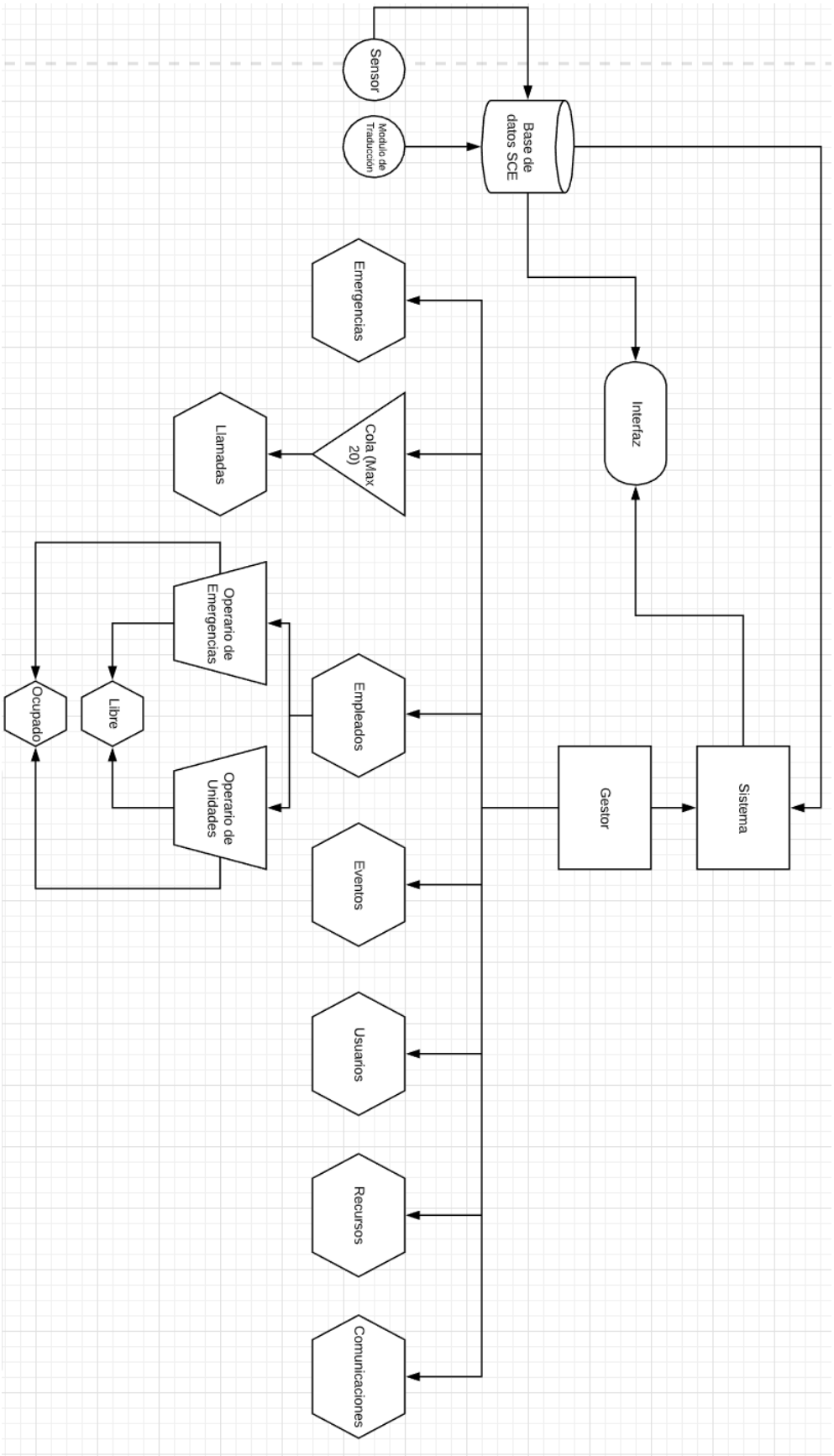
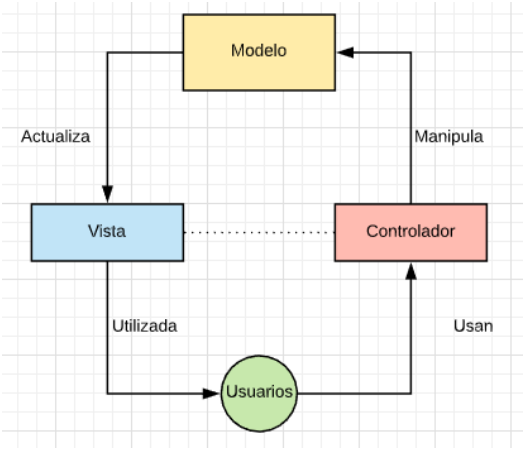
- **Distribución de la App:**



El estilo general de nuestra App será estilo M-V-C (Modelo Vista Controlador): Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.



Dada nuestra distribución y el estilo M-V-C encontraremos, en el modelo, la BD y el gestor.

El Modelo es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del controlador.

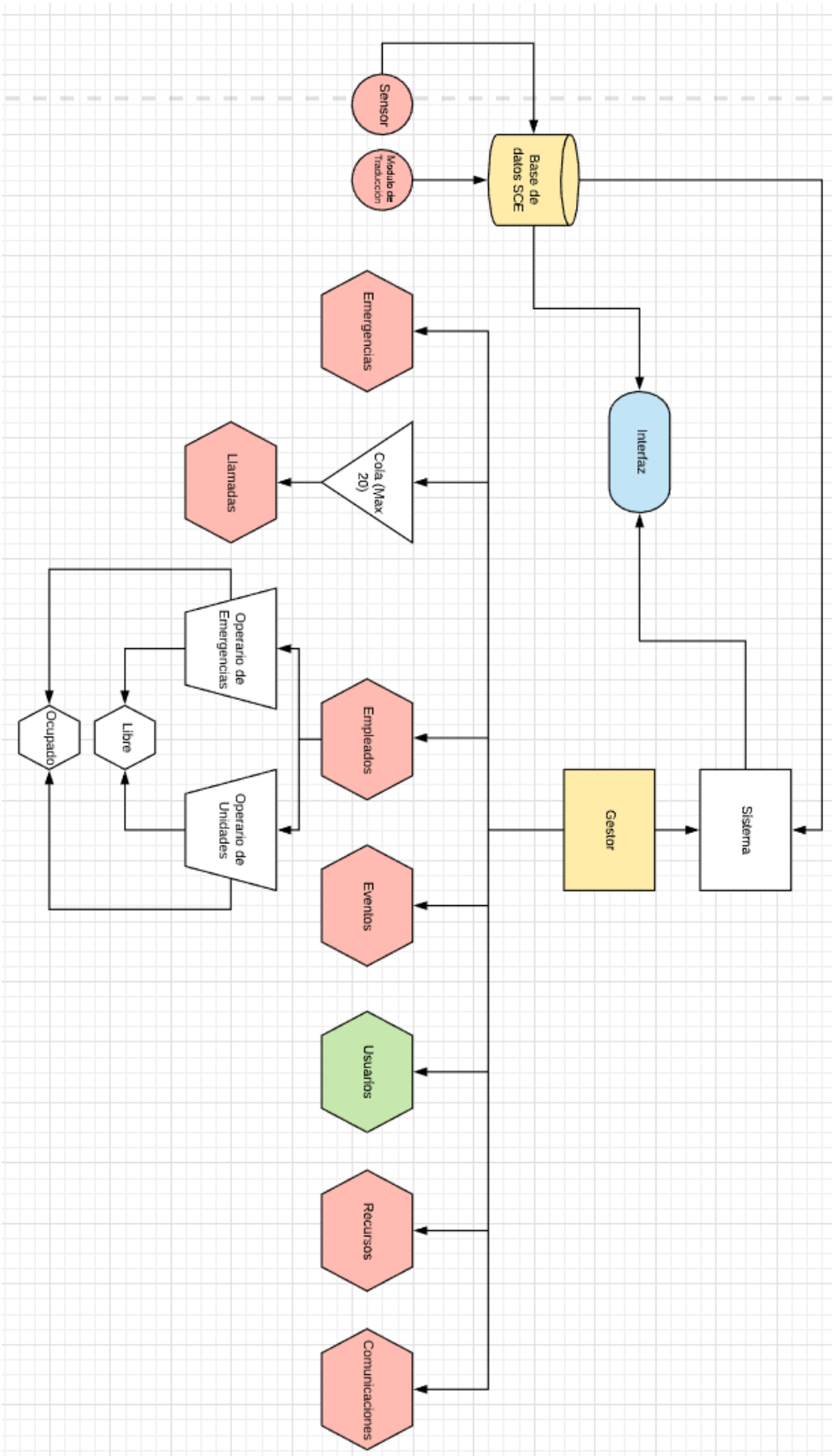
El controlador nos lo encontraremos en el trabajo de gestión del gestor.

El Controlador responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto, se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.

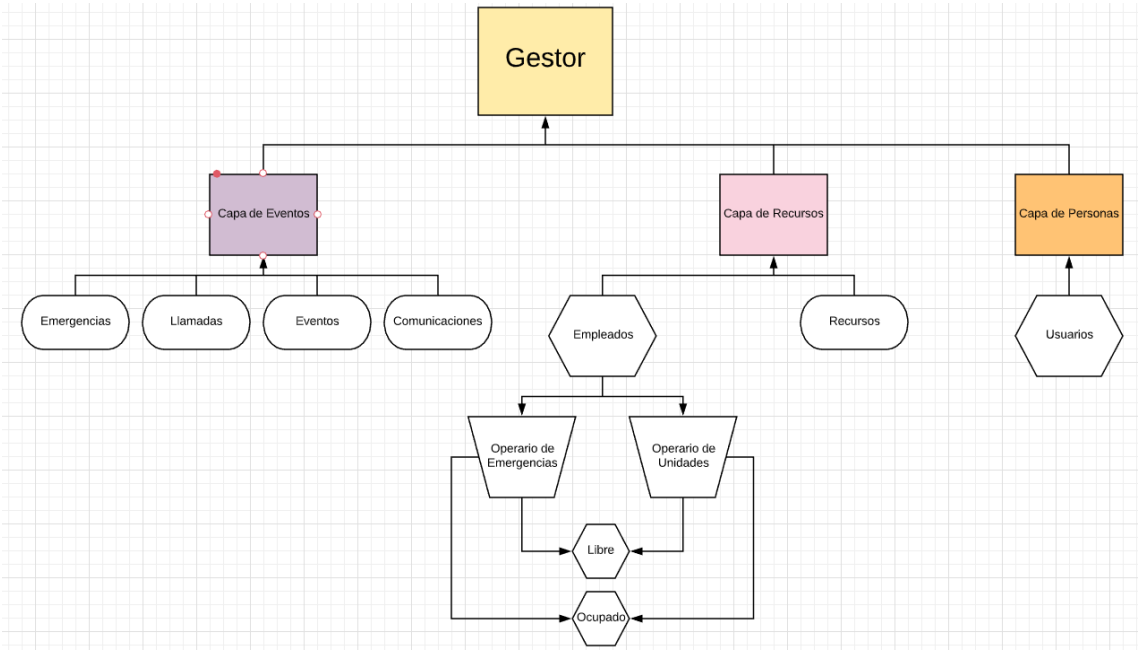
En cuanto a la vista, nos encontraremos con una interfaz centralizada.

La vista Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.

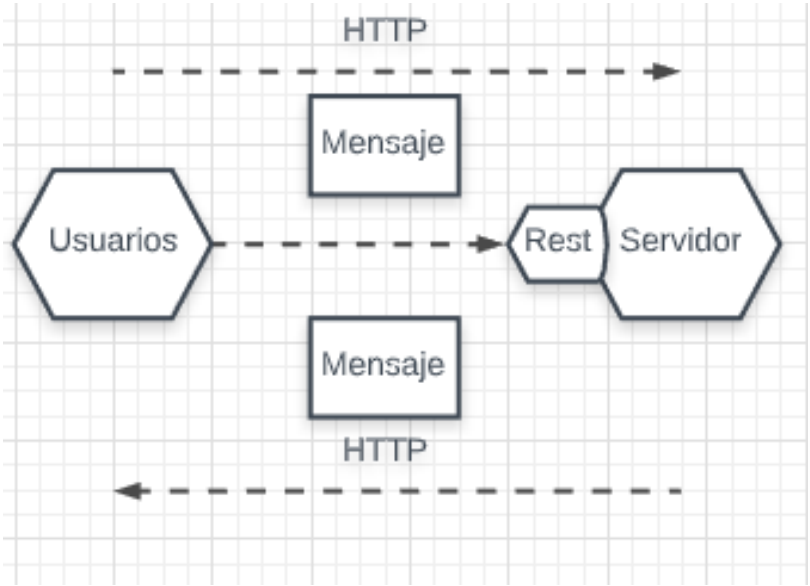
Traducido a nuestra App sería una cosa así:



Una vez definido el diseño general, nos centraremos ahora en algunos sub-diseños, en cuanto al gestor usaremos un estilo por capas, para diseñar cada una de sus subpartes.

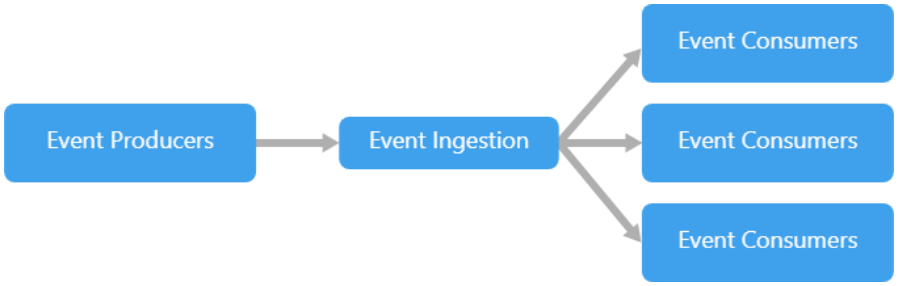


Las interacciones de los usuarios del gestor serán diseñadas mediante el estilo cliente-servidor. El estilo cliente-servidor los clientes acceden a datos y aplicaciones mediante servidores, estos servidores serán redes peer-to-peer con peticiones asíncronas y respuestas a eventos. Además, utilizaremos el estilo REST.



Dentro de nuestra capa de eventos, del estilo de capas de nuestro gestor, usaremos el estilo basado en eventos.

Una arquitectura basada en eventos consta de productores de eventos que generan un flujo de eventos y consumidores de eventos que escuchan los eventos.



Los eventos se entregan casi en tiempo real, de modo que los consumidores pueden responder inmediatamente a los eventos cuando se producen. Los productores se desconectan de los consumidores y los consumidores se desconectan entre sí.

Esta arquitectura se debe usar cuando:

- Varios subsistemas deben procesar los mismos eventos.
- Procesamiento en tiempo real con retardo mínimo.
- Procesamiento de eventos complejos, como coincidencia de patrones o agregación durante ventanas de tiempo.
- Gran volumen y alta velocidad de datos.

En nuestro caso es apropiado hacer uso de esta arquitectura ya que podemos englobar aquí múltiples requisitos como: el gestor regula los eventos de manera simultánea, el gestor de eventos enviará un SMS y una alerta al sistema, gestor de usuarios que regulará un sistema de suscripciones para usuarios, gestor de usuarios que les notificará eventos en tiempo real a Smartphones y tablets, gestor de eventos que los organizará por prioridad, etc.

Este estilo tiene unas claras ventajas ya que desvincula productores y consumidores, es fácil agregar nuevos usuarios y suscriptores, los eventos pueden ser consultados inmediatamente y en nuestros casos de emergencia esto es importante y los subsistemas tienen vistas independientes del flujo de eventos.

Por otro lado, hay escasas desventajas como la posibilidad de desborde y no garantiza por el lado del publicador que el suscriptor responderá ante el evento.

Una vez definidos los estilos, debemos definir los patrones.

Mediante el patrón Singleton nos encargaremos de crear una única instancia del gestor, a la hora de crear emergencias, llamadas, empleador, eventos, usuarios, recursos y comunicaciones usaremos el patrón builder.

La gestión de objetos en la clase gestor se hará mediante el patrón Abstract Factory para que no se mezclen y se vea el tipo de familia a la que pertenecen.

En el caso de empleados o usuarios además, usaremos el patrón factory Method, así ocultamos la diversidad de casos particulares que pueden tener tanto un empleado como un usuario.

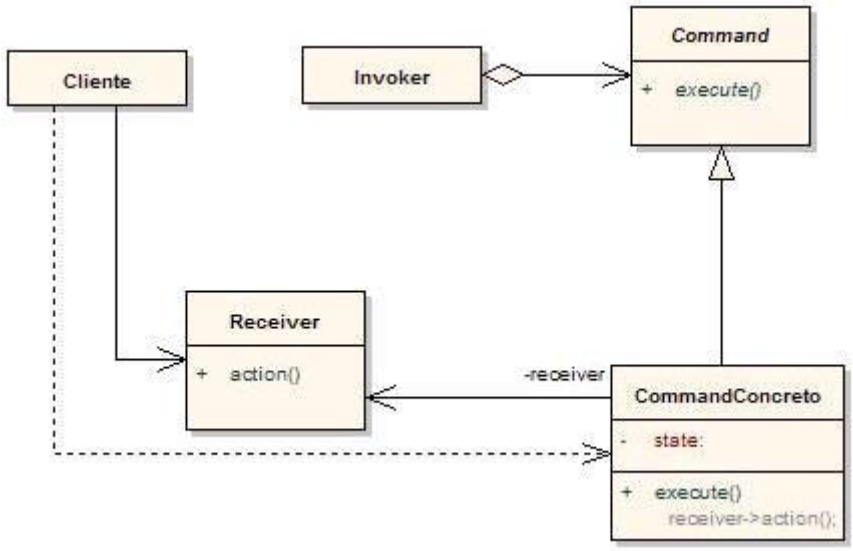
Mediante el patrón Facade proveeremos de una interfaz unificada simple para acceder a cada grupo de interfaces del gestor.

Todo tipo de comunicación entre objetos de diferentes clases deberán llevar el patrón Chain of Responsibility para definir una estructura común en los mismo.

En el caso de los empleados, modificar su comportamiento, en función de si estan libre o no, utilizando el patrón state.También usaremos el patrón Observer para notificar los cambios de estos empleados a todos los objetos que lo necesiten saber.

Las comunicaciones a los usuarios suscritos se harán mediante el patrón Publish-Subscribe, así, se les notificará cambios no programados.

A la hora de encolar o encapsular peticiones podemos usar el patrón Command, permite solicitar una operación a un objeto sin conocer el contenido ni el receptor real de la misma.



Las ventajas de este patrón es que podremos reutilizarlo en nuevos controladores y es fácil de ampliar.