**Software Architecture Documentation (SAD)**

**Equipo: πpo**

**Sergio Cárdenas**

**Juan Sebastián Díaz**

**Sergio Guzmán**

**Julián Manrique**

**Carlos Mario Sarmiento**

**Arquitectura y Diseño de Software**

**2018-10**

**Universidad de los Andes**

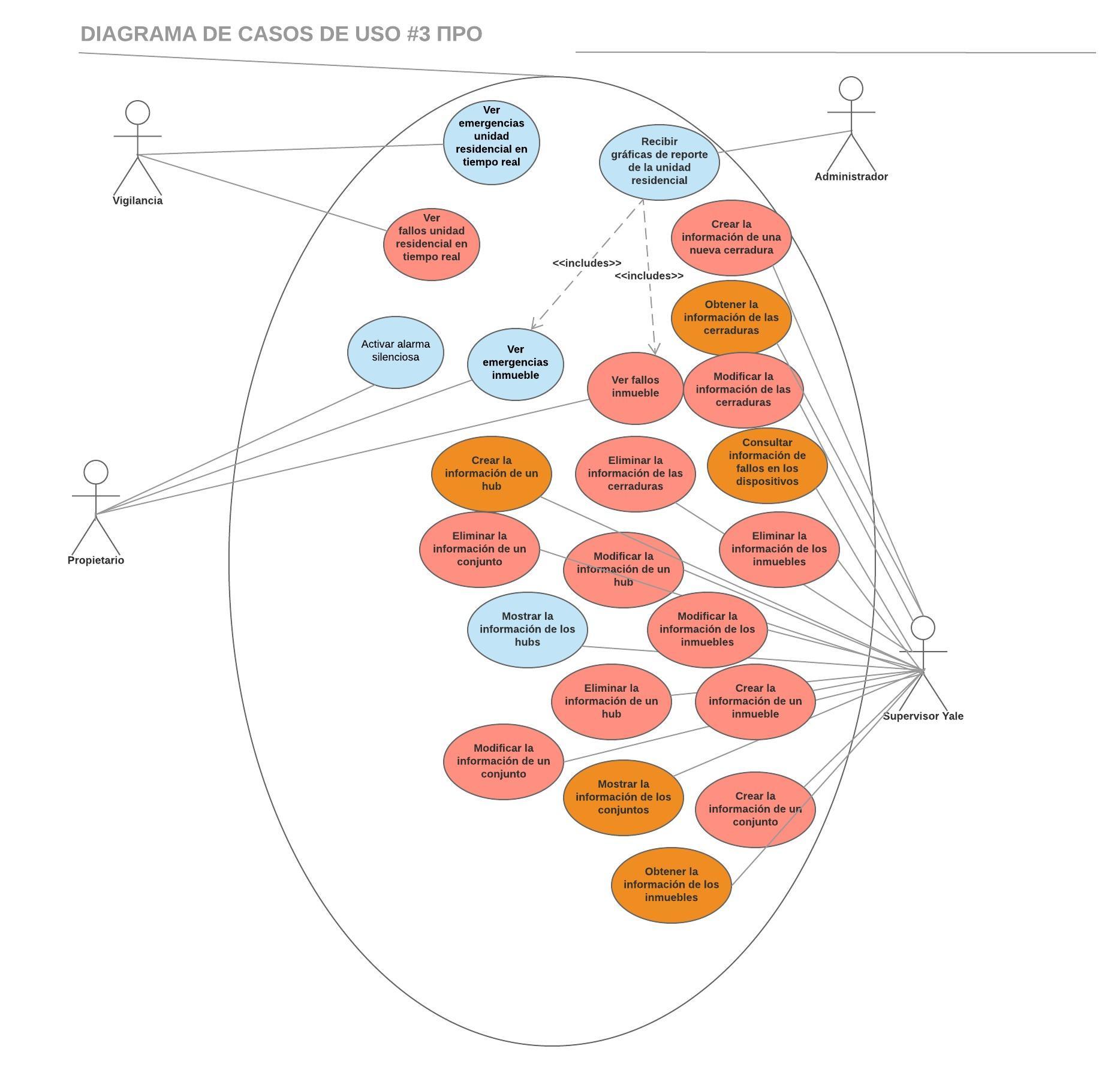
1. **Restricciones**

Según el documento de descripción del sistema, solo se tienen restricciones de tecnología en la comunicación, siendo estas las siguientes:

* El Hub y la cerradura se comunican a través de la red inalámbrica interna del inmueble.
* El Hub se comunica con el sistema central de Yale a través de la conexión de internet de un proveedor externo.
* El sistema de Yale se comunica con el sistema de vigilancia a través de un canal de Internet dedicado.

1. **Casos de Uso**

De acuerdo con el documento de especificaciones dado e instrucciones posteriores, se determinó que el sistema tenía cuatro actores principales: el propietario de un inmueble, el administrador de una unidad residencial, un operario de vigilancia de la misma, y el administrador de Yale. Los casos de uso de estos son los siguientes:



1. **Escenarios de Calidad**

Los escenarios de calidad encontrados son los siguientes:

|  |  |  |
| --- | --- | --- |
| Identificador | Tipo | Prioridad |
| EC1 | Escalabilidad | Alta |
| Fuente | | |
| Cerraduras | | |
| Estímulo | | |
| Notificador de emergencias y fallos. | | |
| Ambiente | | |
| Sobrecargado | | |
| Medida Esperada | | |
| El sistema debe poder recibir 300.000 peticiones en una ventana de 1 minuto con 0% de error y tiempo de respuesta de máximo 1 segundo. | | |

Según el documento, “el sistema debe estar disponible 24 horas, los 7 días a la semana” dado que se trata de un sistema de seguridad, determinamos que esto es equivalente a una disponibilidad de sigma 6.

|  |  |  |
| --- | --- | --- |
| Identificador | Tipo | Prioridad |
| DI1 | Disponibilidad | Alta |
| Fuente | | |
| Sistema | | |
| Estímulo | | |
| Notificador de emergencias y fallos. | | |
| Ambiente | | |
| Normal | | |
| Medida Esperada | | |
| El sistema debe tener una disponibilidad de sigma 6, o 99.9999%. | | |

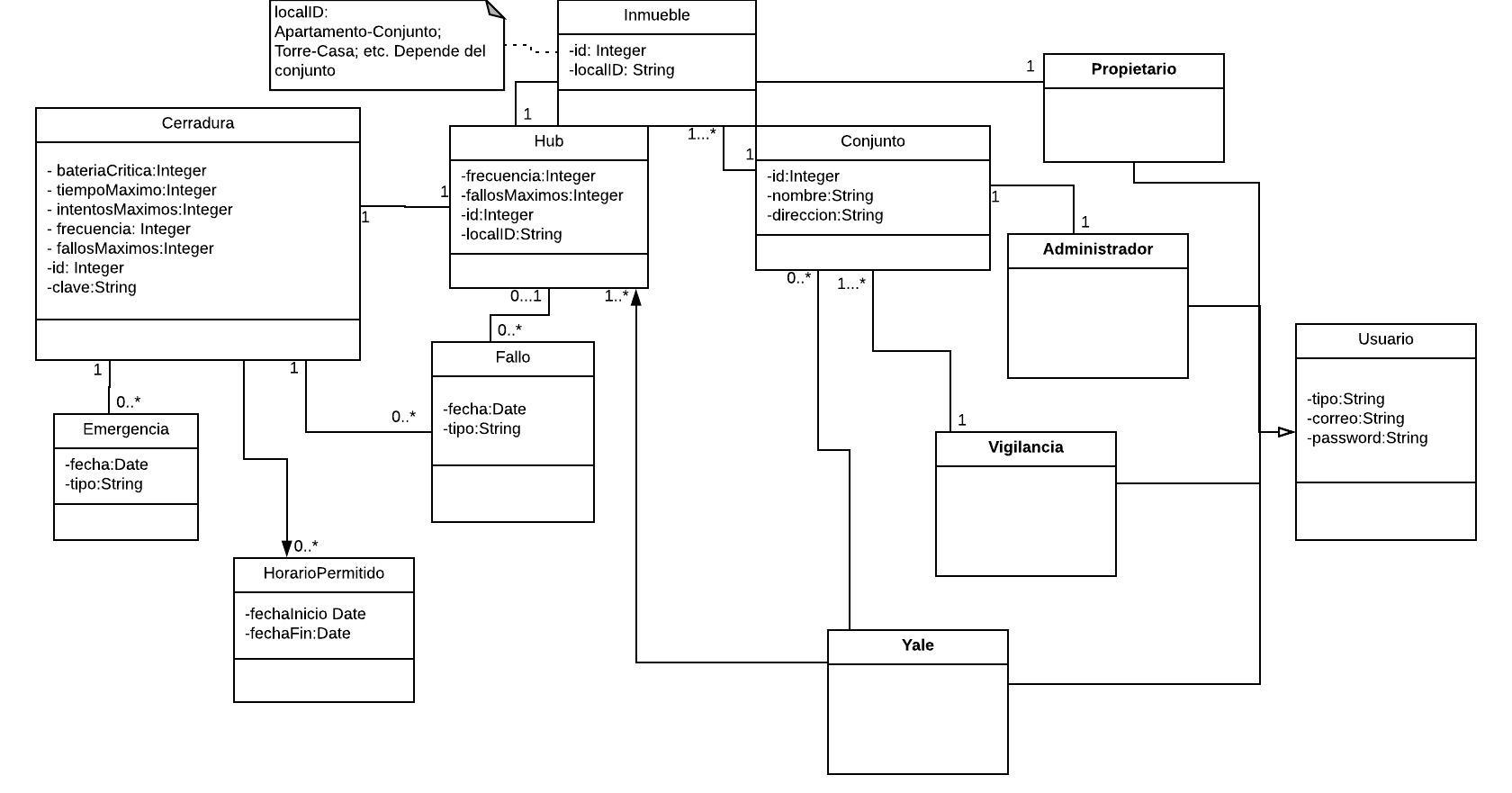
|  |  |  |
| --- | --- | --- |
| Identificador | Tipo | Prioridad |
| SE1 | Seguridad | Alta |
| Fuente | | |
| Todos los actores | | |
| Estímulo | | |
| Ingreso al portal Web | | |
| Ambiente | | |
| Normal | | |
| Medida Esperada | | |
| El portal Web debe tener autenticación y autorización dependiendo de los 4 roles (propietario, administrador, seguridad, Yale). | | |

|  |  |  |
| --- | --- | --- |
| Identificador | Tipo | Prioridad |
| SE2 | Seguridad | Media |
| Fuente | | |
| Cerraduras | | |
| Estímulo | | |
| Envío de alarmas al sistema | | |
| Ambiente | | |
| Normal | | |
| Medida Esperada | | |
| El sistema central de Yale solo debe recibir notificaciones y alarmas de dispositivos registrados y asegurados. | | |

|  |  |  |
| --- | --- | --- |
| Identificador | Tipo | Prioridad |
| US1 | Usabilidad | Alta |
| Fuente | | |
| Seguridad Privada | | |
| Estímulo | | |
| Tablero de Control | | |
| Ambiente | | |
| Normal | | |
| Medida Esperada | | |
| La seguridad privada debe tener un tablero de control tipo dashboard en el cual se vean claramente los estados de los dispositivos del inmueble, así como sus alarmas. | | |

1. **Modelo Conceptual**

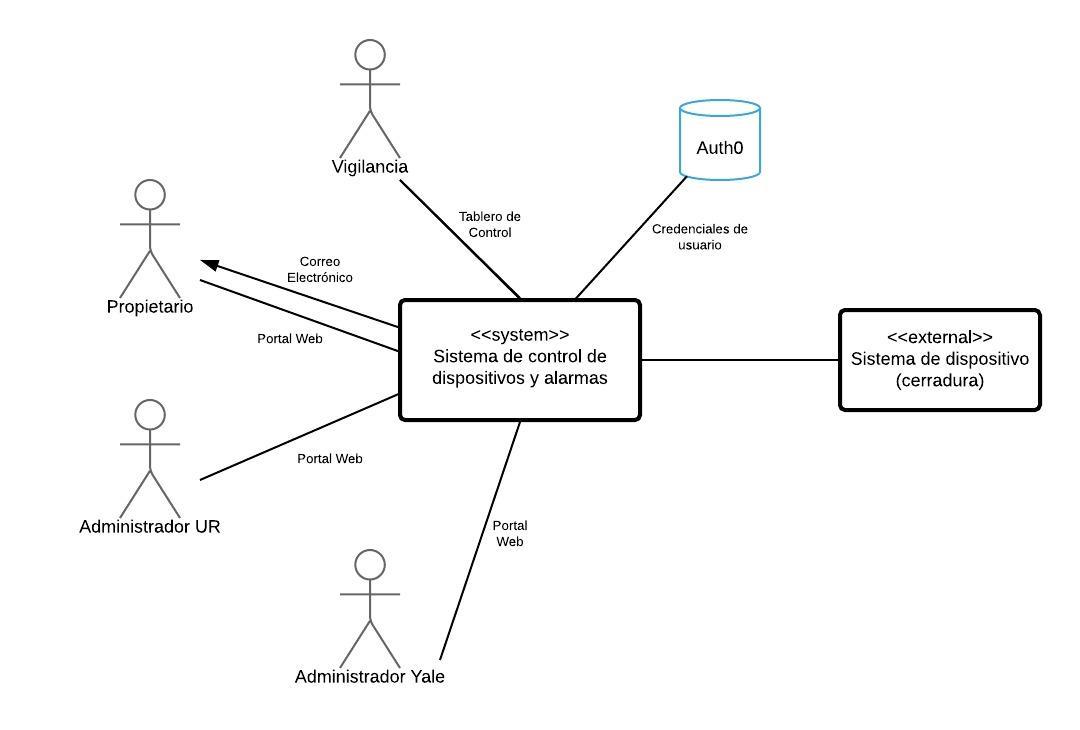
Analizando todo lo anterior, se determinó usar el siguiente modelo conceptual:



Mostrado como diagrama de clases, ya que es más claro así que a partir de un modelo E/R. Estas no son las únicas clases del proyecto, únicamente aquellas que corresponden a los objetos persistidos en Base de Datos.

1. **Vista de Contexto**

La vista de contexto del sistema construido es la siguiente:

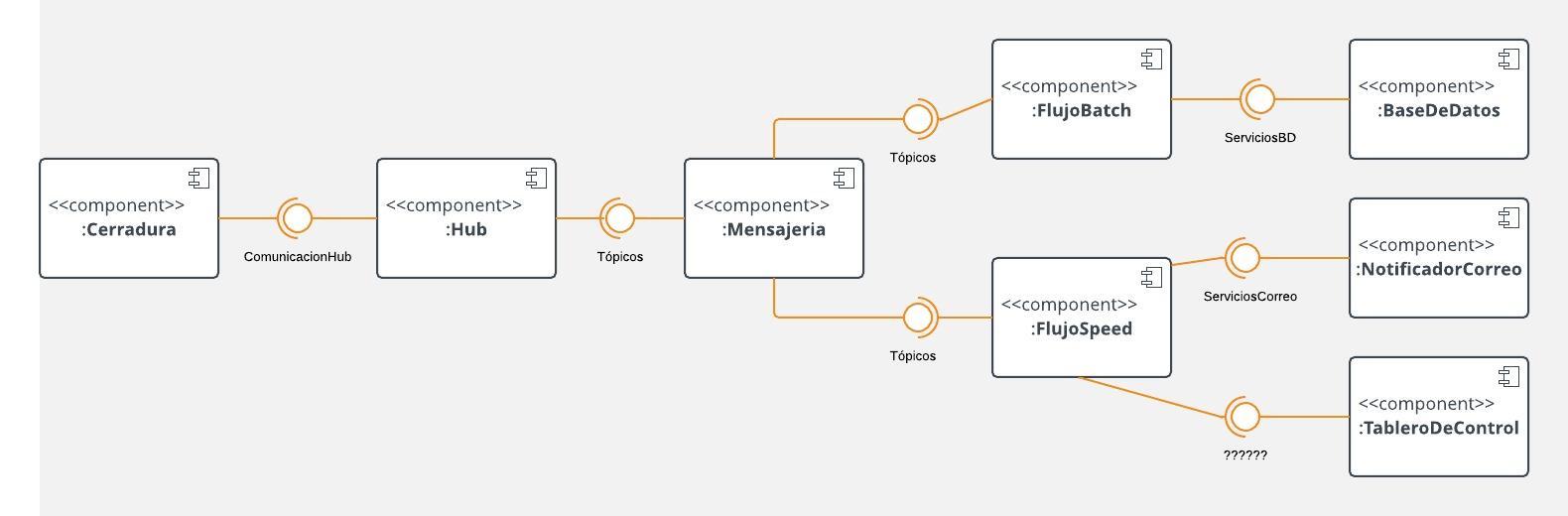


Se decidió una base de datos externa para almacenar credenciales ya que se consideraba más segura para propósitos del proyecto. También vale la pena destacar que el sistema envía notificaciones de correo al propietario de un inmueble, y este además puede acceder al sistema por medio del portal web (como todos los demás actores).

Por otro lado, aunque para esta versión se desarrolló el sistema de control de la cerradura, en la aplicación real se espera que esta tenga un sistema externo, aparte al construido.

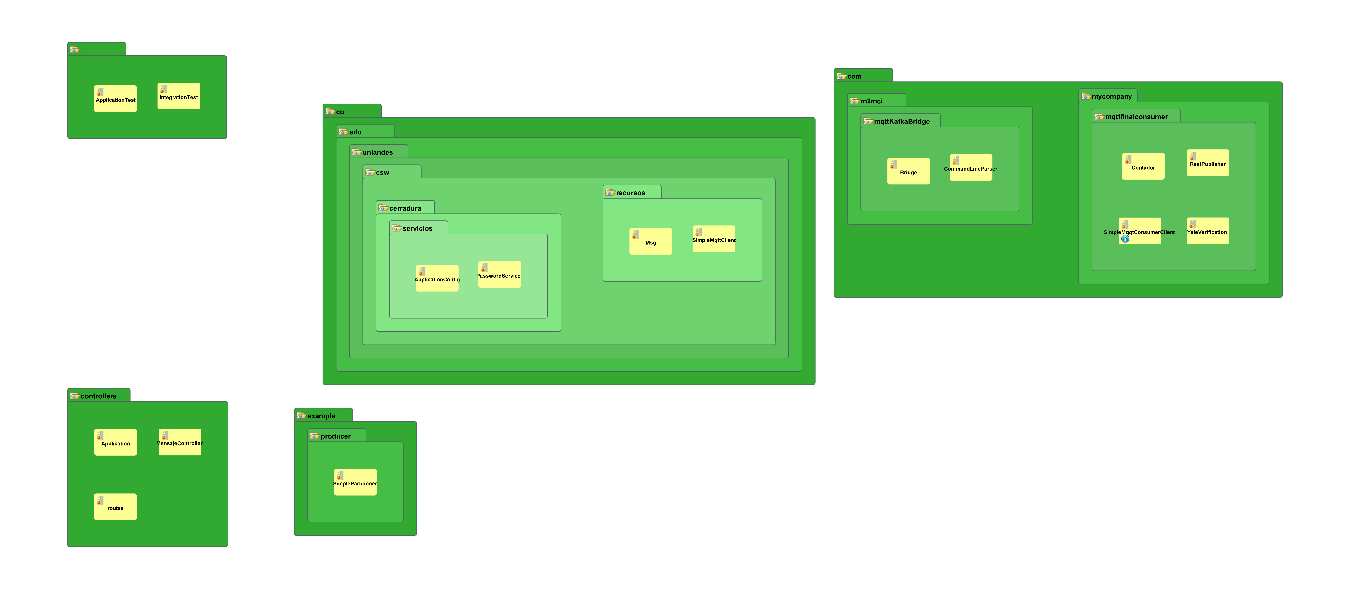
1. **Vista de Componentes**

Desde la vista de componentes, el sistema se ve de la siguiente manera:



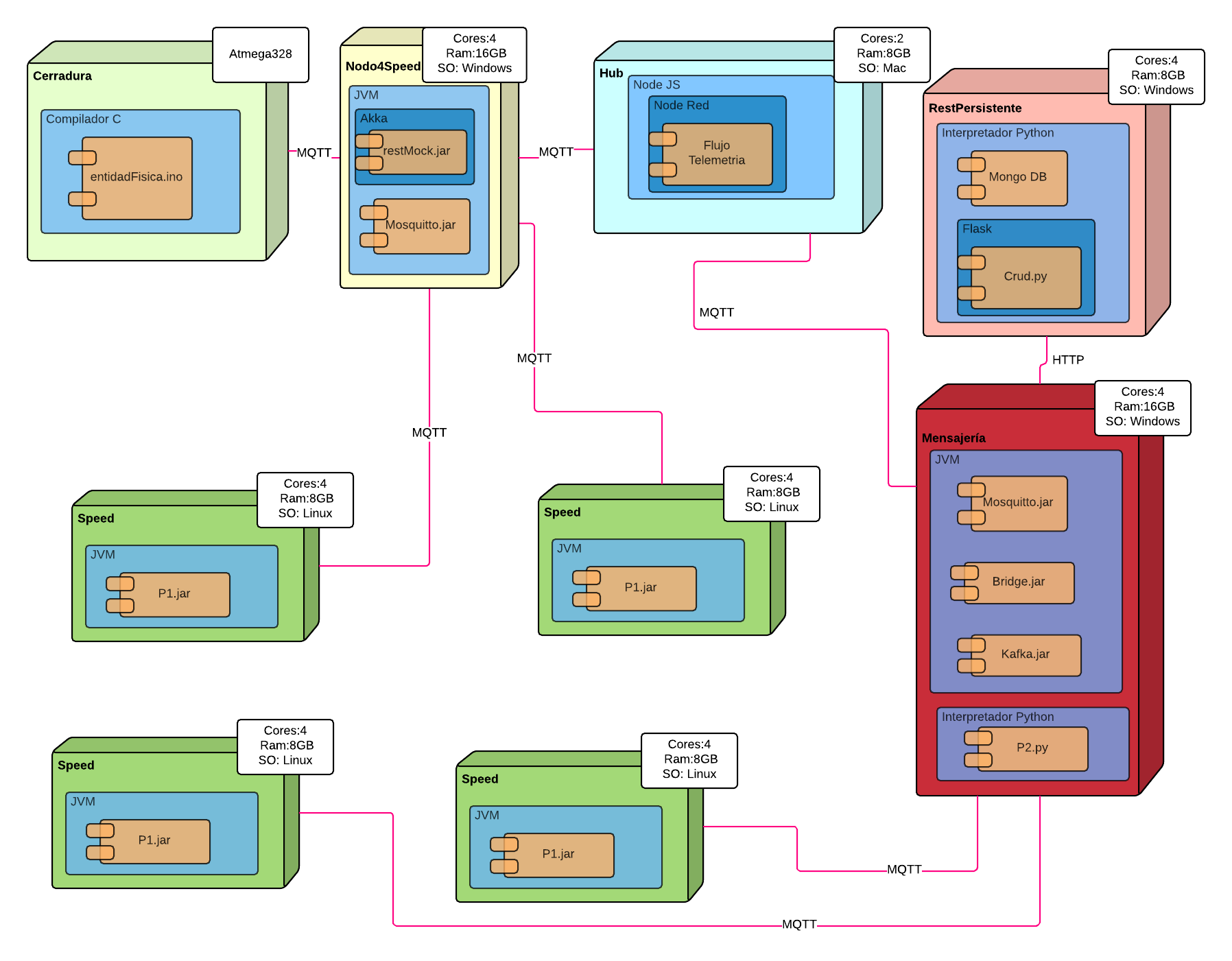
1. **Vista de Desarrollo**

La vista de desarrollo del sistema (o diagrama de paquetes) es:



1. **Vista de Despliegue**

A continuación, se presenta el diagrama de despliegue de nuestra arquitectura en donde se omite parte de la capa *Speed*:

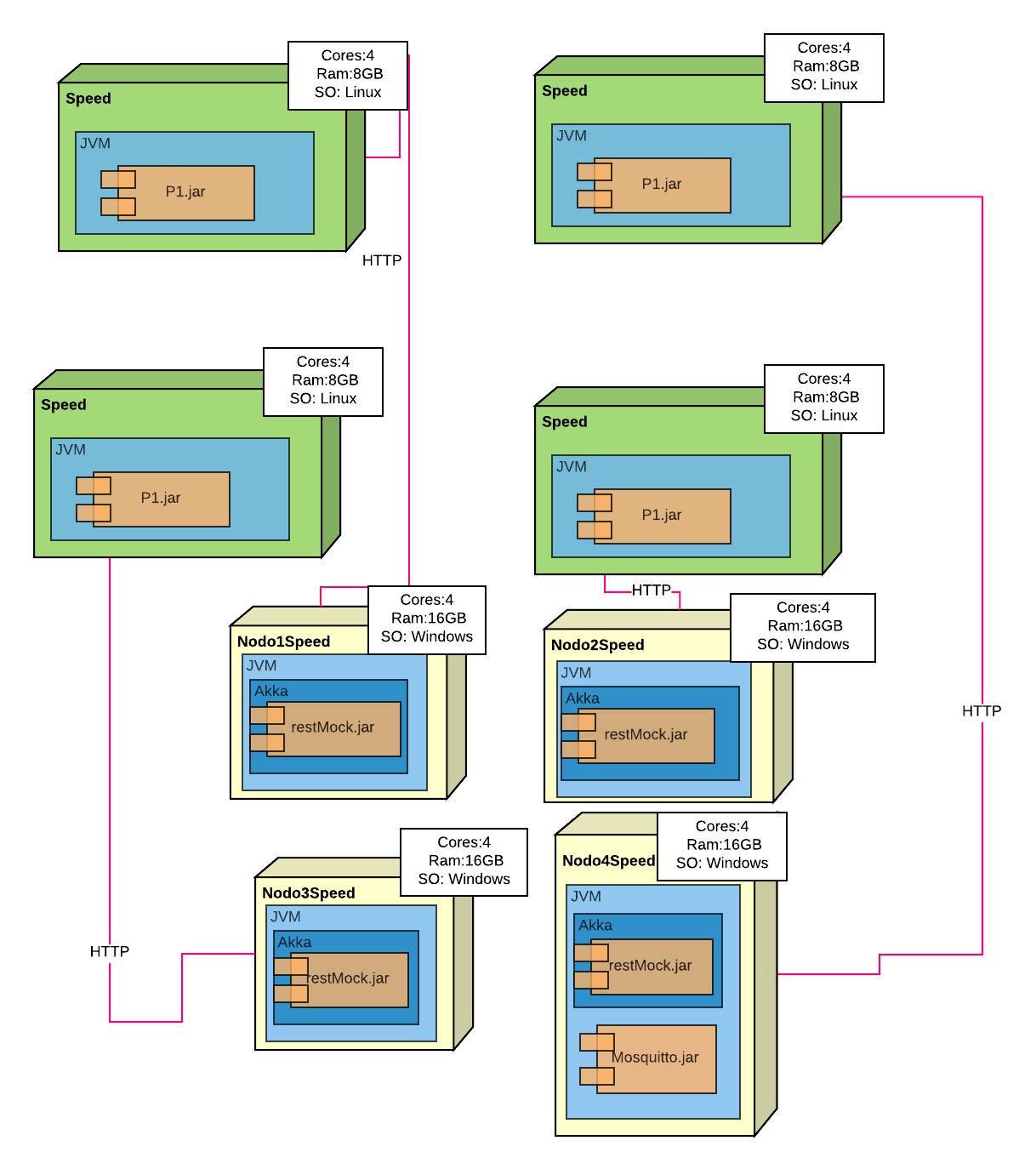


Se decidió usar un servidor de Mosquitto para poder establecer una comunicación inalámbrica entre la cerradura (simulada por el Arduino) y el Hub (Simulado por Node Red). El Hub procesará la información a través de un flujo de telemetría y la publicará en uno de los dos servidores de Mosquitto dependiendo de la región en la que se encuentra el Hub, esto con el fin de distribuir la información de emergencias para que los servidores de mensajería no colapsen en caso en que se envíen muchas emergencias al mismo tiempo. Se puede observar que también se tiene un servidor de mensajería Kafka, del que el programa P2 leerá los mensajes para persistir la información de las emergencias a través de un servicio REST que está sobre Flask.

En cuanto a la capa *Speed*, se tienen 4 copias de P1 corriendo en máquinas distintas, en donde 2 están suscritas a un servidor de Mosquitto y las otras 2 al otro. Para asegurarnos de que cada alarma no llegue a distintas copias de P1, cada copia está suscrita a tópicos distintos para distintas regiones.

Es necesario aclarar que para la arquitectura es necesario que los dos servidores cuenten con un bridge a Kafka para poder persistir las alarmas que provienen de todas las regiones, pero dada la limitación en la cantidad de máquinas disponibles y ya que simulamos únicamente la cerradura y el Hub de un solo apartamento, decidimos implementar la persistencia solo para uno de los servidores de Mosquitto.

Se muestra la parte restante de la capa *Speed* a continuación:



Se observa que tenemos varias copias de computación del REST Mock, en donde cada una está asociada a una copia de P1 para poder soportar una gran cantidad de peticiones en un periodo corto de tiempo. Debido a la poca disponibilidad de máquinas, se decidió ejecutar en una misma máquina un servidor de Mosquitto y una copia del REST Mock; de igual manera, el otro servidor de Mosquitto, Kafka, el bridge entre estos dos, y el programa P2 corren en una sola máquina.