Ambito de decision (Objetos / Arquitectura / Persistencia / Otro)	Componente/s impactado/s	Decisión	Otras Alternativas	Justificación de la decisión
Lenguaje de Programación	Codigo del sistema	Utilizar JAVA	.net, C#, etc.	Teniamos mas conocimientos previos sobre JAVA y nos parecio el más adecuado para afrontar el TP.
Código	Main	Poner la menor cantidad de métodos posibles		Por ahora solo pusimos tres métodos, que son para pedir por pantalla: String, Double o Int. Estos son los tipos de datos que más pedimos por pantalla. Cada vez que necesitemos pedir algo por pantalla en otras clases, lo hacemos con estos. Además, llamamos a la clase singleton Sistema donde está el método para que empiece el programa, y llamamos al Scheduler para que comience a ejecutar.
Código	Sistema	Crear clase Sistema	No crear la clase	Decidimos crear la clase Sistema, con la funcion de ser el nucleo de nuestro programa. En dicha clase, se llama a la instancia del gestor de Usuario, para poder mostrar la consola del Usuario, mediante el método "arrancar", mencionado previamente en la justificación del Main. Esta consola no será necesaria cuando implementemos la interfaz de usuario.
Codigo	Clases Gestoras	Gestor de: Usuario, Password, Egresos, Criterios, Ingresos		Ninguna de estas Clases fueron agregadas al diagrama de clases ya que no consideramos que sea parte de la lógica de negocio del sistema.
Diagrama De Clases	Clase Producto	Clase Producto y clase Item	No hacer la clase item y calcular directamente en Compra	Hicimos una clase Producto, la cual tiene el producto (una descripcion) junto con su Proveedor, ya que un mismo producto puede ser fabricado por distintos proveedores, y su valor. Por el otro lado, una clase Item, que es la que se relaciona con la Compra, contará con un Producto (consecuentemente, también contará con un proveedor) y la cantidad de dicho Producto que se compró. Por último, tendrá un método "Valor total" que calcula el Precio del Producto por la Cantidad de dicho Poducto que hay en la compra. Así, en la Compra se tendrá una lista de Items, y para saber el valor total de toda la compra habrá que hacer una sumatoria del valorTotal() de cada item, y nada más.
Diagrama De Clases	Clase Compra	Dividimos a la clase compra en dos: Compra y Egreso	Considerar a la compra y al egreso como lo mismo	Consideramos que el Egreso y la Compra son dos cosas distintas. La Compra es la que contiene todos los datos: Items, Presupuestos entre los que puede elegir, Usuarios revisores suscriptos a la bandeja de mensajes de esa compra, documentos, etc. Además, tiene los métodos tales como el encargado de elegir el presupuesto, el valor total (que será el del presupuesto elegido), elegir la estrategia en base a la cual se eligirá el Presupuesto, etc. Por último, tenemos un método efectuarCompra() que crea un nuevo Egreso, y toma la fecha actual, para pasársela a dicho constructor.
Diagrama De Clases	Clase Egreso	Dividimos a la clase compra en dos: Compra y Egreso	Considerra a la compra y al egreso como lo mismo	A nuestro criterio, el Egreso será la Compra efectuada, por eso es que se crea cuando se ejecuta el metodo efectuarCompra() en la clase Compra, y se le pasará la fecha actual. El Egreso contendrá, por lo tanto, una Compra y su fecha de pago. El Gestor de Egresos es el que se encargará de hacer más cosas que el Egreso en sí.

	Codigo	Gestor de Egresos	Singleton		Contiene una lista de Egresos vinculados y otra de Egresos no vinculados (se explica más adelante en la vinculación), una lista de Compras Validadas, una lista de Compras No Validadas y una lista de Presupuestos. La razón por la que creamos dos listas distintas para cada tipo de compra se explica mejor cuando hablamos sobre el Scheduler. La clase Gestor de Egresos es la que se encarga de registrar Egresos, Compras y Presupuestos a partir de métodos en la clase Egresos. También tiene un método que busca compra por ID. Esto es para que el Usuario pueda subscribirse como validador/revisor de dicha compra. Utilizaremos la lista de Egresos al momento de asociarlos a Ingresos.
	Codigo	Compra	Lista de usuarios revisores		Cada Compra tendrá una lista de Usuarios revisores. Por lo tanto, la Compra cuenta con un metodo para hacer que a cada uno de ellos les llegue la validación. Agregamos la variable booleana requierePresupuesto ya que si no lo requiere, hay varias cosas que no se deben validar. El metodo validar llama a los validadores dentro de la clase Validador.
	Diagrama De Clases	Presupuesto	Agregamos la clase Presupuesto		La Compra contiene un Presupuesto. El Presupuesto contiene los mismos Items que la Compra, los Documentos comerciales que contiene una Compra, y el valor total del Presupuesto. Asumimos que una Compra contendrá dentro de su lista de Presupuestos, únicamente Presupuestos que tengan los mismos Items que ella. Es decir, no hacemos ninguna validación al respecto, porque se da por hecho. Cuando se selecciona el presupuesto para la Compra, se settea el valorTotal de la compra con el valorTotal del Presupuesto.
	Codigo	Validador	Singleton		Cuenta con una variable que es la de presupuestos máximos. Esto cambiará más adelante con la aparición del Proyecto. Esta es setteada y es la cantidad de presupuestos máximos que puede tener una Compra. También tiene los tres métodos que validan lo pedido en el enunciado de la Entrega 2. La compra tendrá un método validar() que utilizará dicho validador, y que se ejecutará en el Scheduler. Lo que hace es validar y guardar el resultado de la validación en la bandeja de mensajes de los Usuarios suscriptos a la Compra.
	Diagrama De Clases	Criterio	Agregamos la clase Criterio, utilizando el patrón Strategy, para los Criterios en base al cual se eligirá el Presupuesto	Elegir un presupuesto dentro de la clase Compra, usando IFs	Por ahora solo tenemos un Criterio que nos lo dice el enunciado y ese es el de menor valor. Sin embargo, asumimos que en el futuro habrán más, por eso utilizamos este patrón. El método seleccionarPresupuesto() dentro de Compra lo que hace es llamar al método elegriPresupuesto dentro de su Criterio elegido.
	Código	Verificar password	Verifica que se cumplan todas las restricciones mencionadas en la Entrega 1 sobre las contrsaeñas.		Verifica que no se encuentre entre las 10.000 peores, que tenga entre 8 y 64 caracteres, que la seguridad sea mayor o igual a 80 (esto implica que tenga, al menos, minusculas, una mayuscula, un numero y un simbolo), y que no sea igual al nombre de usuario. Si no cumple con alguna de estas condiciones, pide otra. La verificación de contraseña se realiza en el Gestor de Password.

	Archivo de texto	10.000 peores contraseñas	10.000 peores contraseñas encontradas en Wikipedia		Wikipedia fue el único lugar donde encontramos 10.000 peores contraseñas, el resto solían ser mucho menos, y por más que estén en inglés usamos este archivo. En un futuro podemos traducirlas si no es lo que se solicita.
	Código	Hashear contraseña	Método SHA-384		Fue el algoritmo que nos resultó más fácil de entender.
	Diagrama De Clases	Categorizador	Agregamos clase Categorizador		Esta clase se encarga de categorizar el tipo de empresa (Micro, Pequeña, Mediana Tramo 1 o Mediana Tramo 2) a partir de el promedio de ventas anuales, el personal asignado, y el sector al que pertenece (datos ingresados al darse de alta la empresa). El categorizador se ejecuta apenas se da de alta la empresa y utiliza los topes del Sector que haya sido setteado al darse de alta. Contamos, entonces, con las clases que corresponden a los Sectores (Industria y minera, Agropecuaria, Servicios, Comercio, y Construccion), las cuales tendrán los topes de ventas y cantidad de personal, que marcan los valores necesarios para que la empresa pertenezca a cada categoría, en base a su sector.
	Diagrama De Clases	Entidad	Composite	Herencia	Consideramos que aplicar el patron de diseño composite, era mas adecuado para resolver el problema que simplemente usando herencia. Esto fue porque el enunciado nos aclara que las entidades juridicas estan compuestas por las entidades bases, y estas a su vez, son entidades.
	Código	Gestor de Usuarios y Gestor de Password	Crear clases Singleton	No abstraerlo en clases	Preferimos crear una clase Gestor de Usuarios y Gestor de Password. La clase Gestor De Usuarios, se encarga de controlar todo lo relacionado con los usuarios, como lo es, por ejemplo, darlo de alta, mientras que la clase Gestor De Password, se encarga de validar y crear las nuevas Passwords. Como mencionamos, ninguna de las dos clases las agregamos al Diagrama de Clases porque no pertenecen a la lógica de negocio. Consecuentemente, el validador de contraseñas no se verá reflejado en el Diagrama de Clases.
	Codigo	Interfaz	Abstraimos de la parte de negocio, lo relacionado con la lógica de interfaz, creando dos clases nuevas: Interfaz Usuario e Interfaz Password.		La decisión se debe a que no es correcto mezclar la capa de negocio con la capa de interfaz. La interfaz de Usuario la usamos en muchas partes del código, no solo en la clase Usuario.
	Diagrama De Clases	Criterio de Producto y Categoría de Producto	Es lo que la Entrega 3 define como criterio de categorización pero utilizamos otro nombre ya que tenemos clases "Criterio" y "Categorizador", por lo que nos resultaría confuso ponerle de nombre a esta nueva clase "Criterio de Categorización". Lo que hicimos fue que el Producto tenga una Categoria, la cual conoce su Criterio.	Hacer que el Producto tenga un Criterio, y que el Criteiro tenga Categorias.	Creamos las clases Criterio de Producto y Categoría de Producto. La categoría conoce el nombre del Criterio al que pertenece, y el Item contiene varias Categorías, las cuales solo podrán pertenecer a distintos Criterios, ya que no puede asociarse a más de una Categoría de un mismo Criterio. Esto lo verificaremos cada vez que se quiere asociar una Categoría al Producto, mediante el método asociarCategoria(). Dentro de este metodo, se llama a otro estaAsociadoACriterio(), que lo que hace es verificar si el Criterio al que pertenece dicha Categoría a asociar, ya pertenece a la lista de Criterios que tiene el Producto. De no ser así, se asocia y se agrega dicho criterio a traves del metodo agregarCriterio() a la lista. Por el otro lado, la clase Criterio tiene una lista de Categorias.

	Codigo	Gestor de Criterios de Producto	Crear clase singleton	No crerarlo	Al igual que el resto gestores, no lo agregamos al diagrama de clases. Esta vez lo creamos por lo que dice el segundo punto de la Entrega 3. La organización que implementa el sistema podrá optar por su propio sistema personalizado de criterios de categorización y categorías. Entonces, en este gestor hicimos la lógica para que el usuario, es decir, la organización, genere los criterios y sus respectivas categorías, la cantidad que desee. Por lo tanto, el gestor contará con una lista de Criterios, que se irán agregando a medida que el usuario desee.
	Codigo	Egreso y Presupuesto	Se debe permitir asociar egresos y presupuestos a categorías		Al tener tanto el Egreso como el Presupuesto a los Items y estos a Productos, como los Prodcutos tienen sus respectivas Categorías, ya están entonces asociados los Egresos y Presupuestos a Cateogrías, y esto lo hacen a traves de dichos Productos.
	Código	Gestor de Ingresos y Clase Ingreso	Crear clase Singleton		Decidimos que el usuario pueda registrar el Ingreso, con su descripcion y su monto total. Por el otro lado, el Gestor de Ingresos guardara el Ingreso en una lista de ingresos no vinculados. A su vez, tiene otra lista de Ingresos vinculados (explicado más adelante en la sección de vinculación).
	Codigo	Scheduler	Crear Scheduler para ejecutar el validador		Creamos un Scheduler que ejecuta el validador cada 24 horas. El tiempo no fue especificado en el enunciado pero nos pareció el más lógico para este tipo de sistema. Igualmente, podría variar. En el Scheduler lo que se hace es validar todas las compras dentro del Gestor de Compras que no están validadas, tomando las compras pertenecientes a la lista Compras No Validadas. Esto se hace ejecutando el método validar() dentro de cada Compra. Se las agregar a la lista de Compras Validadas y se las eliminar de la lista a la que pertenecían. Esto es porque no deberían validarse más de una vez cada Compra.
	Código	Dirección Postal	Utilizar datos de la API		Creamos la clase Dirección Postal que está compuesta por atributos de tipo String, pero a su vez está compuesta por los atributos País, Provincia y Ciudad, que son obtenidos de la API de Mercado libre.
	Código	Listado de Paises, Listado de Provincias y Listado de Ciudades	Crear clases "gestores"		Creamos estas tres clases para guardar los listados de Paises, Provincias y Ciudades respectivamente, obtenidos de la API de Mercado Libre
	Base de Datos	Interfaz de Usuario, Interfaz de Password, Listado de Paises, Listado de Provincias, Listado de Ciudades, Sistema, Main, Gestor de Usuario, Gestor de Password, Gestor de Ingresos, Gestor de Egresos, Gestor de Criterios, Componente, Administrador, OSC, Categprozador, Criterio (para eleccion de presupuesto), Vinculador, Criterios de vinculacion, y Validador	No persistir las clases	Persistirlas	Estas clases que utilizamos para el funcionamiento del sistema, no nos parecerion relevantes como para persistirlas en la base de datos. Algunas todavía no estamos seguros si las agregaremos a la Base de Datos o no en algún futuro. Dudamos también si persisitir o no la clase Mensaje, pero decidimos hacerlo, ya que tal vez en algun momento nos convenga tener los mensajes que reciben los usuarios guardados en nuestra base de datos.

	Base de Datos	Pais, Provincia y Ciudad	Persistirlas	No persistirlas y acceder a la API cada vez que se las necesite usar	Decidimos guardar toda la infromación correspondiente a estos tres listados en la base de datos ya que acceder constantemente a la API cada vez que se necesite usar alguno de esos datos o consultarlos, geraría mucho overhead. En caso de actualizaciones, se debería actualiar la base de datos.
	Código	Entidad Persistente	Crear esta clase de la que heredan las persisitidas	No crearla y repetir la lógica del ID en cada clase	Agregamos una clase llamada Entidad Persistente, para que todas las clases que sí decidimos persistir hereden de la misma. Esto fue para evitar la repetición de lógica al momento de crear el ID para cada entidad. Con esta herencia, nos aseguramos de que todas las clases persistidas cuenten con un atributo ID incremental en una unidad en la base de datos.
	Código	Vinculador	Strategy y mecanismo		Decidimos utilizar el patrón Strategy para manejear la vinculación de Ingresos con Egresos. Esto fue a partir de que notamos que existen cuatro formas de vincular, es decir, cuatro formas de hacer lo mismo, por lo que este patrón fue el más indicado. El proceso de vinculación que hará es revisar las listas de "no vinculados" dentro de los Gestores de Ingreso y Egreso, y vincularlos según el criterio (estrategia) que se desee. Una vez que se vincule cada egreso, como solo podrá vincularse a un ingreso, se lo pasará a la lista de "vinculados" dentro de su respectivo Gestor. Por el otro lado, para los Ingresos, como este pueden tener muchos Egresos vinculados, solamente lo pasaremos de la lista de "no vinculados" a "vinculados" si la sumatoria de los montos de los egresos vinculados supera el monto total del ingreso.
	Código	Vinculación	Botón en el menú	Scheduler	Decidimos que el proceso de vinculación se ejecute cuando el Administrador elija dicha opción en el menú. Se verifica si es o no el administrador ya que cuando se presiona dicho botón y se pide la contraseña de administrador (GESOC). Una vez que la contraseña sea ingresada correctamente, se le solicita elegir el criterio de vinculación para proceder a vincular a partir de este. Esta decisión la tomamos por sobre el scheduelr porque nos pareció más simple y porque, además, siempre tuvimos la clase "administrador" pero nunca le encontramos un uso específco, por lo que decidimos darle este.
	Código	Bitácora			Creamos una clase llamada Bitácora utilizando el patrón Singleton. En esta tendremos una lista de tipo Operación, donde se guardarán todas las operaciones de alta, baja y modificación en la base de datos. Esta clase podríamos considerarla un Observer, la cual frente a una modificación, alta o baja del resto de las clases, se actualiza con ayuda de una clase auxiliar llamada GuardadorDeLog.
	Código	GuardadorDeLog			Esta clase es una clase específica que recibe una operación de alta, baja o modificación y a partir de esta se encarga de guardar la nueva operación en aquella lista de Operaciones que contiene la bitácora, como fue previamente mencionado. Entonces, este método se llama GuardarEnBitácora y se le pasa por parámetro un Objeto y un Tipo de Operación ("ALTA", "BAJA" o "MODIFICACIÓn"). Dentro de este método, entonces, se hace el new para crear una nueva operación, y una vez hecha, se la agrega a la bitácora.

	Código	Operación			Por último, contamos con esta clase la cual se compone de 3 atributos: tipo de operación, nombre de la entidad sobre la cual se realizó la operación, y fecha en la cual se realizó la operación. Esta clase no tiene comportamiento, lo único que tiene es su constructor.
	Código	Repositorio			Esta clase fue una clase que nos ayudó al momento de realizar las modificaciones, altas y bajas que hace el usuario en la página web, para impactar sobre nuestras tablas de las bases de datos. Sin embargo, le dimos otro uso adicional. Esta clase cuenta con los métodos de "guardar" (alta), "borrar" (baja) y "modificar" (modificación). Además de realizar dicha acción, tiene la responsabilidad de llamar al GuardadorDeLog y guardar en bitácora.
	Código	Proyecto			Al Scheduler, con la validación de Presupuestos de las Compras, agregamos ahora la validación de las compras pertenecientes a los Proyectos. Esto es porque ahora los Proyectos ponen ciertas restricciones al momento de validar las compras que lo componen. Entonces, este va a settear si es necesario o no un presupuesto en cada compra que lo compone, y de ser así, la cantidad necesaria de los mismos al momento de validarse.