Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Ingeniería en Sistemas

Arquitectura de Software

Obligatorio 1

Link al Repositorio

https://github.com/IngSoft-AR-2023-2/256345_267919_241195

Matías Olivera (267919)

Mateo Giraz (241195)

Santiago Villar (256345)

Grupo M7A

Docentes: Gastón Mousqués, Tomas Piaggio, Damian Moretti

Índice

1. Introducción

El presente documento de Descripción de Arquitectura (DA) tiene como objetivo proporcionar una especificación completa de la arquitectura del sistema Inmo 2.0. Este documento está estructurado para ofrecer una visión detallada de los diversos componentes del sistema, sus interacciones y las decisiones de diseño que se han tomado para garantizar su funcionalidad, mantenibilidad y escalabilidad.

La estructura del documento se organiza en varias secciones, cada una de las cuales aborda un aspecto específico de la arquitectura del sistema. Estas secciones incluyen una descripción general del propósito del sistema, los antecedentes y requisitos significativos de arquitectura, así como las distintas vistas de módulos, componentes y conectores, y asignación. Este enfoque estructurado permite a los usuarios del documento obtener una comprensión clara y detallada de cómo está construido el sistema y cómo se espera que funcione.

Propósito

El propósito del presente documento es proveer una especificación completa de la arquitectura del sistema Inmo 2.0, detallando los componentes y sus interacciones, las decisiones de diseño adoptadas, y cómo estas contribuyen a cumplir con los requisitos funcionales y no funcionales del sistema.

2. Antecedentes

Nota: Los ADR realizados para este obligatorio se encuentran en nuestro reposiotrio de Github bajo la carpeta **docs/architecture**

Propósito del sistema

El propósito del sistema es ofrecer una plataforma integral de gestión de inmuebles, permitiendo a distintos usuarios interactuar con los recursos y servicios que ofrece. Este sistema está diseñado para facilitar diversas funciones relacionadas con la administración, búsqueda, reserva y mantenimiento de propiedades. Los usuarios principales del sistema incluyen administradores, operarios de inmuebles, inquilinos y propietarios, cada uno con roles y responsabilidades específicas.

Usuarios del sistema y sus funciones

- **Administrador**: Tiene control total sobre el sistema, incluyendo la gestión de inmuebles, sensores y transacciones. Los administradores pueden agregar, modificar o eliminar propiedades y gestionar a otros usuarios del sistema.
- **Operario de Inmuebles**: Encargado del mantenimiento y resolución de incidencias en las propiedades. Este rol incluye tareas como la verificación del estado de los inmuebles y la atención de reportes de mantenimiento.
- **Inquilino**: Puede buscar, reservar y pagar alquileres de propiedades. Los inquilinos tienen acceso a una interfaz que les permite gestionar sus reservas y ver detalles de las propiedades disponibles.
- **Propietario**: Puede registrar nuevas propiedades y gestionar los periodos en los que estarán disponibles para alquilar. Los propietarios pueden ver el estado de sus propiedades y las reservas realizadas.

Funcionalidades principales del sistema

Gestión de Propiedades

Permite a los propietarios registrar, editar y eliminar propiedades.

- **Alta de propiedades**: Permite a los propietarios registrar nuevas propiedades en el sistema.
- **Edición y eliminación**: Los administradores pueden modificar o eliminar propiedades existentes.
- **Gestión de imágenes y documentos**: Soporte para subir imágenes de las propiedades y documentos relacionados, como contratos o reportes.
- **Actor**: Propietario

Búsqueda de Propiedades

Permite a los inquilinos buscar propiedades utilizando diversos filtros.

- **Búsqueda y filtrado**: Inquilinos pueden buscar propiedades basándose en diversos filtros como ubicación, precio y disponibilidad.
- **Actor**: Inquilino

Reservas y Pagos

Permite a los inquilinos reservar propiedades y realizar pagos en línea, así como cancelarlas y visualizar otras reservas.

- **Realización de reservas**: Funcionalidad para que los inquilinos reserven propiedades y realicen pagos en línea.
- **Gestión de disponibilidad**: Herramientas para que los propietarios y administradores gestionen la disponibilidad de las propiedades.
- **Actor**: Inquilino

Notificaciones y Alertas

Notifica a los usuarios sobre el estado de sus reservas, pagos y reportes de mantenimiento, así como enviar alertas relacionadas a los sensores.

- **Sistema de notificaciones**: Los usuarios reciben notificaciones sobre el estado de sus reservas, pagos y reportes de mantenimiento.
- **Alertas de disponibilidad**: Los propietarios reciben alertas sobre nuevas reservas y cambios en la disponibilidad de sus propiedades.
- **Actor**: Todos los usuarios

Gestión de Disponibilidad

Permite a los propietarios y administradores gestionar la disponibilidad de las propiedades.

- **Gestión de disponibilidad**: Permite ajustar y gestionar la disponibilidad de las propiedades en el sistema.
- **Actor**: Propietario, Administrador

Monitoreo de Sensores

Permite la integración y monitoreo de sensores para supervisar el estado de las propiedades y alertar problemas con las mismas.

- **Integración de sensores**: Facilita la integración de diversos sensores para monitorear las propiedades.
- **Alertas de sensores**: Genera alertas en caso de detección de problemas o condiciones anómalas.
- **Actor**: Administrador, Propietario

Generación de Reportes

Permite la generación de reportes detallados sobre las reservas, pagos y estado de las propiedades.

- **Generación de reportes**: Los administradores y propietarios pueden generar reportes detallados para analizar la información de reservas, pagos y estado de las propiedades.
- **Actor**: Administrador, Operario de Inmuebles

Autenticación y Autorización

Permite a los usuarios autenticarse en el sistema y gestionar permisos basados en roles.

- **Autenticación de usuarios**: Los usuarios pueden autenticarse en el sistema utilizando métodos seguros.
- **Gestión de permisos**: Administración de permisos y roles para controlar el acceso a diferentes partes del sistema.
- **Actor**: Todos los usuarios

Requerimientos significativos de Arquitectura

Resumen de Requerimientos Funcionales

ID Requerimiento	Descripción	Actor
RF1 Gestión de Propiedades	*Permite a los propietarios registrar, editar y eliminar propiedades.*	*Propietario*
RF 2 Búsqueda de Propiedades	*Permite a los inquilinos buscar propiedades utilizando diversos filtros.*	*Inquilino*
RF3 Reservas y Pagos	*Permite a los inquilinos reservar propiedades y realizar pagos en línea, así como cancelarlas y visualizar otras reservas.*	*Inquilino*
RF4 Notificaciones y Alertas	*Notifica a los usuarios sobre el estado de sus reservas, pagos y reportes de mantenimiento, así como enviar alertas relacionadas a los sensores.*	*Todos los usuarios*
RF5 Gestión de Disponibilidad	*Permite a los propietarios y administradores gestionar la disponibilidad de las propiedades.*	*Propietario, Administrador*
RF6 Monitoreo de Sensores	*Permite la integración y monitoreo de sensores para supervisar el estado de las propiedades y alertar problemas con las mismas.*	*Administrador, Propietario*

RF7 Generación de Reportes	*Permite la generación de reportes detallados sobre las reservas, pagos y estado de las propiedades.*	*Administrador, Propietario*
RF8 Autenticación y Autorización	*Permite a los usuarios autenticarse en el sistema y gestionar permisos basados en roles.*	*Todos los usuarios*

Resumen de Requerimientos no funcionales

ID Requerimiento No Funcional	ID Requerimiento Funcional de Referencia	ID Requerimiento de Calidad o restricción	Descripción
RNF1	*RF1*	*Confiabilidad*	*El sistema debe asegurar que los datos de las propiedades no se pierdan durante el procesamiento y sean consistentes.*
RNF2	*RF1*	*Mantenibilidad*	*El sistema debe permitir la adición de propiedades con distintas características con el mínimo impacto.*
RNF3	*RF2*	*Rendimiento*	*Las búsquedas de propiedades deben retornar resultados en menos de 1 segundo.*
RNF4	*RF2*	*Rendimiento*	*Al ser tantas las propiedades hará falta paginar para poder devolver una rápida respuesta al usuario*
RNF5	*RF2*	*Disponibilidad*	*El servicio de búsqueda debe estar disponible la mayor cantidad de tiempo posible para que los usuarios puedan navegar por los inmuebles.*
RNF6	*RF3*	*Disponibilidad*	*El sistema de reservas y pagos debe estar siempre levantado, si los usuarios no pueden hacer reservas o pagarlas cuando quieran perdemos negocio.*
RNF7	*RF3*	*Mantenibilidad*	*El sistema debe ser capaz de responder al usuario independientemente del funcionamiento del sistema de pago externo.*

RNF8	*RF3*	*Performance*	*El sistema debe ser capaz de responder rápidamente a las miles de reservas que se realizan por minuto y con poca latencia.*
RNF9	*RF3*	*Confiabilidad*	*El sistema debe asegurar que los datos de las reservas no se pierdan durante el procesamiento y sean consistentes.*
RNF10	*RF3*	*Escalabilidad*	*El sistema debe ser capaz de enviar miles de mensajes por minuto para comunicar las diferentes acciones de reservas a los usuarios*
RNF11	*RF4*	*Escalabilidad*	*El sistema de mensajería debe poder aumentar su capacidad para atender a la cantidad de eventos que transcurren.*
RNF12	*RF4*	*Modificabilidad y Compatibilidad*	*El sistema debe poder fácilmente pasar a usar distintos proveedores de mensajería como Slack, WhatsApp, etc.*
RNF13	*RF4*	*Performance*	*Es crucial que se notifique a los usuarios en el momento que hay un incidente, sobre todo si se trata de problemas importantes con los inmuebles.*
RNF14	*RF5*	*Confiabilidad*	*Los ajustes de disponibilidad deben cambiarse correctamente.*
RNF15	*RF6*	*Seguridad*	*Los accesos a los datos de los sensores y reservas deben estar restringidos a usuarios autorizados.*
RNF16	*RF6*	*Mantenibilidad*	*El sistema debe permitir la adición y eliminación de sensores y módulos de notificación con el mínimo impacto.*

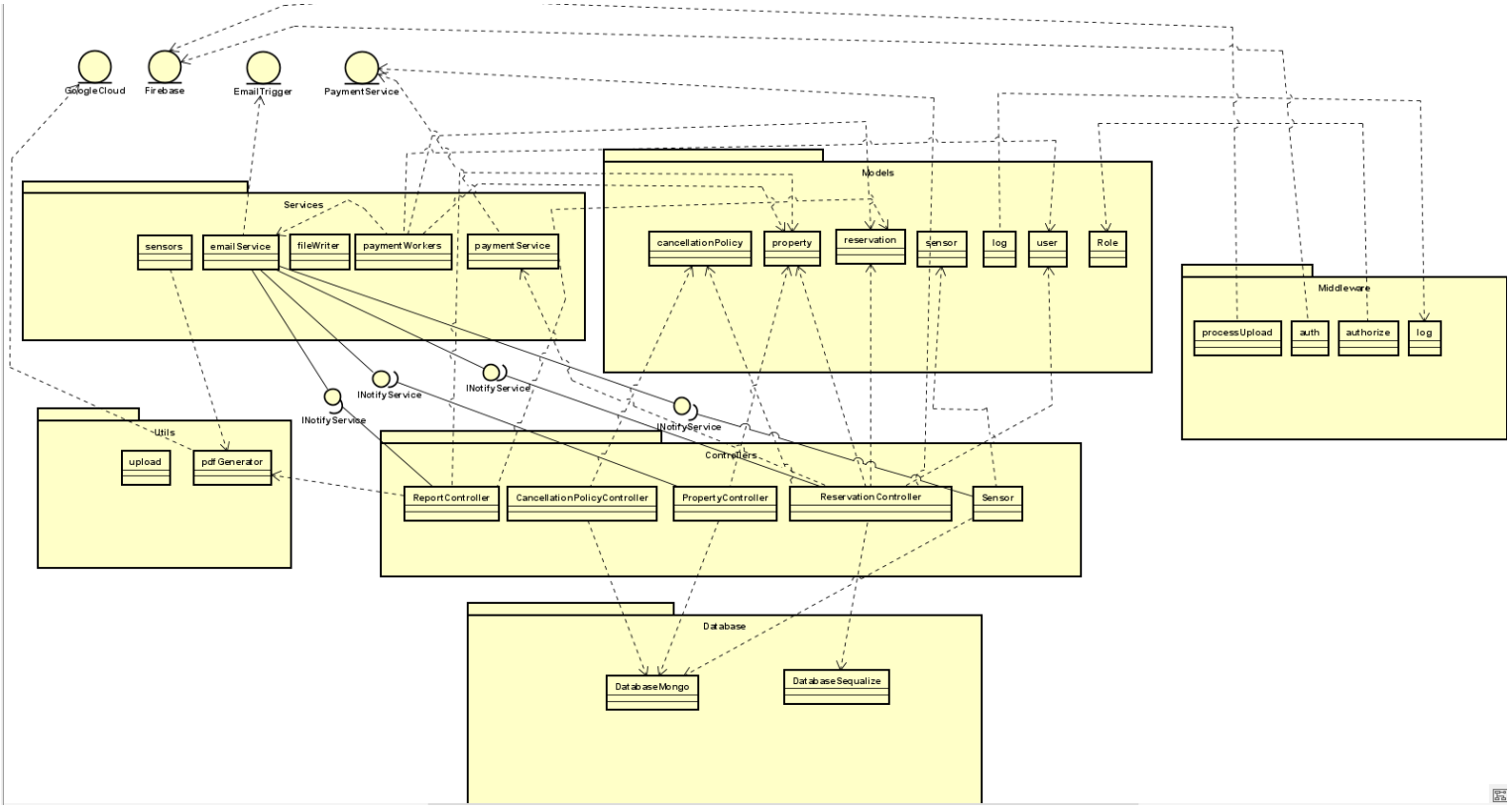
RNF17	*RF6*	*Escalabilidad*	*Se debe poder escalar para manejar incrementos en el tráfico de mediciones.*
RNF18	*RF6*	*Disponibilidad*	*El manejo de sensores no puede caerse en ningún momento para evitar que se pierdan mediciones o que se alerte al usuario de un problema cuando ya es muy tarde.*
RNF19	*RF6*	*Performance*	*Se esperan recibir 100 mediciones por segundo y el sistema debe poder manejar todas sin perder los datos.*
RNF20	*RF6*	*Performance*	*Debe ser capaz de priorizar las alertas más importantes de forma de notificar al propietario y/o admin lo antes posible.*
RNF21	*RF7*	*Performance*	*Los reportes deben ser generados rápidamente, tanto el procesamiento de datos como el envío del reporte al operario o administrador.*
RNF22	*RF7*	*Modificabilidad*	*El sistema debe estar abierto a la implementación de distintos métodos de presentación de reportes.*
RNF23	*RF7*	*Mantenibilidad*	*El sistema de reportes debe permitir la adición de nuevos tipos de reportes en el futuro.*
RNF24	*RF7*	*Escalabilidad*	*El sistema de reportes debe permitir la adición de nuevos tipos de reportes en el futuro.*
RNF25	*RF8*	*Seguridad*	*El sistema debe correctamente manejar el rol y las credenciales del usuario*

RNF26	*RF8*	*Performance*	*El LogIn debe realizarse en no menos de 2 segundos para acceder rápidamente a las demás funcionalidades.*

3. Documentación de la arquitectura

Vista de Módulos

Representación primaria



Catálogo de elementos

Componente/conector	**Descripción**
services/sensores	Gestiona las operaciones relacionadas con sensores.
services/emailService	Administra las notificaciones por correo electrónico.

services/fileWriter	Responsable de las operaciones de escritura de archivos.
services/paymentWorkers	Maneja las tareas de procesamiento de pagos.
services/paymentService	Administra las operaciones de pago.
models/cancellationPolicy	Representa las políticas de cancelación.
models/property	Representa las propiedades
models/reservation	Representa las reservas.
models/sensor	Representa los datos de los sensores.
models/log	Gestiona la información de registro..
models/user models/role	Representa los datos de los usuarios. .Representa los roles y permisos de los usuarios.
controller/ReportController	Administra la generación de informes.
controller/CancellationPolicyController	Gestiona las operaciones de políticas de cancelación.
controller/PropertyController	Administra las operaciones relacionadas con propiedades.
controller/ReservationController	Gestiona las operaciones de reservas.s

controller/SensorController	Administra las operaciones relacionadas con sensores.
utils/upload	Gestiona las operaciones de subida de archivos localmente usando redis previo a que sean subidos a firebase storage
utils/pdfGenerator	Genera documentos PDF.
middleware/processUpload	Middleware para gestionar la subida de archivos.
middleware/auth	Middleware de autenticación.
middleware/authorize	Middleware de autorización.
middleware/log	Middleware de registro.
DatabaseMongo	Gestiona las operaciones de la base de datos MongoDB
DatabaseSequelize	Gestiona las operaciones de la base de datos Sequelize
Servicios Externos/Firebase	Gestiona los servicios de Firebase, firebase authentication,firebase storage, firebase email trigger
Servicios Externos/Payment Service	Gestiona los servicios de pago mediante una aplicación externa
Servicios Externos/Firebase	Administra las operaciones de activación de correos electrónicos mediante firebase

Decisiones de diseño

Una decisión clave en el diseño es la arquitectura orientada a servicios. El módulo de Servicios maneja la lógica de negocio específica relacionada con sensores, correos electrónicos, archivos y pagos, desacoplando así la lógica de negocio de otras partes de la aplicación y promoviendo la reutilización y la modularidad. Por ejemplo, en el código proporcionado, los servicios como *emailService*, *fileWriter* y *paymentService* están claramente definidos y encapsulan funciones específicas, facilitando la gestión y actualización de cada uno de estos servicios sin afectar a otras partes del sistema.

Los controladores son responsables de manejar las solicitudes e invocar los servicios apropiados, manteniendo la lógica de negocio encapsulada dentro de la capa de Servicios. Esto asegura una separación clara entre la capa de API y la lógica de negocio. Por ejemplo, *PropertyController* interactúa con servicios y modelos para gestionar las operaciones relacionadas con propiedades, manteniendo la lógica de negocio separada de la lógica de presentación.

El módulo de Utilidades contiene funciones auxiliares como la subida de archivos y la generación de PDFs, centralizando las funcionalidades comunes utilizadas en toda la aplicación. En el diagrama de módulos, se puede ver cómo *upload* y *pdfGenerator* están definidos como utilidades independientes, proporcionando servicios comunes que pueden ser utilizados por otros módulos sin duplicar código.

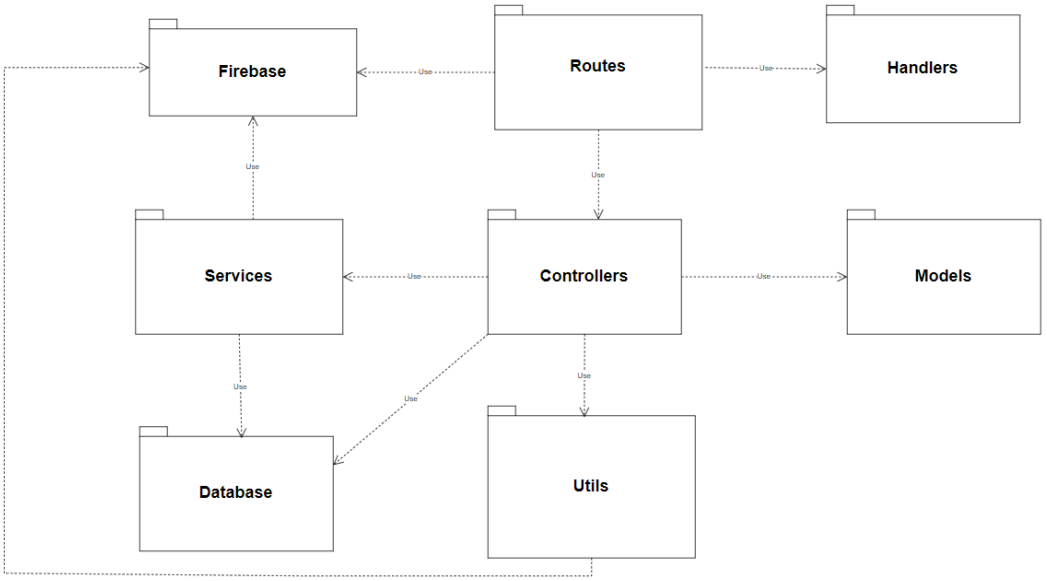
Las funciones de middleware manejan asuntos transversales como la autenticación, autorización, registro y subida de archivos, asegurando que estos asuntos se gestionen de manera consistente en toda la aplicación. Por ejemplo, el middleware auth y authorize garantizan que sólo los usuarios autenticados y autorizados puedan acceder a ciertas funcionalidades del sistema, asegurando la seguridad y consistencia en el manejo de permisos.

La integración con servicios externos como *Google Cloud*, *Firebase*, *EmailTrigger* y *PaymentService* permite que el sistema aproveche funcionalidades de terceros para el almacenamiento en la nube, gestión de bases de datos en tiempo real, notificaciones por correo electrónico y procesamiento de pagos. En el diagrama de módulos, se puede ver cómo estos servicios externos están representados e interactúan con los módulos internos del sistema, proporcionando una integración fluida y aprovechando sus capacidades avanzadas.

Finalmente, la interfaz *INotifyService* estandariza los servicios de notificación, permitiendo una fácil integración de diferentes mecanismos de notificación en el futuro. En el código del servicio de notificaciones, se utiliza *INotifyService* para definir cómo se deben enviar las notificaciones, lo que permite añadir fácilmente nuevos tipos de notificaciones sin modificar la lógica de negocio existente.

Vista de Uso

Representación primaria



Catálogo de elementos

Elemento	Responsabilidades
Firebase	Es el módulo encargado de la comunicación con los servicios de Firebase. Sus responsabilidades incluyen manejar la autenticación de usuarios con Firebase Authentication, gestionar el envío de notificaciones por Email Trigger y almacenar y recuperar datos en Google Cloud. Este módulo asegura que todas las interacciones con los servicios de Firebase sean eficientes y seguras, proporcionando funcionalidades esenciales para la operación del sistema.
Routes	En él se definen los endpoints de la API y redirige las peticiones a los controladores correspondientes. Su responsabilidad principal es asegurarse de que todas las peticiones entrantes sean dirigidas al lugar correcto para su procesamiento, facilitando así la correcta distribución del tráfico y el acceso a las funcionalidades del sistema.

Handlers	Maneja la lógica de negocio en respuesta a eventos específicos y procesar las respuestas, así como manejar los errores.
Services	Gestiona distintos tipos de servicio, entre ellos de implementar la lógica de envío de correos, lo cual lleva a que use tanto a Database como a Firebase
Controllers	Encargado de recibir peticiones desde las rutas y coordinar la lógica de respuesta. Interactúa con los servicios y modelos para procesar la solicitud.
Models	Define las estructuras de datos y las interacciones con la base de datos para operaciones
Database	Maneja la conexión y las operaciones con la base de datos.
Utils	Contiene funciones y utilidades comunes utilizadas en todo el sistema, lo cual simplifica tareas repetitivas y comunes.

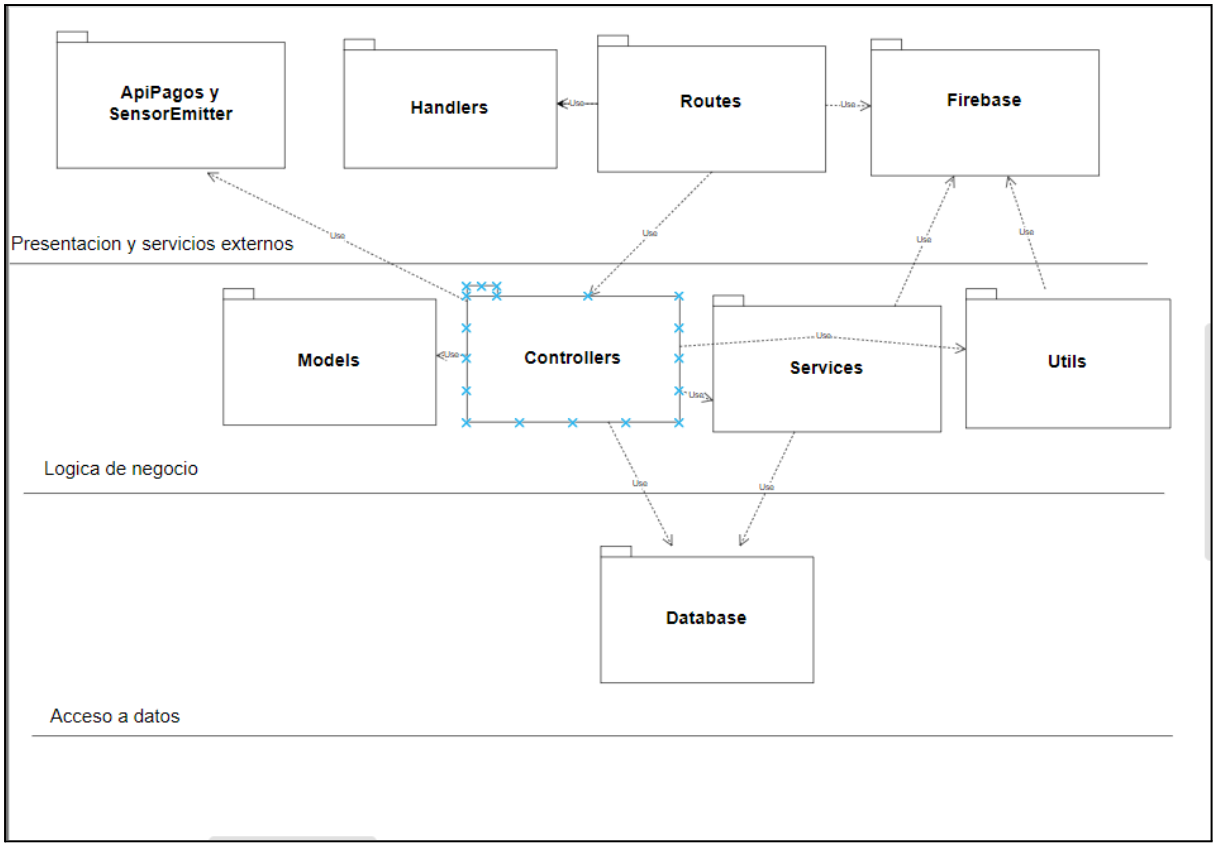
Decisiones de diseño

En el diagrama se puede identificar una de las decisiones de diseño que tomamos, como lo fue el uso de un proveedor externo, en nuestro caso Firebase, para manejar tareas como la autenticación y el envío de notificaciones. Esta decisión fue tomada para favorecer atributos de calidad como la seguridad, la fiabilidad y la performance. Firebase proporciona mecanismos robustos para la autenticación y gestión de datos, lo que asegura una alta seguridad. Además, los servicios gestionados de Firebase son altamente disponibles y escalables, contribuyendo significativamente a la fiabilidad del sistema. Otro argumento a favor también fue que al delegar la autenticación y las notificaciones a Firebase, se reduce la carga en nuestros propios servidores, mejorando así la performance general del sistema .

Otra decisión de diseño importante que se refleja en el diagrama es la gestión centralizada de eventos y errores mediante el módulo Handlers. Manejar eventos y errores de manera uniforme reduce la probabilidad de fallos imprevistos, aumentando así la fiabilidad. Además, una gestión centralizada de errores facilita la localización y corrección de problemas, mejorando la mantenibilidad del sistema.

Vista de Layers

Representación primaria



Catálogo de elementos

Elemento	Responsabilidades
Firebase	Es el módulo encargado de la comunicación con los servicios de Firebase. Sus responsabilidades incluyen manejar la autenticación de usuarios con Firebase Authentication, gestionar el envío de notificaciones por Email Trigger y almacenar y recuperar datos en Google Cloud. Este módulo asegura que todas las interacciones con los servicios de Firebase sean eficientes y seguras, proporcionando funcionalidades esenciales para la operación del sistema.
Routes	En él se definen los endpoints de la API y redirige las peticiones a los controladores correspondientes. Su responsabilidad principal es asegurarse de que todas las peticiones entrantes sean dirigidas al lugar correcto para su procesamiento, facilitando así la correcta distribución del tráfico y el acceso a las funcionalidades del sistema.
Handlers	Maneja la lógica de negocio en respuesta a eventos específicos y procesar las respuestas, así como manejar los errores.
Services	Gestiona distintos tipos de servicio, entre ellos de implementar la lógica de envío de correos, lo cual lleva a que use tanto a Database como a Firebase
Controllers	Encargado de recibir peticiones desde las rutas y coordinar la lógica de respuesta. Interactúa con los servicios y modelos para procesar la solicitud.
Models	Define las estructuras de datos y las interacciones con la base de datos para operaciones
Database	Maneja la conexión y las operaciones con la base de datos.

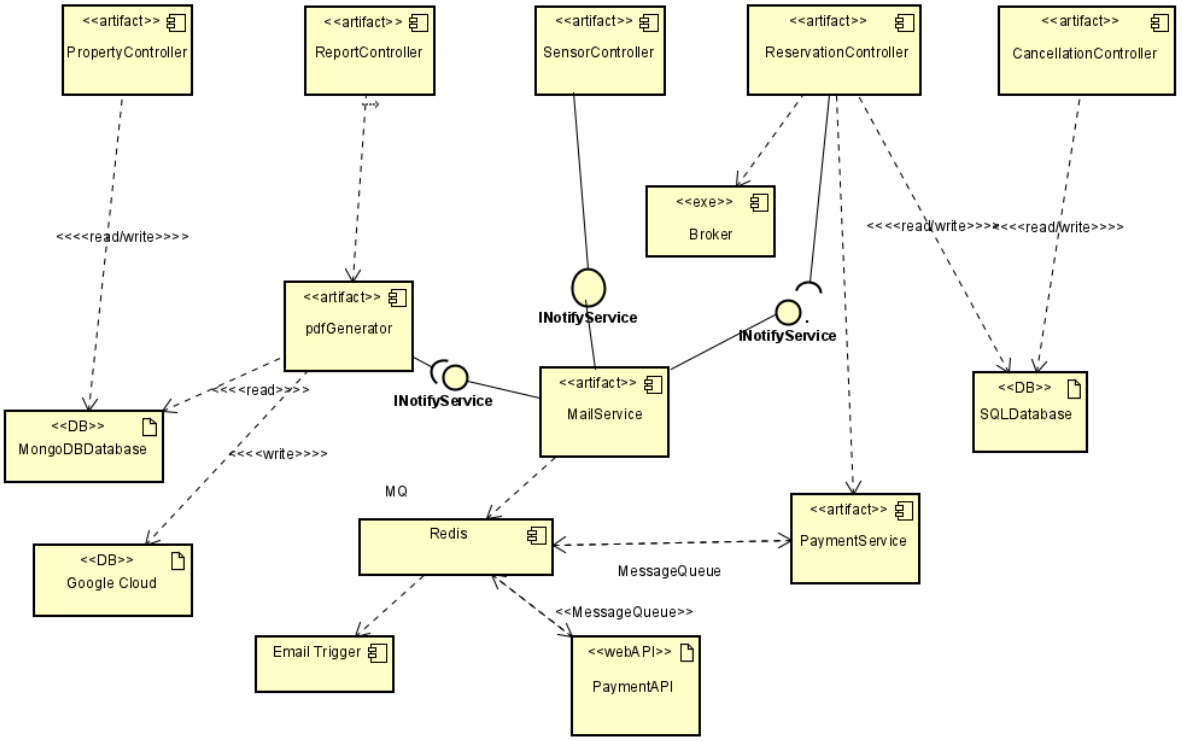
Utils	Contiene funciones y utilidades comunes utilizadas en todo el sistema, lo cual simplifica tareas repetitivas y comunes.
ApiPagos y SensorEmitter	Representan los servicios externos, tanto la API para realizar los pagos correspondientes a las reservas como al servicio para emular los sensores

Decisiones de diseño

Ídem que en la vista de Usos

Vista de Componentes y Conectores

Representación primaria



Catálogo de elementos

Componente/conector	Descripción
PropertyController	Encargado de manejar las solicitudes relacionadas con la gestión de propiedades. Sus responsabilidades incluyen crear, leer, actualizar y eliminar propiedades.
ReportController	Encargado de manejar la generación de reportes. Sus responsabilidades son generar reportes basados en diferentes criterios, lo cual incluye la interacción con el servicio pdfGenerator para crear documentos PDF.
SensorController	Encargado de manejar las solicitudes relacionadas con los sensores. Sus responsabilidades incluyen gestionar los datos de sensores y interactuar con servicios de notificación en caso de eventos específicos.
ReservationController	Encargado de manejar las solicitudes relacionadas con las reservas. Sus responsabilidades incluyen crear, pagar y leer reservas.
CancellationController	Es el controlador encargado de manejar las cancelaciones de reservas. Sus responsabilidades son procesar solicitudes de cancelación de reservas, actualizar la base de datos.
NotifyService	Gestiona las notificaciones del sistema. Sus responsabilidades incluyen enviar notificaciones a través de diferentes canales, como correo electrónico y mensajes. NotifyService interactúa con Redis para gestionar la cola de mensajes, asegurando que todas las notificaciones se envíen de manera eficiente.
MailService	Es el servicio encargado de enviar correos electrónicos. Sus responsabilidades incluyen enviar correos electrónicos basados en eventos

	específicos, interactuando con Email Trigger y Redis para la gestión de mensajes.
PaymentService	Sus responsabilidades son procesar transacciones de pago e interactuar con PaymentAPI y Redis para la gestión de pagos.
pdfGenerator	Crea documentos PDF basados en datos de la BD y usa INotifyService para enviar los PDF a los usuarios. Esto permite generar documentos bien formateados y listos para ser entregados a los usuarios o archivados en el sistema.
MongoDBDatabase	Almacena datos de los inmuebles y sensores. Los mismos cuentan con índices para mejorar la performance
SQLDatabase	Almacena información de las reservas.
Google Cloud Database	Base de Datos en la nube, almacena los reportes.
Redis	Gestiona la comunicación asíncrona entre servicios y mejorar la performance del sistema al desacoplar las operaciones intensivas en tiempo. Tambien el mismo es usado para un cache sobre consultars recientes
Email Trigger	Es el servicio externo de Firebase que usamos para enviar correos
PaymentAPI	Es la API externa para realizar pagos
Broker	Proceso que realiza servicio de Publish-Subscribe para las reservas.

Interfaces

Interfaz:	**INotifyService**
Componente que la provee:	**Service**
Servicio	***Descripción***
INotifyService	*Define los métodos necesarios para enviar notificaciones e implementar la lógica de envío de notificaciones, proporcionando una estructura clara y reutilizable para la gestión de notificaciones en el sistema.*

Decisiones de diseño

En el diagrama se puede identificar una de las tantas decisiones de diseño que tomamos, como lo fue el uso de Firebase para manejar tareas específicas como la autenticación y el envío de notificaciones. Esta decisión fue tomada para favorecer atributos de calidad como la seguridad, la confiabilidad y la performance. Firebase proporciona mecanismos robustos para la autenticación y gestión de datos, lo que asegura una alta seguridad. Además, los servicios gestionados de Firebase son altamente disponibles y escalables, contribuyendo significativamente a la fiabilidad del sistema. Al delegar la autenticación y las notificaciones a Firebase, se reduce la carga en nuestros propios servidores, mejorando así la performance general del sistema.

Otra decisión de diseño importante que se refleja en el diagrama es el uso de interfaces para desacoplar componentes y mantener un mejor código mediante los módulos *NotifyService* e *INotifyService*, mejorando la fiabilidad y mantenibilidad del sistema, ya que manejar eventos y errores de manera uniforme reduce la probabilidad de fallos imprevistos. Además, el uso de interfaces facilita la localización y corrección de problemas, mejorando así la mantenibilidad del sistema, permitiendo que las notificaciones sean manejadas de manera eficiente y asegurando que todos los eventos importantes sean procesados y notificados adecuadamente.

El uso de Redis como sistema de mensajería para gestionar la comunicación asíncrona entre servicios es otra decisión de diseño clave, puesto que favorece la performance del sistema al desacoplar las operaciones intensivas en tiempo, permitiendo que las tareas críticas se procesen en segundo plano sin afectar la respuesta inmediata a los usuarios. Redis actúa como una cola de mensajes, asegurando que todas las notificaciones y eventos sean gestionados de manera eficiente y oportuna, mejorando así la experiencia del usuario final. Justamente de esta forma se envían los pagos a la API de pagos, pues se encolan y van procesando, para luego ser encolados otra vez y consumidos por nuestra aplicación.

También vemos el uso de colas de mensajes para la recepción de señales de los sensores, en este caso se encolan en una cola de prioridad, pues las diferentes mediciones pueden causar alertas con diferente gravedad.

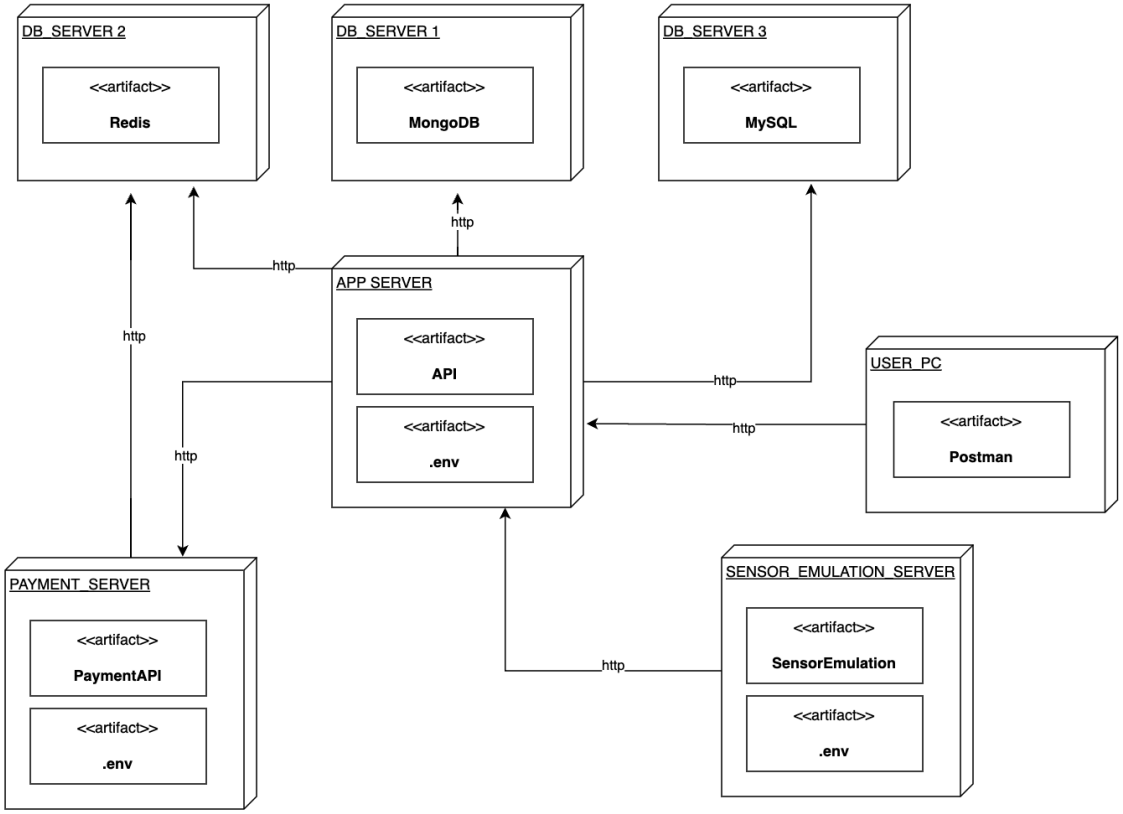
También a través de Redis hacemos uso de un caché sobre consultas de buscar propiedad puesto que estas son bastantes demandantes a nivel de performance, también se actualiza el mismo cuando hace falta de manera de mantener las consultas actualizadas y no desactualizadas por causa de un mal uso del mismo.

La utilización de distribuir los datos en distintas bases de datos permite manejar grandes volúmenes de datos de manera eficiente, favoreciendo la escalabilidad y performance del sistema. MongoDB se utiliza para datos no estructurados, proporcionando flexibilidad en el almacenamiento de datos así mismo también usamos índices en este para hacer que las búsquedas sean más eficientes, mientras que SQL maneja transacciones y consultas complejas, asegurando la integridad y consistencia de los datos. Google Cloud ofrece almacenamiento escalable y altamente disponible, permitiendo que el sistema maneje cargas de trabajo variables y grandes volúmenes de datos sin comprometer la performance. Además, al utilizar diferentes tipos de bases de datos para diferentes necesidades, se distribuye la carga de trabajo, mejorando aún más la performance general del sistema.

Vistas de Asignación

Vista de Despliegue

Representación primaria



Catálogo de elementos

Nodo	**Descripción**
USER PC (Postman)	*Es el encargado de enviar las solicitudes a la aplicación de Inmo 2.0*
API SERVER	*Es el servidor donde se ejecuta la aplicación de Inmo 2.0*
DB SERVER 1	*Es el servidor donde se aloja nuestra base de datos MongoDB*
DB SERVER 2	*Es el servidor donde se aloja nuestra instancia de Redis*
DB SERVER 3	*Es el servidor donde se aloja nuestra base de datos MySQL*
PAYMENT SERVER	*Es el servidor donde se ejecuta la API de pagos*
SENSOR EMULATION SERVER	*Es el servidor donde se ejecuta el script que simula las señales de sensores*