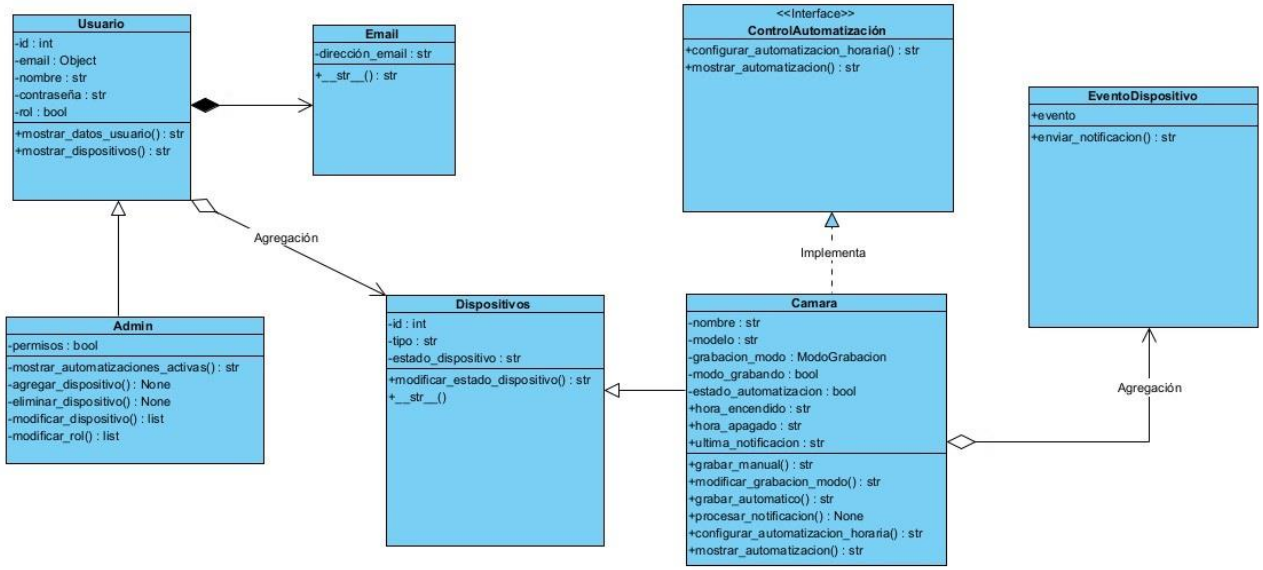
Diagrama de Clases



Entidades:

- Usuario
- Email
- Admin
- Dispositivos
- Camara
- ControlAutomatizacion
- EventoDispositivo

Clases:

class Usuario():

Esta es la clase base para representar cualquier tipo de usuario. Tiene una relación de composición con la clase Email, ya que toda instancia de clase Usuario “todo” contiene un objeto Email “parte”. Su relación es fuerte, porque una instancia de clase Usuario no podrá existir sin el objeto parte Email.
Dentro de la clase Usuario encontramos los atributos: email, nombre, contraseña y rol, que son atributos privados con el fin de proteger los datos internos de la clase y no puedan ser accedidos o modificados directamente desde fuera de ella, sino que deban implementarse los métodos públicos **getters y setters**, principio de **encapsulamiento**.
Así mismo, los métodos que implementa esta clase son mostrar_datos_usuario() y mostrar_dispositivos() que muestran información del usuario y utilizan métodos getters para acceder a ella.

class Email():

Esta clase como se mencionó anteriormente forma “parte” del objeto “todo”. Un Usuario no podría existir si no posee un Email. Su relación es fuerte, ya que es una relación de **composición**. Su atributo es privado ya que contiene información importante y solo se puede acceder a él, mediante **getters y setters**.
Contiene un **método mágico o especial __str__()** que proporciona una descripción legible de la clase.

class Admin():

La clase Admin hereda, por **herencia simple**, de la clase Usuario (**clase base**) todos los atributos y métodos cumpliendo con el principio de **herencia**. Y agrega sus propios atributos y métodos adicionales, que son **privados** para poder ser accedidos según el principio de **encapsulamiento** con **getters y setters**. En este punto vale aclarar, que el atributo permisos es privado porque contiene información sensible, no debería poder modificarse desde fuera de la clase y se evita de esta manera que cualquier usuario pueda otorgarse permisos de administrador. Si se necesita leer o cambiar se haría con los métodos getter y setter o mediante la lógica de la misma clase Admin.
Para que esto se logre, el **método** modificar_rol() dentro de la clase Admin, también se mantiene **privado**, al igual que los demás métodos, que solo van a poder ser accedidos si se poseen los permisos de administrador.
Además de lograr el principio de encapsulamiento, ahora también se logra otro principio fundamental de la POO, que es el principio de **abstracción**. Esto se cumple porque se consigue el **nivel de abstracción adecuado** para ambas clases (Usuario y Admin) al enfocarse en los aspectos más relevantes para el **dominio del problema**. De esta manera se simplifican las clases al **omitir los detalles irrelevantes** que cada instancia de clase podría no necesitar.

class Dispositivos():

Esta clase permite instanciar dispositivos que se asignan a un usuario. Por esta razón la clase Dispositivos “parte” mantiene una relación de **agregación** con la clase Usuario “todo”. La relación no es fuerte, ya que el objeto “parte” puede existir aunque el objeto “todo” sea destruido, y viceversa.
La clase cuenta con atributos privados (principio de encapsulamiento), donde uno de ellos es **autoincremental**: **id**. Lo que quiere decir que, automáticamente, cada vez que se instancie la clase, a cada objeto se le asignará un **id único** y que va a **incrementar en 1** respecto a la instancia de clase anterior.
Por otro lado, contiene métodos publicos. El primero, modificar_estado_dispositivo() modifica el atributo privado estado_dispositivo() mediante un método setter. El segundo es el método especial __str__(), mencionado anteriormente.

class Camara():

Esta clase hereda, por **herencia simple**, de la clase Dispositivos. Adquiere los atributos y métodos de la clase base pero agregando atributos y métodos adicionales.
Los atributos adicionales se mantienen privados al igual que los heredados, ya que contienen datos propios de la clase. Esto garantiza que se accedan o modifiquen de manera segura.
En cuánto a los métodos contiene métodos propios de la clase iniciar_grabacion_manual(), detener_grabación_manual(), cambiar_grabacion_modo() y obtener_estado_grabación() que agregan funcionalidades al objeto. Ademas, incluye métodos implementados, los cuales provienen de la interfaz ControlAutomatizacion.
Estos últimos se definen por **contrato de la interfaz**, lo que quiere decir que toda clase que la implemente debe implementar obligatoriamente el conjunto de métodos. Por este motivo, la interfaz que se menciona aquí, es una **Interfaz Formal**.

class ControlAutomatización()

Esta clase es una clase abstracta: **Interfaz Formal**. Como se mencionó anteriormente, define un conjunto de métodos que las clases que la implementan por contrato deben cumplir.

Su propósito en el desarrollo de la aplicación es permitir configurar automatizaciones de cualquier dispositivo que pueda ser automatizado. De esta manera, un mismo método se implementa de forma diferente según el objeto que lo use, logrando así el principio de **polimorfismo**.
Para poder definir la interfaz formal en Python, se utiliza el módulo por defecto ABC (Abstract Base Classes). En esta interfaz se definen métodos pero no se implementan y se obliga a la clase que la utilice a implementarlos.
Por último, los metodos: configurar_automatizacion(), mostrar_automatizacion(), apagar_automatizacion() y enviar_notificacion() definidos por la interfaz, son métodos públicos para garantizar que las clases que utilicen la interfaz puedan cumplir con el contrato y que otros objetos puedan acceder a la funcionalidad definida por la interfaz.

class EventoDispositivo()

La relación de esta clase con la clase Camara es de agregación. Esta entidad notifica cuando ocurre un evento en una instancia de clase de la parte “todo”. Tiene un solo atributo, que es de acceso público, y sólo un método que se encarga de realizar lo mencionado anteriormente.