Valentina Calderón: 202020771.

Daniela Espinosa: 202022615.

Juan Camilo Neira: 201922746.

**Etapa de análisis.**

**Requerimientos funcionales que la aplicación debe soportar.**

Un usuario puede crear un proyecto y agregar a otros usuarios como participantes del proyecto. El participante puede realizar actividades como parte del proyecto. Asimismo, el participante debe indicar el tipo de actividad que realiza y cuando inicia y cuando finaliza de trabajar en esta actividad. Además, el participante puede registrar una actividad que él o ella recién realizó o que realizó en días o horas pasadas. Por último, el usuario puede acceder a un reporte de las actividades realizadas por este o por otros participantes en el proyecto.

**Restricciones del proyecto.**

Tenemos las siguientes restricciones sobre la construcción del software. Debemos utilizar Git, que es un software de control de versiones. En este sentido, todas las extensiones de los archivos que utilicemos deben ser compatibles con este software. Por ejemplo, debemos utilizar archivos planos (csv, txt, etc.), ya que estos son en gran medida compatibles con Git al momento de almacenarlos y sincronizarlos en este repositorio. Asimismo, existe la restricción de que la interfaz de la aplicación debe estar basada en consola. Por último, contamos con tres entregas: análisis, diseño e implementación. Estas tres entregas deben ser completadas en un rango de un mes.

**Etapa de diseño.**

**Primera iteración.**

**Responsability- Driven Design:**

1. Roles:

* Proyecto.
* Participante.
* Actividad.

1. Responsabilidades.

* Proyecto:
  + Consultar el nombre del proyecto.
  + Consultar la descripción del proyecto.
  + Consultar el reporte de un participante.
  + Consultar la fecha estimada de inicio.
  + Consultar la fecha estimada de finalización.
  + Almacenar los participantes del proyecto.
  + Almacenar las actividades.
* Participante:
  + Consultar el nombre del participante que creó el proyecto.
  + Consultar el correo del participante que creó el proyecto.
  + Consultar el nombre de un participante.
  + Consultar el correo de un participante.
  + Agregar un participante al proyecto.
* Actividad:
  + Consultar el nombre.
  + Consultar la descripción.
  + Consultar la fecha.
  + Consultar el tipo de actividad.
  + Almacenar los reportes de cada participante.
  + Conocer el tiempo total invertido.
  + Conocer el tiempo promedio por tipo de actividad.
  + Conocer el tiempo por cada día.

1. Colaboraciones.

* Agregar participantes a un proyecto.
* Consultar el reporte de un participante.
* Consultar el participante que creó el proyecto.

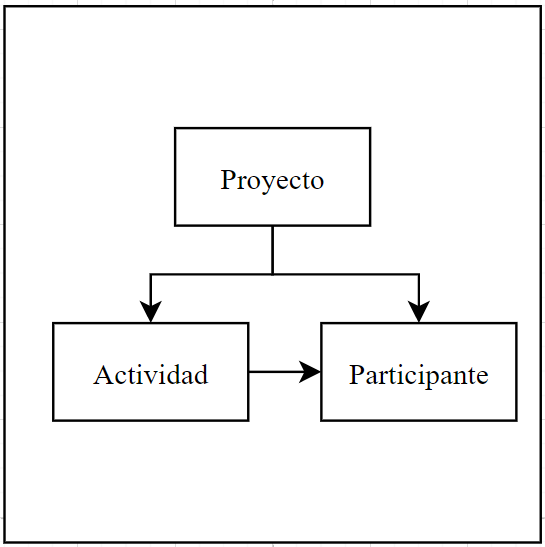


Figura. Esquema general.

**Segunda iteración.**

Realizamos cambios en el esquema general del diseño, debido a que los tres roles (Proyecto, Actividad y Participante) cumplen muchas responsabilidades. Así que, continuando con el diseño iterativo, subdividimos el esquema general de la siguiente forma.

**Responsability- Driven Design:**

1. Roles:

* Aplicación.
* Proyecto.
* Participante.
* Sesión.
* Actividad.
* Estadística.

1. Responsabilidades.

* Aplicación:
  + Consultar la descripción del proyecto.
  + Consultar la fecha estimada de finalización del proyecto.
  + Consultar el nombre del participante que creó el proyecto.
  + Consultar el correo del participante que creó el proyecto.
  + Consultar el correo de un participante.
  + Consultar la cantidad de participantes del proyecto.
  + Consultar la descripción de una actividad
  + Consultar la fecha de una actividad.
  + Consultar el tipo de una actividad.
  + Consultar el reporte de un participante.
* Registro:
  + Crear un proyecto.
  + Crear una actividad.
  + Crear un participante.
* Proyecto:
  + Conocer el nombre del proyecto.
  + Conocer la descripción del proyecto.
  + Conocer la fecha estimada de finalización.
* Participante:
  + Conocer el nombre del participante que creó el proyecto.
  + Conocer el correo del participante que creó el proyecto.
  + Conocer el nombre de un participante.
  + Conocer el correo de un participante.
* Actividad:
  + Conocer el participante.
  + Conocer el nombre de la actividad.
  + Conocer la descripción.
  + Conocer la fecha.
  + Conocer el tipo de actividad.
* Sesión:
  + Conocer el tiempo invertido en una sesión por participante.
* Reporte:
  + Conocer el tiempo total invertido.
  + Conocer el tiempo promedio por tipo de actividad.
  + Conocer el tiempo por cada día.

1. Colaboraciones.

* Calcular las estadísticas del reporte.
* Actualizar el reporte del participante.

Sistemas de persistencia.

1. Archivo con información del proyecto y de las actividades del proyecto.

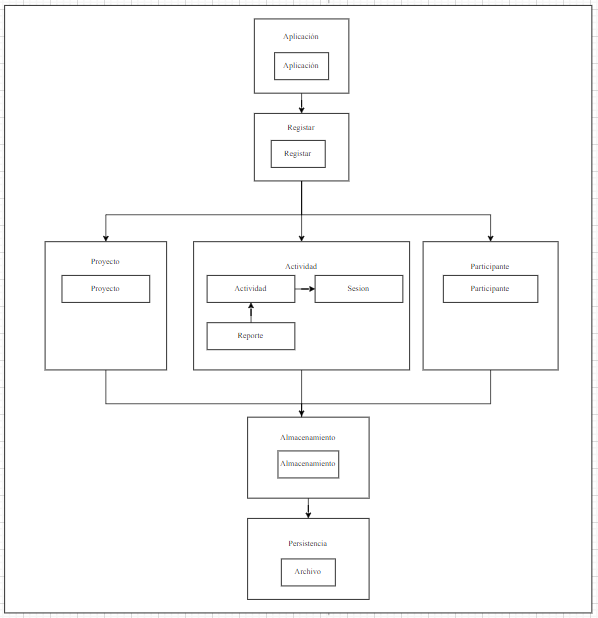


Figura. Componentes candidatos y sus estereotipos.

**Colaboraciones y Diagramas de secuencia.**

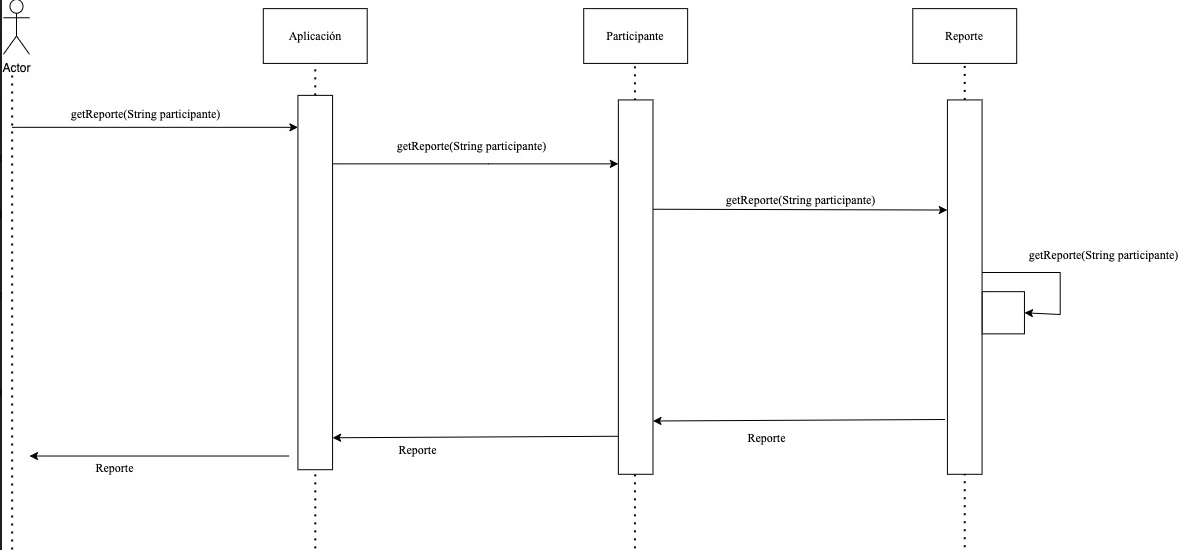


Figura. Diagrama de secuencia de buscar un reporte.

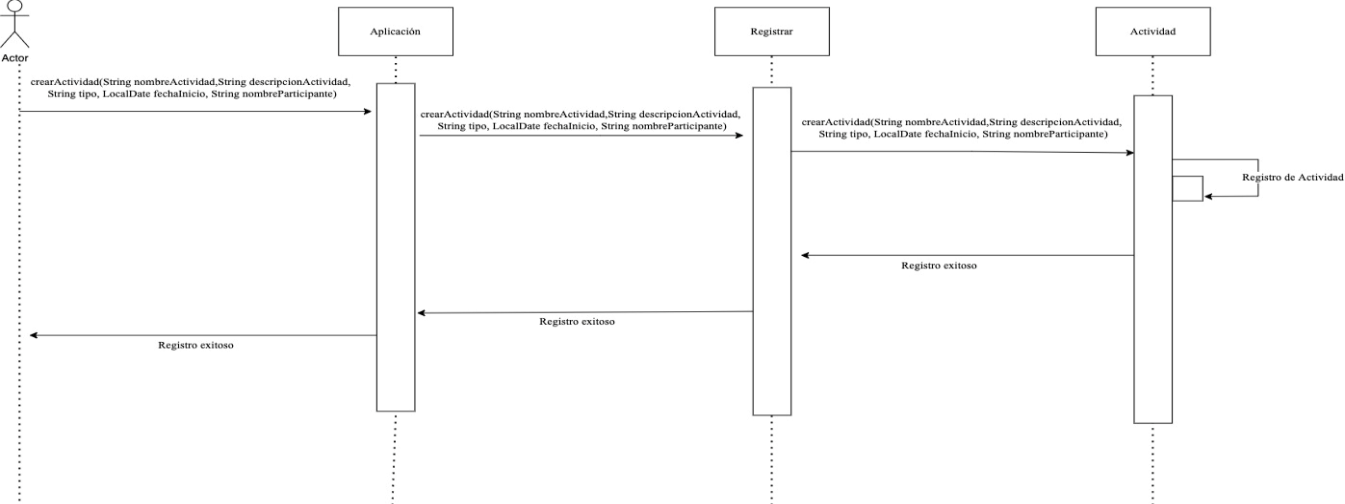


Figura. Diagrama de secuencia de crear una actividad.

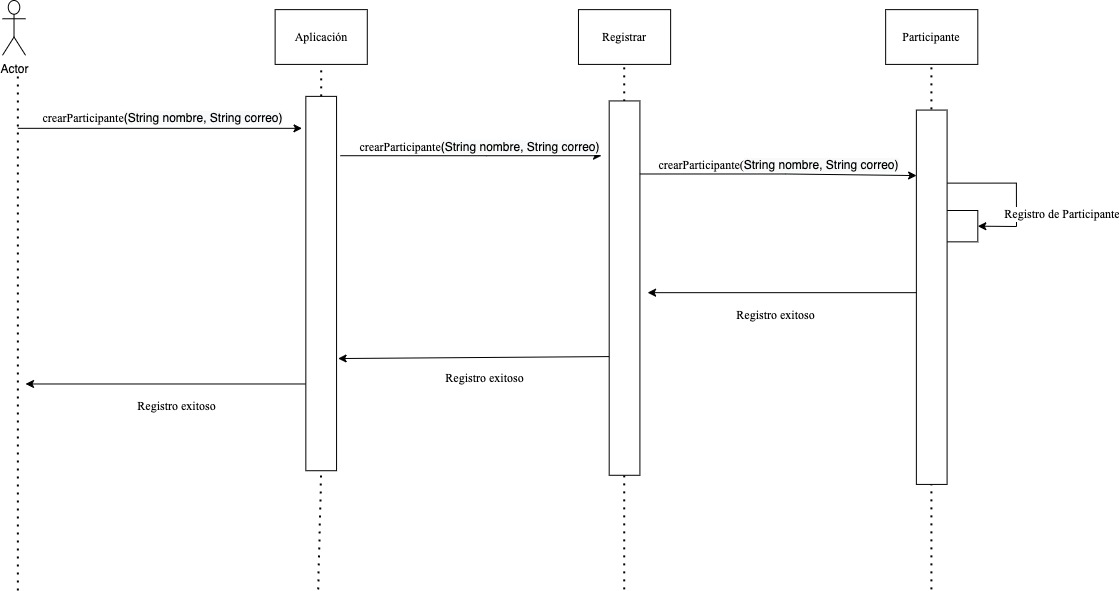


Figura. Diagrama de secuencia de crear un participante.

**Tercera iteración.**

Esta es una actualización del documento de diseño después de terminar la implementación de la lógica del proyecto. Como era de esperarse, hubo cambios en el diagrama de clases, por ejemplo, se añadió nuevos métodos y atributos, se crearon nuevas relaciones entre clases e incluso se decidió no implementar algunas clases.

**Decisiones relevantes durante la implementación del modelo.**

En primer lugar, en la clase App, que es la clase que interactúa con el usuario, decidimos que el participante se identificaría principalmente por su correo. Esto se debe a que los nombres se pueden repetir, pero los correos no. Por esta razón, la App solicita, en algunas ocasiones, solo el correo y no el nombre.

En segundo lugar, la funcionalidad de cronómetro se llevó a cabo con la clase java.time y con métodos propios para obtener el tiempo en minutos entre un tiempo inicial y un tiempo final. Por ejemplo, se utilizó LocalDate para obtener el tiempo actual en formato (Hora: Minuto: Segundo). Asimismo, Duration se utilizó para encontrar la diferencia en minutos entre dos tiempos en el formato antes mencionado. Ahora bien, el cronómetro consistió en almacenar estas diferencias de tiempo en un ArrayList y posteriormente recorrer esta estructura de datos e ir sumando estas diferencias para calcular el tiempo total invertido.

En tercer lugar, se decidió, por un lado, registrar actividades pasadas y, por otro lado, realizar una actividad a partir del momento en que se ejecuta esta opción. En este último caso es cuando se utilizó el cronometro, a diferencia del primer caso en el cual solo es necesario determinar la diferencia entre la hora de inicio y la hora de fin ingresada por parámetro.

En cuarto lugar, no se implementó la clase Sesión que se encuentra en la segunda iteración de este documento. Esto se debe a que la clase java.time y los métodos creados en la clase Actividad fueron suficientes para calcular y almacenar una hora de inicio y una hora de fin, y por tanto no estuvimos en la necesidad de crear otra clase que tuviera esta responsabilidad.

En quinto lugar, no se implementó la clase Almacenamiento que se encuentra en la segunda iteración de este documento. Esto se debe a que lo más lógico, en la programación orientada a objetos, el que objeto principal, que en este caso es la clase Proyecto, almacene los objetos que le pertenecen, como la clase Actividad o Participante.

En sexto lugar, en la clase Registro y la clase Reporte se implementó solo atributos y métodos de tipo static. Esto se debe a que no es lógico que exista una instancia de alguna de estas dos clases.

En séptimo lugar, decidimos que el reporte de las actividades de un participante no se almacena en una estructura de datos, sino que este reporte se genera cada vez que se solicita. Esto se debe a que el reporte cambia cada ocasión que el participante realiza una nueva actividad y, por tanto, este almacenamiento sería obsoleto con cada nueva actividad registrada.

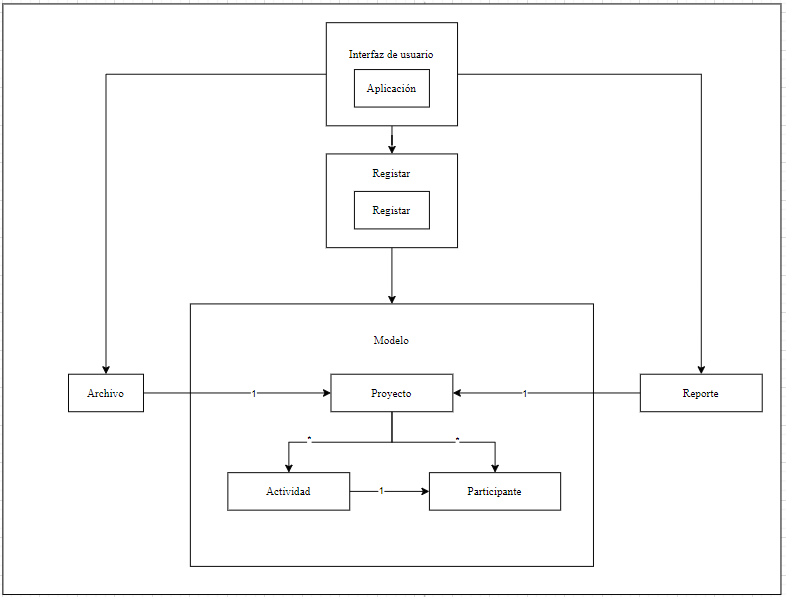


Figura. Diagrama de clases de alto nivel de la implementación.

**Colaboraciones y Diagramas de secuencia.**

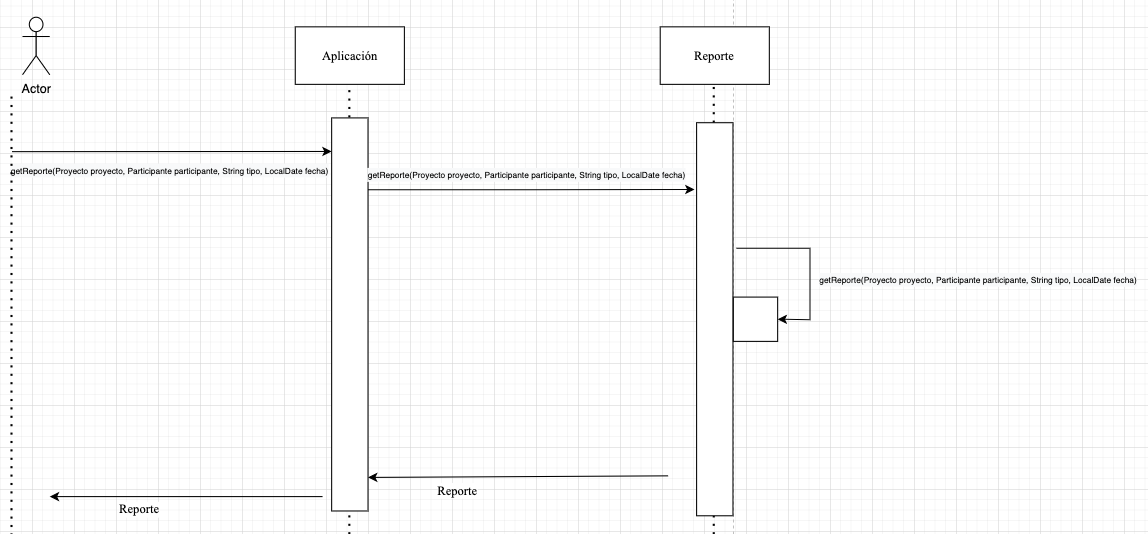


Figura. Diagrama de secuencia de buscar un reporte.

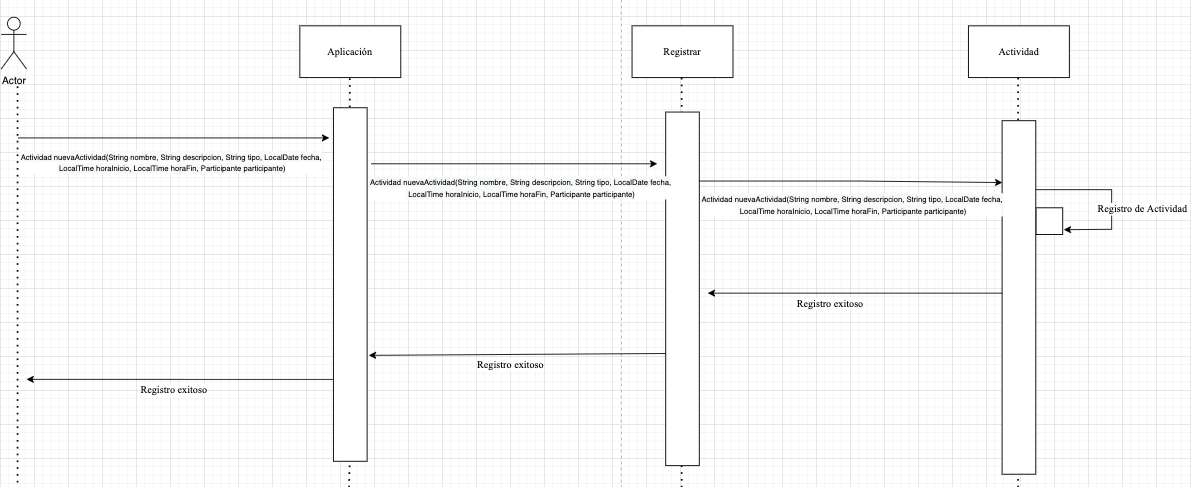


Figura. Diagrama de secuencia de crear una actividad.

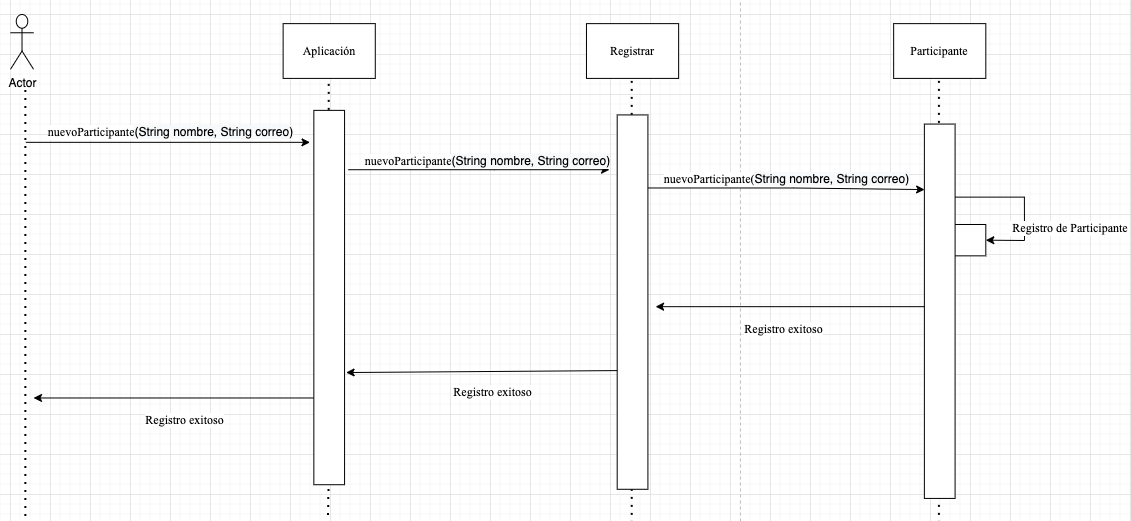


Figura. Diagrama de secuencia de crear un participante.

**Cuarta iteración.**

Esta iteración es una actualización del documento de diseño después de implementar la interfaz gráfica de usuario. No hubo cambios en el modelo, es decir en los métodos e interacciones de las clases Proyecto, Actividad y Participante. En este caso, se añadió nuevas clases con el propósito de crear una interfaz gráfica con el usuario.

**Decisiones relevantes durante la implementación de la interfaz gráfica de usuario.**

En primer lugar, se tomó la decisión de cada clase de la interfaz gráfica de usuario fuera un JFrame, en cambio de reutilizar un mismo JFrame e ir agregando y eliminado distintos JPanel. Esto tuvo la implicación de que en cada cambio de pestaña se puede evidenciar que se cierra y se abre un nuevo JFrame.

En segundo lugar, se tomó la decisión de utilizar librerías externas, las cuales se almacenaron en la carpeta lib del proyecto. En este caso, utilizamos un calendario desplegable para que el usuario seleccione la fecha y un menú desplegable para que el usuario seleccione una hora. Únicamente para estos dos casos decidimos utilizar estos componentes ya implementados y probados.

En tercer lugar, tomamos la decisión de implementar el patrón de diseño de fachada con la clase Registrar. Consideramos que este es un patrón de diseño adecuado en nuestro proyecto porque, si bien no es una interfaz estrictamente, es una forma de unificar la comunicación entre la interfaz gráfica de usuario y el modelo del programa.

**Diagrama de clases.**

Debido a la multitud de clases y relaciones se decidió no copiar directamente el diagrama de clases sino redirigir al siguiente enlace. Recordar oprimir ctrl + click.

[Diadrama de clases.svg](https://uniandes-my.sharepoint.com/:u:/g/personal/j_neira_uniandes_edu_co/ERuhpxnHlfFNiO_1CExYGCMBGKKhIm0nXy2SJYLsXi-4iw?e=vBR4rp)