Biblioteca Comunitaria 2017

- Integrantes:
  - Barbá, Dante
  - Riglos, Juan Ignacio

1. Objetivos del proyecto

Finalidad del proyecto
Crear un sitio web que permita a los usuarios poder realizar préstamos de los distintos libros que cada uno de ellos posee publicado. Para ésto, el sistema permitirá la creación de distintos grupos, los cuales serán regidos por las reglas que el administrador del mismo decida al momento de crearlo. Dichas reglas pueden ser: reglas para la aceptación de un usuario al grupo (ej: cantidad de libros mínimos que el usuario debe introducir al grupo) o reglas de aceptación de préstamos (ej: que el usuario que solicite el libro no posea otro libro prestado al mismo tiempo). Los usuarios podrán cargar tantos libros al sistema como así lo deseen y acceder a un listado que mostrará todos sus libros cargados con un pequeño detalle de los mismos. Con el objetivo de facilitar el login de los usuarios y estar más conectados con ellos, se integrará el sistema con la red social Facebook, permitiendo que los usuarios se logueen con las cuentas que ya poseen en dicha red.

Objetivos propuestos en 2017
Realizar la integración con Telegram, donde se puedan obtener recursos a través de consultas al bot. Creación de nuevas reglas para el intercambio de libros.

2. Logros alcanzados

- Funcionamiento correcto del sistema, con sus respectivas reglas.
- Creación de reglas
    - ReglaSolidaria
    - ReglaFamiliar
- Creación de un tutorial para usar Telegram
- Corrección del tutorial de instalación efectivo utilizando las herramientas de gestión de la configuración del software descritos a continuación
- Virtualización de la aplicación utilizando Docker.

3. Descripción de los frameworks o aplicaciones de base utilizados:

MongoDB con Voyage
MongoDB es una base de datos NoSQL, Open Source y orientada a documentos, los cuales se guardan en un formato de tipo JSON. Voyage es un pequeño framework de persistencia orientado puramente a objetos. Brinda una pequeña capa de abstracción entre el sistema de almacenamiento y los objetos, y provee un lenguaje útil de manejar con objetos

Seaside

Es un framework de aplicaciones web, libre y de código abierto para desarrollar aplicaciones en Smalltalk. Provee una arquitectura de componentes en la que cada página web es construida como un árbol de componentes individuales y estable, donde cada uno encapsula una pequeña porción de la página.

Seaside REST
Servicio REST proporcionado por Seaside para la creación de endpoints REST en una aplicación.

REST
Modelo de arquitectura para el desarrollo de webserivces. Proporciona una interfaz sencilla, montada sobre HTTP, que permite la interacción entre cliente-servidor. Al ser implementada sobre HTTP, todos los navegadores son compatibles, lo cual facilita la comunicación y facilita la compatibilidad y desarrollo.

Monticiello
Es un sistema de control de versiones que ayuda a almacenar y registrar múltiples versiones de su código. Monticiello posee la capacidad de mantener una lista de repositorios y nos permite acceder a ellos, ya sea para guardar nuestros Paquetes o para, a partir de una lista, seleccionar si queremos descargar algún paquete de algún repositorio y agregarlo a nuestra imagen de trabajo. Además, ayuda a gestionar los accesos concurrentes a un repositorio común de código fuente y permite tener un seguimiento de todos los cambios que se hacen sobre el código.

Metacello
Metacello es un sistema de gestión de paquetes para monticello. Eso significa que es un sistema que permite seleccionar distintas versiones de código de moniticello, para permitir así la creación o versionado de paquetes para el sistema completo, permitiendo la gestión de versiones estables o de desarrollo, para las cuales se puede simplificar su instalación con solo correr un método de una clase en particular, con lo cual pueden obtenerse versiones funcionales de un sistema de manera sencillo.

Bootstrap
Framework para el desarrollo del frontend de aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS. Útil además, para desarrollar el sistema de manera que sea adaptable a cualquier dispositivo (responsive).

Docker
Proporciona automatización para el despliegue de aplicaciones en contenedores, permite versatilidad, escalabilidad y una mayor predictibilidad en el comportamiento de la aplicación, a través de la virtualización del software. Las aplicaciones son contenidas dentro de imágenes, las cuales son montadas en contenedores. Las imágenes son inmutables durante la ejecución en un contenedor, lo cual mantiene la uniformidad de la misma en cada despliegue. Esto permite distribuir las imágenes en distintas plataformas y permite ejecuciones predecibles en distintos contextos.

Telegram Bot
Los bots son aplicaciones externas que se ejecutan dentro de Telegram. Los usuarios pueden interactuar con los bots a través de mensajes y comandos. La comunicación es de tipo productor-consumidor, la aplicación de Telegram se transforma en intermediario entre quien produce el contenido y quien lo consulta. El acceso para desarrolladores se da a través de la API que proporciona Telegram.

4. Diseño del proyecto - Decisiones de diseño relevantes

En el momento de encarar el proyecto, uno de los principales desafíos con el que nos encontramos, fue el hecho de que Biblioteca Comunitaria era una iniciativa nueva comenzada en el año 2015, con lo cual, el trabajo que recibimos no contaba con una extensa funcionalidad. Sin embargo, más allá de que no había una amplia variedad de operatividad en el sistema, cierta parte de la lógica estaba implementada, como así también un modelo de clases el cual nos sirvió de referencia para poder desarrollar la funcionalidad faltante en el proyecto y extender o modificar el diagrama a nuestro modo, para representar nuevas ideas que fueron surgiendo durante el desarrollo.

Como primer paso en la continuidad del proyecto, asumimos una de las sugerencias planteadas en el trabajo anterior: Agregar nuevas reglas. Está acción de pensar y proponer nuevas reglas, utilizamos el modelado de lógica que redefinieron en el 2016.

Otra de las sugerencias tomadas fue el de incorporar Telegram como bot de respuestas a ciertas consultas, a la hora de preguntar por un libro este te devuelve su estado, junto a un conjunto de variables de ese libro.

<u>Descripción de la solución de escalabilidad</u>
A continuación, se tratará de describir detalladamente la solución antes mencionada. Es importante destacar, que ésta solución solo fue empleada en las reglas de aceptación de usuario. Sin embargo, podría implementarse la misma lógica en las reglas de aceptación de préstamos, si se encontrara una relación entre varias reglas de préstamos.

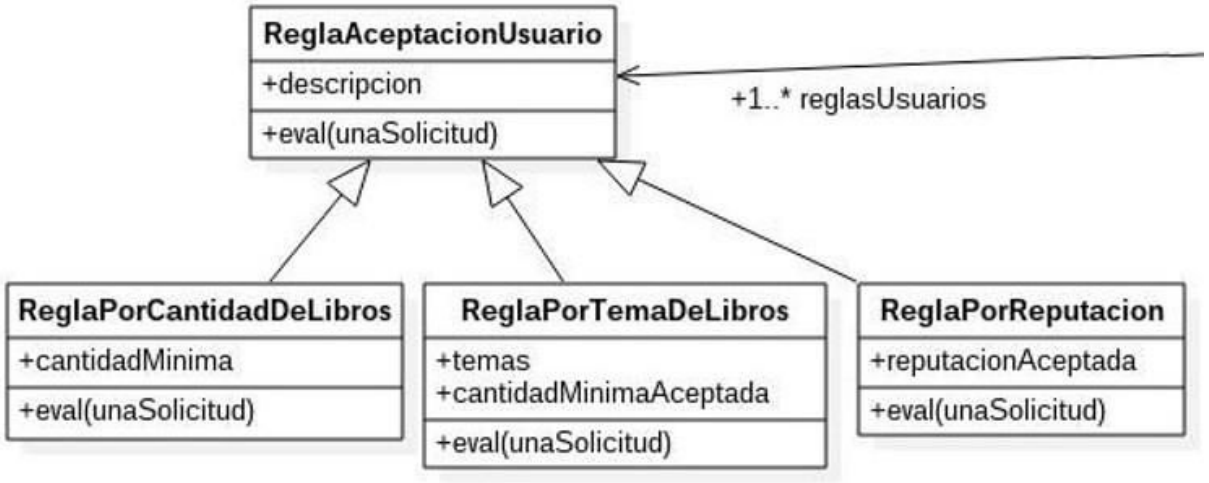
A la hora de tomar el proyecto, las reglas de aceptación de usuario presentes eran las siguientes:

- ReglaPorCantidadDeLibros
- ReglaPorTemaDeLibros
- ReglaPorReputación.

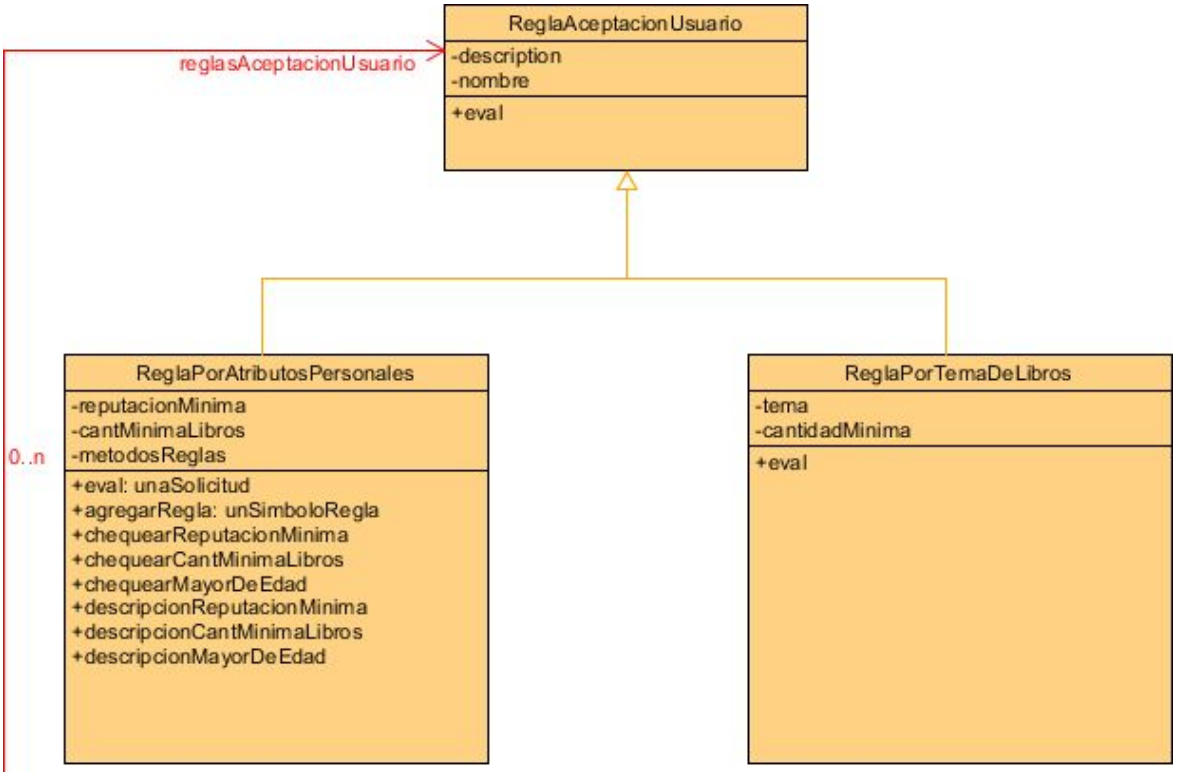
Por otro lado, una de las reglas propuestas por parte nuestra fue la de que el administrador de un grupo pueda requerir que los usuarios que quieran ingresar al mismo, deban ser mayores de 18 años. (ReglaMayorDeEdad).

Al querer incorporar la regla de mayor de edad, observamos que tanto ésta regla, como las reglas existentes , respondían a algo que era propio del usuario. Teniendo en cuenta ésto, hallamos que este aspecto en común podía expresarse mediante la clase ReglasPorAtributosPersonales (que comprendería la clase genérica que se describió anteriormente) e iba a ser subclase de ReglasAceptacionUsuario

Lo dicho se puede ver mejor en la comparación de los 2 modelos:



Modelo del 2015: Las 3 reglas modeladas como subclases



Modelo Propuesto

Lógica de Préstamos

La clase préstamo ya no existe más como tal. Ahora los préstamos se tratan como subclase de *Solicitud* representados como ***SolicitudLibro***.

Un usuario, al realizar una solicitud de préstamo de un libro, crea un objeto de la clase ***SolicitudLibro*** que se agrega en una colección solicitudesLibros ***perteneciente al dueño de la copia del libro solicitado***.
El usuario dueño de la copia del libro solicitado, aprobará -o no- esa SolicitudLibro de manera manual.
La aprobación no puede ser automática ya que la entrega del libro se produce de manera física e interfieren otros factores como lo son las reglas de los préstamos.

Supongamos el siguiente caso:
Dos usuarios distintos me solicitan el mismo libro. En mi colección de solicitudesLibro tengo dos solicitudes para la misma copia. Entonces puedo -por ejemplo- aceptar una de las solicitudes porque ya conocía al usuario, o bien porque su reputación es más alta que la del otro usuario.

En cualquier caso, el usuario dueño y el usuario solicitante del préstamo deberán reunirse para hacer la entrega del libro. Y luego de eso el usuario dueño efectivizar en el sistema que se aceptó la SolicitudLibro.
Al efectivizar la SolicitudLibro el objeto que representaba a la solicitud cambia su ubicación. De la colección de solicitudesLibros, pasa a guardarse en librosPrestados, es decir que el objeto sigue siendo una SolicitudLibro, pero ya aprobada. Del mismo, modo la solicitud se guarda también en la colección de librosAdquiridos que se utilizará desde el apartado "Mis Vencimientos" donde el usuario al que le prestaron el libro podrá ver cuanto tiempo le queda para que los préstamos que le hicieron continúen vigentes, pida renovación e incluso pueda aumentar su reputación si el libro es devuelto a tiempo.

Se decidió entre los dos integrantes no crear una clase "LibroPrestado" porque una SolicitudLibro contiene todos los datos necesarios -usuarioSolicitante, copia, dueño de la copia, fechaDevolucion, etc) para ser un préstamo (realizado o no).
Sólo cambia la perspectiva. Es decir, su estado. Que no se modeló como tal ya que el objeto no cambia su comportamiento sino la visión que se tiene sobre el mismo.

<u>Lógica de renovaciones de préstamos</u>
En el proyecto anterior, las renovaciones fueron desarrolladas como regla de aceptación de préstamos.
En esta regla, se le aplicaba una restricción en la cantidad máxima de renovaciones sobre una copia de un libro al usuario solicitante. Esta lógica fue modificada con la finalidad de que la renovación de préstamos ya no sea tratada como una regla, sino como un atributo propio de un grupo. Por lo tanto, al momento de crear un grupo, se solicita ingresar la cantidad máxima de renovaciones que se permiten por libro. En el caso de no especificar la cantidad, por defecto se establece que la cantidad máxima de renovaciones que un usuario puede solicitar de un libro, es "infinita" (999). Por el contrario, si el usuario que esta creando el grupo establece una cantidad, el maximo de renovaciones por libro dentro del grupo se limita a la cantidad especificada. Es decir que siempre se permiten renovaciones de préstamos, excepto en el caso de que el administrador del grupo (al momento de crearlo) establezca el valor de renovaciones en 0.

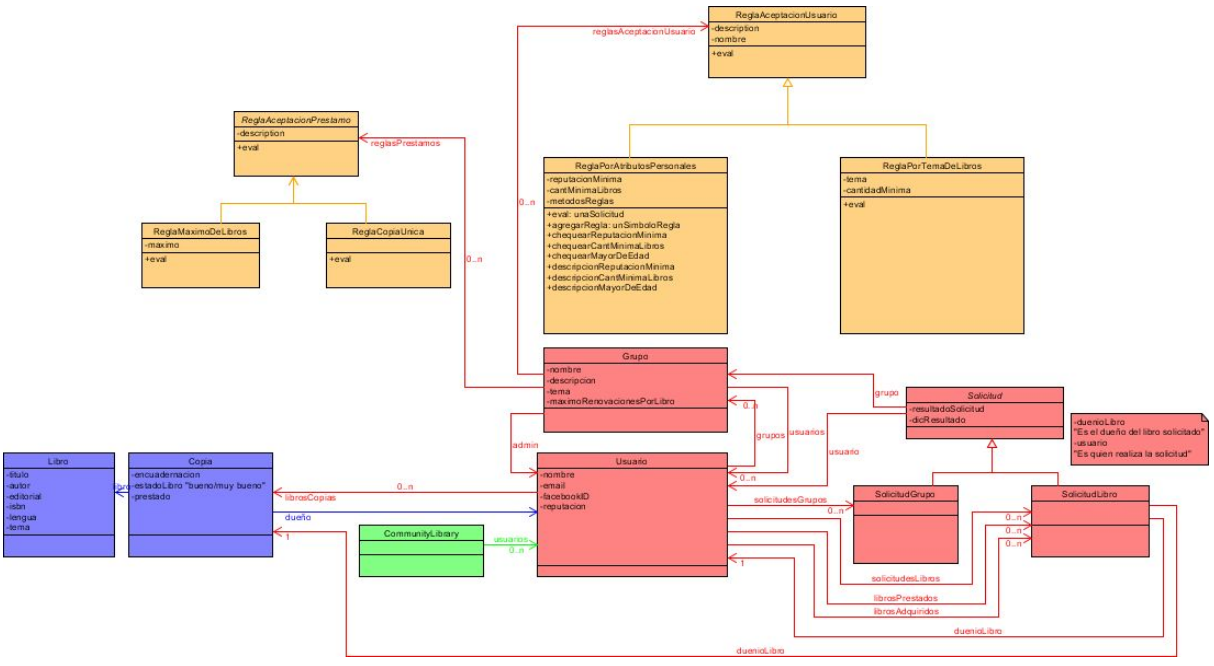
Módulos de la Arquitectura

Antes de describir las clases más relevantes del modelo, es importante destacar que el sistema está compuesto por 3 módulos principales.

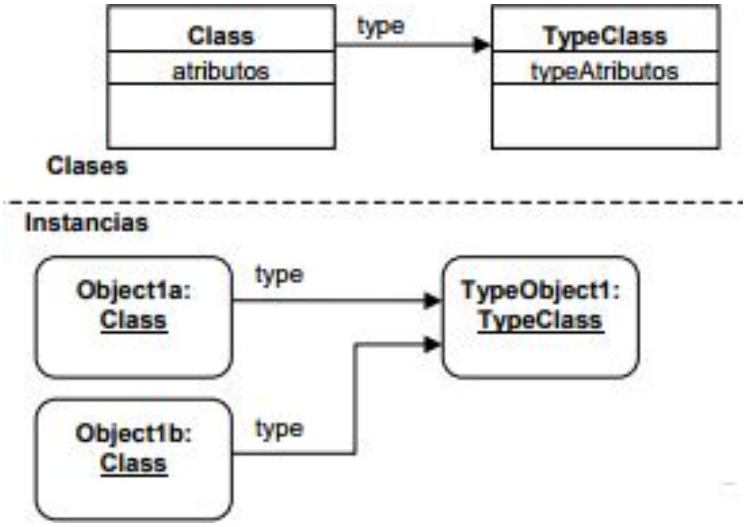
1. **Módulo de Libros:** Este módulo se encarga de la gestión de libroscopias, es aquí donde se aplica el patrón Type Object y se diferencian las copias de los libros.
2. **Módulo de reglas:**
    a. Reglas de Aceptación de Préstamo:Encargado de la gestión de préstamos, en este módulo se decide si el préstamo cumple con las condiciones necesarias para llevarse a cabo o no

    b. Reglas de Aceptación de Usuario:Encargado de la gestión de Usuarios en los grupos, en este módulo se decide si un usuario cumple con las condiciones necesarias para ingresar al grupo o no.

3. **Módulo de Grupos/Usuarios/solicitudes(de grupos y de préstamos):**Este módulo es el core de la aplicación, se encarga de interconectar los otros módulos. Cuando un usuario crea un grupo, este módulo se encarga de relacionar ese grupo con las reglas elegidas y luego en cada acción que tenga ese grupo verificar que las reglas que tiene asociadas sean cumplidas. También se encarga de comunicar a los usuarios con los distintos libros/copias, haciendo un correcto uso del patrón Typeobject, para traer las copias que tiene un usuario y ofrecer esas copias a los distintos grupos. Por último, también se encarga de los préstamos dentro de los grupos(solicitudes), relaciona los préstamos con los usuarios, las copias, los grupos y verifica que las reglas del mismo sean cumplidas.

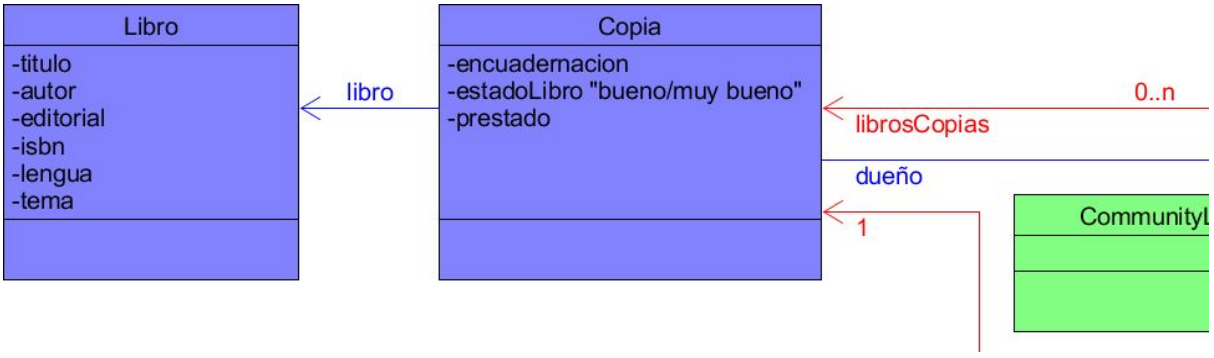


Este módulo se implementa haciendo uso del patrón *Type Object*.



Para plantear la relación entre una copia y su "tipo" (Libro) se utiliza el ***Type Object Pattern [Johnson 96]*** que permite que varias instancias de una clase -en este caso Copia- sean agrupadas de acuerdo con comunes atributos y/o comportamiento

1. Módulo de Libros



Clase Libro

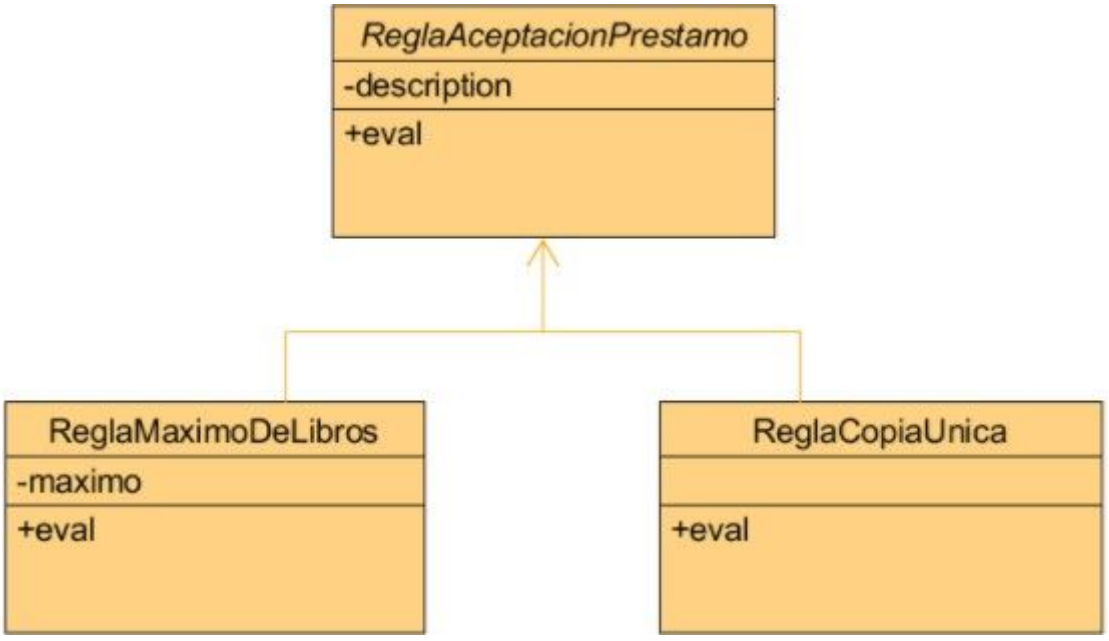
Tiene el propósito de concentrar los datos comunes de un libro, evitando así que se repitan en cada copia existente. Su rol dentro del patrón Type Object es el de TypeClass, determinando las características generales de un libro, que son compartidas por todas las copias del mismo. Su instancia será el TypeObject (dentro de patrón), al cual referenciarán múltiples copias, y al que delegarán las peticiones correspondientes.

Clase Copia:

Tiene el propósito de relacionar al Usuario con su copia de un determinado libro. Su rol dentro del patron Type Object, es el de Class, manteniendo las características particulares de una copia, de un libro determinado, cuyo dueño es un usuario.

Su instancia será el Object (dentro del patrón), que recibirá las peticiones particulares a esta copia y delegará las correspondientes a su libro. Así una instancia de Copia mantendrá una referencia a una instancia de Libro, siendo este su Type Object.

2. Módulo de Reglas



a. Reglas de Aceptación de Préstamo:

Clase ReglaAceptacionPrestamo:
Es una generalización abstracta de las reglas de aceptación de préstamos de libros. Aquí, se define el método abstracto *eval: unaSolicitud*. unaSolicitud será una instancia de la clase SolicitudLibro -explicada más adelante en el módulo de Solicitudes/Préstamos-.

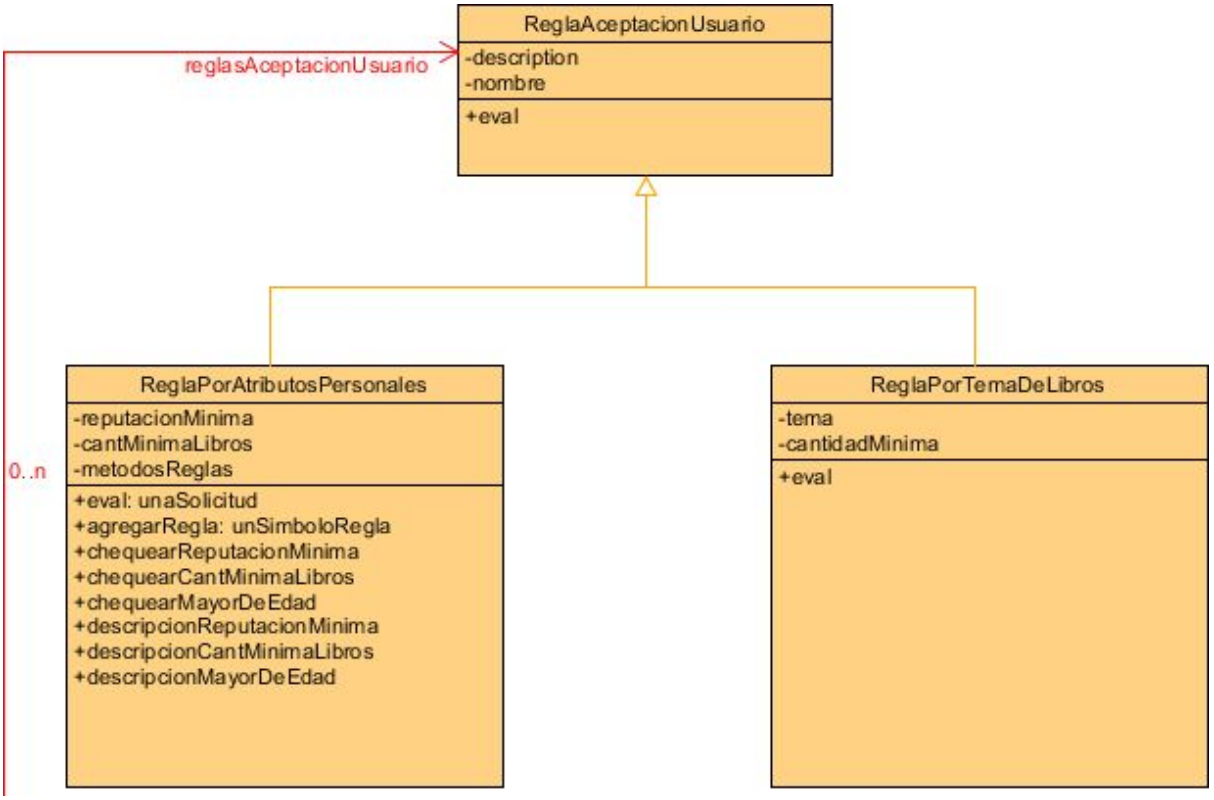
El método eval es el encargado de evaluar la solicitud que se reciba como parámetro. Al ser la clase abstracta de préstamos, se obliga a redefinir la responsabilidad del método a las subclases (ReglaMaximoDeLibros y ReglaCopiaUnica) quienes implementan el método teniendo en cuenta su propio comportamiento.

Además, la clase posee el atributo descripción, que será utilizado por cada subclase para describir brevemente el objetivo de la regla.

Clase ReglaMaximoDeLibros *superclass:* ***ReglaAceptacionPrestamo***
Esta regla tiene como objetivo limitar el maximo de libros que puede adquirir un usuario al mismo tiempo, en un cierto grupo. Para ésto, se establece el atributo “maximo”.

Si esta regla está presente en un grupo, al momento de realizarse una solicitud de un libro, se valida que el usuario solicitante no sobrepase la cantidad máxima de libros que pueden adquirirse definidos en la regla.

Clase ReglaCopiaUnica *superclass:* ***ReglaAceptaciónUsuario***
Esta regla tiene como objetivo permitir como maximo, a un usuario en un determinado grupo, una sola copia del mismo libro.
Si esta regla está presente en un grupo, al momento de realizarse una petición de un libro, se valida que el usuario solicitante no tenga préstamos vigentes del libro que está solicitando.



b. Reglas de Aceptación de Usuario

Clase ReglaAceptacionUsuario

Es una generalización abstracta de las reglas de aceptación de usuarios a un grupo. Aquí, se define el método abstracto *eval: unaSolicitud*. -unaSolicitud será una instancia de la clase SolicitudGrupo -explicada más adelante en el módulo de Solicitudes/Préstamos-. El método eval es el encargado de evaluar la solicitud que se reciba como parámetro. Al ser la clase abstracta de aceptación de usuarios al grupo, obliga a delegar la responsabilidad del método a las subclases (ReglaPorAtributosPersonales y ReglaPorTemaDeLibros) quienes implementan el método teniendo en cuenta sus particularidades. Es importante resaltar que *ReglaPorAtributosPersonales* **no es una regla en sí misma**, sino es una clase que agrupa varias reglas, todas ellas caracterizadas por representar un atributo personal de los usuarios.

Actualmente se ven agrupadas 3 reglas en esta clase:

- ReglaMayorDeEdad
- ReglaCantMinimaLibros
- ReglaReputacionMinima

Estas reglas **no se implementaron como clases** sino como métodos, mediante reflexión como se explica en el punto 4.

Dentro de la clase **ReglaPorAtributosPersonales** *superclass:* **ReglaAceptacionUsuario** se pueden encontrar los siguientes métodos:

chequearMayorDeEdad: Esta regla tiene como objetivo establecer que los usuarios en un grupo tengan más de 18 años. Si esta regla está presente en un grupo, al momento de realizarse una petición de ingreso al mismo, se valida que el usuario solicitante sea mayor de 18 años. La validación se realiza con la fecha que el usuario posee en Facebook.

chequearReputacionMinima: Esta regla tiene como objetivo limitar la reputación de los usuarios en un grupo. Si esta regla está presente en un grupo, al momento de realizarse una petición de ingreso al mismo, se valida que el usuario solicitante tenga la reputación mínima requerida por el grupo. La validación se realiza comparando la reputación del usuario en el sistema, con la reputación mínima del grupo ***(variable de instancia de la clase ReglaPorAtributosPersonales)***.

chequearCantMinimaLibros: Esta regla tiene como objetivo establecer que los usuarios en un grupo tengan de antemano una cantidad mínima de libros cargados en el sistema. Si esta regla está presente en un grupo, al momento de realizarse una petición de ingreso al mismo, se valida que el usuario solicitante tenga como mínimo la cantidad de libros requerida por el grupo. La validación se realiza comparando la cantidad de libros que el usuario tiene cargados en el sistema, con la cantidad mínima del grupo ***(variable de instancia de la clase ReglaPorAtributosPersonales)***.

Clase ReglaPorTemaDeLibros *superclass:* **ReglaAceptacionUsuario**

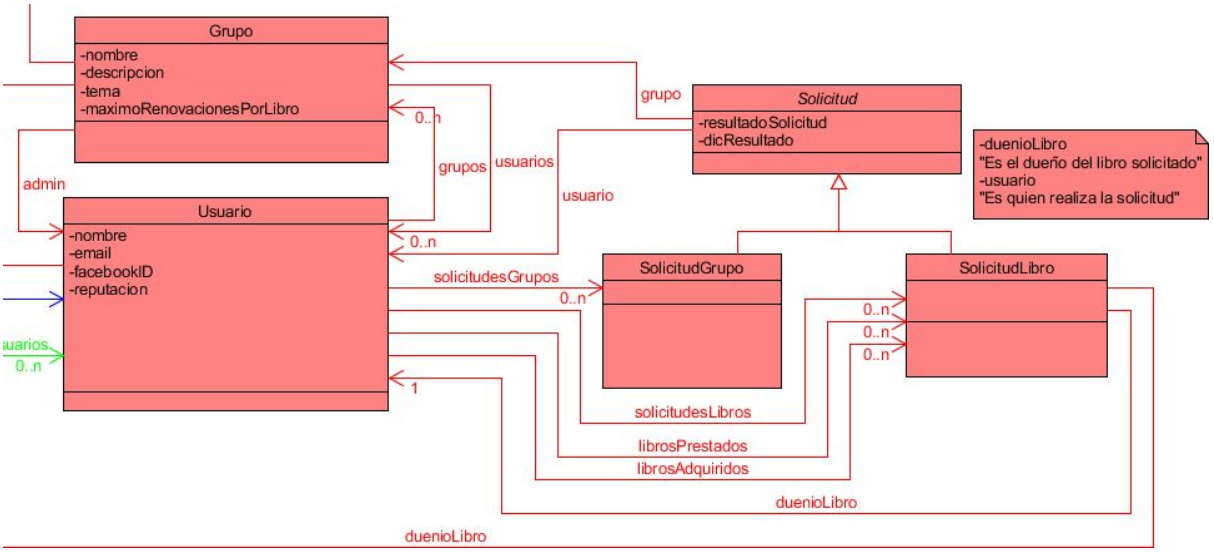
Esta regla tiene como objetivo establecer una temática específica en un grupo. Cuando el usuario que está creando el grupo incorpora esta regla, indicará además del tema, una cantidad mínima de libros (sobre el tema seleccionado) que los usuarios deben poseer para poder ingresar al grupo. Si esta regla está presente en un grupo, al momento de realizarse una petición de ingreso al mismo, el sistema valida que el usuario solicitante tenga cargados en el sistema una cantidad de libros (sobre el tema en cuestión) mayor o igual a la cantidad mínima establecida por el administrador del grupo.
Si el usuario solicitante cumple con la condición de poseer como mínimo tantos libros del tema como requiere la regla, el usuario no pasa a formar parte del grupo automáticamente, sino que se realizan las siguientes acciones:

- Se agrega la solicitud a la lista de solicitudes pendientes del administrador del grupo

- Se muestra un mensaje al usuario solicitante informandole que el administrador del grupo se pondrá en contacto con el. (Esto es para que el administrador del grupo pueda corroborar que el usuario solicitante realmente cuenta con la cantidad de libros del tema especificado que se requieren para ingresar al grupo). Luego, el admin puede decidir aceptar o rechazar la solicitud cuando lo desee.

Por lo tanto es importante destacar que esta regla es tratada de manera particular a las demás. Ya que como se dijo anteriormente, la validación del cumplimiento o no de la regla, no se realiza totalmente de manera automática por el sistema.

3. Módulo de Grupos/Usuarios/solicitudes (de grupos y de préstamos)



Clase Usuario:
Modela a un usuario del sistema, con grupos a los que pertenece, y con copias de libros que le pertenecen.
El usuario puede realizar búsquedas de distintos grupos y hacer peticiones para pertenecer a ellos, puede crear distintos grupos y asignarles las reglas de préstamos y de usuarios que desee. En cuanto a libros, puede agregar nuevos libros que posea y también puede solicitar distintos préstamos en los grupos a los cuales pertenezca.

Clase Grupo:

Modela un conjunto de usuarios relacionados para compartir libros de una índole o tema específico. Esta clase conoce a todos los usuarios que pertenecen al grupo y diferencia de ellos al administrador del mismo, también indirectamente tiene acceso a todas las copias de libros que posee el grupo y de la misma manera al historial de todos los préstamos

realizados en el mismo. Además esta clase se encarga de aplicar las reglas de aceptación, tanto las de usuarios al grupo como las de préstamo de libros. Por lo tanto, cada vez que se quiera realizar un préstamo se le pedirá a esta clase que verifique que dicho préstamo cumple con las reglas preestablecidas en el grupo, y ocurrirá lo mismo al momento de que un usuario trate de unirse a un grupo: Esta clase verificará que el usuario solicitante cumpla con los requisitos para pertenecer.

Clase Solicitud

Es una generalización abstracta de las solicitudes realizadas por los usuarios(Solicitud de grupo y solicitud de préstamo de libro). Aquí, se definen los atributos comunes a los 2 tipos de solicitudes existentes. En particular, es importante detenerse en el propósito y el funcionamiento de los atributos “dicResultado” y “resultadoSolicitud”. Por un lado, el atributo *dicResultado* es un diccionario donde se almacena :

- claves = reglas que tiene el grupo. El diccionario va poseer tantas claves como reglas tenga el grupo.
- valor asociado a cada clave = booleano. True o False , para representar si la regla indicada en la respectiva ‘clave’, es cumplida por el usuario solicitante.

Por otro lado, el atributo resultadoSolicitud, es un string que almacena los diferentes resultados que obtiene el usuario para la evaluación de cada regla. Por ejemplo, si un grupo posee la regla “Mayor de 18 años” y “reputación mínima”, y un usuario solicita la inscripción a ese grupo, se crea una SolicitudGrupo y su correspondiente dicResultado va a contener las siguientes claves:

- “informeMayorDeEdad”
- “informeReputacionMinima”

Es importante destacar que el nombre de las claves se estipulan como ‘informe..’ , ya que cada una representa a su vez un método para ser invocado por reflexión. ***Este método retorna un mensaje (informe) del no cumplimiento de la clave***.

Luego, se verifica el cumplimiento de cada una de las reglas por parte del usuario solicitante, y como se dijo anteriormente, se indica como valor de clave:

- True en caso de cumplir con la regla representada en la clave
- False en caso contrario.

Cuando se finaliza de procesar la solicitud, se toman todas las claves *No cumplidas* por el usuario (es decir, reglas no cumplidas) y por cada una se ejecuta mediante reflexión el mensaje de su clave.

Siguiendo con el ejemplo, si el usuario que solicita cumple con la reputación mínima requerida para ingresar al grupo, pero no es mayor de 18 años, la clave "*informeMayorDeEdad*" toma un valor *false*, y por tanto se ejecuta el método(informeMayorDeEdad) el cual retorna el string "Ud no es Mayor de edad." que es almacenado en la variable *resultadoSolicitud* para luego informar al usuario el resultado de su solicitud.

Clase SolicitudGrupo *superclass:* **Solicitud**
Tiene como objetivo modelar las solicitudes realizadas por los usuario al momento de querer ingresar a un grupo. En esta clase, es importante detenerse en el comportamiento del sistema al momento de que el usuario solicitante genere la solicitud de grupo(al querer ingresar al grupo), cuando este mismo tiene presente la regla "Tema de libro". Si el admin de un cierto grupo impuso esta regla al momento de crearlo, el sistema validará que el usuario solicitante tenga cargados en esta red social, los libros del tema en cuestión. Esto, se realiza mediante el mensaje *"tieneLibrosDelTema:unTema";* presente en esta clase. La evaluación de este mensaje provoca lógicamente 2 resultados posibles:

1. Si el usuario solicitante no posee libros del tema, la clave correspondiente a la regla de tema de libro (de dicResultado), tomará el valor False, y se informará el resultado al usuario tomando el valor del método "informeTemaDeLibro" que retorna el String "Ud. No cuenta con la cantidad mínima de libros con el tema del grupo" de igual manera en la que se explicó anteriormente en la descripción de la superclase - Clase Solicitud-.

2. Por el contrario, si el usuario solicitante posee libros del tema, la clave correspondiente a la regla de tema de libro (de dicResultado), tomará el valor True. En este caso, el sistema agregara la solicitud de ingreso al grupo en la colección solicitudesGrupos del administrador del grupo e informará al usuario solicitante que el administrador del grupo se pondrá en contacto con él. Esto es principalmente para que el admin pueda verificar, por ejemplo mediante una reunión con el solicitante, que el mismo tenga efectivamente los libros del tema que ha cargado en el sistema. Luego de esto, el administrador podrá decidir aceptar o denegar la solicitud de ingreso al grupo.
Esto también es explicado más arriba donde se describe el comportamiento de la clase ReglaPorTemaDeLibros.

Clase SolicitudLibro *superclass:* **Solicitud**
Tiene como objetivo modelar las solicitudes realizadas por los usuario al momento de pedir el préstamo de una copia de un libro en un grupo determinado. Aquí, además de los atributos heredados de la superclase, se definen entre otros:

- copia → libro que se solicita
- duenioLibro → usuario dueño del libro que se solicita (al que le llega la solicitud)

- fechaInicio → Se define cuando se crea la solicitud. Sirve para identificar quien solicitó primero un libro en caso de que dos o más usuarios pidan el préstamo de un libro particular al mismo usuario.<
  No es la fecha de inicio del préstamo, sino la fecha de inicio de la solicitud
- fechaDevolucion → Es definida por el administrador al momento de aprobar la solicitud. Es la fecha de devolución del préstamo.
- cantRenovaciones → contador de la cantidad de renovaciones que va solicitando el usuario sobre el mismo préstamo
  El máximo de renovaciones está definido en la regla MaximoRenovacion
- solicitoRenovacion → valor booleano para indicar si el usuario al que se le prestó el libro, ha pedido o no una renovación del mismo. Este valor va cambiando de true a false y viceversa, a medida que se solicitan renovaciones.
    - Si pidió una renovación el valor se encontrará en *True*, podrá acceder o denegar la extensión del préstamo desde la vista *"LibrosPrestadosView"*.

La evaluación de la solicitud se implementa de la misma manera que se explicó en la superclase -Clase Solicitud-.
En este caso, las claves que representan al diccionario "*dicResultado*", serán las reglas correspondientes a la aceptación de préstamos:

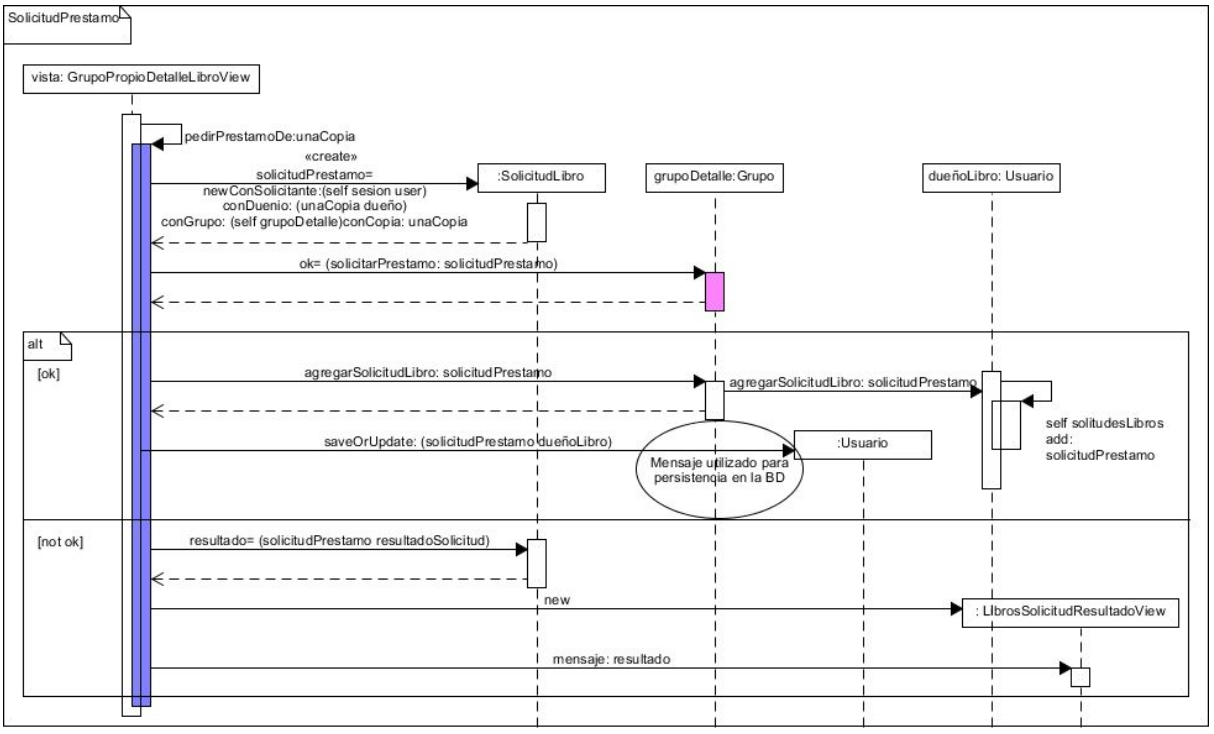
- informeCopiaUnica
- informeMaximoDeLibros

Por otro lado, se encuentran los métodos ***puedeRenovar*** y ***extenderPrestamo***.

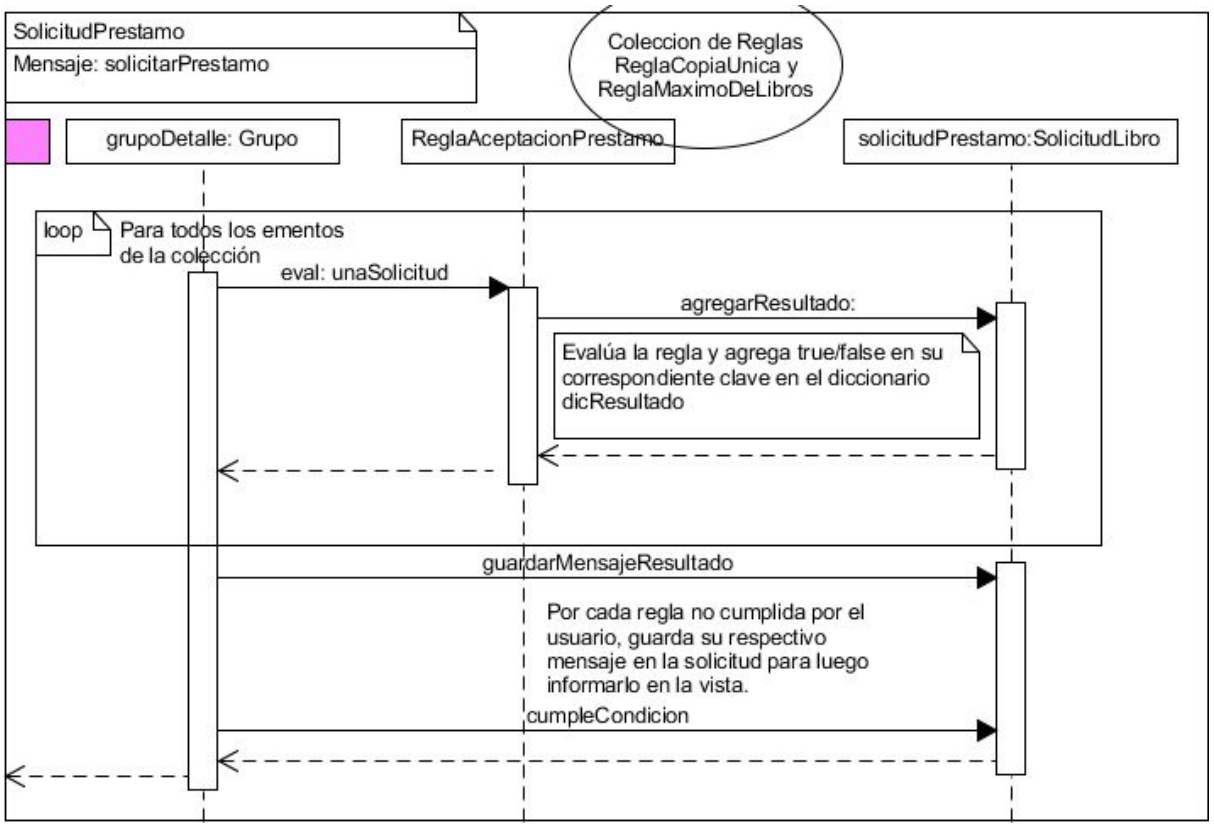
El método **puedeRenovar**, verifica si el usuario puede solicitar una renovación del préstamo, comparando el atributo cantRenovaciones, con el máximo de renovaciones definido en el atributo del grupo maximoRenovacionesPorLibro. Si el usuario no puede renovar, se le informará que ya excedio el maximo de renovaciones. Si puede renovar, quedará pendiente de aprobación por el usuario dueño del libro.
El método **extenderPrestamo**, representa la accion de extender el prestamo. Esta acción, es ejecutada por el usuario dueño del libro, quien responde a una solicitud de renovación encontrada en la vista de librosPrestados (en este caso del usuario dueño del libro).
Por defecto, la renovación, extiende el préstamo del libro por 7 días.

Diagramas de Secuencia

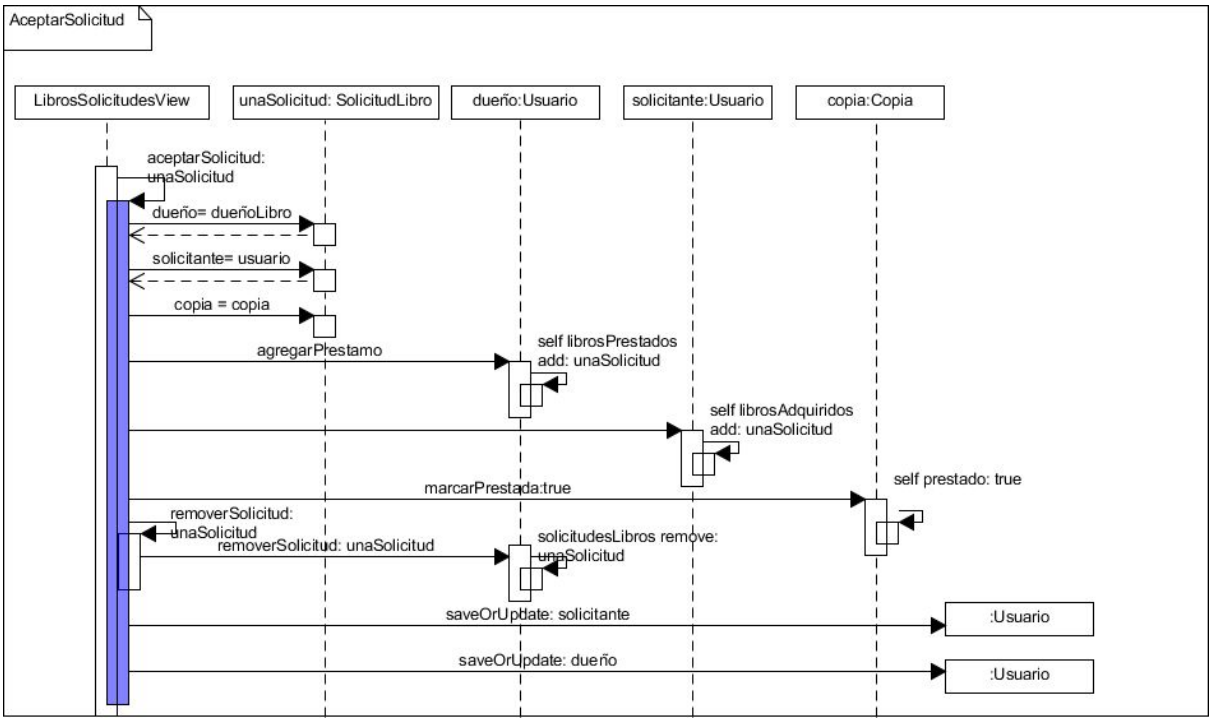
SOLICITAR UN PRÉSTAMO



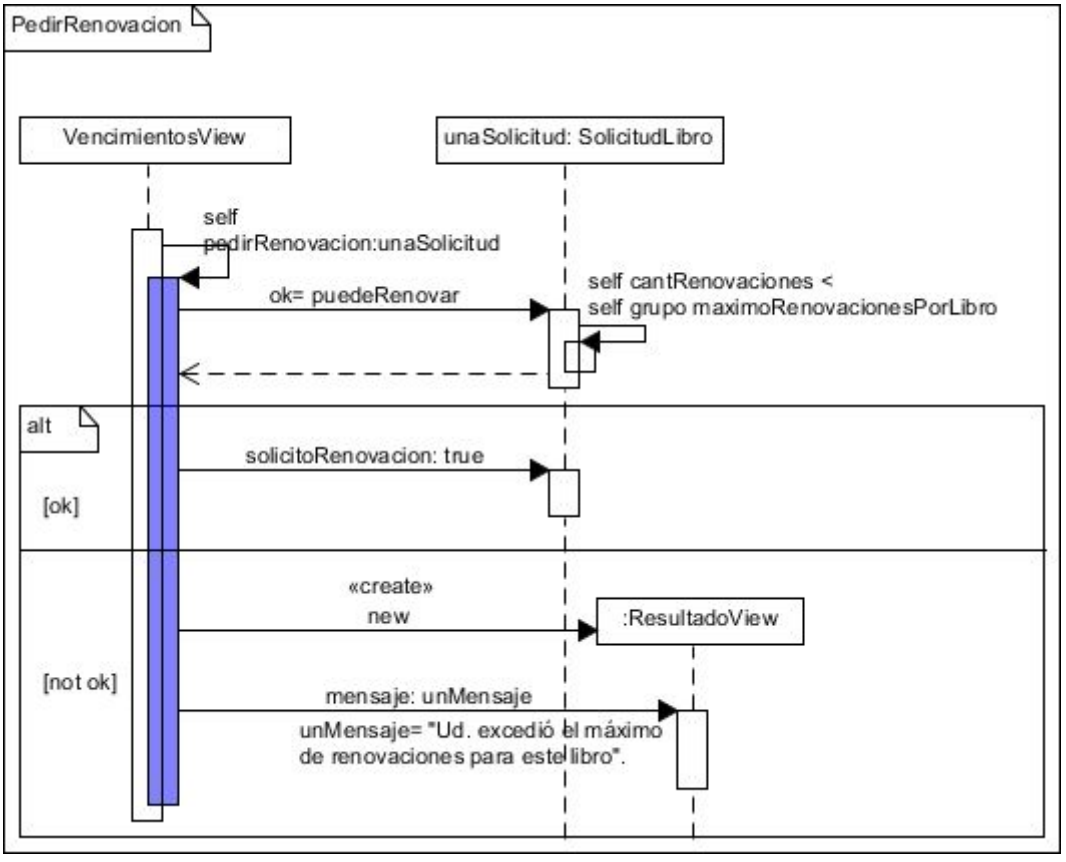
SOLICITAR UN PRÉSTAMO (CONTINUACIÓN)



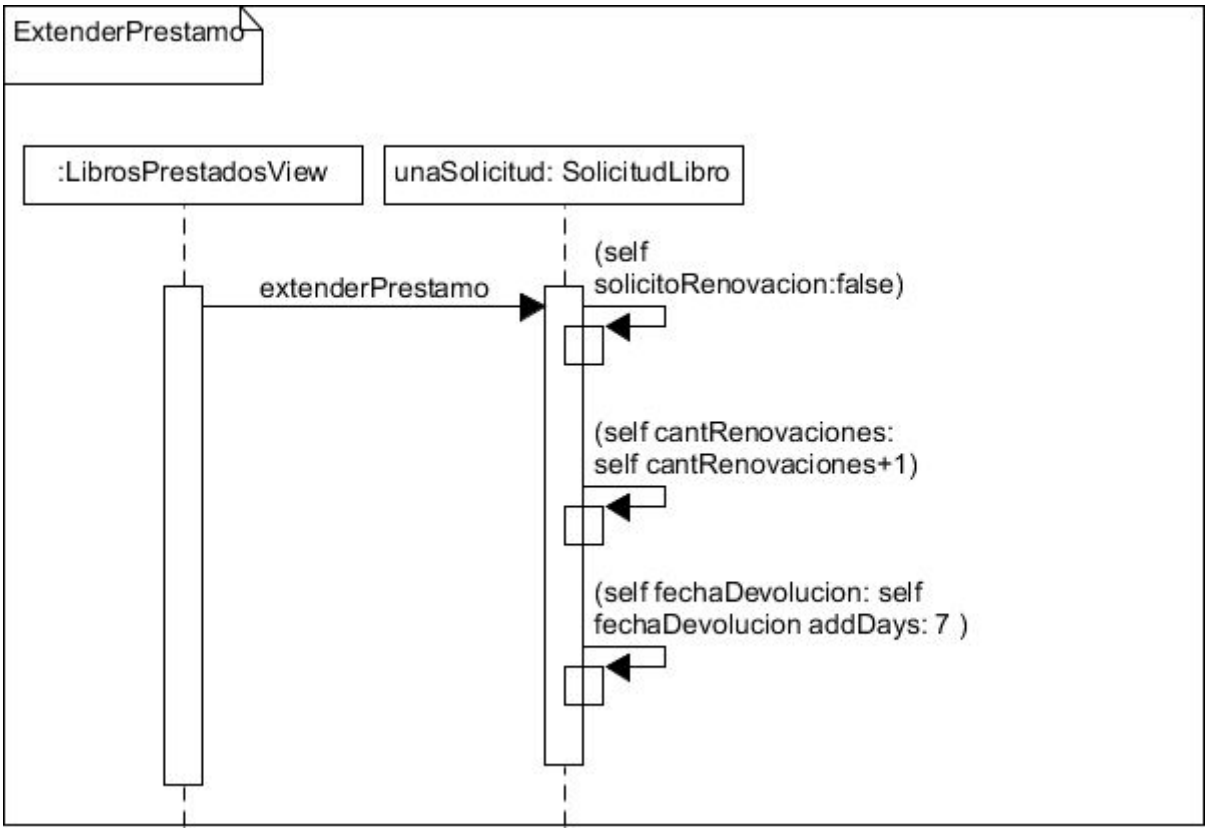
ACEPTAR SOLICITUD DE UN PRÉSTAMO



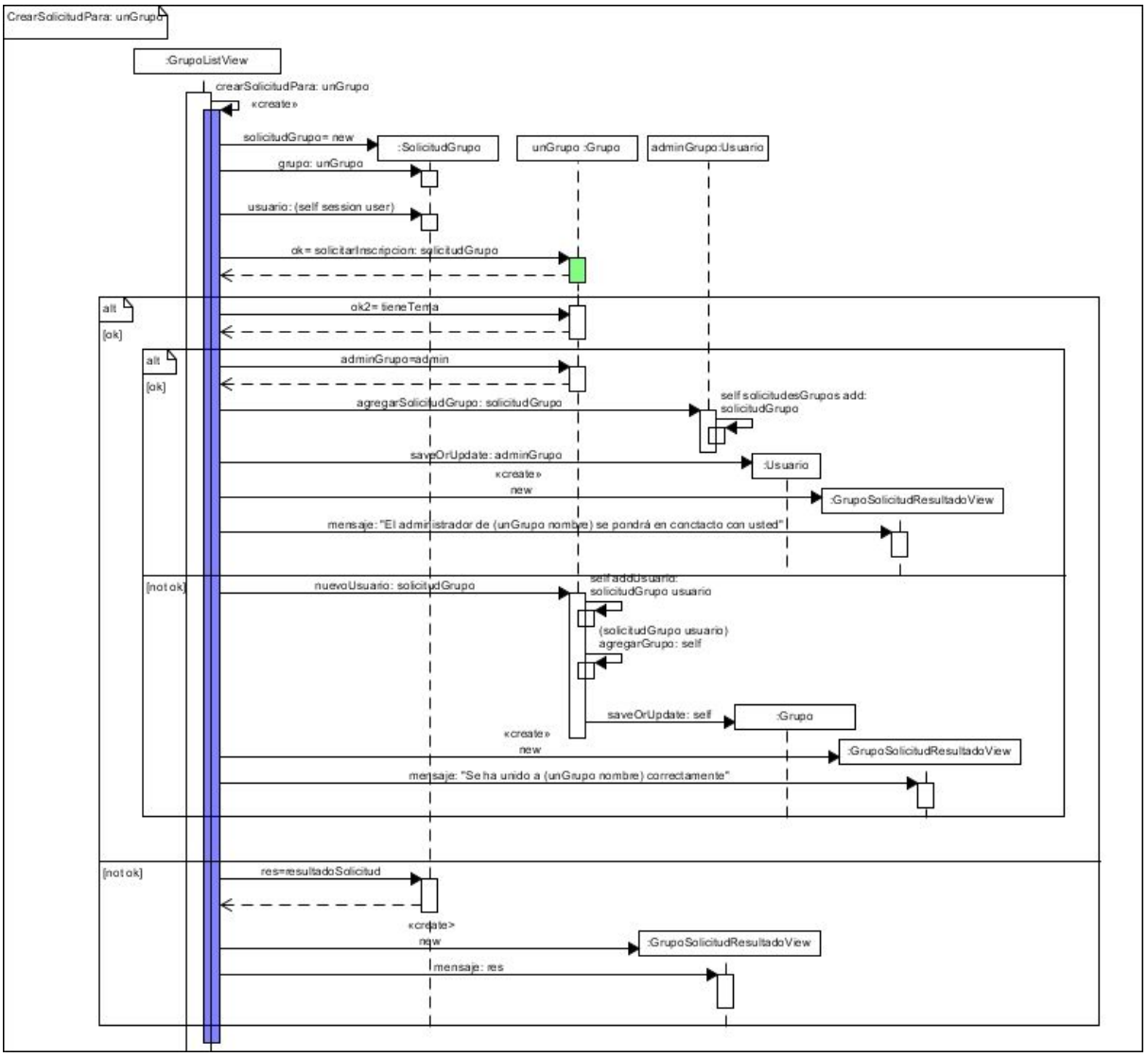
PEDIR RENOVACIÓN



ACEPTAR RENOVACIÓN



SOLICITAR INGRESO A UN GRUPO



SOLICITAR INGRESO A UN GRUPO (CONTINUACIÓN)

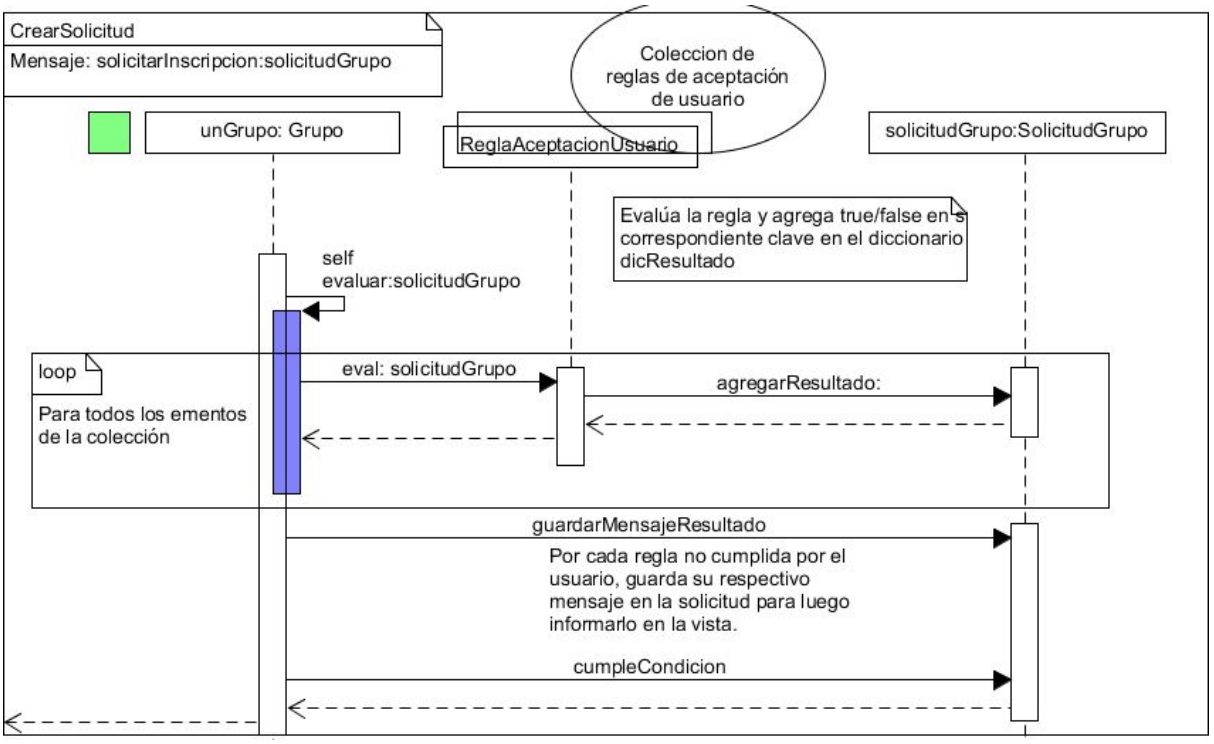
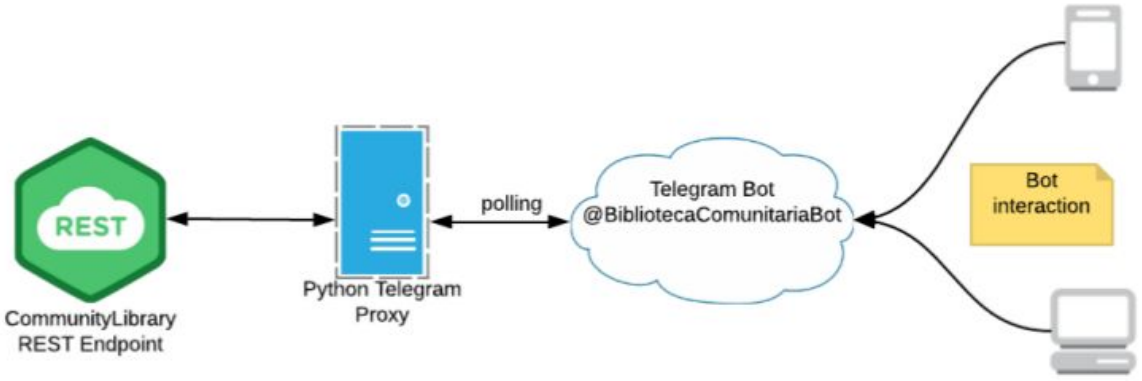


Diagrama de interacción entre API REST y Telegram:



5. Instrucciones de instalación paso a paso:

API Facebook: Antes de comenzar con el tutorial de instalación cabe aclarar que deberán crear una aplicación de Facebook que permita el login de los usuarios.

Para hacerlo tiene que realizar los siguiente:

a. Loguearse normalmente en Facebook e ingresar en la siguiente url: https://developers.facebook.com/apps/
b. Ingresar en “Mis aplicaciones” → “Agregar aplicación”.
c. Luego seleccionar como tipo de aplicación “Inicio de sesión con Facebook” → “www”(web)
d. Como URL del sitio web colocar: http://localhost:8080/hello
e. Luego de crear la aplicación, en el panel de administración busque el ID de la aplicación creada. Algo como esto: 848529051971750
f. En la clase del proyecto de pharo CommunityLibrary en el initialize deberá reemplazar los valores:
g. newClientID por el ID de su aplicación y SecretID por la clave secreta de su aplicación.
h. Nota: Sólo el creador de la aplicación podrá loguearse ya que es para pruebas de desarrollo. Para loguear otros usuarios deberá darles permiso desde el panel de administración en la opción “roles”.

A) Mongo DB

Descargar e instalar la última versión correspondiente a la arquitectura de su pc de MongoDB de la página o cial.

https://www.mongodb.com/download-center#community

B) Crear la siguiente carpeta: C:\data\db de no existir “data”, también crearla.

C) Descargar e instalar el un gestor de base de datos para mongodb como lo es el gestor “robombongo” (ya que va a facilitar el manejo de la base de datos durante las pruebas) de su página o cial:

https://robomongo.org/download

Repositorios de Pharo.

Sobre una imagen nueva de Pharo 4.0 hacer lo siguiente: Click izquierdo → Tools → Conguration Browser y realizar las instalaciones en dicho orden:

1. Instalar la versión estable de Voyage mongo (EstebanLorenzano.47). ES IMPORTANTE INSTALAR PRIMERO VOYAGE MONGO ANTES DE MONGO TALK.
2. Instalar la version estable de Mongotalk (EstebanLorenzano.43)
3. Instalar la versión estable de Seaside3 (topa.278)
4. Ejecutar el siguiente comando en **playground**

```
Gofer it
  smalltalkhubUser: 'SvenVanCaekenberghe' project: 'Neo';
  configurationOf: 'NeoJSON';
  loadStable.
Gofer new
  squeaksource: 'Seaside30Addons';
  package: 'Seaside-REST-Core';
  package: 'Seaside-Pharo-REST-Core';
  package: 'Seaside-Tests-REST-Core';
  load.
```

5. Agregar el repositorio de la cátedra en Monticello.

Click izquierdo →Monticello Browser → +Repository (Seleccionar HTTP), pegar el siguiente contenido:

```
MCHttpRepository
    location: 'https://catedras.lifia.info.unlp.edu.ar/monticello'
    user: 'tpoo2012'
    password: 'tpoo2012'
```

Una vez hecho esto podrás acceder a el repositorio del proyecto en la ventana de diálogo que se abre.

6. Buscar a la izquierda en los paquetes

“CommunityLibraryFacebook”

clickearlo y a la derecha seleccionar la versión 41

(CommunityLibraryFacebook-DanteBarba.41.mcz) y clickear “load”.

Esto descargara sobre la imagen, la última versión del proyecto.

Configuración de base de datos:

A. Se deberá crear la base de datos siguiendo estos pasos: Click derecho sobre la conexión → Create database y ponerle de nombre “comunidad”.
B. Click izquierdo → Tools → Seaside Control Panel → Click derecho sobre el espacio blanco → Add adaptor → seleccionar “ZnZincServerAdapter y click en “ok”. Quedará en el puerto 8080. Seleccionar el adaptor creado y click en "start".
C. Estos pasos pueden ser reemplazados por el siguiente comando unificado
D. CommunityLibrary start.
E. Ejecutar el siguiente código en un “playground”.

```
DBRepository conectar.
```

Verificar que en el mongod que aceptó la conexión. Luego ejecutar en el playground el siguiente código: (WAAdmin register: LoginComponent asApplicationAt: 'hello')preferenceAt: #sessionClass put: Sesion.

Ir al navegador, entrar en: localhost:8080/hello y loguearse con Facebook

- Mocking

  Si en vez de utilizar base de datos, deseamos simplemente inicializar los servicios, con datos precargados en memoria, podemos ejecutar el siguiente comando:

  CommunityLibrary mock.

- Docker

  El proyecto se encuentra parcialmente dockerizado. Para utilizar las funcionalidades de telegram, se debe Ejecutar una micro aplicación montada sobre un contenedor de docker.

  Requisitos:

  - docker
  - docker-compose 3.1 o superior.

  Sobre la carpeta donde se encuentra el archivo docker-compose.yml

  $ docker-compose build

  $ docker-compose up

Configuración de Telegram

La aplicación permite consultar reservas utilizando la app de mensajería Telegram. Esto se logra a través de los llamados bots. Los bots son aplicaciones con inteligencia automatizada que permiten la interacción entre usuarios. Estas aplicaciones son configurables a gusto del programador o quien las utilice. Para

poder interactuar con un bot, primero es necesario crear uno. Para la aplicación Biblioteca Comunitaria se ha preconfigurado un bot llamado @BibliotecaComunitariaBot que puede ser buscado en la aplicación de Telegram de la plataforma que se desee utilizar.

- El primer comando /start nos permite iniciar la secuencia de pasos.
- Luego, con el comando /isbn se permitirá buscar un libro

Ejemplo de comando: /isbn 1234

Requisitos:
- Tener la microaplicacion "telegram_webserver" en ejecución
- Tener Pharo configurado con el servidor funcionando (probar ingresar a localhost:8080/hello)
- Tener una cuenta activa en Telegram.

6. Minutas de cada sprint

Reunión 1

Fecha	04/09/2017, 18:30hs
Lugar	Aula 1-3, Fac. Infórmatica, UNLP
Integrantes	Dante Barbá, Juan Riglos y Federico Balaguer
Objetivo	Presentación: • Presentación del proyecto, se habló acerca de la modalidad de trabajo. • Se debe presentar una minuta por cada reunión. • Todas las minutas estarán contenidas en este archivo • EL trabajo consta de la extensión del proyecto de Bibliotecas, el cual se viene ampliando hace dos años. • Se perfila Telegram como opción de integración para la Biblioteca. Se conversarán las formas de esta integración en la próxima reunión.

Reunión 2

Fecha	18/09/2017, 19:hs.

Lugar	Aula 1-3, Fac. Infórmatica,UNLP
Integrantes	Juan Riglos, Dante Barbá. Federico Balaguer
Objetivo	Lineamientos generales sobre el trabajo: ● Explicar cómo poder integrar telegram con el sistema. ● Patrón command: Se utilizó para establecer las reglas de negocio. Aceptación de grupos y validación de préstamos. ● La documentación es independiente, no se debe anexar la documentación previa. ● La próxima minuta será el día Lunes 2 de Octubre.

Reunión 3

Fecha	02/10/2017
Lugar	Aula 1-3, Fac. Informática, UNLP
Integrantes	Juan Riglos, Dante Barba, Federico Balaguer
Objetivo	Integración con Telegram. Cliente para Pharo: ● Problemas del servidor para descargar el repo. ● PharoUsers: Consultar por levantar servidor HTTPs para webhook de Telegram. ● Librería de Cliente de Telegram.

Reunión 4

Fecha	23/10/2017
Lugar	Aula 1-3, Fac. Informática, UNLP.
Integrantes	Juan RIglos, Dante Barba y Federico Balaguer
Objetivo	Integración con Telegram. Cliente para Pharo: ● Headless. Enterprise pharo. ● NeoJson: Manejo JSON para Pharo. ● Pensar reglas de admisión y préstamos de libros.

	Verificar si falta agregar relaciones al modelo. ● Analizar el impacto de reglas en el modelo. ● Metacello.

7. Conclusiones

Descripción de las dificultades encontradas en el desarrollo del proyecto y cómo fueron superadas

- **Instalación del proyecto:**
  Dado que el tutorial de instalación propuesto en la documentación anterior del proyecto no funcionó, organizamos una reunión con Federico para intentar acordar una solución al problema. Se acordó solicitarle la imagen de Pharo sobre la cual trabajó el grupo anterior, y partir de esa base.
  Probamos con la versión de Pharo que nos dijeron y no anduvo, así que tuvimos que empezar a debuguear y el problema estaba en el orden de los pasos de instalación, modificamos el orden y anduvo.

- **Dedicación de tiempo:**
  No disponer del tiempo deseado para poder involucrarnos como hubiésemos querido fue una de las trabas a la hora de desarrollar.

- **Problemas configuración con Base de Datos:** El adaptador a la base de datos no funciona correctamente no funciona a la hora de guardar datos, no encontramos una solución a este problema con lo cual utilizamos datos estáticos. El módulo de conexión con MongoDb no funciona de forma adecuada en Pharo4.

- **Falta de soporte:**
  Se nos hizo muy difícil encontrar documentación actualizada. También resultó complejo entender el funcionamiento de algunos frameworks y librerías, como ZnServer.

Lecciones aprendidas:

1. Comprendimos que a la hora de encarar un proyecto, es necesario una fase previa de comprensión del sistema e intentar pensar que fue lo que quiso hacer o porque se tomó esta decisión para luego poder continuar con algún criterio similar el proyecto.
2. Comprendimos las dificultades de encarar un proyecto con tecnlogías desconocidas para nosotros como lo son las **Base de datos no relacionales (MongoDB) y Framework web para Smalltalk (Seaside).**

Trabajo futuro: cómo podría continuarse con el proyecto

Nuestra propuesta para la continuación del proyecto involucra una serie de items.

- Como ítem principal proponemos la realización del requisito que propuso el grupo anterior y que no logramos cumplir: la migración del login con API de Facebook a Twitter y la comunicación por mensaje directo en Twitter entre los usuarios.
- Como segundo ítem podría agregarse el acceso al sistema con una cuenta de Google que todos los usuarios del sistema operativo “Android” poseen.
- Durante el desarrollo del sistema nos abocamos a la implementación de funcionalidades y la verificación del correcto funcionamiento del sistema en general pero no dedicamos el tiempo necesario al apartado visual. Por lo tanto se solicita una renovación de los layouts de las vistas y -de ser posible- migrar del framework Seaside a la utilización de html5 puro y Javascript como una web clásica, ya que por lo menos a nosotros no nos gustó generar con objetos html.
- Crear e implementar reglas que involucren aspectos de la red social. Como por ejemplo los seguidores de un usuario en Twitter. Tener en cuenta la posible necesidad de refactorizar ambas jerarquías para obtener una escalabilidad.
- Permitir que los usuarios se comuniquen con los desarrolladores (mediante mensaje directo a la cuenta oficial de Twitter de la biblioteca).

Aspectos a mejorar:

Implementar botón “cerrar sesión” que mate la sesión con la aplicación y redirecciónе a la página principal. Ahora está vinculado con Facebook, y para cerrar la sesión, hay que cerrar la sesión de Facebook.

Realizar PopUp en la eliminación del libro del estilo: ¿Desea eliminar “Libro Policial 1”?

- Baja y modificación de grupo.
- Cancelar una solicitud a un libro.
- Cancelar una solicitud a un grupo.