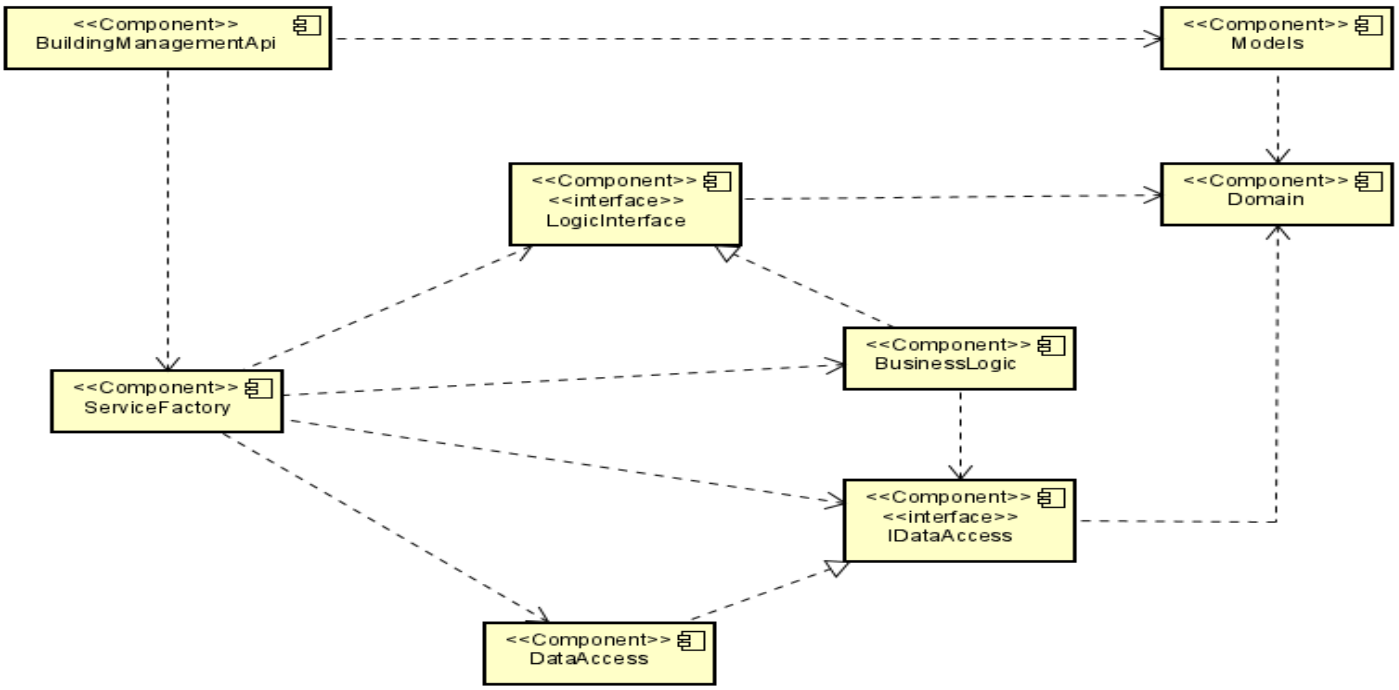
- Controladores:
- Lógica de negocio

- Repositorios
- Modelos del dominio
- Modelos de entrada y salida

Flujo de datos

1. Solicitud
2. Controlador
3. Lógica de Negocio
4. Repositorio
5. Respuesta

2b. Diagramas de componentes y paquetes



Componente Principal (BuildingManagementApi):
Función: Actúa como el punto de entrada de la aplicación y maneja las solicitudes HTTP de los clientes.
Justificación: Es el núcleo de la API RESTful que expone los endpoints y delega las operaciones a los componentes pertinentes, garantizando que la interfaz externa siga los principios REST como la comunicación sin estado y la manipulación de recursos a través de representaciones.

Componentes de Dominio y Modelos:
Dominio: Define las entidades del negocio como edificios, apartamentos, etc.
Modelos: Contienen las representaciones de datos que se transfieren entre los diferentes componentes, especialmente entre la API y los clientes.

Justificación: Separar el dominio y los modelos permite una clara distinción entre la lógica de negocio y los datos transferidos, facilitando la validación, el mantenimiento y la escalabilidad del sistema.

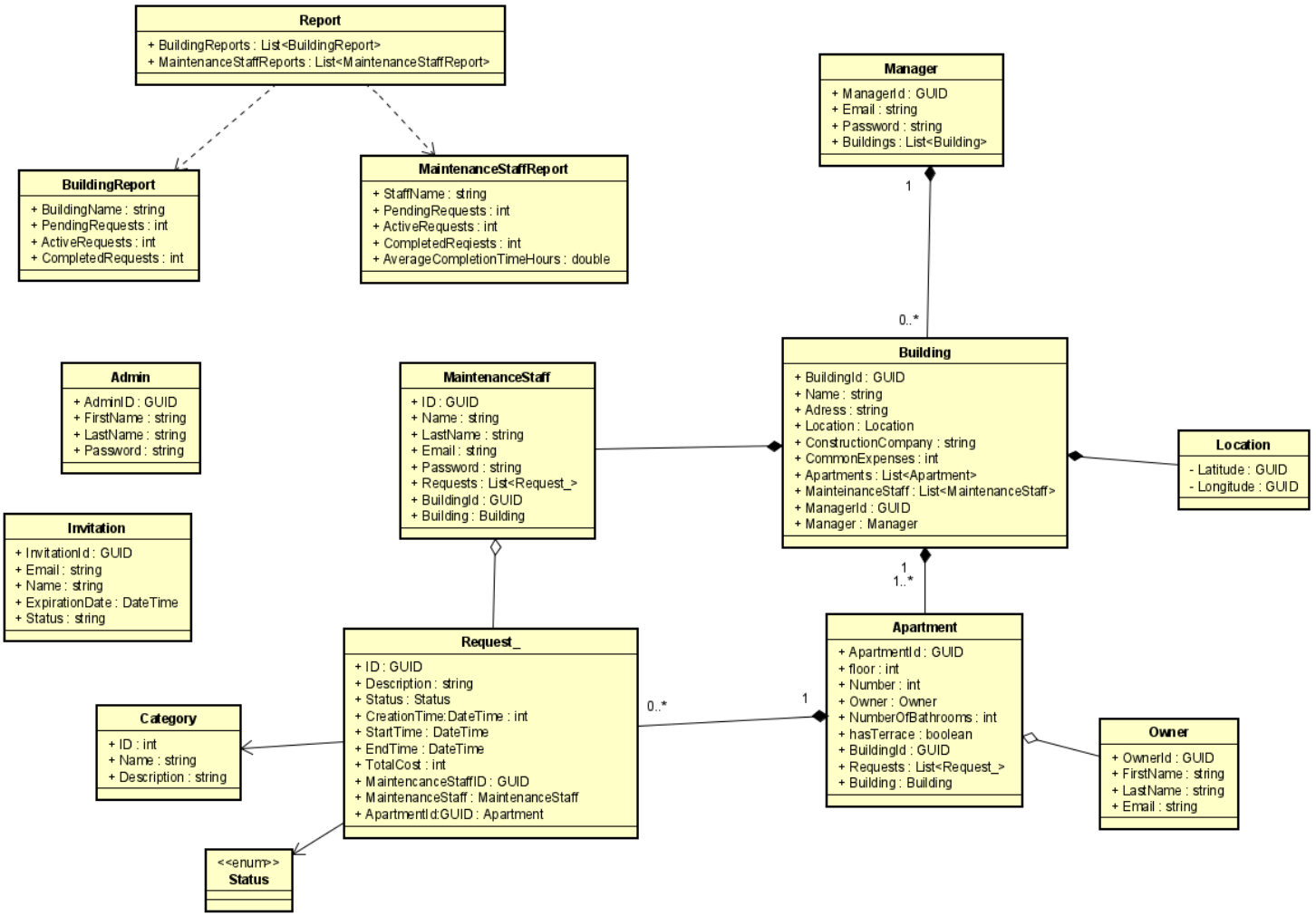
Capa de Lógica de Negocio (BusinessLogic):
Función: Contiene la lógica de negocio que manipula datos del dominio en respuesta a las operaciones API.
Justificación: Centraliza las reglas del negocio en un solo lugar, facilitando su gestión y reutilización. Esto asegura que todas las decisiones importantes del negocio se manejen de manera coherente y conforme a las políticas del sistema.

Interfaces (LogicInterface y IDataAccess):
LogicInterface: Define los contratos para las operaciones de lógica de negocio.
IDataAccess: Define los contratos para las operaciones de acceso a datos.
Justificación: Usar interfaces promueve un diseño desacoplado y mejora la testabilidad del sistema. Las implementaciones pueden ser intercambiadas fácilmente sin modificar los consumidores, lo que es esencial para el mantenimiento y la escalabilidad.

Capa de Acceso a Datos (DataAccess):
Función: Implementa las interfaces de acceso a datos, interactuando directamente con la base de datos.
Justificación: Separar el acceso a datos de la lógica de negocio permite cambiar la fuente de datos o la tecnología de base de datos sin afectar la lógica de negocio, lo que proporciona flexibilidad y robustez al sistema.

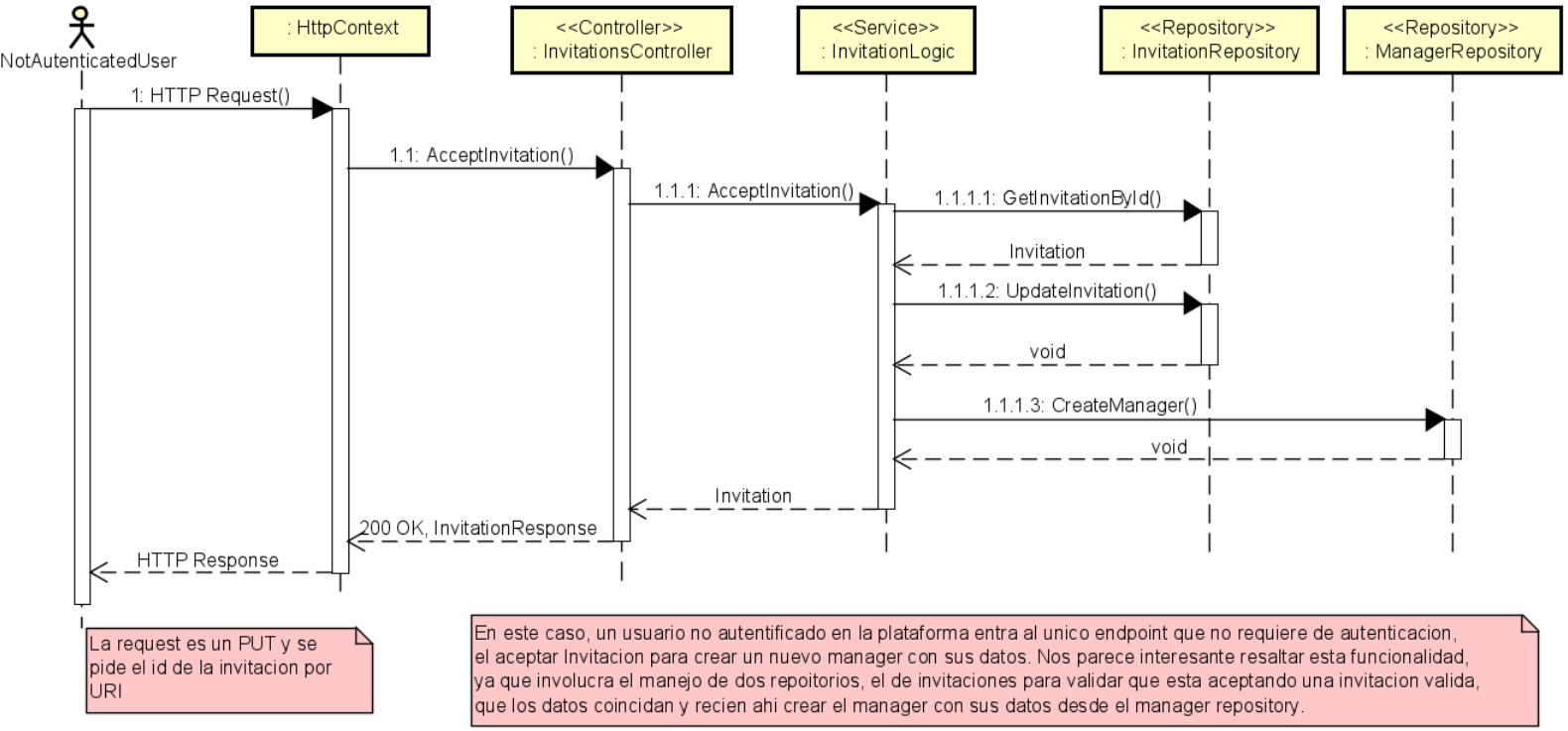
Service Factory:
Función: Es responsable de la creación de objetos, especialmente servicios utilizados en la aplicación.
Justificación: Implementar una fábrica de servicios centraliza la creación de objetos, reduciendo la complejidad en otras partes de la aplicación y facilitando la inyección de dependencias.
Este diseño promueve una arquitectura limpia y separada que sigue los principios SOLID, haciendo el sistema más fácil de entender, mantener y escalar. Cada componente tiene responsabilidades bien definidas, cumpliendo con los principios de la arquitectura RESTful donde los recursos son manejados a través de representaciones y manipulados mediante interfaces bien definidas, asegurando una alta cohesión y bajo acoplamiento entre componentes.

2c. Diagrama de clases del dominio

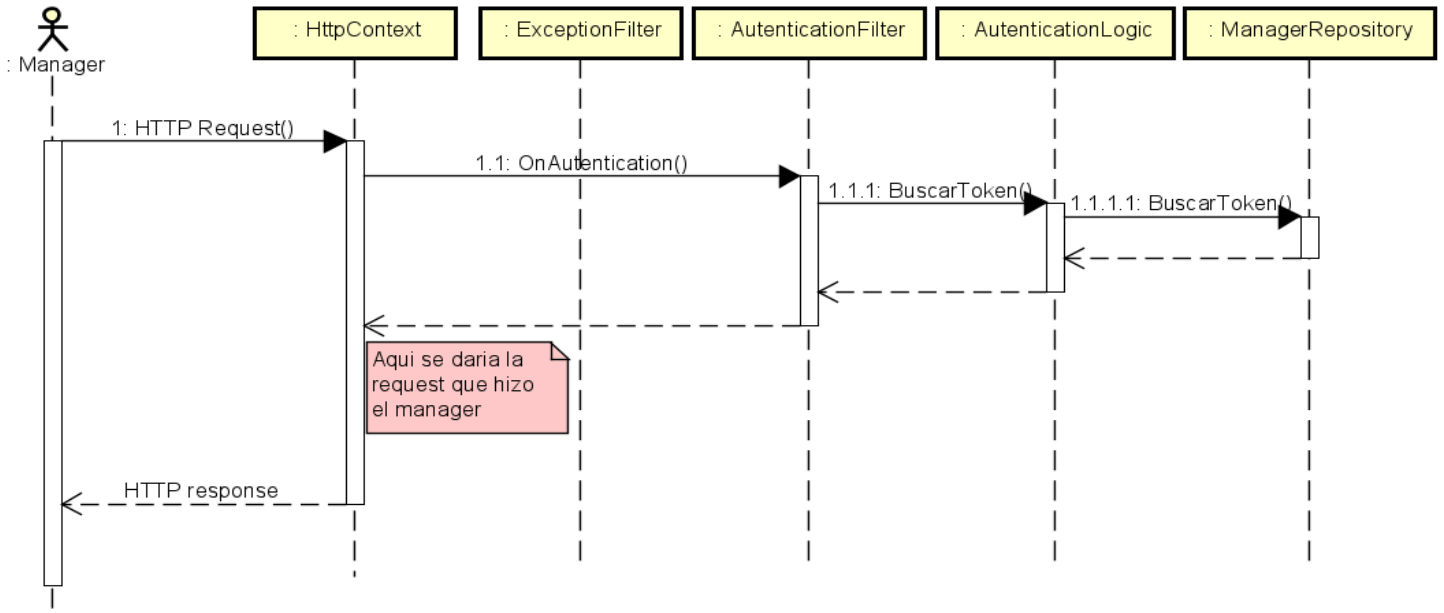


2d. Diagramas de comunicación e interacción

2da. Funcionalidad de invitación, diagrama de interacción

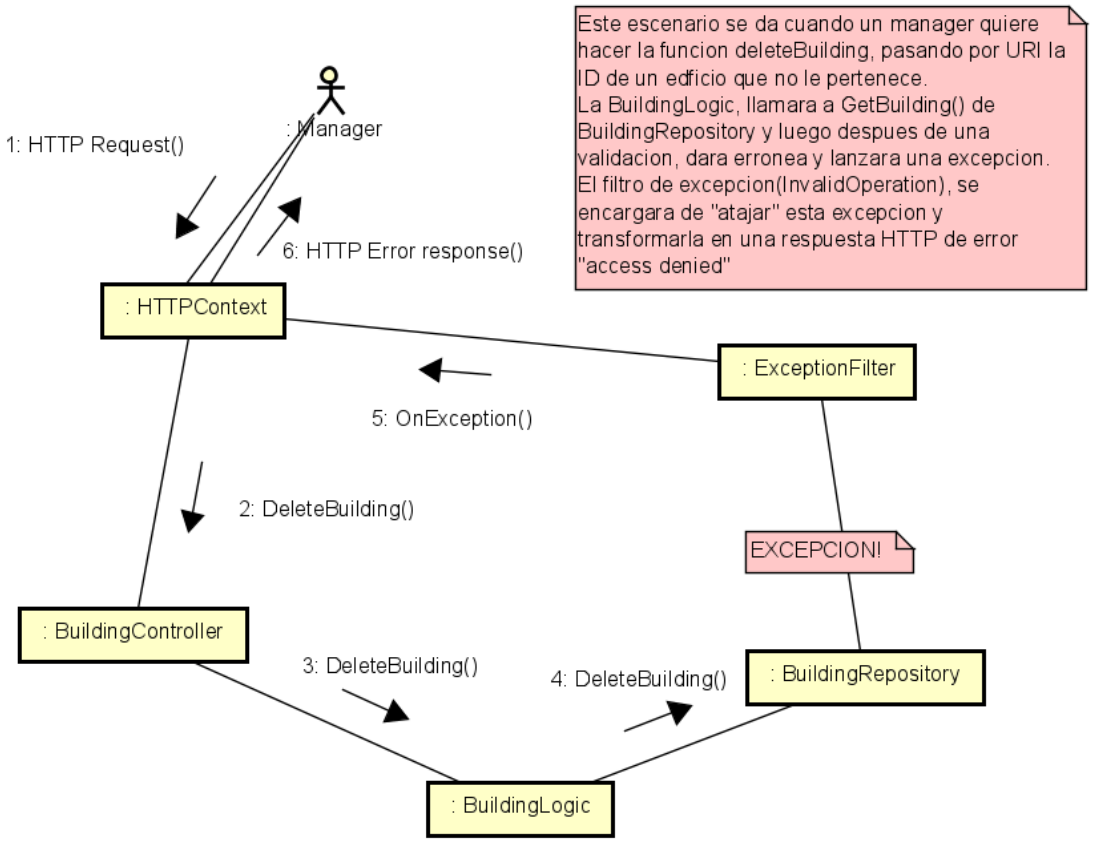


2db. Funcionalidad autenticación, diagrama de interacción:

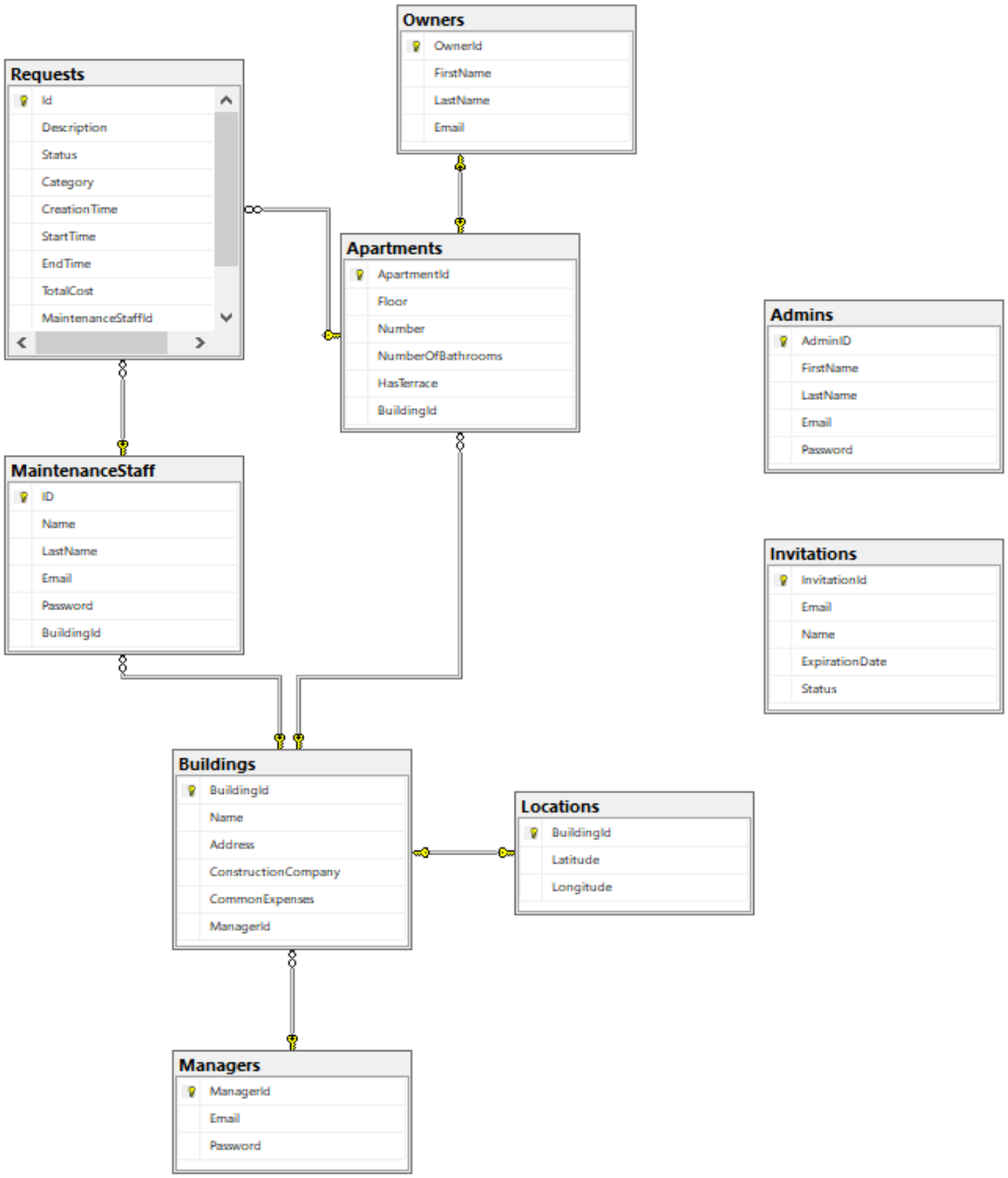


En este caso, estamos viendo que pasa cuando un manager solicita alguna de las acciones que solo su ROL puede ejecutar, y como se maneja esto con el filtro de autenticacion. Una vez validado, despues del punto 1.1.1.1, Vuelve al httpcontext y finalmente ejecuta el metodo del controlador que se habia llamado.

2dc. Funcionalidad filtro de excepciones, diagrama de comunicación



2e. Modelo de la base de datos de SQL Server Management Studio.



3. Implementación de la API

3a. Aclaración sobre la documentación de la API

El archivo que subimos es el boceto que hicimos en previamente a empezar la api, tiene muchos cambios que implementar.

3b. Seguridad, autenticacion y autorizacion

Manejo de token de autenticación:
El sistema utiliza un token de autenticación basado en GUID, que es enviado en el encabezado (header) de las solicitudes HTTP. Este token es esencial para identificar y autenticar a cada usuario dentro de la API. El GUID utilizado como token corresponde al ID del usuario específico, que es el identificador único de cada tipo de usuario tiene dentro del sistema.

Filtro de Autenticación
Para asegurar que cada solicitud sea autenticada de manera efectiva, se implementó un filtro de autenticación en los controladores. Este filtro intercepta las solicitudes entrantes para validar la presencia y la validez del token de autenticación en el encabezado, o sea que pertenezca a la tabla del tipo de usuario(Manager, Admin o MaintenanceStaff) que puede acceder a este endpoint. La implementación de este filtro asegura que todas las operaciones que requieren autenticación sean procesadas solo después de validar satisfactoriamente la identidad del solicitante.

Autorización
Una vez autenticado el usuario y su rol, el sistema realiza una verificación de autorización a nivel de la BusinessLogic para asegurar que el usuario tenga permiso para acceder a los recursos solicitados. Este proceso de autorización implica verificar que el ID extraído del token de autenticación corresponda al propietario o administrador de los recursos que intenta acceder o modificar específicos. Esto es crucial para prevenir que los usuarios interactúen con recursos fuera de su dominio asignado, como por ejemplo,un manager queriendo gestionar edificios o apartamentos que no están bajo su administración.

4. Manejo de Excepciones

4a. Filtro de excepciones

Este filtro garantiza que cualquier excepción que se pueda dar, ya sea en la capa de acceso a datos o en la lógica de negocio, sea transformado en una respuesta HTTP adecuada y coherente. Este enfoque no solo mejora la robustez de la aplicación sino que también proporciona una experiencia de error más controlada y predecible para los consumidores de la API. Además, ayuda en el diseño a que los controladores no tengan que encargarse del manejo de errores, ya que todas las validaciones se realizan en capas inferiores.

4b. Personalización de las respuestas de error en la API

Respuestas de error que implementamos (se pueden ver en los filtros):
401 Unauthorized(autorizacion), 400 Bad Request, 404 not found, 403 Forbidden(autenticación) y 500 Internal server error

5. Postman y Swagger

Se usaron PostMan y Swagger para hacer las pruebas y se incluyo un archivo con muchas requests para probar.

6.Consideraciones Finales y Futuras Mejoras

Somos conscientes de que hay validaciones que no se han realizado exhaustivamente, que procuraremos mejorar en la siguiente entrega.
En determinados errores, no se muestra correctamente el mensaje de error que debería mostrarse. Procuraremos mejorarlo en la siguiente entrega.
A su vez, hay algunos modelos de salida que podrían ser mejores, como el de accept invitation que debería darnos el token nuevo. Así sabemos nuestro token
¿Debíamos implementar un login?
El apartamento no tiene número de rooms
Podríamos tratar construction Company como algo más en vez de string
Podríamos haber agregado un polimorfismo entre los usuarios
Manager no tiene nombre
En nuestro sistema un owner no puede tener varios apartamentos.
El eager loading no se aplica bien, cuando se recupera un edificio no se recuperan automáticamente sus apartamentos por alguna razón