

GeSoc

Decisiones de diseño

**Entrega 2**

[**Introducción**](#_pvrdgv4k41hh) **2**

[**Presupuestos**](#_ygg1o4uqi9xf) **2**

[**Asociación de un presupuesto a un egreso**](#_bnpg5qk33qgx) **3**

[**Validador de egresos**](#_v3pgz569hsqt) **4**

[**Notificaciones a revisores de egresos**](#_qmsagkii6far) **7**

[**Direcciones postales**](#_ta8j7ogdr6ga) **8**

[**Monedas**](#_fg3pvmbv95rl) **8**

# Introducción

Para esta entrega, si bien aún no contamos con UI y estamos pensando sólo en el dominio de la aplicación, consideramos que van a existir tres tiempos a la hora de cargar las entidades del dominio:

* Tiempo de carga del egreso
* Tiempo de carga del presupuesto
* Tiempo de validación de egresos

Basados en estos tres tiempos, construimos las nuevas abstracciones y funciones de la entrega.

Utilizamos además para esta entrega, distintos branches por funcionalidad:

* Conexión a API’s monedas y geografías.
* Validador de egresos.

Mergeamos los branches con master una vez estabilizado el código.

# Presupuestos

Un presupuesto contiene un proveedor, un documento comercial, y un listado de ítems con la descripción, la cantidad y el importe del producto **[Requerimiento 1]**.

# Asociación de un presupuesto a un egreso

Considerando los tiempos que mencionamos en la introducción, un egreso puede crearse sin presupuestos.

Agregamos al constructor de un egreso un booleano que indica si el egreso creado requiere obligatoriamente o no presupuestos (ver Validador de egresos).

Un egreso, tendrá un listado de presupuestos. Para poder asociar un presupuesto a un egreso, se deberá utilizar el builder Presupuestador. El mismo permite cargar un presupuesto borrador, y crear el presupuesto deseado una vez sus datos estén listos. Para poder crear el presupuesto, es necesario informar un egreso para permitir la asociación. Este último dato es obligatorio **[Requerimiento 2]**.

# Validador de egresos

El tercer tiempo que consideramos es el de validación de datos de egresos / presupuestos.

El validador de egresos es un programa que recibe como parámetro el listado de egresos a validar. Es decir, un listado de egresos con estado “Pendiente”.

Para cada egreso se validarán:

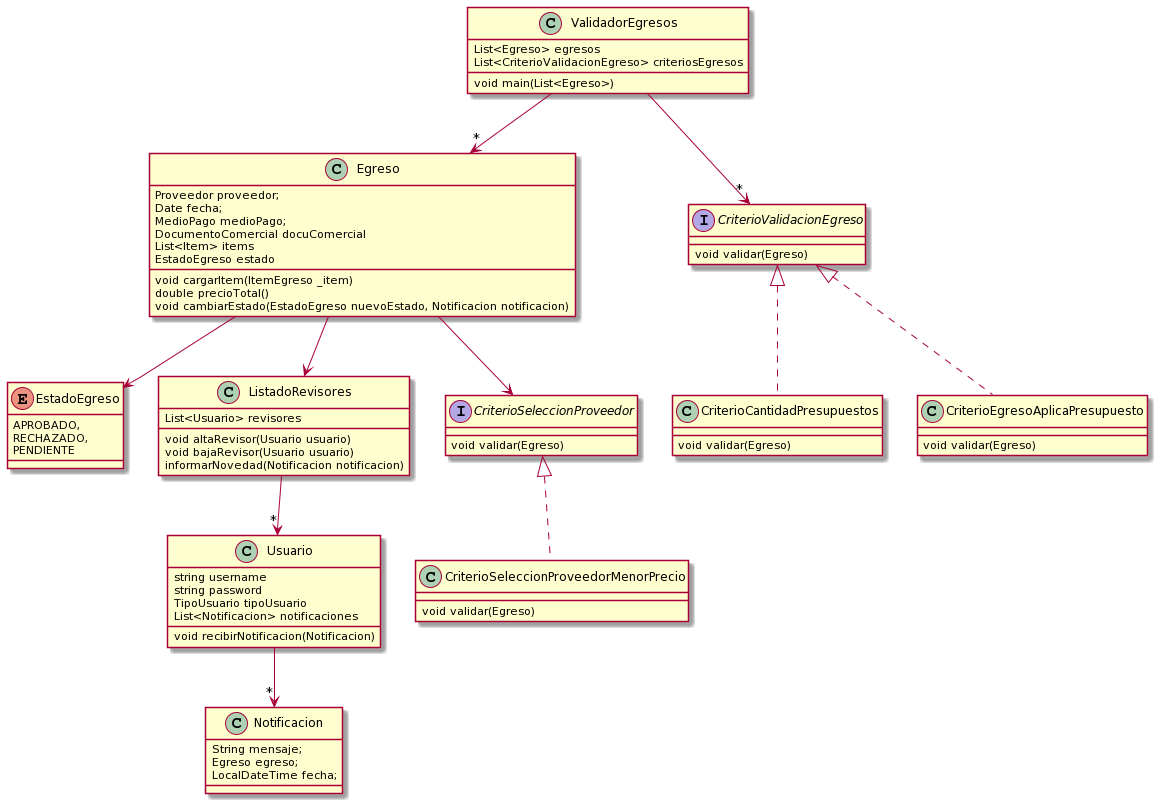
* Criterios generales: son aquellos que aplican para todos los egresos. Es decir, configurables a nivel validador.
  + Cantidad de presupuestos: se valida que el egreso tenga la cantidad de presupuestos requerida por el sistema.
  + El egreso se armó en base a un presupuesto: Se validan los datos del egreso, en comparación con el presupuesto aplicado.
* Criterios particulares: son aquellos que aplican para cada egreso. Es decir, son criterios que se guardan en cada egreso. Por el momento, existe un único criterio particular que es el de selección de proveedor.
  + Menor valor: El criterio de selección de presupuesto considera el menor presupuesto de los cargados para el egreso. Este es el único criterio particular por el momento.

Se valida además para cada egreso, que requiera obligatoriamente presupuestos previo a las validaciones mencionadas.

Se realizó este esquema de Criterios (con sus interfaces correspondientes), del mismo modo que se crearon las “Validaciones de password”. De esta forma creamos validaciones extensibles y encapsuladas **[Requerimiento 3]**.

El validador, como último objetivo, invocará el cambio de estado del egreso a “RECHAZADO” o “APROBADO”.

El programa podrá ejecutarse manualmente desde el sistema operativo. Se generó el .JAR correspondiente al mismo **[Requerimiento 4]**.



# Notificaciones a revisores de egresos

Cada egreso tendrá su “listado de revisores”. Para el egreso entonces, un revisor podrá suscribirse al listado, o darse de baja.

Ante un cambio de estado de un egreso, se disparará la notificación a cada uno de los revisores. La notificación tendrá el mismo egreso, un mensaje, y la fecha de la notificación.

Cada usuario, tendrá un buzón o listado de notificaciones que podrá consultar.

Por el momento, los revisores son de tipo Usuario. En caso de recibir otros tipos de revisores está previsto agregar una interfaz para utilizar polimorfismo **[Requerimiento 5]**.

# Direcciones postales

Considerando que en la entrega se solicita la **obtención** de países, provincias y ciudades, no hemos “conectado” esta función a los datos de las abstracciones. Se considera que los datos servirán de fuente a objetos de la UI, en una futura entrega.

Se utilizó el servicio de Mercado Libre para la obtención de los parámetros estándar.

**[Requerimiento 5].**

# Monedas

Para el caso de las monedas, se utilizó el servicio mencionado anteriormente. En particular, además, se desarrolló una implementación puntual de la invocación al servicio de Mercado Libre para monedas.

Por este punto, se generó la abstracción importe (que contiene la moneda, y el precio unitario asociado). No hemos conectado esta abstracción con el servicio mencionado de obtención de monedas. Entendemos que el usuario final, desde la UI seleccionará la moneda y esta selección se guardará en el objeto Importe, posteriormente **[Requerimiento 5]**.

