# Gestión de Gastos Comunes (Edificio : Propiedad)

**Ingeniería del detalle**

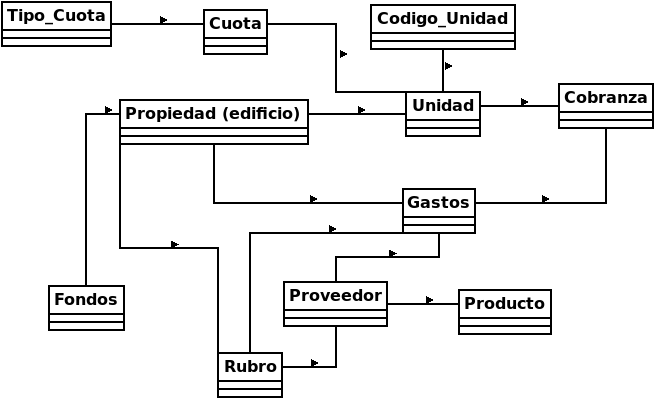


Diagrama No1 - **Modelo de clases**

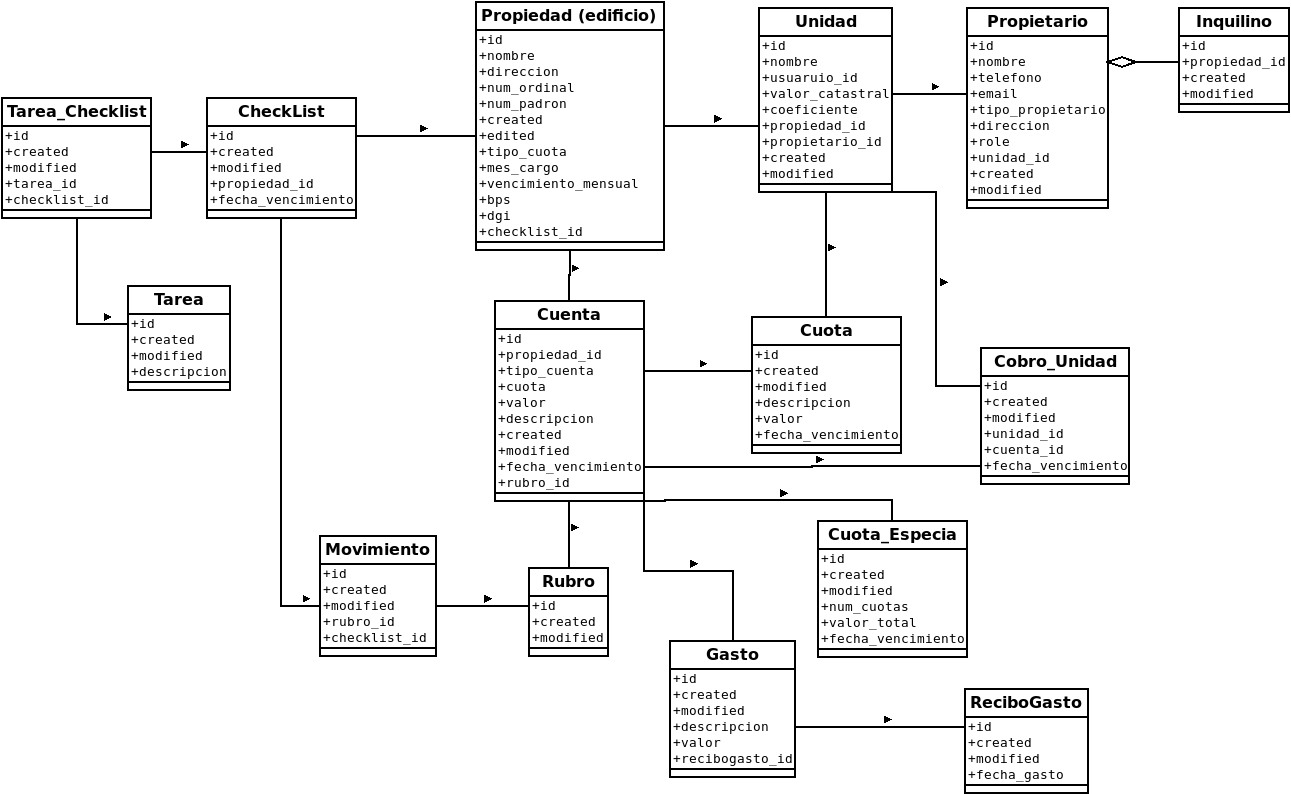


Diagrama No2 - **Modelo de clases – Atributos**

### Generalidades

* El atributo **id** existe en todas las cases. Este atributo es auto-incremental no Nulo y clave primaria. Se gestiona internamente por el Sistema de Gestión de Base de Datos (SGDB) y sera el indice (index) de cada tabla
* El atributo **created\_at**, sera de tipo timeStamp. En el momento de crear un registro (persistir un nuevo objeto en la DB) tomara el valor actual del sistema del servidor. Se gestiona internamente por el Sistema de Gestión de Base de Datos (SGDB)
* El atributo **modifed\_at**, sera de tipo timeStamp. En el momento de modificar un registro (actualizar un objeto en la DB) tomara el valor actual del sistema del servidor. Se gestiona internamente por el Sistema de Gestión de Base de Datos (SGDB)
* Las claves foráneas de las tablas (relaciones en los objetos) serán de tipo entero (integer), no Nulo y apuntara a la clave prmaria de la tabla que relaciona.
* El estándar usado para designar un clase, un atributo, una relacional, o un método sera la nomenclatura CamelCase (JavaDoc).

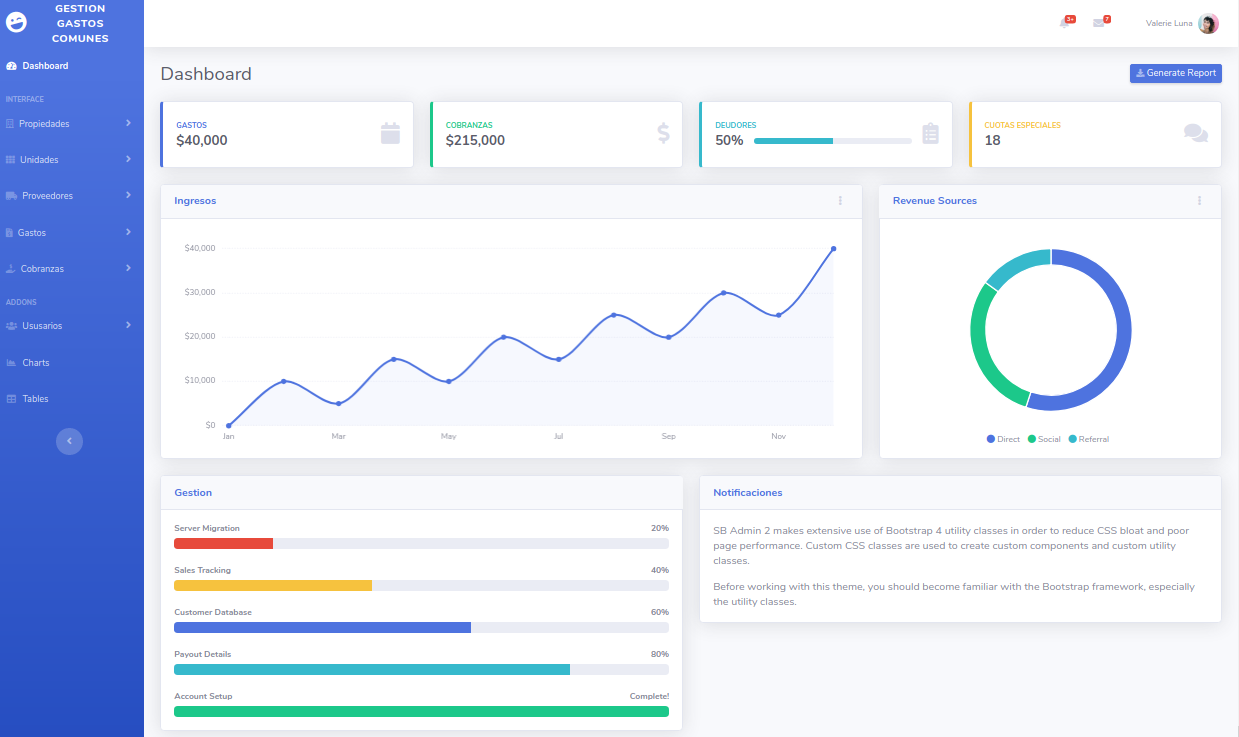
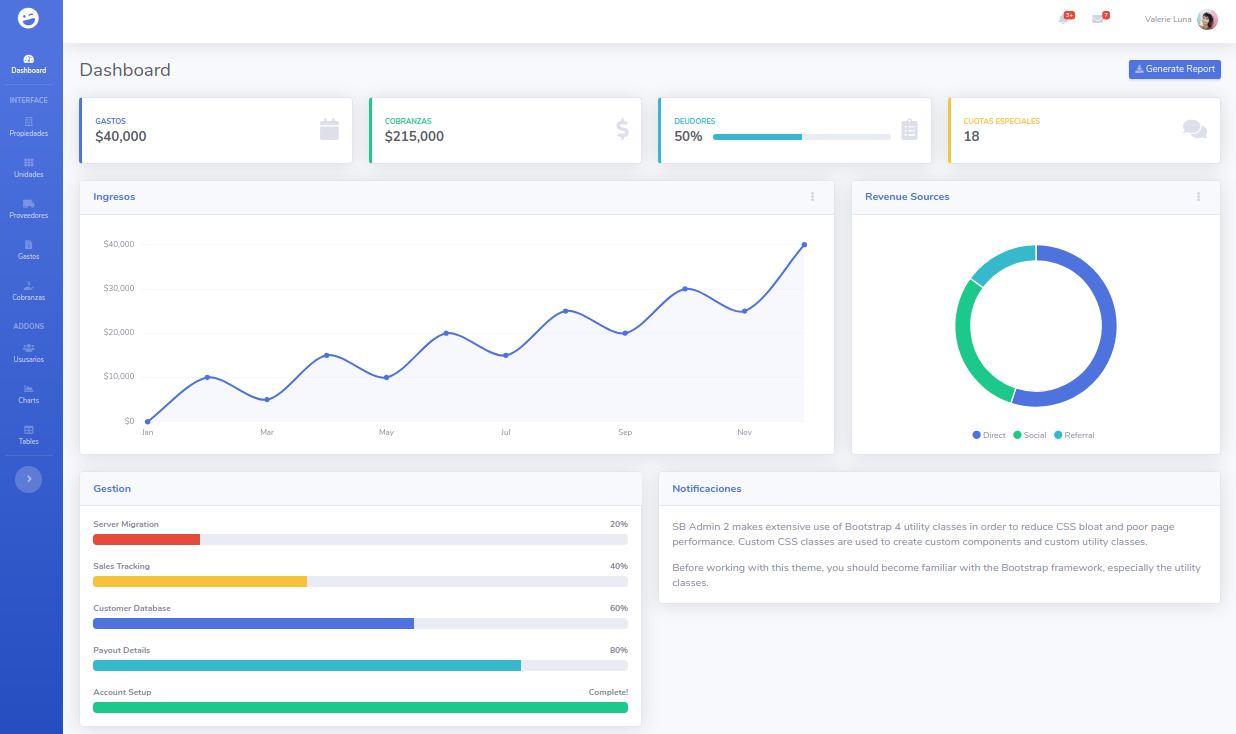


Imagen No 1 – **Vista General**

Imagen No 2 – **Vista General – Menú reducido**

### Funcionalidades por clase (métodos)

Detalles de la Ingeniería

## **PROPIEDAD**

Una “*propiedad”* es la unidad fundamental (singleton) de vivienda.

La “*propiedad”* puede estar compuestas por “*unidades”* (1:n), que son a su vez las subdiviciones de dicha propiedad (p.e. propiedad : Edificio, unidad : Apartamento)

La “*propiedad*” esta asociada a las cuentas, las cuales registran los movimientos de ingresos y egreso

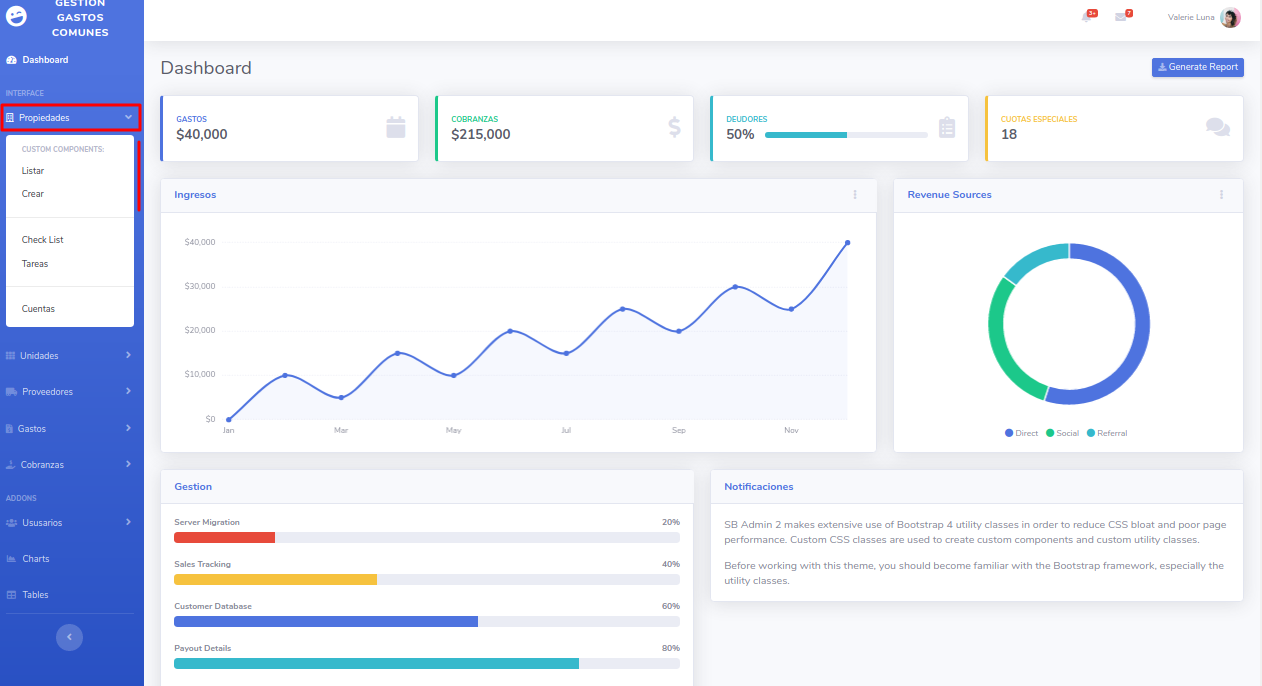


Imagen No 3 – **Vista General – Menú Propiedades**

# PROPIEDAD – Atributos y Métodos

| **Atributos** | **Tipo** | **Parámetros** | | **Retorno** |
| --- | --- | --- | --- | --- |
| nombre | VarChar (45) |  | | |
| direccion | Text |  | | |
| numeroOrdinal | Integer(11) |  | | |
| numeroPadron | Integer(11) |  | | |
| tipoCuota\_id | Integer(11):FK-constrain |  | | |
| fondo\_id | Integer(11):FK-constrain |  | | |
| mesCargo | Date |  | | |
| fechaVencimiento | Date |  | | |
| BPS |  |  | | |
| RUT |  |  | | |
| MTS |  |  | | |
| IMM\_Saneamiento |  |  | | |
| IMM\_Tributo |  |  | | |
| IMM |  |  | | |
| UTE\_General |  |  | | |
| UTE\_Garaje |  |  | | |
| UTE\_Otros |  |  | | |
| OSF |  |  | | |
|  |  |  | | |
| **Método** | **Descripción** | **Parámetros** | **Retorno** | |
| list()  [1]:El método puede implementar un Limit para paginacion (10) – depende del SGBD | Lista todos los objetos de tipo *propiedad* | N/A | arreglo(propiedad) | |
| crear() | Crea un nuevo objeto tipo *propiedad* y persiste sus atributos en la DB | N/A | id – identificador del nuevo objeto persistido | |
| False – fallo inserción en DB | |
| getByID(id) | Recupera un objeto de tipo *propiedad* | id-identificador del objeto *propiedad* | Objeto(*propiedad*) | |
| False - objeto no existe | |
| update(id,data:arreglo(atributo=>valor)) | Actualiza el objeto *propiedad* identificado con el id actualizando los atributos | id-identificador del objeto *propiedad* | True – Actualizo (persistió) el objeto | |
| Data-arreglo (nombre\_atributo => valor\_atributo) | False – No se pudo realizar la operación de actualización | |
| delete(id) | Borra el objeto *propiedad* identificado con el id  [2]:Tomar en cuenta las relaciones de cascada en el SGDB | id-identificador del objeto *propiedad* | True – Borrado el objeto | |
| False – No se pudo realizar la operación de borrado | |
| cuenta(id)  \*\*\* **Ver la clase cuenta** | Recupera la colección de *cuentas* asociadas a la propiedad identificada con el id  \*\*\* La clase *cuenta* esta relacionad con la clase *gasto,* donde se aloja el valor de dicho gasto | id- identificador de la propiedad \*\*\* la relación es : propiedad→cuenta→gasto→ reciboGasto | Colección objetos (*cuenta*) | |
| False – La propiedad no tiene cuentas registradas aun | |
| checkList(id) | Recupera la colección de *tareas* en una relación entre propiedad y tareas a traves de la clse checkList | id- identificador de la propiedad \*\*\* la relación es : propiedad→checkList→tarea | Colección objetos (*tareas*) | |
| False – La propiedad no tiene una lista de tareas (*checkList*) registradas aun | |

# PROPIEDAD – Métodos detalle

# list()

Lista todos los objetos *“propiedad”* almacenados en la base de datos (persistidos).

Este listado sera mostrado de forma ascendente según el total de la deuda que sus “*propiedades”* registren en las “*cuentas”* asociados a dicha *propiedad*.

El listado sera paginado, este parámetro de la longitud de elementos a mostrar por pagina sera configurado en el maestro el sistema

Cada propiedad listada dispondrá de botones de acciones “Modificar”, “Unidades” y “Cerrar Mes”

* Modificar : Recupera los datos de la propiedad y genera un formulario que permite modificar dichos datos (Tener en cuenta que algunos datos serán mostrados pero no podrán ser modificados – disabled)
* Unidades : Muestra el listado de las “*unidades*” asociadas a dicha propiedad. Nuevamente el listado de las unidades preservara el comportamiento mostrado por la propiedad, es decir – paginado – ascendente – botones de acciones
* Cerrar Mes : Procesa las *cuentas* asociadas a dicha *propiedad* y por cada *unidad* en el periodo de tiempo definido (quincena, mes, trimestre, a~no)

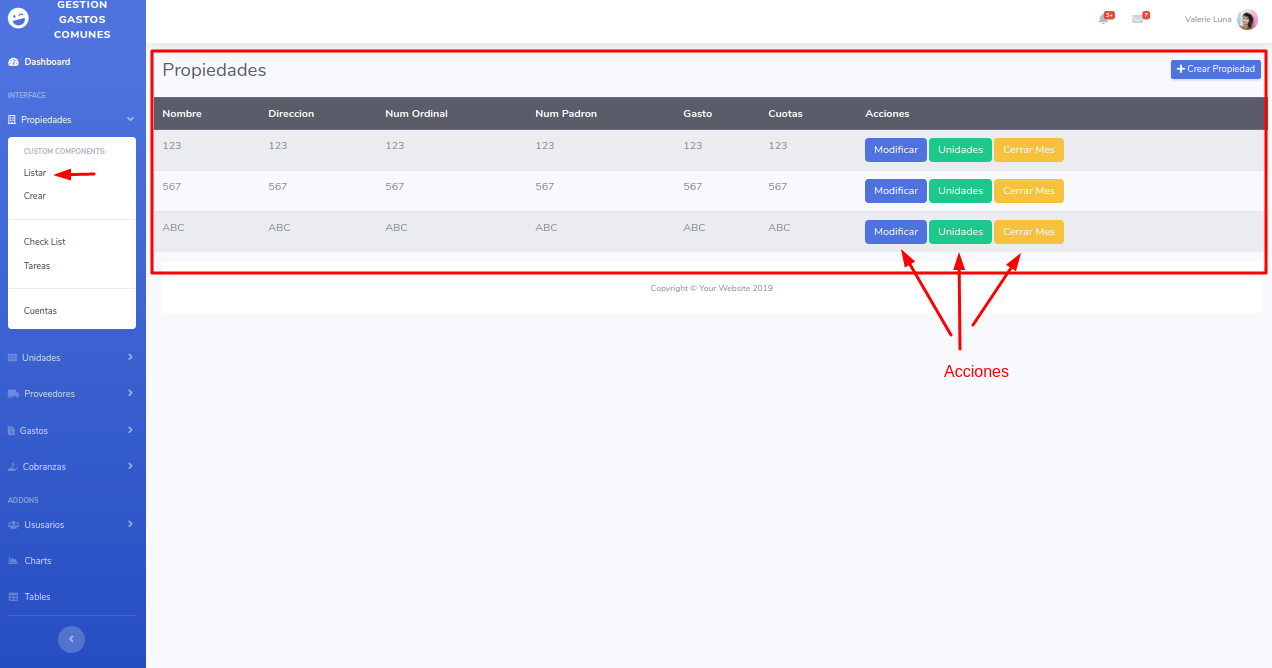


Imagen No 4 – **Menú propiedades – Listado propiedades - Paginado**

# list() → array(obj:propiedad)

|  |  |
| --- | --- |
| **Paso** | **Acción** |
| 1 | Formar el Query haciendo inerJoin con las tablas de los objetos *unidad, gastos, rubro, proveedor, fondos* [5]:Limitar el query en el SGDB al valor parametrizado query\_limit |
| 2 | Retornan un arreglo de objetos *propiedades* . Si el arreglo es vació indica que no hay objetos persistidos en la DB (tomar en cuenta para el render index) |



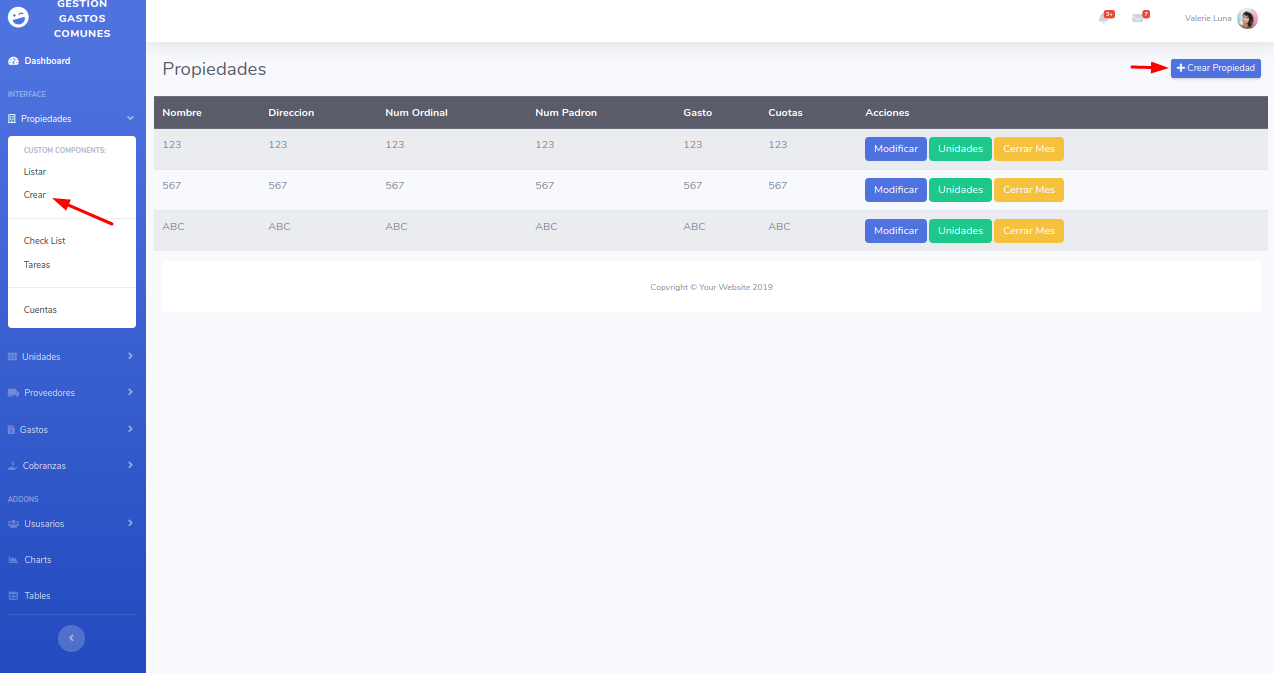
# create()

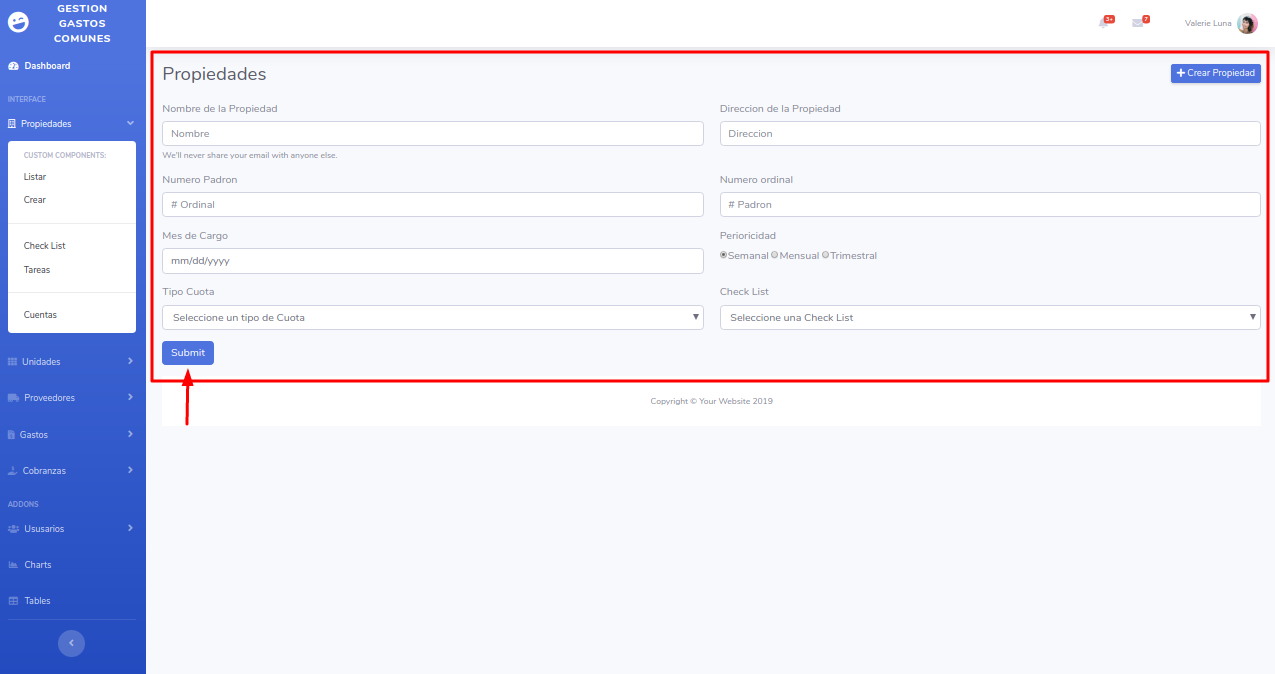
Acción que crea una nueva propiedad

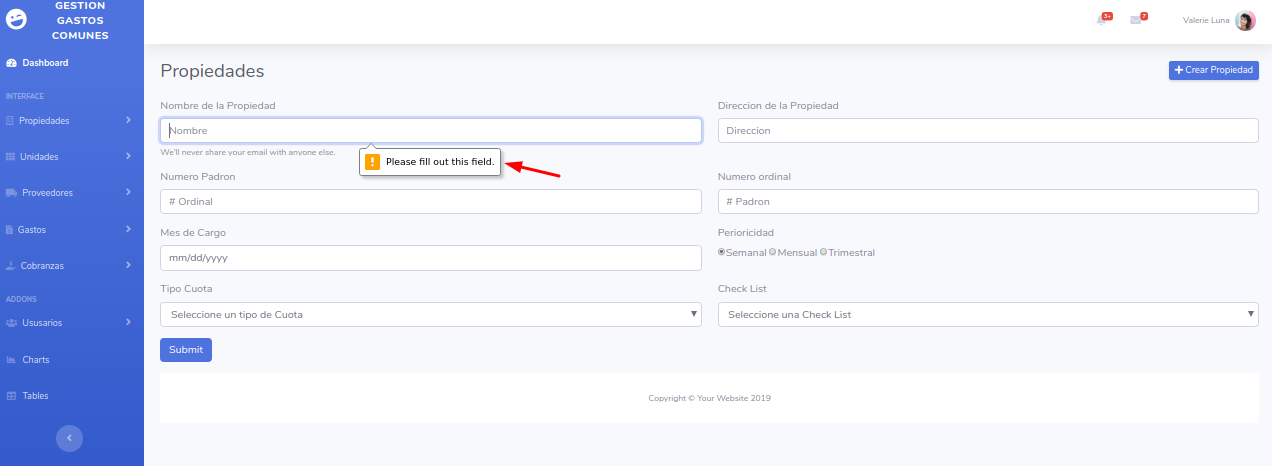
Esta nueva propiedad debe ser única y se usara como clave (index) el atributo numero ordinal

Se generara un formulario para capturar la data de la nueva propiedad. Este formulario sera validad del lado del usuario (frontend: html, jquery) y del lado del servidor (backend: node)

Debe contar con un sistema de gestión de mensajes (error, successfull)

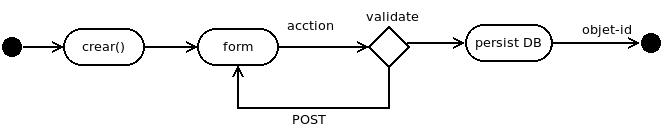
Imagen No 5 – **Crear propiedad**

Imagen No 6 **–**  **Crear propiedad – Formulario**

Imagen No 7 **–**  **Crear propiedad – Manejo de mensajes**

# create() →obj:propiedad

|  |  |
| --- | --- |
| **Paso** | **Acción** |
| 1 | Renderizar formulario con todos los datos de una nueva propiedad |
| 2 | Verificar que el objeto *propiedad* sea único [3]:uniq – SGDB |
| 3 | Validar campos del formulario (request) [4]:requiremt HTML |
| 4 | Persistir data ORM – SGDB |
| 5 | Retornar ID nuevo objeto *propiedad* |



# getByID(id) → obj:propiedad / False

Recupera un objeto propiedad basado en el atributo id (primaryKey : PK) de la base de datos y crea en memoria un objeto de tipo propiedad con los atributos recuperados de la base de datos (ORM)

|  |  |
| --- | --- |
| **Paso** | **Acción** |
| 1 | Obtener el parámetro id via URL (param) |
| 2 | Formar el Query haciendo inerJoin con las tablas de los objetos *unidad, gastos, rubro, proveedor, fondos* |
| 3 | Retornar : objeto si existe la *propiedad* , False en otro caso |



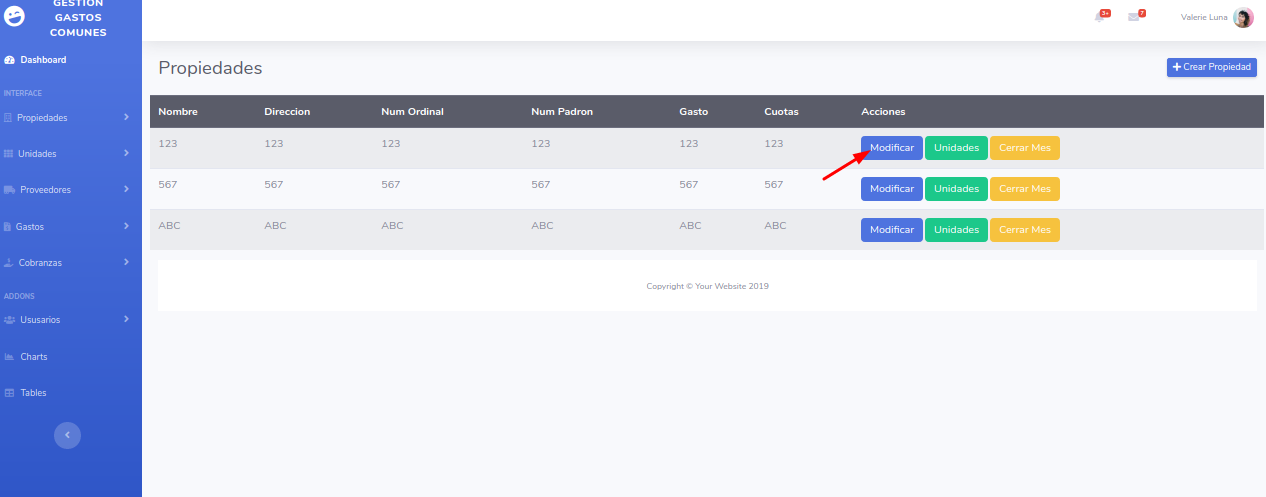
# update(id, data) → obj:propiedad / False

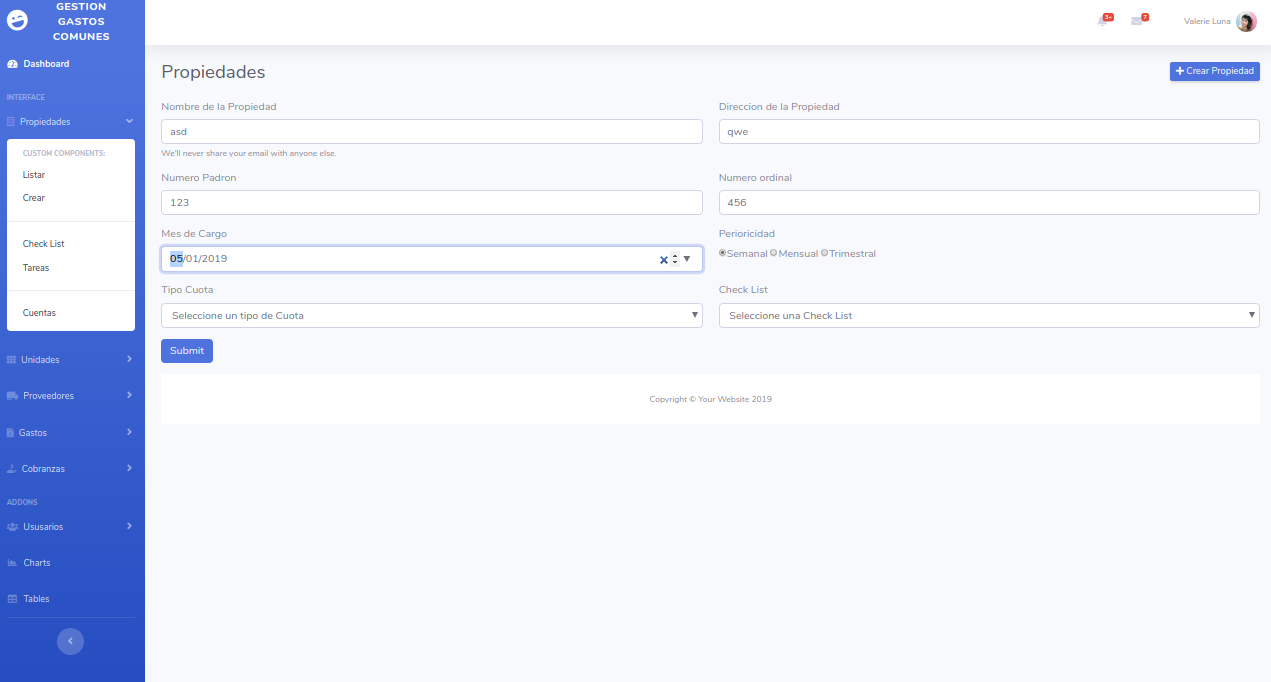
Este método permite modificar un registro de una “*propiedad”* ya creada y almacenada en la base de datos (persistida)

Este método se apoya en el método getByID(id) para recuperar la data que sera expuesta (renderizada) en el formulario

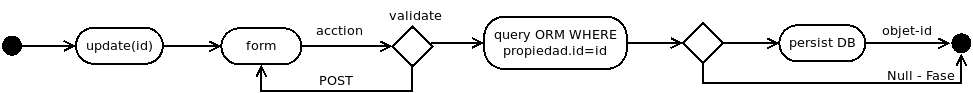
Al igual que el método de create() la data sera validada del lado del usuario (frontend: html, jquery) y del lado del servidor (backend: node)

Tambien deberá contar con un sistema de gestión de mensajes (error, successfull)

Imagen No 8 **–**  Modificar **propiedad**

Imagen No 9 **–**  **Modificar Propiedad - formulario**

|  |  |
| --- | --- |
| **Paso** | **Acción** |
| 1 | Obtener el parámetro id via URL (param) |
| 2 | Formar el Query haciendo inerJoin con las tablas de los objetos *unidad, gastos, rubro, proveedor, fondos* |
| 3 | Si retorna el objeto, renderizar formulario con todos los datos del objeto propiedad. No retorna Null, indicar que el objeto no existe |
| 4 | Validar campos del formulario (request) [4]:requiremt HTML |
| 5 | Actualiza la data ORM para el objeto WHERE (propiedad.id = id) – SGDB |
| 6 | Retornar ID nuevo objeto *propiedad* |



# delete(id)

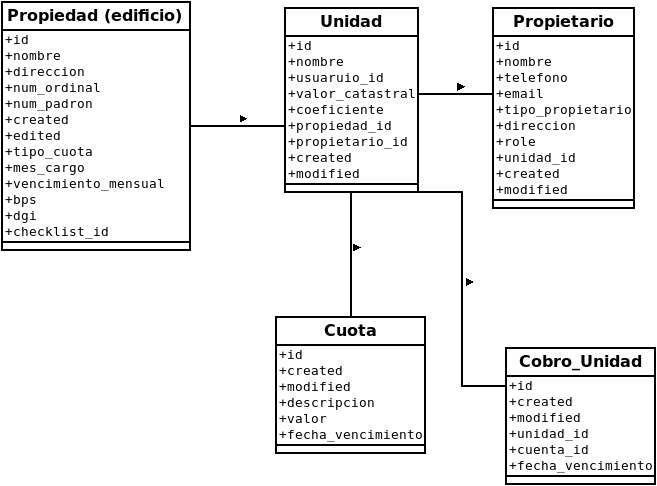
# **\*\*\*\*** Waiting for Referential Integrity

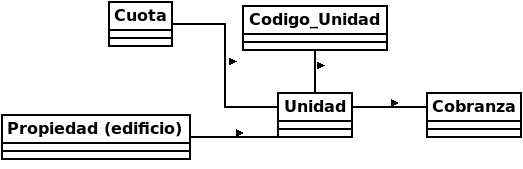
## UNIDAD

Una “*unidad”* es una clase asociada de la clase “*propiedad*” en una relación múltiple (OneToMany 1:N)

La clase *unidad* esta asociada con la clase *propietario* en una relación simple (OneToOne 1:1)

La clase *unidad* esta asociada con la clase *Cuota* en una relación múltiple (ManyToMany N:N) y a través de esta relación se logra establecer la relación con la clase *cuenta*

****



# UNIDAD – Atributos y Métodos

| **Atributos** | **Tipo** | **Parámetros** | | **Retorno** |
| --- | --- | --- | --- | --- |
| nombre | VarChar (45) |  | | |
| direccion | Text |  | | |
| usuario\_id | Integer(11):FK-constrain |  | | |
| valor\_catastral | Real |  | | |
| coeficiente | Real |  | | |
| propiedad\_id | Integer(11):FK-constrain | Asocia una *unidad* con una *propiedad* (1:N) | | |
| propietario\_id | Integer(11):FK-constrain | Asocia una *unidad* con un *propietario* (1:N) | | |
| codigo\_unidad | Integer(11):FK-constrain | Asocia una *unidad* con un *tipoUnidad* (1:N) | | |
| cuota\_id | Integer(11):FK-constrain | Asocia una *unidad* con una *cuota* (N:N) | | |
|  |  |  | | |
| **Método** | **Descripción** | **Parámetros** | **Retorno** | |
| list()  [1]:El método puede implementar un Limit para paginacion (10) – depende del SGBD | Lista todos los objetos de tipo *unidad* | N/A | arreglo(unidad) | |
| crear(propiedad\_id, propietario\_id, tipoUnidad\_id) | Crea un nuevo objeto tipo *unidad* y persiste sus atributos en la DB [6]:cascade on create (propiedad, propietario, tipoUnidad). [7]: la relación con cuota es dinámica, | propiedad\_id – objeto *propiedad* | id – identificador del nuevo objeto persistido | |
| propietario\_id – objeto *propietario* | False – fallo inserción en DB | |
| tipoUnidad\_id – objeto *tipoUnidad* |
| getByID(id) | Recupera un objeto de tipo *unidad* | id-identificador del objeto *unidad* | Objeto(*unidad*) | |
| False - objeto no existe | |
| update(id,data:unidad(atributo=>valor)) | Actualiza el objeto *unidad* identificado con el id actualizando los atributos | id-identificador del objeto *unidad* | True – Actualizo (persistió) el objeto | |
| Data-arreglo (nombre\_atributo => valor\_atributo) | False – No se pudo realizar la operación de actualización | |
| delete(id) | Borra el objeto *unidad* identificado con el id  [2]:Tomar en cuenta las relaciones de cascada en el SGDB | id-identificador del objeto *unidad* | True – Borrado el objeto | |
| False – No se pudo realizar la operación de borrado | |

# UNIDAD – Métodos detalle

# list()

Lista todos los objetos *“unidad”* asociado al objeto *propiedad* seleccionada que se encuentran almacenados en la base de datos (persistidos).

Este listado sera mostrado de forma ascendente según el total de la deuda que las “unidad*”* registren en las “*cuentas”* asociados a dicha *propiedad*.

El listado sera paginado, este parámetro de la longitud de elementos a mostrar por pagina sera configurado en el maestro el sistema

Cada *unidad* listada dispondrá de botones de acciones “Modificar”, “Detalles” y “Cuentas”

* Modificar : Recupera los datos de la *unidad* y genera un formulario que permite modificar dichos datos (Tener en cuenta que algunos datos serán mostrados pero no podrán ser modificados – disabled)
* Detalles : Muestra detalle de la “*unidade*” así como las deudas registradas en las cuentas.
* Cuentas : Muestra el detalle de las *cuentas* asociada a dicha *unidad* en el periodo de tiempo en curso

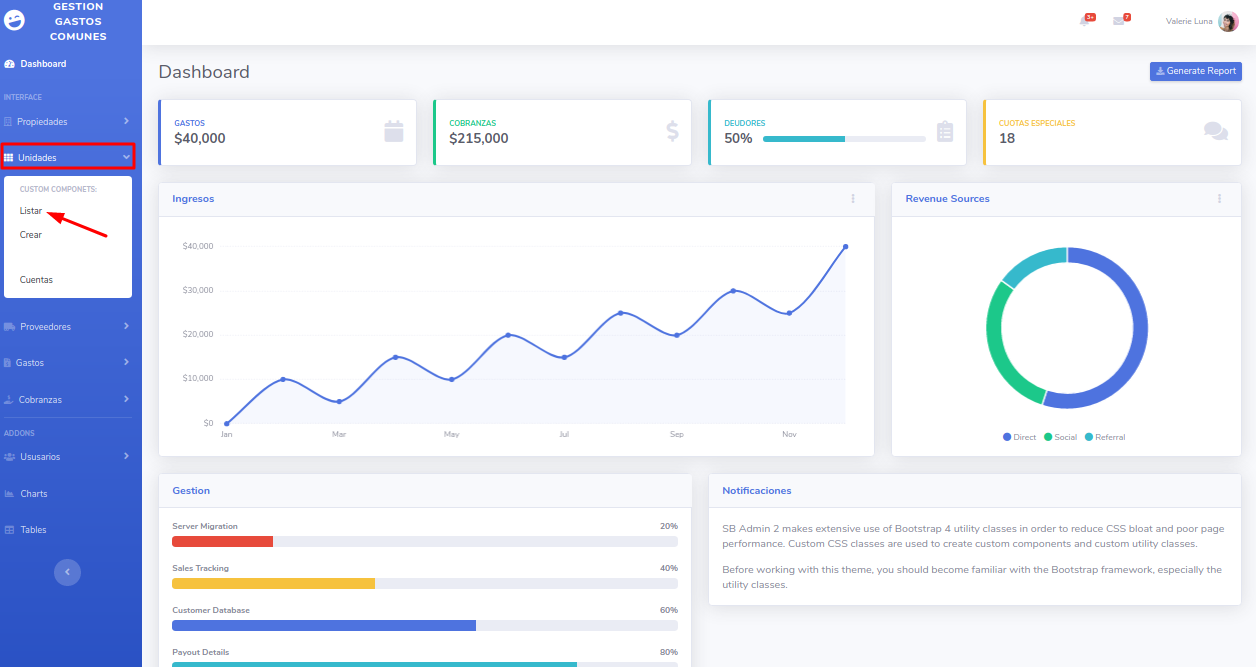
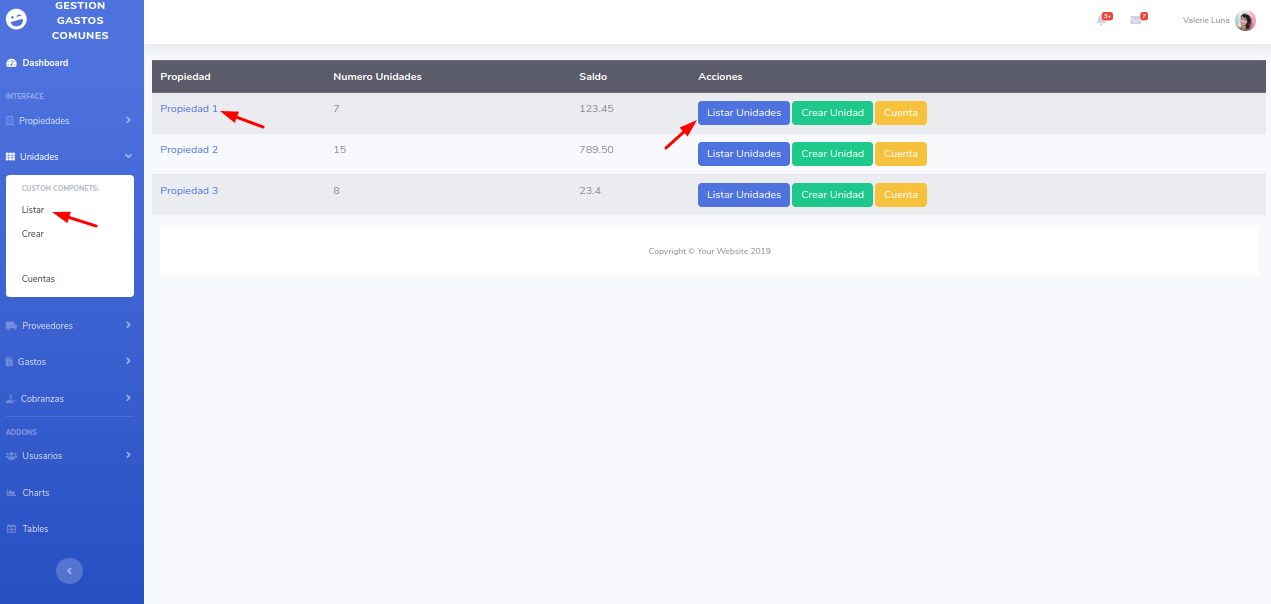


Imagen No 10 **–**   **Listado de todas las unidades**

Imagen No 11 **–**   **Listado de todas las unidades**

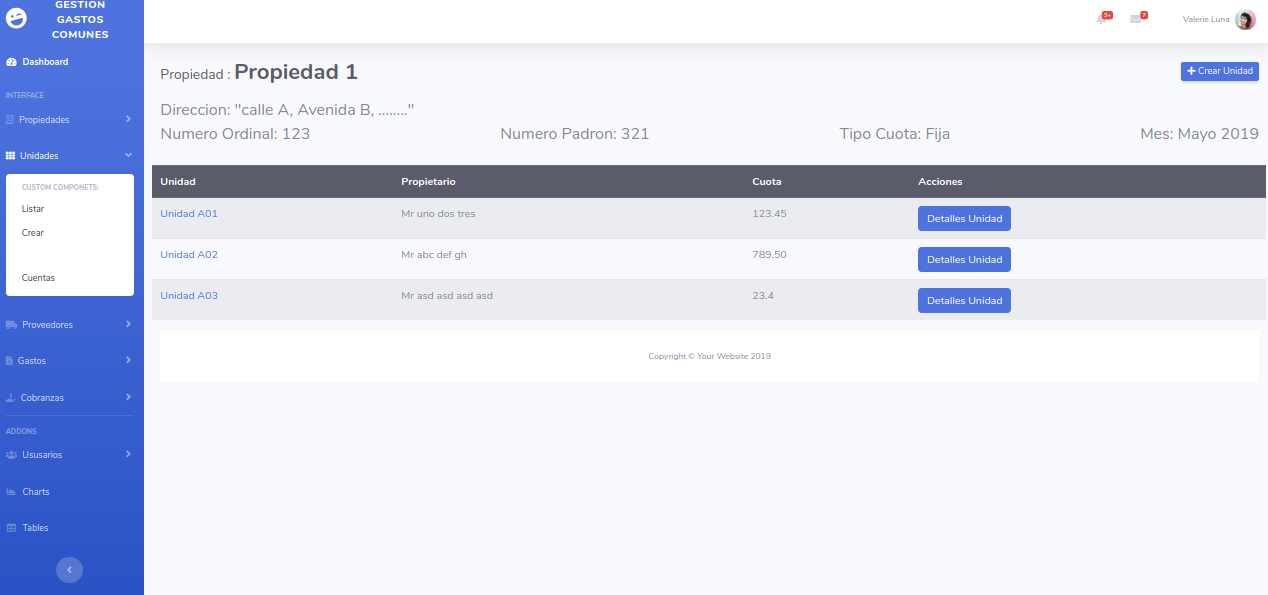


Imagen No 12 **–**   **Listado unidad – Detalle Propiedad**

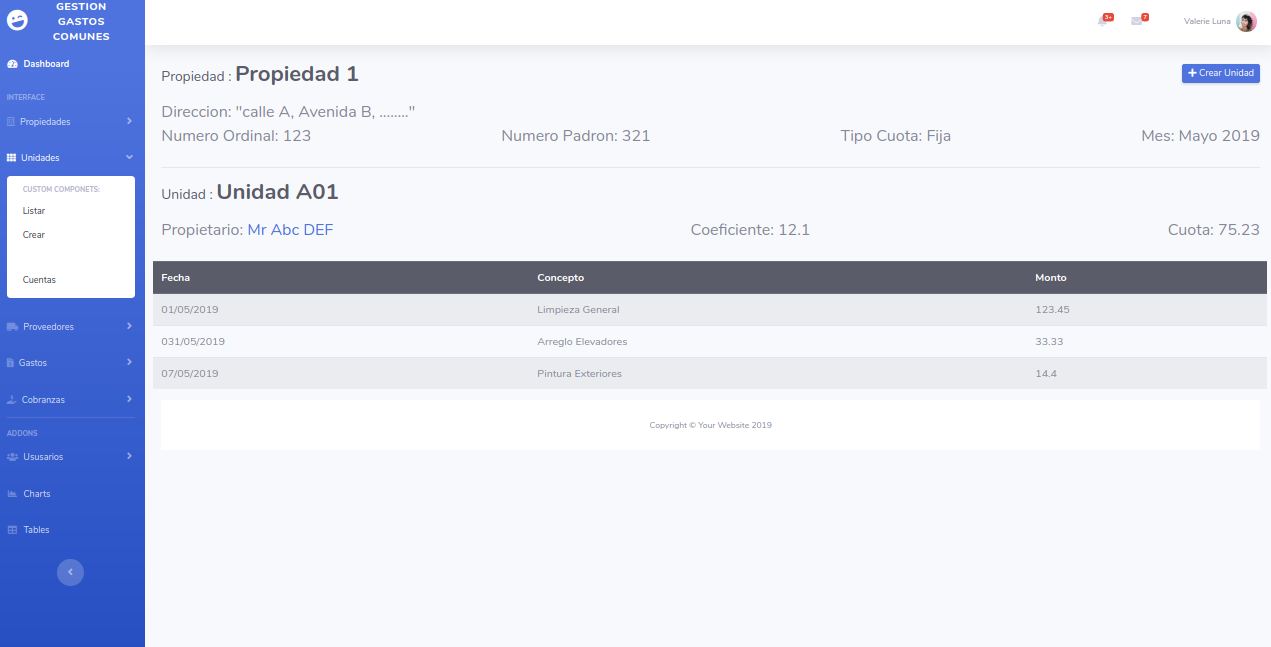


Imagen No 13 **–**   **Listado unidad – Detalle Unidad**

# list() → array(obj:unidad)

|  |  |
| --- | --- |
| **Paso** | **Acción** |
| 1 | Formar el Query haciendo inerJoin con las tablas de los objetos propiedad*, cuota, codigoUnidad, cobranza* [5]:Limitar el query en el SGDB al valor parametrizado query\_limit |
| 2 | Retornan un arreglo de objetos *unidad* . Si el arreglo es vació indica que no hay objetos persistidos en la DB (tomar en cuenta para el render index) |



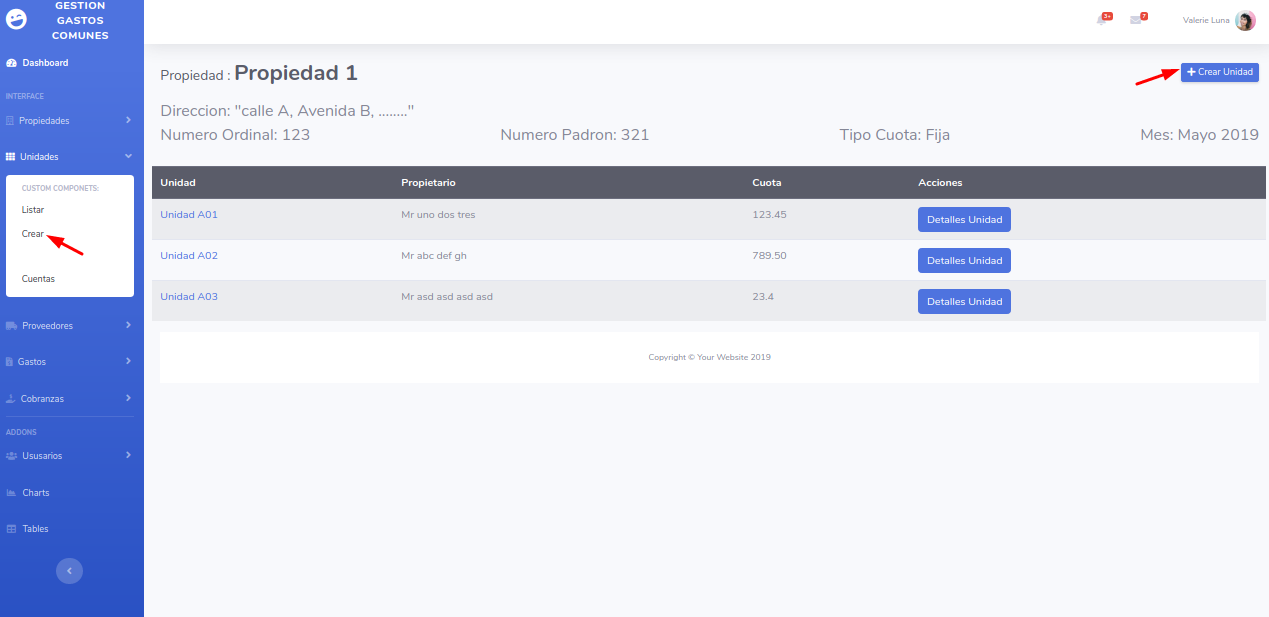
# crear() →obj:unidad : Debe existir un objeto de cada clase *propiedad*, *propietario* y *codigoUnidad.* Si no existen dichos objetos previos (operaciones de cascade en el gestor de base de datos) se debe redirigir el formulario a la creación de dichos objetos y finalmente retornar a la creación del objeto *unidad*

Acción que crea una nueva *unidad* para una *propiedad* seleccionada

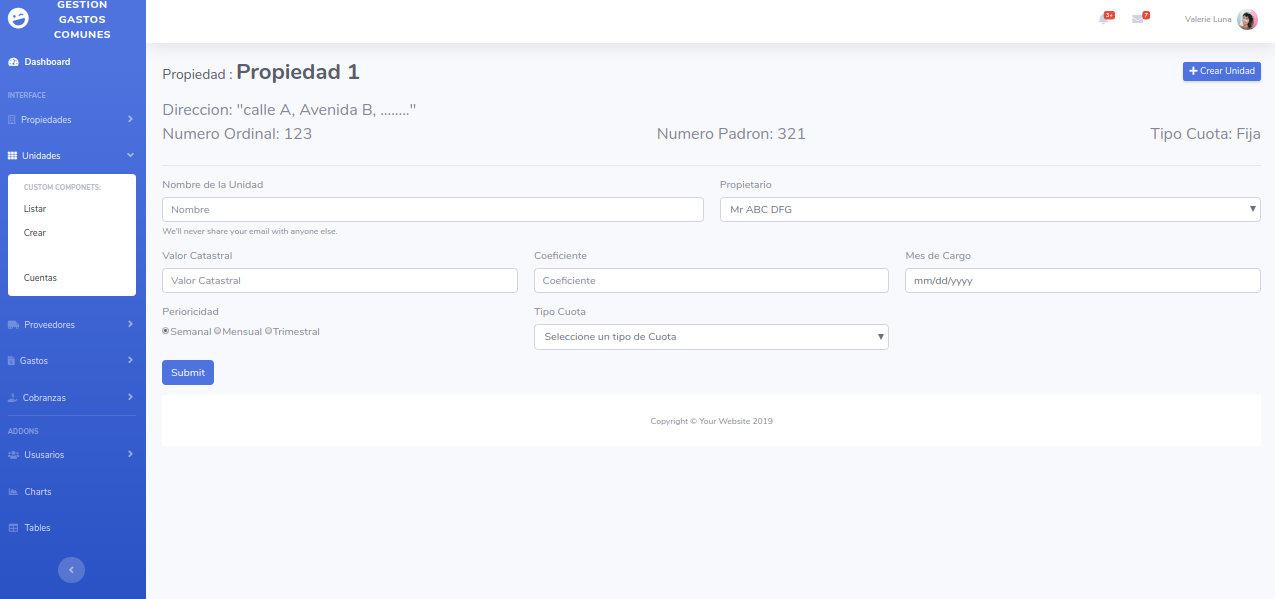
Esta nueva *unidad* debe ser única y se usara como clave (index) el atributo propiedad\_id

Se generara un formulario para capturar la data de la nueva *unidad* asociada a la *propiedad*, por lo que previamente debe ser recuperada la información de la *propiedad* de la base de datos e instanciada en un objeto de tipo *propiedad*. Este formulario sera validad del lado del usuario (frontend: html, jquery) y del lado del servidor (backend: node)

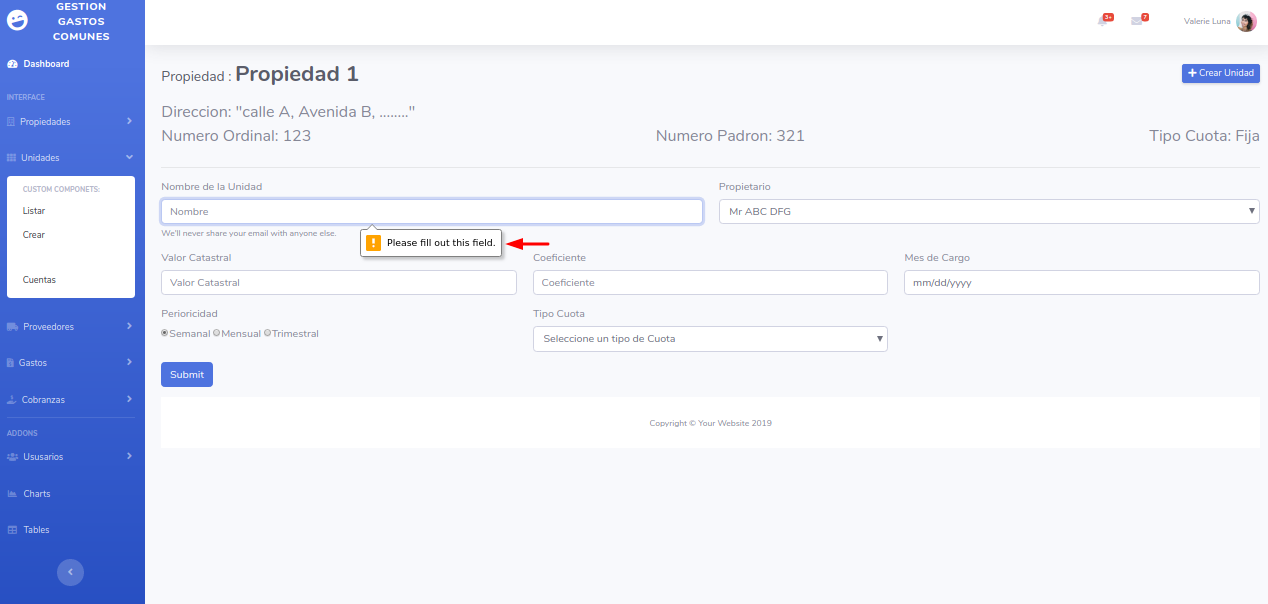
Debe contar con un sistema de gestión de mensajes (error, successfull)

**

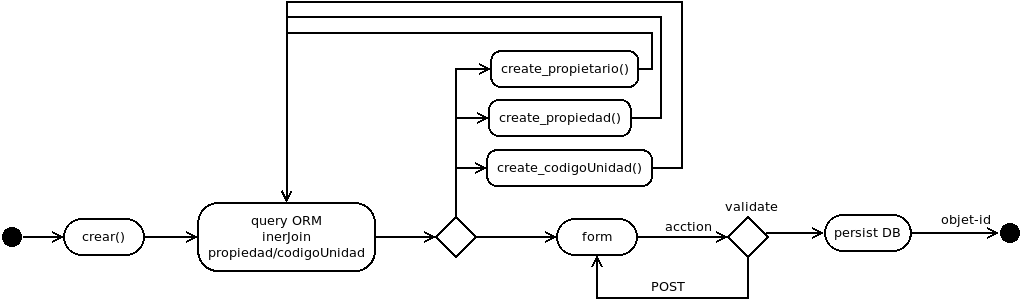
*Imagen No 14* ***–***  ***Crear Unidad***

**

*Imagen No 15* ***–***  ***Crear Unidad – Formulario***

*Imagen No 16* ***–***  ***Crear Unidad – Formulario - Manejo de Mensajes***

|  |  |
| --- | --- |
| **Paso** | **Acción** |
| 1 | Formar el Query haciendo inerJoin con las tablas de los objetos *propiedad, cobranza, coidgoUnidad, cuota* [8]: Si el inerJoin de *propiedad, propietario* y *codigoUnidad* son vacíos, previamente se debe invocar la creación de dichos objetos. |
| 2 | Renderizar formulario con todos los datos de una nueva *unidad* y los datos pertinentes (id, name) de los objetos *propiedad*, propietario y *codigoUnidad* a fin de tener disponible el widget select |
| 3 | Verificar que el objeto *unidad* sea único en la relación (foreing key) con *propiedad* [9]:uniq – SGDB |
| 4 | Validar campos del formulario (request) [10]:requiremt HTML |
| 5 | Persistir data ORM – SGDB |
| 6 | Retornar ID nuevo objeto *unidad* |



# getByID(id) → obj:unidad / False

Recupera un objeto *unidad* basado en el atributo unidad\_id (primaryKey : PK) y el objeto *propiedad* que lo relaciona (InerJoin) de la base de datos y gestiona en memoria el query compuesto de tipo *unidad*->*propiedad* con los atributos recuperados de la base de datos (ORM)

|  |  |
| --- | --- |
| **Paso** | **Acción** |
| 1 | Obtener el parámetro id via URL (param) |
| 2 | Formar el Query haciendo inerJoin con las tablas de los objetos  *cuota, codigoUnidad, cobranza* |
| 3 | Retornar : objeto si existe la *unidad* , False en otro caso |

****

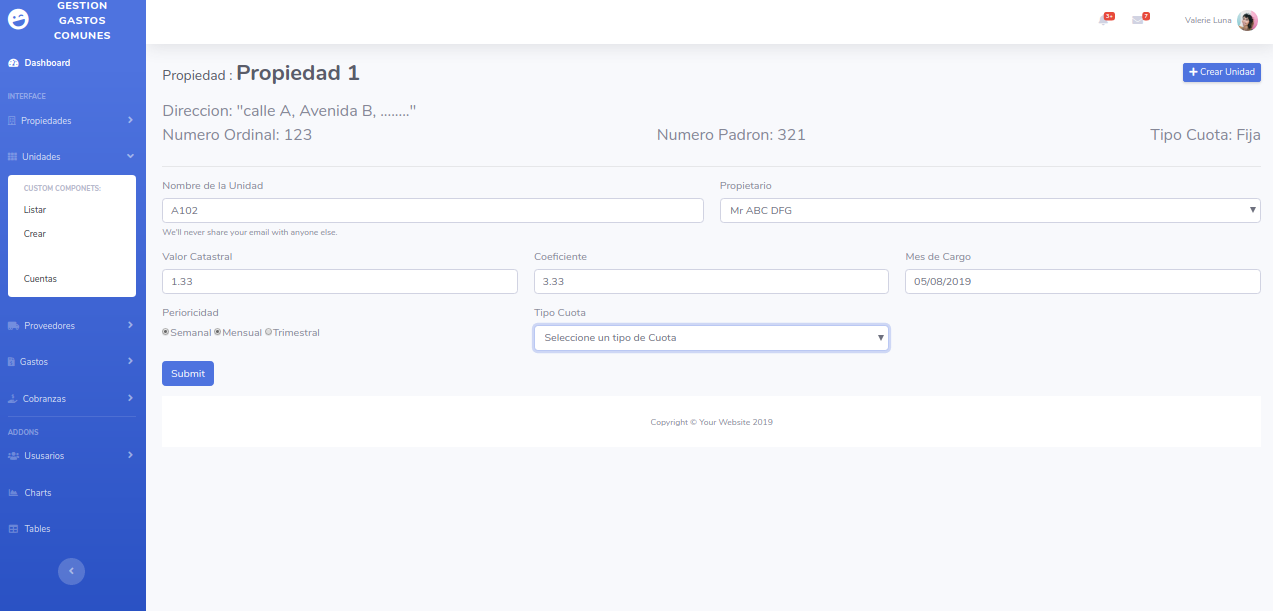
# update(id, data) → obj:unidad / False : La actualización de la unidad implica que se pueda actualizar los valores de las relaciones (*cobranza, unidadTipo, propiedad, cuota)* aquí se debe tomar en cuenta las relaciones de dependencia (cascade) especificadas en el diagrama principal

Este método permite modificar un registro de una *unidad* ya creada y almacenada en la base de datos (persistida) así como su relación la el registro del objeto *propiedad* con el que esta relacionado

Este método se apoya en el método getByID(id) para recuperar la data que sera expuesta (renderizada) en el formulario

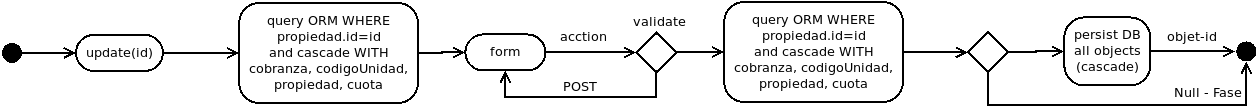
Al igual que el método de create() la data sera validada del lado del usuario (frontend: html, jquery) y del lado del servidor (backend: node)

También deberá contar con un sistema de gestión de mensajes (error, successfull)



*Imagen No 17* ***–***  ***Modificar Unidad – Formulario - Manejo de Mensajes***

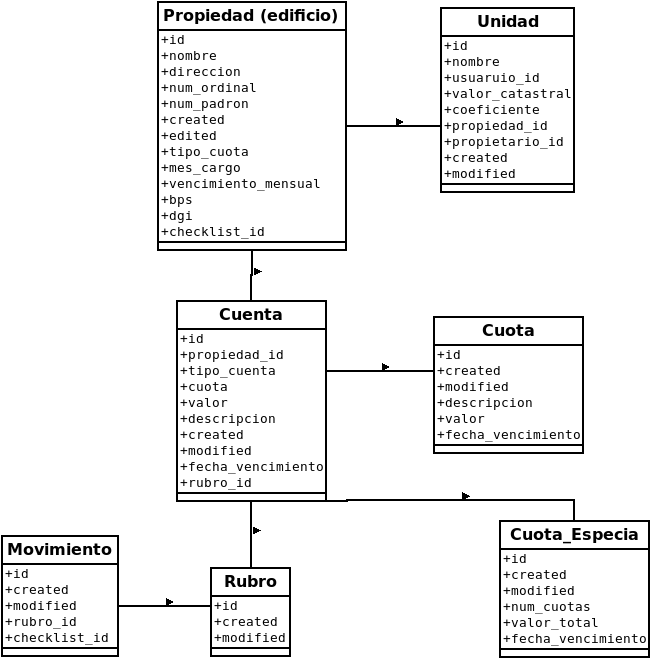
|  |  |
| --- | --- |
| **Paso** | **Acción** |
| 1 | Obtener el parámetro id via URL (param) |
| 2 | Formar el Query haciendo inerJoin con las tablas de los objetos  *cobranza, unidadTipo, propiedad, cuota* |
| 3 | Si retorna la colección de objetos, renderizar formulario con todos los datos de la colección de cada objeto *unidad*, *cobranza, unidadTipo, propiedad, cuota*. Si retorna Null, indicar que el objeto no existe |
| 4 | Validar campos del formulario (request) [4]:requiremt HTML |
| 5 | Actualiza la data ORM para el objeto WHERE (unidad.id = id) y los otros objetos de la relacion (nuevamente tomar en cuenta las relaciones de dependencia : cascade) – SGDB |
| 6 | Retornar ID nuevo objeto *unidad* |

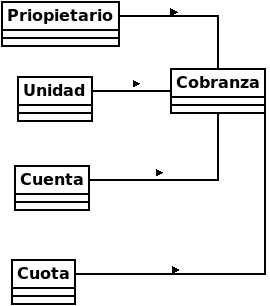


# delete(id)

# **\*\*\*\*** Waiting for Referential Integrity

## **MOVIMIENTOS**





# MOVIMIENTOS **-** Atributos y Métodos

| **Atributos** | **Tipo** | **Parámetros** | | **Retorno** |
| --- | --- | --- | --- | --- |
| propiedad\_id | Integer(11):FK-constrain | Asocia una *cuenta* con una *propiedad* (1:N) | | |
| unidad\_id | Integer(11):FK-constrain | Asocia una *cuenta* con un *unidad* (1:N) | | |
| tipo\_cuenta | Integer(11):FK-constrain | Asocia una *cuenta* con un *tipo\_cuenta* (1:N) | | |
| cuota | Float (real) |  | | |
| valor | Float (real) |  | | |
| descripción | Text (255) |  | | |
| fecha\_vencimiento | TimeStamp |  | | |
|  |  |  | | |
| **Método** | **Descripción** | **Parámetros** | **Retorno** | |
| crear(unidad\_id, cuenta\_id) | Crea un nuevo objeto tipo *cuenta* y persiste sus atributos en la DB [6]:cascade on create (unidad, cuenta, propietario, cuota). | unidad\_id – objeto *unidad* | id – identificador del nuevo objeto persistido | |
| cuenta\_id – objeto *cuenta* | False – fallo inserción en DB | |
| propietario\_id – objeto *propietario \*\** atributo recuperado por la relación (cascade) |
| cuota\_id – objeto *cuota \*\** atributo recuperado por la relación (cascade) |
| list()  [1]:El método puede implementar un Limit para paginacion (10) – depende del SGBD | Lista todos los objetos de tipo *cuenta* | N/A | arreglo(cuenta) | |
| getByID(id) | Recupera un objeto de tipo *cuenta* | id-identificador del objeto *cuenta* | Objeto(*cuenta*) | |
| False - objeto no existe | |
| update(id,data:cuena(atributo=>valor)) | Actualiza el objeto *cuena* identificado con el id actualizando los atributos | id-identificador del objeto *cuenta* | True – Actualizo (persistió) el objeto | |
| Data-arreglo (nombre\_atributo => valor\_atributo) | False – No se pudo realizar la operación de actualización | |
| delete(id) | Borra el objeto *cuenta* identificado con el id  [2]:Tomar en cuenta las relaciones de cascada en el SGDB | id-identificador del objeto *cuenta* | True – Borrado el objeto | |
| False – No se pudo realizar la operación de borrado | |

# MOVIMIENTOS **-** Métodos detalle de la Ingeniería