

ACOES

SISTEMAS DE INFORMACIÓN PARA INTERNET

PROYECTO FINAL ACOES

ACOES

Contenido

[INTRODUCCIÓN. 3](#_Toc10312368)

[Cambios respecto a las entregas anteriores. 3](#_Toc10312369)

[Modificación de los requisitos: 3](#_Toc10312370)

[Modificación de las entidades JPA. 3](#_Toc10312371)

[Al igual que como nos pasaba en la entidad de usuario, hemos tenido que crear otro Enumerado llamado Estado para poder controlar cómo está actualmente. 5](#_Toc10312372)

[Modificación de las vistas XHTML 5](#_Toc10312373)

[Modificación de los controladores de las vistas 7](#_Toc10312374)

[Descripción de los EJB. 9](#_Toc10312375)

#### MIEMBROS DEL GRUPO

Andrés Valentín Suárez Mediavilla - 77425032V

Sergio González Sicilia - 77225653W

Ignacio Pascual Gutiérrez - 54236255R

Katia Moreno Berrocal - 79038916Z

**URL Repositorio (los archivos finales se encuentran en la carpeta: Tarea 3):**

<https://github.com/Lilithsoul/ACOES>

**Commit final del proyecto:**

<https://github.com/Lilithsoul/ACOES/commit/b4ef0d0bab9c65d01e64a93e66fc87a22b964e9e>

Dicho commit es la última modificación del proyecto, la carpeta del proyecto ACOES también se encuentra dentro de la tarea 3, llamada ACOES.

# INTRODUCCIÓN.

Este documento está elaborado con el propósito de actualizar y presentar toda la información referente al proyecto ACOES, incluyendo modificaciones respecto a las tareas anteriores.

Dichas modificaciones son respecto a la anterior entrega, situada en Tarea 2.

# Cambios respecto a las entregas anteriores.

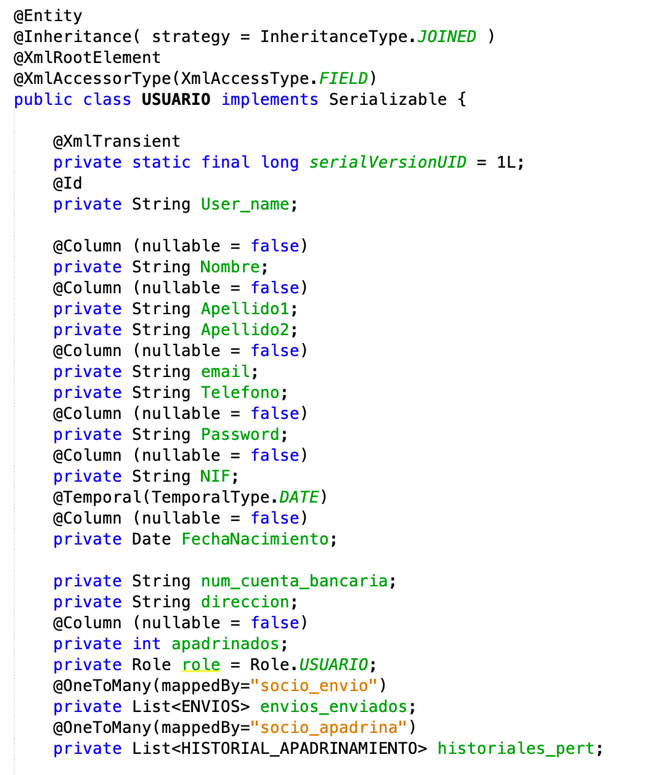
## Modificación de los requisitos:

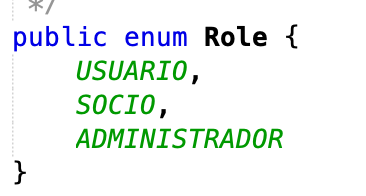
Se encuentran en el DGR dentro de Tarea 3. Commit correspondiente:

https://github.com/Lilithsoul/ACOES/commit/9135560198f78eb1113a8cd8cde7d645e481aaf4

## Modificación de las entidades JPA.

**Entidad: USUARIO**





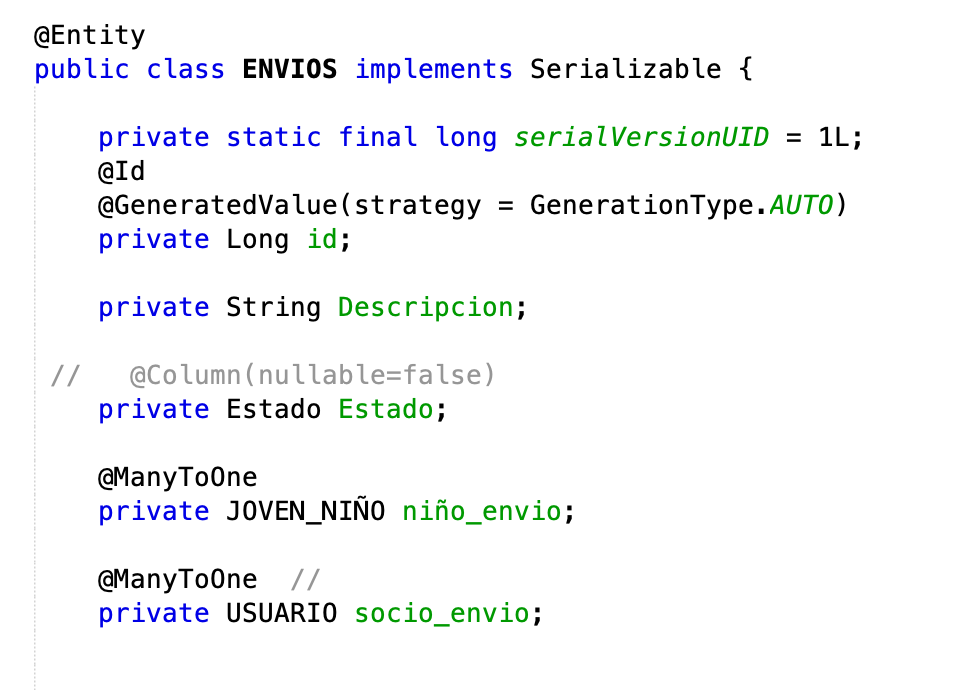
La entidad Usuario ha sido modificada debido a que el grupo ha elegido eliminar la entidad de ADMINISTRADOR y SOCIO. Estas dos últimas entidades extendían de USUARIO.

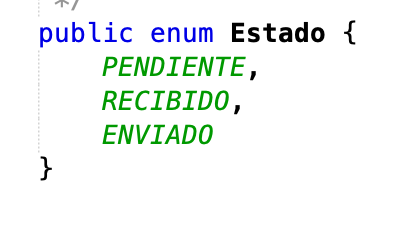
En lugar de usar este método, hemos optado por tener un Enumerado llamado Role compuesta por los siguientes roles: USUARIO, SOCIO y ADMINISTRADOR.

La entidad ADMINISTRADOR tenía información sobre el número de seguridad social, fecha cuando empezó a trabajar y la nómina que percibía. Pensando lo que se pedía en el trabajo, estos campos eran totalmente innecesarios dado que en esta aplicación se nos pedía un gestor de niños, apadrinamiento y escolarización, no una aplicación de recursos humanos.

Estas dos entidades son las que usamos en la entrega anterior y han sido cambiadas.

**Entidad: ENVIOS**





## Al igual que como nos pasaba en la entidad de usuario, hemos tenido que crear otro Enumerado llamado Estado para poder controlar cómo está actualmente.

## Modificación de las vistas XHTML

En esta entrega hemos visto necesario añadir las diferentes vistas:

**Vistas por cada excepción existente.**

Dado que pueden existir distintas excepciones, ej: apadrinar un niño cuando no hay disponibles, error eliminando un ccj, etc. Hemos visto necesario cambiar el mensaje que aparecía en mitad de la pantalla a algo un poco más estético, una vista a la que se te redirige cuando se intenta usar un método y da fallo.

**Vista: error<nombre>.xhtml**

Así mismo, como pueden surgirmás errores como errorApadrinamiento, añadimos las siguientes vistas: errorApadrinamiento.xhtml, errorEliminaciónCCJ.xhtml, errorEliminaciónNinio.xhtml y errorEliminaciónPopulorum.xhtml.

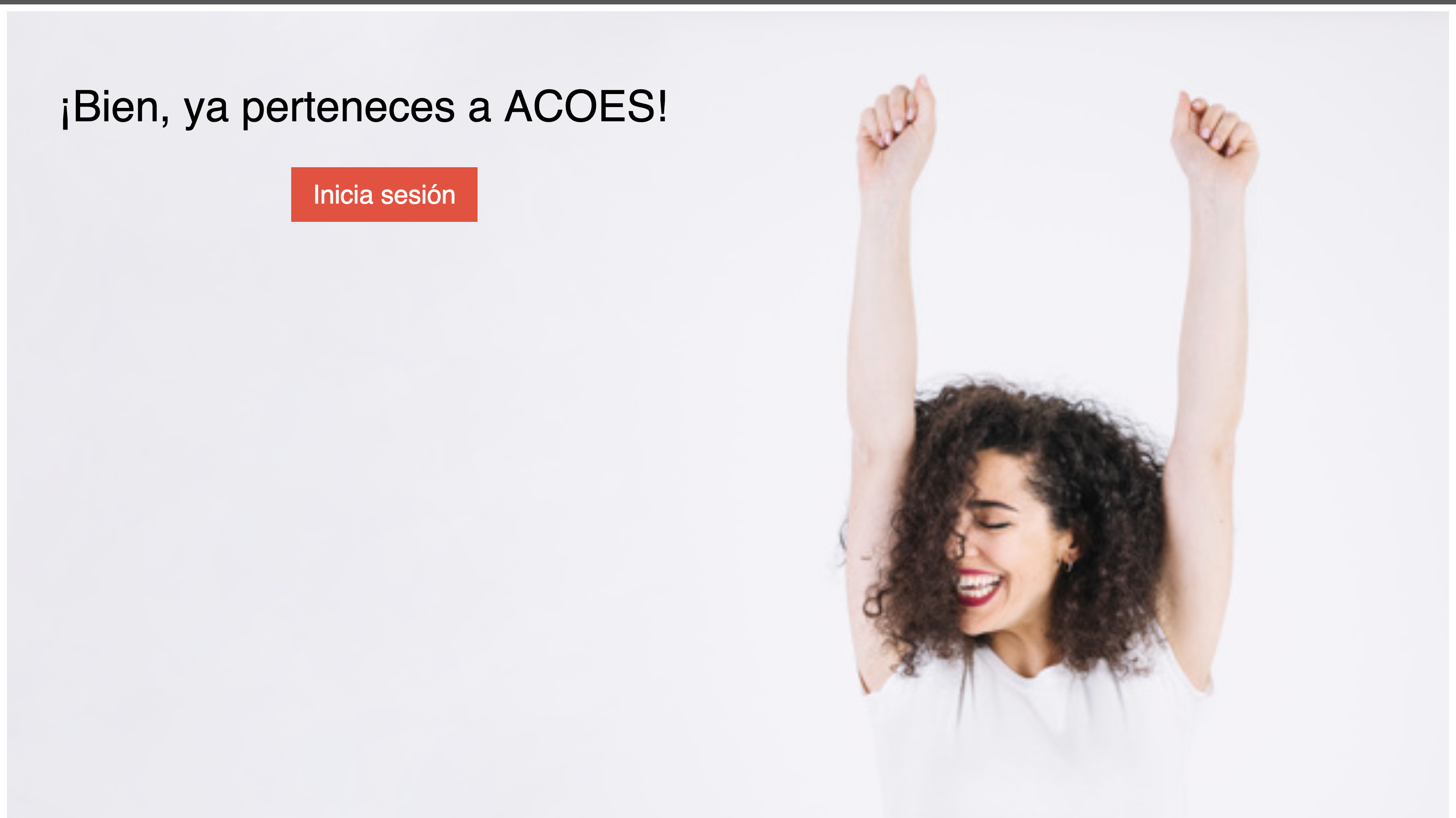
Vista: error errorApadrinamiento.xhtml.

**Ninios.xhtml**

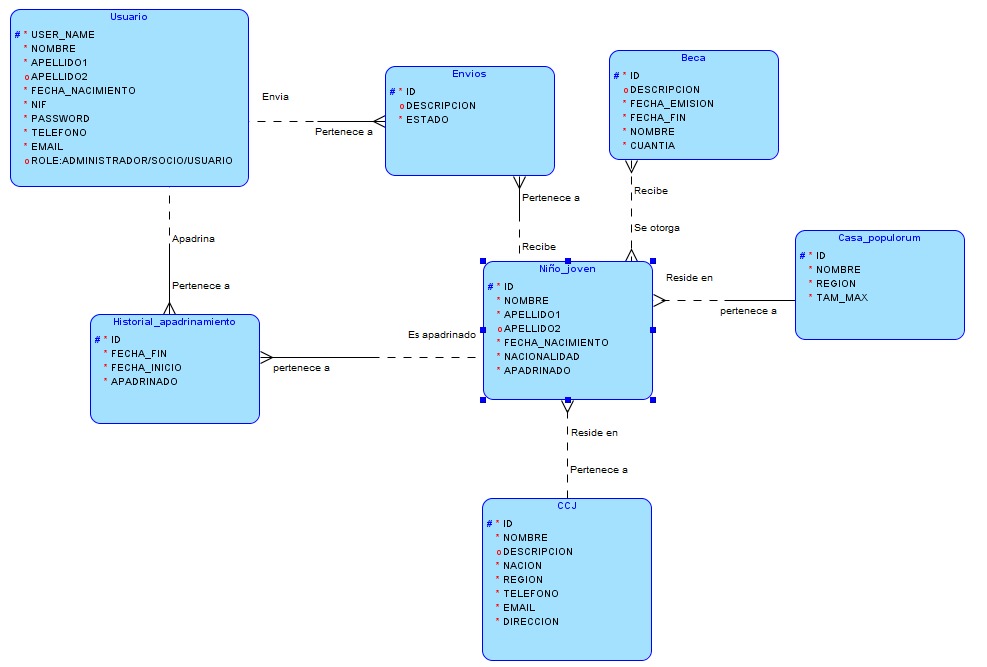
Viendo que era un caos el uso de varias tablas dentro de una misma, decidimos que en la vista ninios.xhtml aparezca una información resumida de los niños que están siendo apadrinamos por un socio en ese momento. Además, si dicho socio desea ver más información del niño en cuestión, solamente tendrá que pulsar sobre el botón: detalles. Detalles redirige a una página con la información del niño en cuestión y además podrá realizar el envío desde ahí.

**Vista: exitoRegistro.xhtml**

Añadimos esta vista para que el usuario se cerciore de que ha sido registrado con éxito en la web. Antes redirigía a la página Home-xhtml.

****

El esquema lógico resultante sería:



## Modificación de los controladores de las vistas

Hemos optado por un modelo-vista-controlador. Los controladores de las vistas son las siguientes:

**Becas.java**

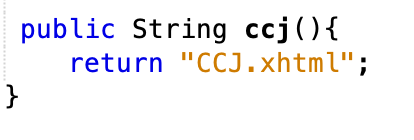
Esta clase contiene todos los controladores relacionado con el manejo de las becas. Sus métodos son: añadir becas, eliminar becas, refrescar la página, ejecutar acción becas (añadir, modificar) para los botones, comprobar si el rol del que intenta acceder una acción es autorizado.

**CCJs.java**

Al igual que la clase anterior, contiene métodos idénticos para los CCJs.

**Control.java**

Su función principal consiste en devolver las páginas que se pidan. Por ejemplo:

En este caso nos devolvería la página de acoes cuando lo llamemos mediante algún action=”…” en la vista pertinente.

**Envios.java**

Contiene todos los métodos necesarios para interactuar con envíos, alguno de los métodos son: eliminar, modificar, insertar envío, refrescar la página (por si añadimos alguno y no aparece), modificar los estados del envío (solo disponibles para administrador), etc.

**Historiales.java**

Se dedica a implementar todos los métodos relacionados con apadrinar, tanto del manejo del historial (lista de apadrinamiento), como la obtención de los padrinos.

**InfoSesion.java**

Su función es devolver los datos que se le piden, como: historiales, usuarios, etc.

**Login.java**

Comprueba que el usuario introducido sea el correcto.

**Ninios.java**

Es idéntica a CCJ y Becas, pero usando métodos especializados a los niños.

**Populorum.java**

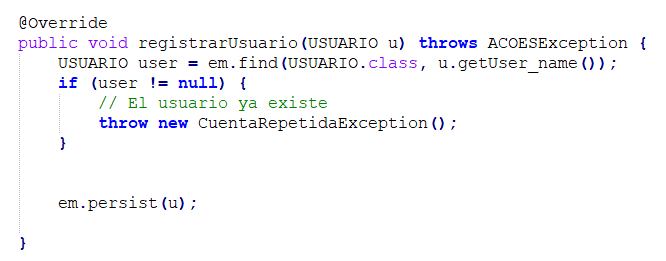
Como lo dicho anteriormente con ninios.java, usa los mismos métodos, pero personalizados en los populorum.

**Registro.java**

Se dedica a comprobar que los campos introducidos sean correctos y completos. Comprueba que las contraseñas coincidan (contraseña y repetir contraseña) y que el usuario no esté registrado ya. El método registerr() añade el usuario a la base de datos. Finalmente registrar socio añade un socio a partir de un usuario cuando se añaden los campos cuenta bancaria y dirección.

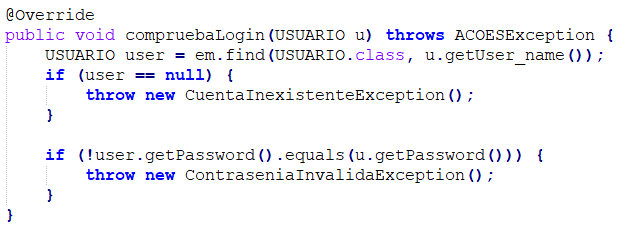
# Descripción de los EJB.

A continuación, vamos a describir los EJB implementados en la capa de negocio, la cual nos sirve para conectar la capa de acceso a datos (JPA) y la capa de presentación (JSF). Para ello tenemos una interfaz Negocio y una clase NegocioImpl que implementa los métodos que se definen en Negocio, los cuáles son:

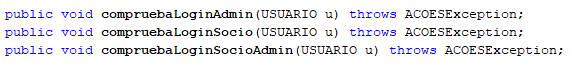


Este método registra un usuario haciendo un em.persist() del usuario pasado por parámetro en el caso de que no exista en la BD. En el caso de que el método em.find() encuentre el Usuario (es decir, que esté registrado) saltará una excepción.

---

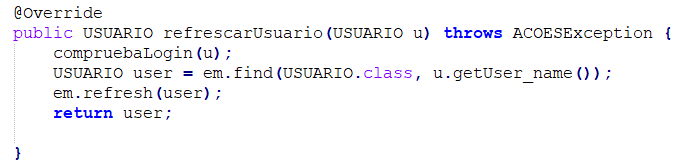


Este método comprueba que el usuario ha introducido sus credenciales correctamente (buscando en la BD al Usuario por su clave primaria (user\_name en este caso)). En el caso de no haberlo encontrado, saltará una excepción. Además, comprueba que la contraseña introducida coincide con la guardada en la BD, en caso contrario, saltará otra excepción. Este método sirve para garantizar la integridad a la capa de negocio, por ello lo llamamos en otros métodos.



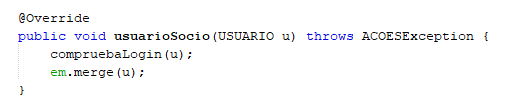
Al igual que compruebaLogin, implementamos estos tres métodos para comprobar cuando un usuario está logeado como Admin, Socio y Admin o Socio respectivamente.

---



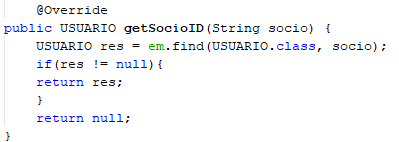
A este método se le pasa por parámetro el usuario de la sesión, comprueba que existe en la BD y le hace un refresh();

---



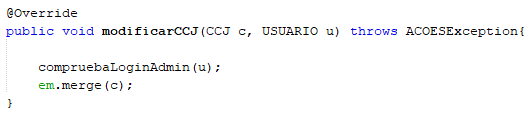
Mediante la variable ‘u’ se llama a compruebaLogin para certificar que el usuario está logeado actualmente en la página. A partir de dicho Usuario se accede a la base de datos y se actualiza el valor del Usuario asociado.

---

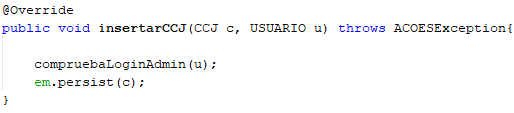


A partir de la clave primaria del SOCIO, en este caso el username de su sesión, se obtiene el SOCIO asociado a dicho username. En caso de no encontrar ningún SOCIO con el username asociado, se devuelve null.

**Métodos asociados a los CCJ:**

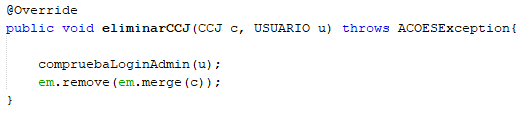


A partir de una variable CCJ se accede a la base de datos y se actualiza el valor del CCJ asociado. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.



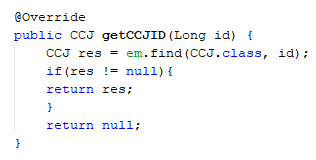
A partir de una variable CCJ se accede a la base de datos y se inserta el valor del CCJ. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

---



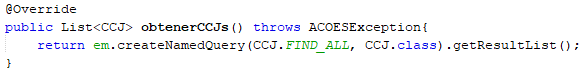
A partir de una variable CCJ se accede a la base de datos y se elimina el valor asociado a este. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

---

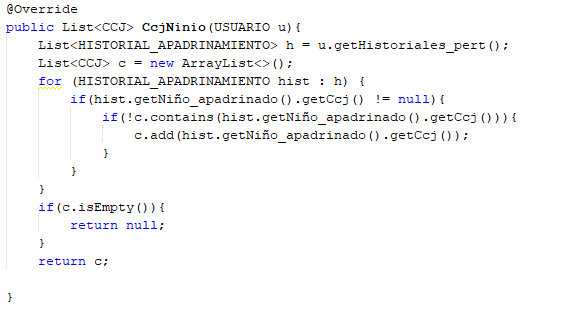


A partir de la clave primaria de CCJ se obtiene el CCJ asociado dentro de la base de datos. Si el valor no se encuentra en la base de datos se devuelve null.

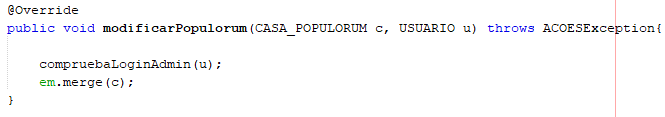
---



Devuelve todos los CCJ almacenados en la base de datos mediante una consulta JPQL. Este método es utilizado para mostrar al administrador todos los CCJ.

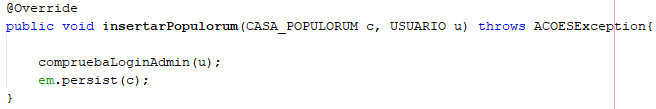
Este método devuelve una lista de CCJ asociadas a los niños apadrinados por el usuario que se le pasa por parámetro, evitando así que tenga acceso a todas las casas ccj existentes y solamente permitiendo su visualización a las que son de su interés. Creamos un arrayList de CCJ c, y para mostrar ccjs sin repeticiones añadimos a la lista solamente los que no estén ya.

**Métodos asociados a las Casas Populorum:**



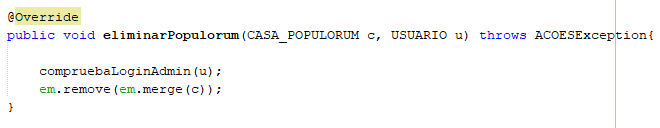
A partir de una variable CASA\_POPULORUM se accede a la base de datos y se actualiza su valor. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

**---**



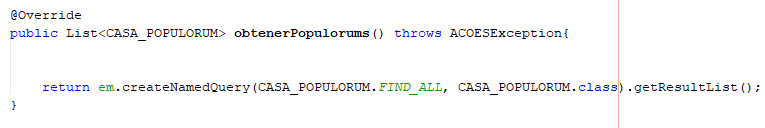
A partir de una variable CASA\_POPULORUM se accede a la base de datos y se inserta el valor de CASA\_POPULORUM. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

---

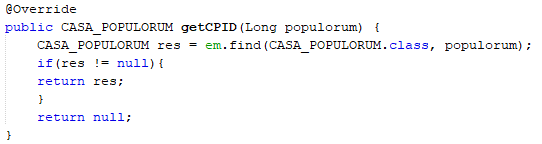


A partir de una variable CASA\_POPULORUM se accede a la base de datos y se elimina el valor de CASA\_POPULORUM. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

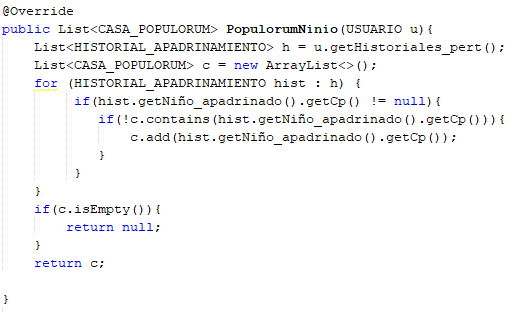
---



Se llama a la base de datos y se obtienen todas las CASAS POPULORUM dentro de ella mediante una consulta JPQL. Este método es utilizado para mostrar al administrador todas las casas populorum.

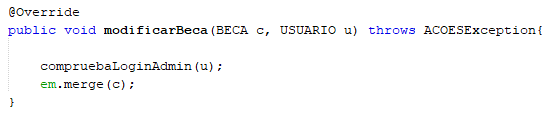


A partir de la clave primaria de CASA\_POPULORUM (id), se accede a la base de datos y se obtiene la CASA\_POPULORUM asociada. Si el valor no se encuentra se devuelve null. Se utiliza en los registros de niños para, a partir del id, obtener la casa populorum asociada.



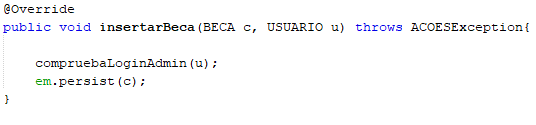
Este método devuelve una lista de CASA\_POPULORUM asociadas a los niños apadrinados por el usuario que se le pasa por parámetro, evitando así que tenga acceso a todas las casas populorum existentes y solamente a las que son de su interés. Creamos un arrayList de CASA\_POPULORUM c, y para mostrar casas populorum sin repeticiones añadimos a la lista solamente los que no estén ya.

**Métodos asociados a las Becas:**

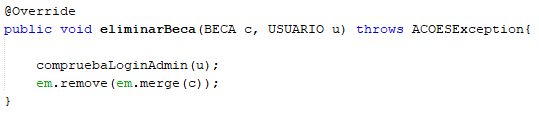


A partir de una variable BECA se accede a la base de datos y se actualiza el valor de BECA asociado. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

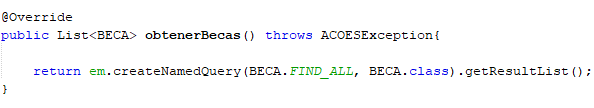
---



A partir de una variable BECA se accede a la base de datos y se inserta el valor de BECA. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

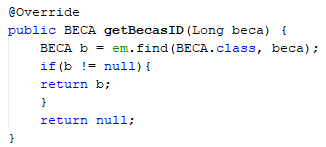


A partir de una variable BECA se accede a la base de datos y se elimina el valor de BECA. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.



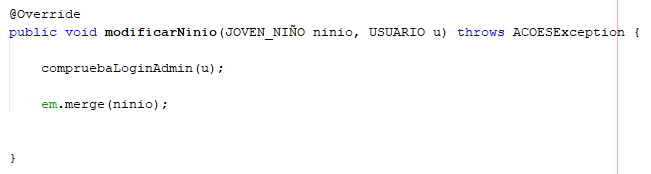
Se llama a la base de datos y se obtienen todas las BECAS mediante una sentencia JPQL. Este método se utiliza para que al administrador se le muestren todas las becas.

---

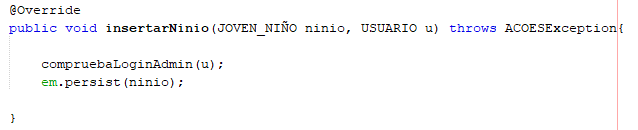


A partir de la clave primaria de BECA (id), se accede a la base de datos y se obtiene la BECA asociada. Si el valor no se encuentra se devuelve null. Se utiliza en los registros de niños para a partir del id obtener la beca.

**Métodos asociados a los Niños o Historiales:**

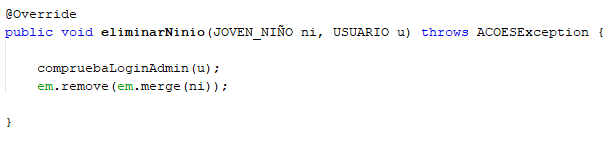


A partir de una variable JOVEN\_NIÑO se accede a la base de datos y se actualiza el valor del JOVEN\_NIÑO asociado. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.



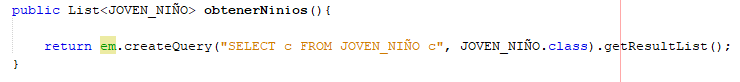
Gracias al persist tomamos la instancia de JOVEN\_NIÑO pasada por parámetro, se añade al contexto de persistencia y hace que la instancia sea manejada.

--



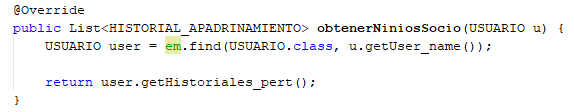
A partir de una variable JOVEN\_NIÑO se accede a la base de datos y se elimina el valor de JOVEN\_NIÑO. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

---



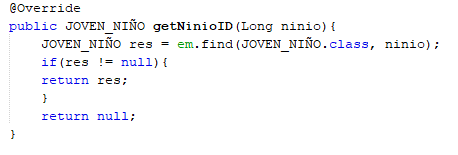
Creamos una Query seleccionando a todos los JOVEN\_NIÑO dentro de la BD y hacemos un return de dicha lista. Dicho método sirve para que el administrador pueda ver a todos los niños en su pestaña correspondiente dentro de la página web y así administrarlos.

---



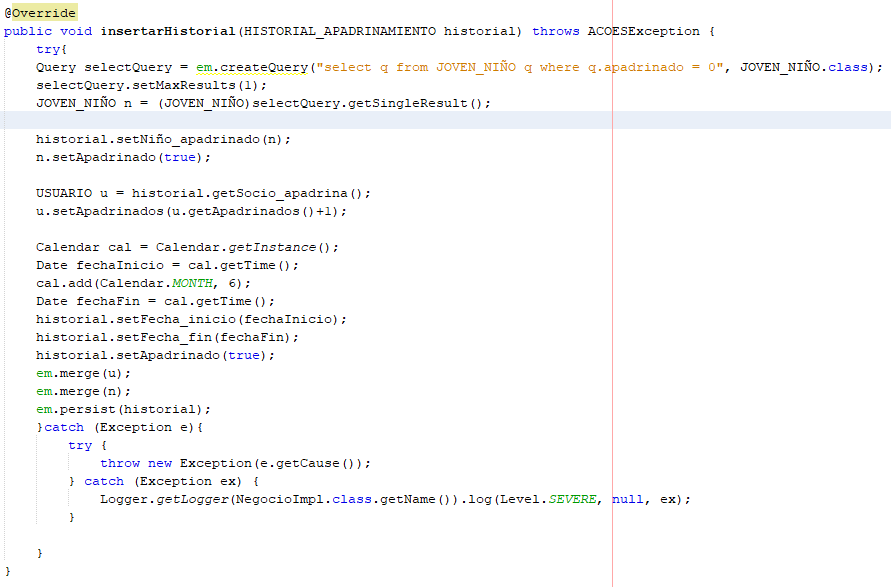
Buscamos al Usuario pasado por parámetro dentro de la BD gracias al método em.find() y devolvemos las filas pertenecientes a su Historial como SOCIO.

Dicho método sirve para que el SOCIO pueda ver a todos los niños apadrinados y no apadrinados en su pestaña correspondiente dentro de la página web y así administrarlos.



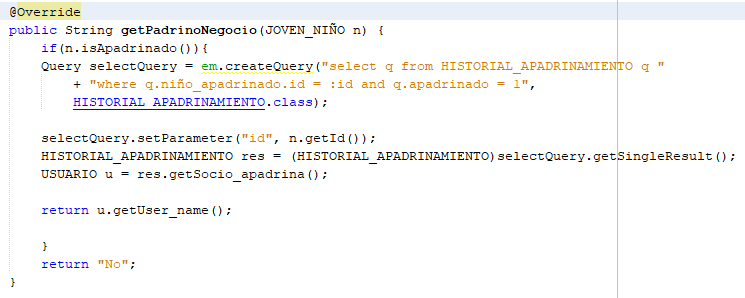
A partir de la clave primaria del JOVEN\_NIÑO, se obtiene el niño asociado a ella. En caso de no encontrar ningún JOVEN\_NIÑO con el id asociado, se devuelve null.

---



Este método es llamado cada vez que un usuario quiera apadrinar a un niño. El método hará una consulta JPQL de los niños que no estén apadrinados, y mediante el uso del método query.setMaxResults(1) obtendremos un único resultado, el cual se almacenará en un nuevo JOVEN\_NIÑO. Se realizan las actualizaciones correspondientes, en el historial se introduce el niño obtenido, el niño se actualiza a ‘apadrinado’ y el número de niños apadrinados del socio que apadrina se incrementará en 1. Aparte, para poder establecer el resto de los componentes de un historial, obtenemos como fecha de inicio de apadrinamiento la del sistema y la fecha fin serán 6 meses más que la fecha de inicio. Se realizan los merge correspondientes al usuario, al historial y al niño.

---

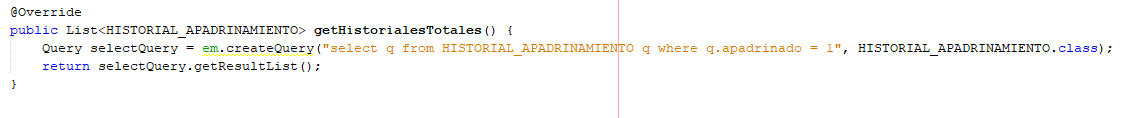


Este método obtendrá el username del padrino asociado a un JOVEN\_NIÑO que se le pasa por parámetro. Primero se comprobará que el niño este apadrinado para poder realizar la consulta JPQL. Nos sirve a la hora de mostrar los niños en el rol ADMINISTRADOR, pues así reflejaremos el padrino del niño en caso de estar apadrinado, y en otro caso se mostrará “No” para indicar que no tiene padrino.

---

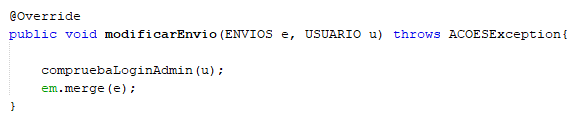


Este método, llamado a través de un botón ‘refrescar apadrinamientos’, se encarga de modificar los parámetros correspondientes cuando un apadrinamiento se ha terminado, es decir, comprobando si la fecha\_fin del apadrinamiento es menor a la del sistema, el niño apadrinado dejará de estarlo, el socio tendrá 1 apadrinado menos y estos cambios se verán reflejados en dicho historial de apadrinamiento.

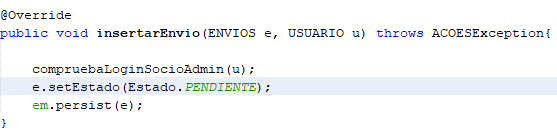


Este método, el cual será llamado desde un rol ‘ADMINISTRADOR’, consulta todos los historiales de apadrinamiento existentes en la BD. Se ha implementado mediante una consulta JPQL.

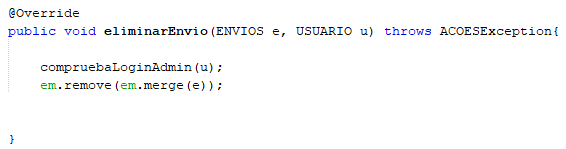
**Métodos asociados a los Envíos:**



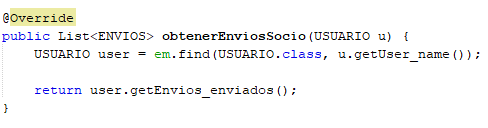
A partir de una variable ENVIOS se accede a la base de datos y se actualiza el valor de ENVIOS asociado. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.



A partir de una variable ENVIOS se le fija el valor de estado de forma predeterminada y se añade a la base de datos. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.

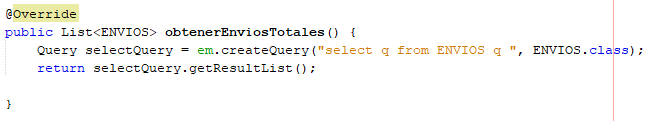


A partir de una variable ENVIO se accede a la base de datos y se elimina el valor de ENVIO. Mediante la variable ‘u’ se llama a compruebaLoginAdmin para certificar que el usuario que esta interactuando posee el rol ADMINISTRADOR.



Este método obtendrá todos los envíos realizados por un usuario (SOCIO) y nos servirá para poder mostrarle sus envíos en la pestaña de Niños apadrinados, evitando así que tenga acceso a todos los envíos de la base de datos y solamente a los de su interés.

---



Este método, el cual será llamado desde un rol ‘ADMINISTRADOR’, consulta todos los envíos existentes en la BD. Se ha implementado mediante una consulta JPQL.



Este método nos es útil para comprobar que, a la hora de apadrinar un nuevo niño, existan niños sin apadrinar disponibles en la base de datos y así poder asignar un nuevo apadrinamiento a un usuario.