

UNIVERSIDAD DE ALCALÁ

Escuela Técnica Superior de Ingeniería Informática

Ingeniero Técnico en Informática de Sistemas



Proyecto Fin de Carrera

**SNSAngelGuard: Una aplicación extensible para el
control parental en Facebook**

Autor: José Javier Blecua de Pedro

Director: Miguel Angel Sicilia Urbán

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

CALIFICACIÓN:..... FECHA:.....

SNSAngelGuard: Una aplicación extensible para el control parental en Facebook

José Javier Blecua de Pedro

15 de julio de 2014

*"Querido George: Un hombre con amigos no es un fracasado"
Clarence, Qué bello es vivir*

A Marisa, allá donde estés, esto es para ti.

A Sandra, por todo el tiempo que la debo y que a partir de ahora podremos disfrutar. Por haberme animado tanto durante todo este tiempo y haber estado a mi lado. Por aguantarme a mí y a mis cambios de humor. Gracias por hacerme ver lo importante que es tener una persona a tu lado. Te quiero.

A mis compañeros de promoción. Porque sin ellos mi paso por la Universidad no habría sido lo mismo. Fuimos compañeros, y ahora somos amigos.

A mis hermanos, Estíbaliz, Lidia, Fátima, Guillermo, Alberto, Alvaro, María y el resto de la gran familia que somos. Porque no me imagino una vida sin vosotros.

A mi madre, Clotí. Por todo lo que me has dado, por estar tan pendiente de mí y hacer que me centre en lo que debo. Por apoyarme en cada paso de mi vida, aunque a veces no haya sido el más correcto. Para la mejor madre del mundo.

A mi padre, Francisco Javier. Por haberme enseñado todo en ésta vida, por ser mi verdadero ejemplo del camino de vida, por confiar en mí en los momentos más duros y por un día decirme que la constancia al final tiene su premio. Lo conseguí Papa!

Agradecimientos

A mis profesores, Miguel-Angel Sicilia y Salvador Sanchez. Por haberme guiado en éste camino.

A mis excompañeros del Laboratorio IERU. Por haberme enseñado tanto en los nueve meses que pasamos codo con codo.

A mi hermano Alvaro, por haberme echado una mano en la parte Abstract del presente libro. No siempre se tiene la suerte de tener un hermano bilingüe.

Resumen

SNSAngelGuard es un software desarrollado por la Universidad de Alcalá cuyo objetivo es controlar la actividad de una determinada persona o perfil dentro de la Red Social Facebook. Una figura denominada Tutor controlará, por medio de informes generados desde SNSAngelGuard, la actividad que un usuario pueda generar en dicha red social. Los filtros actuales dentro de la aplicación permiten controlar los mensajes que se escriben dentro del muro, la edad de amigos potencialmente peligrosos, su configuración de seguridad y el número de visitas que recibe una determinada persona o perfil de Facebook. Con ésta implementación se consigue monitorear potenciales indicadores de comportamientos peligrosos para el menor por terceras personas y a su vez, notificarse automáticamente a su Tutor. Esta aplicación se ejecutará en el entorno de Facebook y utilizará una base de datos externa que, mediante servicios RestFul, controlará el tráfico de datos del usuario y servirá como base de análisis a la aplicación.

Abstract

SNSAngelGuard is a software developed for the University of Alcala de Henares and it's objective is controling the activity of a determinate Facebook user or his/her profile within this social network. A feature called Tutor, will control, through generated researchs from SNSAngelGuard, the activity that a usser could develop in this social network. The current filters within the application allow the control of the messages posted on the wall, the age of the potential dangerous friends, the security settings and the number of visits this user or Facebook profile can recive. With this implementation the application takes control of possible harrasments by third people and at the same time, it communicates them automatically to the Tutor. This application will be run in the Facebook enviroment and it will use an external database which, by using RestFul services, will protect the usser data traffic and will work as a base for the application analysis.

Índice general

Agradecimientos	III
Resumen	V
Abstract	VII
1. Introducción.	17
1.1. Qué es SNSAngelGuard	17
1.2. Objetivos de SNSAngelGuard	18
1.2.1. Dotar de una herramienta a la víctima para que pueda compartir su información	18
1.2.2. Monotorizar las posibles amenazas a un perfil determinado	19
1.2.3. Preservar la identidad de los acosadores a terceras personas	19
1.2.4. Dotar de mecanismos de monotorización sin tener perfil en la Red Social	19
1.2.5. Actualización constante de nuevos ataques	19
1.2.6. Enviar notificaciones en forma de emails	19
1.2.7. La aplicación deberá ser multiidioma	19
1.2.8. Detectar los amigos aislados que pueda tener un perfil	19
1.2.9. Analizar la configuración de la privacidad de un usuario	20
1.3. Requisitos de SNSAngelGuard	20
1.3.1. Obtener la información personal de un perfil tras la aceptación de un acuerdo legal	21
1.3.2. Preservar la identidad de los acosadores	21
1.3.3. Preservar la información del usuario en las notificaciones	21
1.4. Concepto de Red Social	21
2. Estado del Arte	23
2.1. Introducción	23
2.2. Qustodio	23
2.3. ZoneAlarm SocialGuard	24
2.4. Qué aporta SNSAngelGuard	24
2.4.1. Aplicación de Facebook	25
2.4.2. Su configuración será responsabilidad del usuario	25
2.4.3. Información a terceros restringida	25

2.4.4. Multiidioma	25
2.4.5. Control integral modularizable	26
2.4.6. Notificaciones regulares	26
2.4.7. Los tutores pueden no pertenecer a la Red Social	27
2.4.8. Actores de la aplicación	27
2.5. OpenSocial	28
2.5.1. Estructura básica de una aplicación	28
2.5.2. Crear aplicaciones con OpenSocial	28
2.6. Facebook	29
2.6.1. Tipos de APIs	29
2.6.2. RestFB	30
2.7. Servicios Web	30
2.7.1. ¿Qué es un Servicio Web?	30
2.7.2. Arquitectura REST	32
2.7.2.1. Introducción	32
2.7.2.2. Principios básicos	32
2.7.3. Funcionalidad	34
3. Arquitectura de SNSAngelGuard	35
3.1. Introducción	35
3.2. Diseño estructural	35
3.2.1. Estructura de la base de datos	36
3.2.2. Módulo de comunicación con la base de datos: Interfaz RestFul	40
3.2.3. Módulo servidor	41
3.2.4. Módulo cliente	41
3.2.5. Modulo Offline	43
4. Base de Datos SocialNetwork	45
4.1. Introducción	45
4.2. Módulo funcional de Configuración	46
4.2.1. Tabla user_settings	47
4.2.2. Tabla locale_settings	47
4.2.3. Tabla settings_angels	47
4.2.4. Tabla settings_filter	48
4.3. Módulo funcional de Datos Comunes	48
4.3.1. Tabla user	49
4.4. Módulo funcional de OpenSocial	49
4.4.1. Tabla user_openSocial	49
4.4.2. Tabla Phones_openSocial	50
4.4.3. Tabla Smoker_openSocial	50
4.4.4. Tabla LookingFor_openSocial	51
4.4.5. Tabla Email_openSocial	51
4.4.6. Tabla Gender_openSocial	51
4.4.7. Tabla Presence_openSocial	52
4.4.8. Tabla Drinker_openSocial	52

4.4.9. Tablas de direcciones Address_openSocial y Organization_ openSocial	53
4.4.10. Tabla Friends_openSocial	53
4.4.11. Tabla url_openSocial	53
4.4.12. Tabla Name_openSocial	54
4.4.13. Tabla Message_openSocial	54
4.5. Módulo funcional de Facebook	54
4.5.1. Tabla user_facebook	55
4.5.2. Tabla work_history_facebook	55
4.5.3. Tabla education_history_facebook	55
4.5.4. Tabla friends_facebook	56
4.5.5. Tabla stream_facebook	56
4.5.5.1. Tabla likes_facebook	56
4.5.5.2. Tabla privacy_facebook	57
4.5.5.3. Tabla comments_facebook	57
4.5.5.4. Tabla action_links_facebook	57
4.5.6. Tabla affiliations_facebook	57
4.5.7. Tabla family_facebook	58
4.5.8. Tabla location_facebook	58
5. Interface RESTFul	59
5.1. Introducción	59
5.2. Casos de uso para la Interface RestFul	60
5.3. Projecto SNSdataBaseIntegratorServer	62
5.3.1. Paquete es.uah.cc.ie.service	63
5.3.2. Paquete es.uah.cc.ie.converter	65
5.3.3. Paquete es.uah.cc.ie.persistence	68
5.4. Interface RESTFul SNSdataBaseClient.java	69
5.4.1. Atributos de la clase SNSdataBaseClient.java	70
5.4.2. Métodos de la clase SNSdataBaseClient.java	70
5.4.2.1. Recurso userSettings	70
5.4.2.2. Recurso localeSettings	71
5.4.2.3. Recurso settingsFilter	72
5.4.2.4. Recurso settingAngels	72
5.4.2.5. Recurso user	72
5.4.2.6. Recurso userFacebook	74
5.4.2.7. Recurso userOpenSocial	75
6. Módulo Servidor	77
6.1. Introducción	77
6.2. Paquete es.uah.cc.ie.snsangelguardfb	78
6.2.1. Clase SNSAngelGuardFBManager.java	79
6.2.2. Clase ConfigurationManager.java	82
6.2.3. Interface ILifeCycleFilter	85
6.2.4. Paquete es.uah.cc.ie.snsangelguardfb.exception	85

6.2.4.1.	Enumerado CodeException.java	85
6.2.4.2.	Clase ExceptionManager.java	87
6.2.4.3.	Clase GenericException.java	87
6.2.4.4.	Clase InterDataBaseException.java	87
6.2.4.5.	Clase InterEmailException.java	87
6.2.4.6.	Clase InterProcessException.java	88
6.2.5.	Paquete es.uah.cc.ie.snsangelguardfb.facebookclient	88
6.2.5.1.	Clase GenericDataFacebook.java	90
6.2.6.	Paquete es.uah.cc.ie.snsangelguardfb.facebookclient.data	90
6.2.6.1.	Clase DatesAngelsFacebook.java	90
6.2.6.2.	Clase FacebookUrlStadistics.java	92
6.2.6.3.	Clase FacebookWallStadistics.java	93
6.2.6.4.	Clase FriendsFacebook.java	93
6.2.6.5.	Clase GetAppPropertiesFacebook.java	94
6.2.6.6.	Clase InfoCurrentUser.java	95
6.2.6.7.	Clase InfoUser.java	96
6.2.6.8.	Clase InfoUserFacebook.java	97
6.2.6.9.	Clase LocaleUserFacebook.java	99
6.2.6.10.	Clase PostWallFacebook	100
6.2.6.11.	Clase StreamCommentsFacebook	101
6.2.7.	Paquete es.uah.cc.ie.snsangelguardfb.facebookclient.clients	102
6.2.7.1.	Clase FacebookClientLocal.java	103
6.2.8.	Paquete es.uah.cc.ie.snsangelguardfb.googleclient	105
6.2.8.1.	Servlet GoogleContactsServlet	105
6.3.	Paquete es.uah.cc.ie.snsangelguardfb.sources	106
6.3.1.	Paquete es.uah.cc.ie.snsangelguardfb.sources.dao	108
6.3.1.1.	Clase LocaleSettingsDaoManager.java	108
6.3.1.2.	Clase UserSettingsDaoManager.java	111
6.3.2.	Paquete es.uah.cc.ie.snsangelguardfb.sources.dao.entity	116
6.3.2.1.	Clase LocaleSettingsDAO.java	117
6.3.2.2.	Clase UserSettingsDAO.java	119
6.3.2.3.	Clase UserSettings_SettingsFilterDAO.java	129
6.3.3.	Paquete es.uah.cc.ie.snsangelguardfb.sources.email	131
6.3.3.1.	Clase GenericEmailObject.java	131
6.3.3.2.	Clase EmailObject.java	133
6.3.4.	Paquete es.uah.cc.ie.snsangelguardfb.sources.filtersfuncionality	133
6.3.5.	Interface IKeyArgsFilter	134
6.3.5.1.	Clase GenericFilterFuncionality.java	136
6.3.5.2.	Clase WallFilterFuncionality.java	138
6.3.5.3.	Clase FriendsFilterFuncionality.java	139
6.3.5.4.	Clase PrivacyFilterFuncionality.java	142
6.3.5.5.	Clase VisitsFilterFuncionality.java	142
6.3.6.	Paquete es.uah.cc.ie.snsangelguardfb.sources.logs	144
6.3.6.1.	Fichero log4j.properties	145
6.3.6.2.	Clase Log4Init.java	146

6.3.7.	Paquete es.uah.cc.ie.snsangelguardfb.sources.offline	146
6.3.7.1.	Clase HarvestedSNS.java	146
6.3.8.	Paquete es.uah.cc.ie.snsangelguardfb.sources snswebserviceclient .	147
6.3.9.	Paquete es.uah.cc.ie.snsangelguardfb.sources.utilities	147
6.3.9.1.	Clase AngelsUtilities.java	148
6.3.9.2.	Clase DateTimeUtilities.java	150
6.3.9.3.	Clase JSONUtilities.java	150
6.3.9.4.	Clase StringUtilities.java	151
6.3.9.5.	Clase UserUtilities.java	152
6.4.	Paquete es.uah.cc.ie.snsangelguardfb.sources.jspcontroler	152
6.4.1.	Clase GenericJSPControler.java	153
6.4.2.	Paquete entity	153
6.4.2.1.	Clase CheckNowJSPControler.jsp	155
6.4.2.2.	Clase DeleteAngelSelectedJSPControler.java	155
6.4.2.3.	Clase DoOperationWithGoogleContactsJSPControler.java	158
6.4.2.4.	Clase DoOperationWithOtherContactsJSPControler.java	158
6.4.2.5.	Clase IndexJSPControler.java	158
6.4.2.6.	Clase LegalAcceptedJSPControler.java	159
6.4.2.7.	Clase SaveEmailFacebookContactsJSPControler.java . . .	159
6.4.2.8.	Clase SaveNewAngelJSPControler.java	160
6.4.2.9.	Clase SchedulerUserLoggedFacebookJSPControler.java .	160
6.4.2.10.	Clase SettingsSNSAngelGuardJSPControler.jsp	161
6.4.2.11.	Clase SettingsSNSAngelGuardJSPControler_Angels.jsp .	162
6.4.2.12.	Clase SettingsSNSAngelGuardJSPControler_Vigilants.jsp	162
6.4.2.13.	Clase TutorialInicioJSPControler.java	162
6.4.2.14.	Clase UpdateFacebookFriendsOnlineJSPControler.java .	163
6.4.3.	Paquete data	164
6.4.3.1.	Enumerado TypeOperationContact	164
6.4.4.	Paquete threads	164
6.4.4.1.	Clase ThProcessCheckFilter.java	164
6.4.4.2.	Clase ThUpdateInformationUser.java	165
6.4.5.	Paquete resources	165
6.4.5.1.	Clase LegalAcceptedJSPControlerResources.java	171
6.4.5.2.	Clase SettingsSNSAngelGuardJSPControlerResources.java	171
6.4.5.3.	Clase SettingsSNSAngelGuardJSPControlerResourcesAn-	
	gels.java	172
6.4.5.4.	Clase SettingsSNSAngelGuardJSPControlerResourcesAn-	
	gelsMenu.java	172
6.4.5.5.	Clase SettingsSNSAngelGuardJSPControlerResourcesJQuery-	
	MenuResources.java	172
6.4.5.6.	Clase SettingsSNSAngelGuardJSPControlerResourcesVi-	
	gilants.java	172
6.4.5.7.	Clase SettingsSNSAngelGuardJSPControlerResourcesVi-	
	gilantsMenu.java	173
6.4.5.8.	Clase TutorialInicioJSPControlerResources.java	173

6.5.	Operaciones	173
6.5.1.	Login en la aplicación	174
6.5.2.	Comprobación en la página de inicio	181
6.5.3.	Aceptación del acuerdo legal	182
6.5.4.	Guardar datos	182
6.5.5.	Confirmación de un ángel	184
7.	Módulo Cliente	187
7.1.	Introducción	187
7.2.	Organización del módulo	188
7.2.1.	Hojas de estilos	189
7.2.2.	Imágenes e iconos	189
7.2.3.	Ficheros de contenido JavaScript	190
7.2.4.	Ficheros de vocabulario	190
7.3.	Pantallas de la aplicación SNSAngelGuardFB	191
7.3.1.	Pantallas Principales	191
7.3.1.1.	Pantalla de Acuerdo Legal	192
7.3.1.2.	Pantalla de Configuración General	194
7.3.1.3.	Pantalla de Selección de Ángeles	196
7.3.1.4.	Pantalla de Configuración de Vigilantes	199
7.3.1.5.	Pantalla de Selección de contactos de Google	202
7.3.2.	Pantallas Secundarias	204
7.3.2.1.	Pantalla de Ayuda	205
7.3.2.2.	Pantalla de Resultado a la confirmación de un ángel	206
7.3.2.3.	Pantalla de Resultado de la operación	208
7.3.2.4.	Pantalla de Información de Error	209
7.3.2.5.	Pantalla de Identificación del usuario	211
7.4.	Formato de notificaciones de correo	213
7.4.1.	Notificación de confirmación de ángel	213
7.4.2.	Notificación de actividad	215
7.4.3.	Notificación de eliminación	216
8.	Módulo Offline	219
8.1.	Introducción	219
8.2.	Configuración de la tarea programada	221
8.2.1.	Fichero crontab	221
8.2.2.	Modificación del fichero crontab	222
8.3.	Script harvestedSNSAngelGuardFB.sh	222
8.4.	Aplicación harvestedSNSAngelGuard	223
8.5.	Servlet HarvestedSNS	224
8.5.1.	Método doGet()	224
8.5.2.	Método updateUsers()	226

9. Cómo extender SNSAngelGuardFB	229
9.1. Introducción	229
9.2. Añadir nuevos filtros a SNSAngelGuardFB	229
9.2.1. Implementación de la interface ILifeCycleFilter	229
9.2.2. Añadir una entrada al fichero de control de filtros	230
9.2.3. Añadir una imagen al nuevo filtro	231
9.2.4. Actualizar los recursos de idioma de la tabla locale_settings	231
9.3. Traducción de la aplicación a un nuevo idioma	232
9.3.1. Nuevo registro en la tabla locale_settings	232
9.3.2. Generación de un nuevo fichero lexico de idioma	232
9.3.3. Modificaciones en la clase UserSettingsDAO	233
9.3.4. Modificaciones en la clase WallFilterFuncionality	233
10. Trabajo a futuro y conclusiones	235
10.1. Trabajo a futuro	235
10.1.1. Implementación del filtro de configuración	235
10.1.2. Mejorar el algoritmo de control de lenguaje	235
10.1.3. Mejorar el mantenimiento de la aplicación	236
10.1.4. Filtro de fotografías	236
10.1.5. Filtro de chat	236
10.2. Conclusiones	236
A. SNSAngelGuardFB: Manual de usuario y configuración	239
A.1. Instalación	239
A.1.1. Requisitos técnicos	239
A.1.2. Instalando la base de datos SocialNetwork	239
A.1.3. Instalación del Servidor Web Apache Tomcat	240
A.1.3.1. Apache Tomcat	240
A.1.3.2. Modificación para el certificado digital	240
A.1.4. Creación del fichero de configuración config.properties	240
A.1.5. Creación del fichero de configuración de base de datos pool.properties	242
A.1.6. Creación del fichero de filtros activos filters.xml	243
A.1.7. Instalando harvestedSNSAngelGuard	243
A.1.7.1. Instalación del proyecto harvestedSNSAngelGuardFB	243
A.1.7.2. Creación de la carpeta de logs	243
A.1.7.3. Modificación del fichero harvestedSNSAngelGuardFB.sh .	244
A.1.7.4. Modificación del cron del sistema	245
A.1.8. Instalando SNSdataBaseIntegratorServer	245
A.1.9. Instalando SNSAngelGuardFB	245
A.2. SNSAngelGuardFB: Manual de usuario	246
A.2.1. Alta de un nuevo usuario	246
A.2.2. Acceso de usuarios ya dados de alta	262
A.2.3. Confirmación de un ángel y envío de notificaciones	263
B. Base de Datos SocialNetwork: Tablas	269

C. Interface RESTFul: Definición de métodos	289
C.1. Método userSettings_getEntities	289
C.2. Método userSettings_getUserByUidPublic	289
C.3. Método userSettings_setUpdateTime	290
C.4. Método userSettings_setNewEntityUserSettings	291
C.5. Método userSettings_getUserSettingsByUid	291
C.6. Método userSettings_setNewAngelsCollectionByUid	292
C.7. Método localeSettings_getLocaleSettingsByUid	292
C.8. Método settingsFilter_getFiltersByIdFilter	293
C.9. Método settingsFilter_getFiltersByUserSettingsUid	293
C.10. Método settingsFilter_setNewFilter	294
C.11. Método settingsFilter_updateFilterByIdFilter	294
C.12. Método settingsFilter_getAngelsCollectionByIdFilter	295
C.13. Método settingsAngels_setAngelByUid	295
C.14. Método settingsAngels_delAngelByUid	296
C.15. Método settingsAngels_setNewAngel	296
C.16. Método settingsAngels_getAngelsByPropUid	296
C.17. Método settingsAngels_getAngelsByUid	297
C.18. Método settingsAngels_getAngelsByUidFacebook	297
C.19. Método user_getEntities	298
C.20. Método user_getUserByUid	298
C.21. Método user_setNewUser	299
C.22. Método user_setUserByUid	299
C.23. Método userFacebook_getEntities	300
C.24. Método userFacebook_setNewUserFacebook	300
C.25. Método userFacebook_getUserFacebookByUid	300
C.26. Método userFacebook_setUserFacebookByUid	301
C.27. Método userFacebook_setNewStreamComentsByUid	301
C.28. Método userFacebook_getStreamComentByUid	302
C.29. Método userFacebook_setStreamComentsByUid	303
C.30. Método userFacebook_setComentsPost	303
C.31. Método userFacebook_getComentsPostById	304
C.32. Método userFacebook_getComentsPostByTime	304
C.33. Método userFacebook_getStreamFacebookByUid	305
C.34. Método userFacebook_getStreamFacebookByUpdatedTime	305
C.35. Método userFacebook_getFriendsFacebookByUidCollection	306
C.36. Método userFacebook_isNewFriendsFacebookByUid	306
C.37. Método userFacebook_getFriendsFacebookByUid	307
C.38. Método userFacebook_setNewFriendFacebook	307
C.39. Método userFacebook_setFriendsFacebookByUid	308
C.40. Método userOpenSocial_getEntities	308
C.41. Método userOpenSocial_setUserOpenSocialByUid	309

Listado de figuras

2.1. Arquitectura de la tecnología REST	32
3.1. Arquitectura de la aplicación SNSAngelGuard	36
3.2. Modelo Entidad-Relación de la base de datos de SNSAngelGuard, módulo OpenSocial	37
3.3. Modelo Entidad-Relación de la base de datos de SNSAngelGuard, módulo de configuración	38
3.4. Modelo Entidad-Relación de la base de datos de SNSAngelGuard, módulo Facebook	39
5.1. Arquitectura de la Interface RestFul	60
5.2. Diagrama de Casos de Uso de la Interface RestFul	61
5.3. Diagrama de paquetes del proyecto SNSdataBaseIntegratorServer	63
5.4. Diagrama de clases para el recurso UserSettings	76
6.1. Diagrama de despliegue de la aplicación SNSAngelGuardFB	79
6.2. Diagrama del paquete es.uah.cc.ie.snsangelguardfb	80
6.3. Diagrama del paquete es.uah.cc.ie.snsangelguardfb.facebookclient	91
6.4. Paquete es.uah.cc.ie.snsangelguardfb.sources	109
6.5. Paquete es.uah.cc.ie.snsangelguardfb.sources.dao	110
6.6. Paquete es.uah.cc.ie.snsangelguardfb.sources.filtersfuncionality	135
6.7. Paquete es.uah.cc.ie.snsangelguardfb.sources.utilities	148
6.8. Paquete es.uah.cc.ie.snsangelguardfb.jspcontroler	155
6.9. Paquete es.uah.cc.ie.snsangelguardfb.jspcontroler.entity	156
6.10. Paquete es.uah.cc.ie.snsangelguardfb.sources.jspcontroler.resources	170
6.11. Diagrama de operación de login	180
6.12. Diagrama de operación del inicio de la aplicación	181
6.13. Diagrama de operación de aceptación del acuerdo legal	183
6.14. Diagrama de operación de guardado datos	185
6.15. Diagrama de operación de la confirmación de un ángel	186
7.1. Estructura de organización del Módulo Cliente	188
7.2. Pantalla de Acuerdo Legal	192
7.3. Pantalla de Acuerdo Legal, por partes	193
7.4. Pantalla de Configuración General	194
7.5. Pantalla de Configuración General, por partes	195

7.6. Pantalla de Selección de Ángeles	196
7.7. Pantalla de Selección de Ángeles, por partes	198
7.8. Pantalla de Configuración de Vigilantes	199
7.9. Pantalla de Configuración de Vigilantes, por partes	200
7.10. Pantalla de Selección de contactos de Google	202
7.11. Pantalla de Selección de contactos de Google, por partes	203
7.12. Pantalla de Ayuda	205
7.13. Pantalla de Resultado a la confirmación de un ángel	206
7.14. Pantalla de Resultado a la confirmación de un ángel, por partes.	207
7.15. Pantalla de Resultado de la operación	208
7.16. Pantalla de Resultado de la operación, por partes.	209
7.17. Pantalla de Información de Error.	209
7.18. Pantalla de Información de Error, por partes.	210
7.19. Pantalla de Identificación del usuario	211
7.20. Pantalla de Identificación del usuario, por partes.	212
7.21. Notificación de confirmación de ángel.	213
7.22. Notificación de confirmación de ángel, por partes.	214
7.23. Notificación de actividad.	215
7.24. Notificación de actividad, por partes.	216
7.25. Notificación de eliminación de ángel.	217
7.26. Notificación de eliminación de ángel, por partes.	217
8.1. Diagrama de ejecución del Módulo Offline	220
8.2. Diagrama de actividad del método doGet()	225
8.3. Diagrama de actividad del método updateUsers()	227
A.1. Pantalla de Aceptación del Acuerdo Legal.	247
A.2. Pantalla de Aceptación del Acuerdo Legal, con loader.	248
A.3. Pantalla de Selección de Ángeles inicial.	249
A.4. Mensaje de aviso para seleccionar ángeles.	250
A.5. Mensaje de post en Facebook.	251
A.6. Ventana de selección de contactos de Gmail inicial.	252
A.7. Ventana de Login de Gmail.	253
A.8. Selección de un contacto de Gmail.	254
A.9. Pantalla de Selección de Ángeles con un contacto de Gmail.	255
A.10. Selección de un nuevo contacto manual habilitado.	256
A.11. Mensaje de confirmación para nuevo ángel introducido manualmente.	257
A.12. Habilitada una nueva estructura para introducir un nuevo ángel manualmente.	258
A.13. Pantalla inicial de Configuración de Vigilantes.	259
A.14. Configuración del filtro de Control de Muro.	260
A.15. Datos guardados correctamente.	261
A.16. Pantalla inicial para usuarios de la aplicación.	262
A.17. Notificación de confirmación de ángel.	263
A.18. Pantalla de Identificación del usuario	264

A.19.Pantalla de Identificación del angel de Facebook	265
A.20.Pantalla de Resultado a la confirmación de un ángel	266
A.21.Notificación de actividad.	267

Lista de tablas

5.1.	Tabla de Servicios RESTFul del recurso userSettings	71
5.2.	Tabla de Servicios RESTFul del recurso localeSettings	72
5.3.	Tabla de Servicios RESTFul del recurso settingsFilter	72
5.4.	Tabla de Servicios RESTFul del recurso settingsAngels	73
5.5.	Tabla de Servicios RESTFul del recurso user	73
5.6.	Tabla de Servicios RESTFul del recurso userOpenSocial	74
5.7.	Tabla de Servicios RESTFul del recurso userFacebook	74
6.1.	Atributos de SNSAngelGuardFBManager.java	81
6.2.	Métodos de SNSAngelGuardFBManager.java	82
6.3.	Atributos de ConfigurationManager.java	82
6.4.	Métodos de la clase ConfigurationManager.java	85
6.5.	Métodos de la clase ILifeCycleFilter	86
6.6.	Métodos de la clase ExceptionManager.java	87
6.7.	Atributos de la clase DatesAngelsFacebook.java	92
6.8.	Métodos de la clase DatesAngelsFacebook.java	92
6.9.	Atributos de la clase FacebookUrlStadistics.java	93
6.10.	Métodos de la clase FacebookUrlStadistics.java	93
6.11.	Atributos de la clase FacebookUrlStadistics.java	94
6.12.	Métodos de la clase FacebookUrlStadistics.java	94
6.13.	Atributos de la clase FriendsFacebook.java	94
6.14.	Métodos de la clase FriendsFacebook.java	95
6.15.	Atributos de la clase GetAppPropertiesFacebook.java	95
6.16.	Métodos de la clase GetAppPropertiesFacebook.java	95
6.17.	Atributos de la clase InfoCurrentUser.java	96
6.18.	Métodos de la clase InfoCurrentUser.java	96
6.19.	Atributos de la clase InfoUser.java	97
6.20.	Métodos de la clase InfoUser.java	97
6.21.	Atributos de la clase InfoUserFacebook.java	98
6.22.	Métodos de la clase InfoUserFacebook.java	99
6.23.	Atributos de la clase LocaleUserFacebook.java	100
6.24.	Métodos de la clase LocaleUserFacebook.java	100
6.25.	Atributos de la clase PostWallFacebook.java	101
6.26.	Métodos de la clase PostWallFacebook.java	102
6.27.	Atributos de la clase PostWallFacebook.java	102

6.28. Métodos de la clase StreamCommentsFacebook.java	102
6.29. Atributos de la clase FacebookClientLocal.java	105
6.30. Métodos de la clase FacebookClientLocal.java	106
6.31. Atributos de la clase GoogleContactsServlet.java	106
6.32. Métodos de la clase GoogleContactsServlet.java	107
6.33. Atributos de la clase LocaleSettingsDaoManager.java	110
6.34. Métodos de la clase LocaleSettingsDaoManager.java	110
6.36. Métodos de la clase UserSettingsDaoManager.java	111
6.35. Atributos de la clase UserSettingsDaoManager.java	117
6.37. Atributos de la clase LocaleSettingsDao.java	117
6.38. Atributos de la clase UserSettingsDAO.java	120
6.39. Métodos de la clase UserSettingsDAO.java	122
6.40. Atributos de la clase UserSettings_SettingsFilterDAO.java	130
6.41. Métodos de la clase UserSettings_SettingsFilterDAO.java	130
6.42. Atributos de la clase GenericEmailObject.java	132
6.43. Métodos de la clase GenericEmailObject.java	132
6.44. Atributos de la clase EmailObject.java	134
6.45. Métodos de la clase EmailObject.java	134
6.46. Atributos de la clase GenericFilterFuncionality.java	136
6.47. Métodos de la clase GenericFilterFuncionality.java	136
6.48. Atributos de la clase WallFilterFuncionality.java	138
6.49. Métodos de la clase WallFilterFuncionality.java	138
6.50. Atributos de la clase FriendsFilterFuncionality.java	140
6.51. Métodos de la clase FriendsFilterFuncionality.java	140
6.52. Atributos de la clase VisitsFilterFuncionality.java	142
6.53. Métodos de la clase VisitsFilterFuncionality.java	142
6.55. Métodos de la clase HarvestedSNS.java	146
6.54. Atributos de la clase HarvestedSNS.java	148
6.57. Métodos de la clase AngelsUtilities.java	148
6.59. Métodos de la clase DateTimeUtilities.java	150
6.61. Métodos de la clase JSONUtilities.java	150
6.63. Métodos de la clase StringUtilities.java	151
6.65. Métodos de la clase UserUtilities.java	152
6.56. Atributos de la clase AngelsUtilities.java	153
6.58. Atributos de la clase DateTimeUtilities.java	153
6.60. Atributos de la clase JSONUtilities.java	154
6.62. Atributos de la clase StringUtilities.java	154
6.64. Atributos de la clase UserUtilities.java	155
6.66. Atributos de la clase CheckNowJSPControler.jsp	156
6.67. Métodos de la clase CheckNowJSPControler.jsp	157
6.68. Atributos de la clase DeleteAngelSelectedJSPControler.jsp	158
6.69. Metodos de la clase DeleteAngelSelectedJSPControler.jsp	158
6.70. Atributos de la clase DoOperationWithGoogleContactsJSPControler.java .	159
6.71. Metodos de la clase DoOperationWithGoogleContactsJSPControler.java .	160
6.72. Atributos de la clase DoOperationWithOtherContactsJSPControler.java .	160

6.73. Metodos de la clase DoOperationWithOtherContactsJSPControler.java	161
6.74. Atributos de la clase IndexJSPControler.java	161
6.75. Metodos de la clase IndexJSPControler.java	161
6.76. Atributos de la clase LegalAcceptedJSPControler.java	162
6.77. Metodos de la clase LegalAcceptedJSPControler.java	162
6.78. Atributos de la clase SaveEmailFacebookContactsJSPControler.java	163
6.79. Metodos de la clase SaveEmailFacebookContactsJSPControler.java	163
6.80. Atributos de la clase SaveNewAngelJSPControler.java	164
6.81. Metodos de la clase SaveNewAngelJSPControler.java	164
6.82. Atributos de la clase SchedulerUserLoggedFacebookJSPControler.java	165
6.83. Metodos de la clase SchedulerUserLoggedFacebookJSPControler.java	165
6.84. Atributos de la clase SettingsSNSAngelGuardJSPControler.jsp	166
6.85. Métodos de la clase SettingsSNSAngelGuardJSPControler.jsp	166
6.86. Atributos de la clase SettingsSNSAngelGuardJSPControler_Angels.jsp	167
6.87. Métodos de la clase SettingsSNSAngelGuardJSPControler_Angels.jsp	168
6.88. Atributos de la clase SettingsSNSAngelGuardJSPControler_Vigilants.jsp	169
6.89. Métodos de la clase SettingsSNSAngelGuardJSPControler_Vigilants.jsp	170
6.90. Atributos de la clase TutorialInicioJSPControler.java	171
6.91. Metodos de la clase TutorialInicioJSPControler.java	171
6.92. Atributos de la clase UpdateFacebookFriendsOnlineJSPControler.java	172
6.93. Metodos de la clase UpdateFacebookFriendsOnlineJSPControler.java	172
6.94. Atributos de la clase ThProcessCheckFilter.java	173
6.95. Metodos de la clase ThProcessCheckFilter.java	173
6.96. Atributos de la clase ThUpdateInformationUser.java	174
6.97. Metodos de la clase ThUpdateInformationUser.java	174
6.98. Atributos de la clase LegalAcceptedJSPControlerResources.java	175
6.99. Métodos de la clase LegalAcceptedJSPControlerResources.java	175
6.100Atributos de la clase SettingsSNSAngelGuardJSPControlerResources.java	176
6.101Métodos de la clase SettingsSNSAngelGuardJSPControlerResources.java	176
6.102Atributos de la clase SettingsSNSAngelGuardJSPControlerResources.java	177
6.103Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesAngels.java	177
6.104Atributos de la clase SettingsSNSAngelGuardJSPControlerResourcesAn-	
gelsMenu.java	177
6.105Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesAngels-	
Menu.java	177
6.106Atributos de la clase SettingsSNSAngelGuardJSPControlerResourcesJQuery-	
MenuResources.java	178
6.107Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesJQuery-	
MenuResources.java	178
6.108Atributos de la clase SettingsSNSAngelGuardJSPControlerResources.java	179
6.109Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesAngels.java	179
6.110Atributos de la clase SettingsSNSAngelGuardJSPControlerResourcesVigi-	
lantsMenu.java	179
6.111Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesVigi-	
lantsMenu.java	179

B.2. Tabla locale_settings	269
B.1. Tabla user_settings	272
B.3. Tabla settings_angels	273
B.6. Tabla user_openSocial	273
B.4. Tabla settings_filter	275
B.5. Tabla user	275
B.7. Tabla Phones_openSocial	276
B.8. Tabla Smoker_openSocial	276
B.9. Tabla LookingFor_openSocial	276
B.10. Tabla Email_openSocial	276
B.11. Tabla Gender_openSocial	276
B.12. Tabla Presence_openSocial	277
B.13. Tabla Drinker_openSocial	277
B.14. Tabla Address_openSocial	277
B.15. Tabla Organization_openSocial	278
B.16. Tabla Friends_openSocial	278
B.17. Tabla url_openSocial	278
B.18. Tabla Name_openSocial	279
B.19. Tabla Message_Type_openSocial	279
B.20. Tabla Message_openSocial	279
B.21. Tabla user_facebook	280
B.22. Tabla work_history_facebook	282
B.23. Tabla education_history_facebook	282
B.25. Tabla stream_facebook	282
B.24. Tabla friends_facebook	284
B.26. Tabla likes_facebook	284
B.27. Tabla friends_likes_facebook	284
B.28. Tabla privacy_facebook	284
B.29. Tabla comments_facebook	285
B.30. Tabla comment_facebook	285
B.31. Tabla action_links_facebook	285
B.32. Tabla affiliations_facebook	286
B.33. Tabla type_affiliations_facebook	286
B.34. Tabla family_facebook	286
B.35. Tabla relationship_facebook	287
B.36. Tabla location_facebook	287

Capítulo 1

Introducción.

1.1. Qué es SNSAngelGuard

El término **Bullying** o **acoso escolar** se define como cualquier forma de maltrato psicológico, verbal o físico producido entre escolares, de forma reiterada o a lo largo de un tiempo determinado, normalmente en su entorno escolar. **Cyberbullying: el acoso a través de las redes sociales**, (Cabal, 2014).

Cuando éste maltrato se produce en el ámbito de las redes sociales, blogs, mensajería instantánea, websites, teléfonos móviles, .etc, éste fenómeno se conoce como **Cyberbullying** o **Ciberacoso**.

Muchos autores inciden en la importancia de éste tipo de acosos. Muchos lo consideran más importante que en el caso del acoso convencional, *El acoso escolar no es nuevo pero, gracias a Internet, los adolescentes están siendo acosados en casa, What is Cyberbullying?*, (Hardcastle, 2014). La Universidad de Gotemburgo también corrobora esta visión en su artículo *Cyberbullying: A growing problem*, (of Gothenburg, 2010).

Los investigadores inciden en la importancia de Internet como medio para desarrollar acosos, ya que en éste contexto se podrían realizar acciones que en otros medios no serían permitidos. *Sería impensable para nosotros ver una revista en la que el título fuera 'Quien de vosotros odia a Stina Johansson'*, (of Gothenburg, 2010).

Las consecuencias de éste tipo de acoso pueden marcar el resto de la vida de la víctima, deteriorando su autoestima, provocándole una tendencia a la introversión, angustia, depresiones y fracaso escolar.

Este tipo de ataques suelen tener estas consecuencias principalmente por la poca implicación de los agentes sociales en la vida de las víctimas:

1. Los adolescentes que sufren éste tipo de ataques no suelen compartirlo con su familia.
2. Existe poca convicción en que los colegios o institutos puedan ayudar a las víctimas en su ámbito de acción.

3. Muchos padres consideran 'normales' este tipo de ataques.

Bajo este contexto, necesitamos herramientas para realizar estudios sobre el **Ciberacoso** y que, por un lado, puedan preservar la dignidad de la víctima y por otro, sean capaces de informar al entorno de la víctima de la manera menos intrusiva posible.

SNSAngelGuard es una aplicación software diseñada en la Universidad de Alcalá en colaboración con otras universidades a nivel nacional con el objetivo de crear una aplicación que permita profundizar en el amplio universo de las redes sociales. Este proyecto forma parte de un proyecto global cuyo objetivo es desarrollar herramientas para controlar y monitorear posibles acosos o ataques en una Red Social, sea cual fuere su naturaleza. En este proyecto se pretenden dos objetivos:

1. Construcción de un modelo de datos general para el conjunto de redes sociales existentes, haciendo un estudio de aquellas partes en las que dichas redes sociales pueden llegar a un punto en común en el almacenamiento de datos.
2. Diseñar y desarrollar una aplicación que extraiga datos de una determinada Red Social y que trate dichos datos con una funcionalidad determinada. En éste caso, la red social a tratar será Facebook.

El **objetivo primordial** de éste proyecto será construir una aplicación totalmente modularizable que sea capaz de detectar posibles acosos a un perfil de una Red Social y los monitoree por medio de informes que serán enviados a los tutores de una aplicación para que éstos sean conscientes de la actividad social de sus tutelados y tomen decisiones al respecto.

1.2. Objetivos de SNSAngelGuard

SNSAngelGuard ha sido diseñado para denunciar las posibles amenazas que un determinado perfil de una Red Social pueda sufrir por parte de acosadores desconocidos e informar a las personas responsables de éste, para que siempre estén en contacto con la actividad social del perfil y puedan tomar decisiones respecto a estos ataques ya que, el usuario afectado, ya sea por causas ajenas a su voluntad, no suele revelar estas amenazas hasta que son demasiado flagrantes para no hacerles frente.

1.2.1. Dotar de una herramienta a la víctima para que pueda compartir su información

Con ésta premisa, conseguimos dos objetivos:

1. Dotar a los padres o tutores legales de un adolescente de mecanismos para que puedan implicarse en la actividad de un adolescente dentro de una Red Social.
2. Solucionar la posible animadversión, por parte del adolescente, a compartir sus posibles problemas de **Ciberacoso**.

1.2.2. Monotorizar las posibles amenazas a un perfil determinado

La aplicación debe en todo momento ser consciente de las amenazas que sufra un determinado perfil. Para ello, se diseñarán módulos específicos que analicen parte de la información y detecten posibles amenazas en diferentes sectores de la información. Estos módulos tomarán el nombre de **filtros** o **vigilantes**.

1.2.3. Preservar la identidad de los acosadores a tercera personas

Los datos serán analizados y se enviarán informes a los tutores de la aplicación. Nunca se identificarán posibles acosadores en las notificaciones que los tutores reciba, por lo que nunca se proporcionará información a terceros.

1.2.4. Dotar de mecanismos de monotorización sin tener perfil en la Red Social

Los tutores podrán recibir informes aunque no formen parte de la Red Social que se esté analizando para un determinado usuario de la aplicación.

1.2.5. Actualización constante de nuevos ataques

La aplicación, debido a su arquitectura modularizable, deberá estar siempre preparada para posibles nuevas formas de acoso, implementando o actualizando **vigilantes** que puedan ser útiles para ésta labor.

1.2.6. Enviar notificaciones en forma de emails

Los informes de actividad de los filtros configurados por cada usuario serán enviados por medio de emails a los tutores de la persona en cuestión, siendo la periodicidad de éstos totalmente configurable para cada **vigilante**.

1.2.7. La aplicación deberá ser multiidioma

La aplicación deberá estar traducida y preparada para que, personas de diferentes idiomas, puedan utilizarla de forma automática.

1.2.8. Detectar los amigos aislados que pueda tener un perfil

Las Redes Sociales parten de la base proporcionada por el teorema de los **Seis Grados de Separación**, formulado por primera vez en el libro *Chains* (Karinthy, 1930), en el que se postula que *cualquier persona puede estar conectado a cualquier otra persona del planeta a través de una cadena de conocidos que no tiene más de cinco intermediarios* (Crespo, 2011).

A partir de éste teorema podemos probar la fiabilidad de los contactos de un usuario de una Red Social ya que, lo que puede probar es el hecho de que para poder estar

conectado con personas del otro lado del planeta, debemos tener contactos con amigos en común para poder formar parte de uno de los seis eslabones de la cadena. Si un usuario tiene contactos poco relacionados puede ser considerados como una posible amenaza, ya que los acosadores suelen crear perfiles nuevos para atacar a sus posibles víctimas, por lo que no son muy populares en cuanto a amigos se trata. Se deberá detectar si un usuario tiene contactos **aislados**, es decir, contactos que no están relacionados con el resto de contactos que forman parte del grupo de amigos de un usuario, para poder ser estudiados y detectados como una posible amenaza para el usuario de una Red Social.

1.2.9. Analizar la configuración de la privacidad de un usuario

Cualquier usuario ha 'regalado' datos a una Red Social. Cada vez que generamos un nuevo estado, subimos una foto o comentamos cualquier muro de un amigo, estamos generando una información que es almacenada en los servidores de una Red Social. Esta información, en el momento en que la producimos, deja de ser nuestra y pasa a ser propiedad de la Red Social. Incluso si damos de baja nuestra cuenta, esa información seguirá persistida en los servidores de información y puede recorrer toda la red sin ninguna restricción, por lo que cualquier usuario podrá acceder a ella.

En una Red Social, una de los primeros objetivos que debemos cumplir es definir qué o quién puede acceder a nuestra información y a nuestros datos personales. Si no se definen estos parámetros, podemos encontrarnos sorpresas en cuanto a la vulneración de nuestra privacidad sin apenas darnos cuenta. La mayoría de los usuarios no tienen un conocimiento pleno para poder configurar las opciones de privacidad de una Red Social, por lo que un acosador puede valerse de éste asunto y acceder a cualquier información de un usuario sin tener éste conocimiento de ello. Esto puede derivar en ataques o acosos indeseables a cualquier usuario que no esté protegido convenientemente.

Muchos investigadores se han hecho eco de estos riesgos y han publicado estudios identificando los posibles riesgos a los que están sometidos los usuarios de una Red Social, tal como se muestra en el estudio *What Anyone Can Know: The Privacy Risks of Social Networking Sites* (Rosenblum, 2007), en el cual se enumeran todas las posibles amenazas que un usuario puede recibir a partir de una mala configuración de su privacidad dentro de la Red Social.

Será uno de los objetivos de **SNSAngelGuard** el detectar una posible configuración poco fiable de privacidad y alertar a los tutores de un usuario de éste asunto, para que ellos, en última instancia, puedan tomar medidas de cualquier tipo para mejorar el control del acceso a la información de un usuario.

1.3. Requisitos de SNSAngelGuard

La aplicación **SNSAngelGuard** debe tener una serie de premisas a la hora de tratar la información que puede ser considerada como potencialmente peligrosa. Estas premisas parten de la base de información relativa a los acosadores y a los propios usuarios.

1.3.1. Obtener la información personal de un perfil tras la aceptación de un acuerdo legal

La aplicación no podrá obtener ningún dato personal del usuario hasta que éste acepte un acuerdo legal formulado para proteger su información y cumplir con la *Ley Orgánica de Protección de Datos* (B.O.E., 1999). Sin éste acuerdo aceptado, la aplicación será inutil y **en ningún caso podrá descargar ni almacenar ningún dato propio del usuario sin su consentimiento.**

1.3.2. Preservar la identidad de los acosadores

En el momento de analizar la información, **SNSAngelGuard** podrá detectar irregularidades en la información del usuario y marcarla como posible ataque. En el momento en el que se envían las notificaciones a los tutores del usuario informando de éstas irregularidades, **no se podrá informar las identidades de los usuarios que hayan podido realizar ataques o acosos a los tutores**, ya que no es posible proporcionar información relativa a los contactos de un usuario a terceras personas.

El objeto de ésta premisa es el aviso a los tutores de que el usuario está siendo acosado, sin llegar a revelar la identidad de su atacante, y a partir de ése momento, será el tutor quien tome las decisiones oportunas para solventar éste acoso.

1.3.3. Preservar la información del usuario en las notificaciones

Los datos que se obtienen de una Red Social son propios de ésta y pueden estar en formatos preparados según sus tablas de información. Estos datos, si llegan a manos inadecuadas, pueden usarse fraudulentamente para atacar al usuario propietario de éstos.

Será un requisito de la aplicación **SNSAngelGuard** el **no mostrar ningún dato del usuario propio de la Red Social salvo su información de identificación**, tales como su nombre y apellidos, **en las notificaciones enviadas a los tutores de éste**.

1.4. Concepto de Red Social

El fenómeno de las redes sociales ha revolucionado nuestro concepto de relación social clásica y nuestra inversión en tiempo libre. En ellas buscamos contactos con aquellos con los que perdimos trato, mantenemos amistades, nos ponemos al día de la vida de los demás, conocemos a gente nueva o incluso encontramos trabajo. Las Redes Sociales. Tipología, uso y consumo de las redes 2.0 en la sociedad digital actual (Domínguez, 2010).

Como concepto, una Red Social puede entenderse como un ente formado por un conjunto de personas que comparten algún tipo de relación (familiares, amigos, trabajo, ocio..., etc.) o simplemente, por personas que quieren compartir su conocimiento para llevar a

cabo un objetivo común. Dentro de una Red Social, cada persona puede comunicarse con las personas que forman su entorno, además de tener nuevas relaciones y darse a conocer dentro de otros entornos que a su vez, todos juntos, conformen dicha red.

Las redes sociales tienen su origen en la teoría de los **Seis Grados de Separación**, formulado por primera vez en el libro *Chains* (Karinthy, 1930), en el que se postula que *cualquier persona puede estar conectado a cualquier otra persona del planeta a través de una cadena de conocidos que no tiene más de cinco intermediarios* (Crespo, 2011). Dicho de otra forma, las redes sociales se basan en el simple hecho de que el número de conocidos crece exponencialmente con el número de enlaces de la cadena, y sólo un pequeño número de enlaces son necesarios para que el conjunto de conocidos se convierta en una población humana entera.

Si ponemos ésta idea en práctica, podemos comprobar que no es nada descabellada. Cada persona se suele relacionar con aproximadamente unas 100 personas, entre familiares, amigos, compañeros de trabajo,..., etc. Si cada una de estas personas se relaciona con otras 100 personas no comunes, no es difícil imaginarse que una persona puede difundir un mensaje entre 10000 personas. Si seguimos recorriendo eslabones en la cadena de relaciones, en unos cuantos pasos estaríamos preparados para mandar un mensaje a cualquier persona que estuviera conectada a una Red Social. Dicho de otra manera, estaríamos preparados para entrar en contacto con cualquier persona del planeta.

El verdadero impacto que tiene una Red Social sobre la sociedad actual moderna no tiene límites. Nos encontramos ante un fenómeno de masas, con una capacidad de comunicación imposible de calcular. Por esta razón, el estudio de las Redes Sociales se ha convertido en un elemento esencial para las Universidades y Centros de Conocimiento y, sin ir más lejos, para todas aquellas empresas, tanto grandes, medianas o pequeñas, que quieran expandirse y dar a conocer sus productos. Cualquier departamento de marketing incorpora especialistas en redes sociales para calcular el impacto de sus productos en la sociedad y difundirlos de la manera más eficaz posible.

Evidentemente, todas estas relaciones serían imposibles sin un vehículo idóneo de comunicación. Gracias al desarrollo tecnológico del sector de la informática y de las telecomunicaciones, cualquier persona puede disponer de un equipo informático moderadamente asequible y una conexión a Internet lo suficientemente rápida como para poder estar en constante comunicación con otra, y poder enviar y recibir mensajes simultáneamente, como si ésta comunicación se produjera “cara a cara”.

Capítulo 2

Estado del Arte

2.1. Introducción

En éste capítulo se hará un estudio de las herramientas que ejercen un control parental en Facebook hasta la fecha y lo que aporta la aplicación **SNSAngelGuardFB** a éste mercado. Incluimos también un estudio de dos herramientas de control parental que actualmente hay en el mercado y, por último, se profundizará en las Redes Sociales, que serán objeto de estudio en el documento y de las tecnologías que se utilizarán para llevar a cabo la construcción de la aplicación.

2.2. Qustodio

Qustodio es una herramienta de control parental. Es una aplicación de escritorio que controla toda la actividad de un adolescente en Internet por parte de sus padres. Controla toda la actividad web de un usuario, supervisando las páginas que visita, con quien se escribe o quienes son sus amigos.

Las funcionalidades que dispone son las siguientes:

1. Ofrece una supervisión social para proteger a los adolescentes de abusos y depredadores online.
2. Puede bloquear contenidos inapropiados.
3. Puede limitar el tiempo que un adolescente pasa en internet o en sus dispositivos electrónicos.
4. Puede localizar geográficamente la ubicación del móvil de la persona que se está controlando.
5. Permite conocer con quién se escriben y llaman las personas que se están controlando.
6. Prevenir que se acceda a páginas adultas.

7. Proteger y supervisar más de un adolescente.
8. Ver y controlar las actividades de los adolescentes desde cualquier lugar (modo web).
9. Instalar y operar de manera invisible.

Como podemos observar, es una herramienta de control bastante completa. Tiene muchas ventajas pero, la principal desventaja es que es una herramienta bloqueante, es decir, controla todos los contenidos y puede restringir contenidos a los adolescentes. También parte de la premisa, otra de las desventajas principales, de que la configuración se realiza por parte de los padres o tutores legales de la persona a controlar, por lo que ésta no tiene ningún control sobre la herramienta, incluso se puede estar ejecutando en sus dispositivos (modo invisible) sin que ellos sean conscientes.

2.3. ZoneAlarm SocialGuard

ZoneAlarm SocialGuard es uno de los plugins proporcionados por el antivirus **ZoneAlarm**. Puede controlar la actividad en Facebook de un usuario por parte de sus padres.

Como nuestra aplicación, también dispone de filtros de control de actividad, que son los siguientes:

1. Control del lenguaje, previniendo a los padres del vocabulario ofensivo que puedan estar sufriendo sus hijos en Facebook.
2. Control del rango de fechas del nacimiento de sus amigos.
3. Control de la configuración.
4. Control de la edad real del perfil que se está controlando.

En éste caso, la aplicación cuenta con filtros que se adaptan mejor a nuestra aplicación, pero cuenta con dos desventajas. La primera es esencial, la aplicación se basa en un control en tiempo real, es decir, si no estás utilizando la aplicación, ésta no generará ni monitorizará ningún informe. La segunda es más importante aún, también del estilo de **Qustodio**, y es que la aplicación sigue siendo configurada por el tutor del adolescente, no por él mismo.

2.4. Qué aporta SNSAngelGuard

Actualmente, todos las aplicaciones están orientadas a arquitecturas específicas más que a Redes Sociales. Facebook es consciente de ésta situación y está creando mecanismos para que se puedan desarrollar aplicaciones que permitan este tipo de controles. Hasta ahora, los controles parentales existentes se dedicaban a un campo específico (controlar

los mensajes del muro, organizar los contactos, etc...) y estaban orientados a un único idioma, por lo que están muy limitados y es un campo aún por explorar.

En ésta sección estudiaremos qué aporta a éste mundo nuestra aplicación, explicando paso por paso las razones que nos han motivado para llevar a cabo este desarrollo.

2.4.1. Aplicación de Facebook

La aplicación será ejecutada desde el entorno de aplicaciones de Facebook, **AppCenter**. Sólo será posible su ejecución si el usuario en cuestión posee una cuenta de Facebook, por lo que es requisito indispensable.

2.4.2. Su configuración será responsabilidad del usuario

La responsabilidad de la configuración de la aplicación y de los contenidos que desea mostrar a sus padres o cualquier otra persona de su entorno corresponderá al **propio usuario**. Ésta es una de las desventajas de las aplicaciones que hemos analizado con anterioridad y que, en nuestro caso, se convertirá en ventaja.

El usuario siempre tendrá el control total de la aplicación, desactivando ésta en cualquier momento si ya no quiere compartir más información o su utilidad ya no es necesaria. Con ésto se consigue que el propio usuario decida qué contenido quiere compartir y cuáles no, dotándole de independencia y sin que haya terceras personas restringiéndole su actividad.

El papel de los tutores se verá reducido únicamente a recibir la información que el usuario crea conveniente que deban conocer, respetando siempre su privacidad y sin mostrar información a terceros en las notificaciones que los tutores reciban.

2.4.3. Información a terceros restringida

En ningún caso, la aplicación mostrará datos de terceras personas implicadas en casos de acoso del usuario en las notificaciones que los tutores reciban. Únicamente se les avisará de que hay contenido no apropiado en la actividad del usuario en cuestión.

2.4.4. Multiidioma

SNSAngelGuard es capaz de procesar cualquier tipo de idioma. La aplicación está preparada para ser traducida en cualquier idioma con sólo incluir la traducción en la base de datos. Así, si en algún momento se desea ampliar el mercado en otros países, se podrá tomar el idioma original en que se construyó la aplicación, es decir, en castellano, y traducirlo para un idioma en concreto. Además, los controles de idioma llevan asociados un fichero de vocabulario propio de éste, así que para controlar expresiones malintencionadas en cualquier otro idioma bastará con incluir un fichero con éstas expresiones en el idioma deseado.

Actualmente se dá soporte a dos idiomas:

1. Castellano: Idioma oficial de la aplicación.
2. Inglés: Idioma por defecto. Si el idioma del usuario no es castellano, la aplicación se ejecutará en inglés.

2.4.5. Control integral modularizable

Desde la aplicación, vamos a tener centralizado todos los aspectos y funcionalidades que deseemos. En la actualidad, se controlan cuatro aspectos:

1. **Control de los mensajes del muro:** Se controlará si en los mensajes del muro del usuario existe vocabulario ofensivo.
2. **Control de amigos:** Se controlarán todas las edades de los contactos del usuario y se detectará cuales de ellos exceden la edad mínima aconsejable como contacto para un usuario.
3. **Control de privacidad:** Se detectará si el perfil tiene una configuración demasiado permisiva para su privacidad.
4. **Control de visitas:** Se controlará, por medio de estadísticas de los mensajes de muro y el número de contactos del usuario, cuales de éstos tienen una actividad importante dentro de su perfil, obteniendo el numero de mensajes de un contacto determinado o detectando si el usuario tiene contactos aislados, siendo potencialmente considerados como amenaza.

Además de toda ésta funcionalidad, el programa será capaz de ser ampliado con nuevas funcionalidades, no siendo acotado a una en concreto, sino que está preparado mediante una arquitectura modularizable para incluir nuevos filtros según las necesidades de los usuarios.

2.4.6. Notificaciones regulares

Las posibles anomalías que se detecten en un determinado perfil se enviarán regularmente a los tutores del usuario de forma regular. La aplicación **SNSAngelGuard** cuenta con un mecanismo **offline** al margen del módulo principal de la aplicación que se ejecuta periódicamente en el servidor. Con ésto conseguimos que la información de la base de datos siempre esté actualizada con la última información producida en la Red Social por parte del usuario y a la hora de ejecutar los filtros se utilice ésta. El módulo **offline** realiza las siguientes acciones:

1. Obtiene la nueva información del usuario en la Red Social y la almacena en la base de datos.
2. Ejecuta los filtros para detectar anomalías en la información del usuario o posibles vulnerabilidades de ésta.
3. Si se detectan anomalías, se enviará, mediante correos electrónicos de la aplicación, notificaciones con el detalle de las anomalías detectadas.

2.4.7. Los tutores pueden no pertenecer a la Red Social

En la aplicación **SNSAngelGuard** un tutor puede no pertenecer a la misma Red Social que el usuario que tutela. La aplicación es capaz de almacenar contactos externos a la Red Social, aparte de los propios usuarios de ésta, y enviarles notificaciones de información.

Actualmente, para el caso de **Facebook**, los contactos que pueden ser elegidos como tutores pueden ser de los siguientes tipos:

1. Contactos de Facebook: La aplicación pintará todos los amigos actuales del usuario en la aplicación y podrán ser elegidos cualquiera de ellos. Notesé que para ser contacto de éste tipo, anteriormente el contacto de Facebook deberá haber utilizado la aplicación.
2. Contactos de una cuenta de Google: El programa será capaz de obtener los contactos de una cuenta de Google logueada por el propio usuario y elegir de ella los contactos que se van a utilizar de tutores.
3. Cualquier otra cuenta de correo electrónico: El programa habilitará estructuras para introducir manualmente una cuenta de correo electrónico válida para que puedan llegarle las notificaciones producidas por la aplicación.

Además, el número de contactos es ilimitado y se podrán configurar dependiendo de los filtros que se vayan a ejecutar, es decir, se pueden elegir contactos distintos para el control de los mensajes del muro o el resto de filtros.

2.4.8. Actores de la aplicación

En la aplicación se configurarán dos actores funcionales indispensables, que serán las que definan el comportamiento de la aplicación:

1. **Ángeles**: Son las figuras seleccionadas como tutores. Si aceptan controlar al usuario de la aplicación, recibirán informes de su actividad en una frecuencia definida por éste. Como hemos introducido anteriormente, pueden ser de tres tipos:
 - a) Contactos de Facebook.
 - b) Contactos de Gmail.
 - c) Direcciones de email de cualquier otro tipo.
2. **Vigilantes**: Son filtros software que analizan la información del usuario y mandan informes a sus ángeles. Cada vigilante controlará un tipo de información y estará parametrizado de forma totalmente independiente al resto, es decir, para cada filtro se podrá elegir a qué ángeles del usuario se le enviarán notificaciones y con qué frecuencia. La aplicación no estará delimitada a un número máximo de vigilantes, pudiendo ser éstos incluidos de forma flexible por los desarrolladores de la aplicación.

2.5. OpenSocial

OpenSocial¹(Wikipedia, 2007) es un conjunto de API comunes destinadas a la creación de aplicaciones sociales en múltiples sitios web. OpenSocial está compuesto por API de JavaScript y API de datos de Google. La existencia de este modelo de programación único resulta de gran utilidad tanto para los desarrolladores como para los sitios web.

En primer lugar, los desarrolladores sólo tienen que aprender las API una vez para crear aplicaciones que funcionen con cualquier sitio web compatible con OpenSocial.

En segundo lugar, como cualquier sitio web puede implementar OpenSocial, los desarrolladores disponen de una amplia red de distribución para llegar a los usuarios. Los sitios web también se benefician mediante la participación de un conjunto mucho más numeroso de desarrolladores externos que el que podrían conseguir sin un conjunto estándar de API.

La Compañía Google y sus asociados, ofrecen algunas tecnologías para que Internet en su conjunto llegue a ser un medio más social, respondiendo así al claro interés de los usuarios. Productos, como orkut, son sólo uno de los distintos sitios web que implementan OpenSocial. Actualmente, el código de ejemplo se ofrece con la licencia de Apache 2.0. Además, las licencias de toda la documentación de OpenSocial proceden de Creative Commons, por lo que se puede reutilizar y combinar los servicios como estimes oportuno. En el futuro, se plantea ofrecer el software libre de los componentes necesarios para ejecutar OpenSocial en tu propio sitio web.

2.5.1. Estructura básica de una aplicación

Las aplicaciones de OpenSocial utilizan la estructura de gadgets de Google, pero con extensiones que proporcionan acceso programático a datos sociales dentro de su entorno de contenedor. De forma similar a los gadgets de Google, las aplicaciones de OpenSocial alojan documentos XML con lenguaje HTML/JavaScript integrado. Las aplicaciones sociales disponen de la mayor parte de la infraestructura de los gadgets de Google, pero con algunas pequeñas excepciones. Uno de los primeros entornos de las aplicaciones sociales que utilizan las API de OpenSocial es orkut. Se espera que otros sitios web compatibles con OpenSocial admitan pronto la participación de desarrolladores.

2.5.2. Crear aplicaciones con OpenSocial

Las aplicaciones sociales se crean en principio de la misma forma que los gadgets de Google: con tu editor de texto favorito o con el Editor de gadgets de Google. A continuación, se pueden aumentar con las API JavaScript de OpenSocial, donde estas aplicaciones pueden obtener y enviar datos sociales sobre amigos y actividades.

¹OpenSocial es un servicio de Google, Yahoo!, MySpace y otros muchos asociados, que mediante un conjunto de APIs permite construir aplicaciones o redes sociales. De tal manera que la forma de programar aplicaciones para diferentes sitios de redes sociales sea común. Fuente: <http://es.wikipedia.org/wiki/OpenSocial>

2.6. Facebook

Facebook(Zuckerberg, 2002) es el ejemplo de Red Social, mundialmente conocido. Facebook es una aplicación que engloba una conjunto de redes sociales creada por Mark Zuckerberg. Se trata de una aplicación web en la cualquier usuario puede acceder a ella a través de únicamente una dirección de correo válida. En ella, el usuario puede ponerse en contacto con otros usuarios de una o más redes sociales, dependiendo de su situación académica, su lugar de trabajo o su región geográfica.

Originalmente se creó específicamente para estudiantes de Harvard, Universidad en la que inició sus estudios su creador, pero posteriormente, tras su éxito a nivel mundial, fue ampliada a cualquier persona que tuviera una dirección de correo electrónico.

Una de las características principales de Facebook es su expansión hacia el mundo de las aplicaciones web. Se ha convertido en una plataforma en la que terceros pueden desarrollar aplicaciones y hacer negocio a partir de la red social.

Para conseguir dicho objetivo, el equipo de Facebook ha ido publicando diferentes APIs, basadas en Servicios Web, para la creación de aplicaciones y el acceso a datos de usuario, las cuales van adaptándose a las tecnologías actuales y marcando un perfil de aplicación determinado. Cualquier desarrollador tiene acceso a ellas y constituyen una de las mayores riquezas actuales de la compañía.

Actualmente, un desarrollador puede empezar una aplicación enfocándola directamente sobre el tipo de dispositivo para el que va dirigido. Las interfaces publicadas por Facebook así lo establecen, dejando a la elección del programador el tipo de aplicación que desea realizar, tales como páginas web, aplicaciones para dispositivos móviles o aplicaciones para escritorio.

2.6.1. Tipos de APIs

Actualmente, Facebook dispone de las siguientes interfaces para el desarrollo de aplicaciones:

1. Login: Es una interfaz que integra los métodos de logueo necesarios para una aplicación web. Puede ser utilizada desde cualquier lenguaje de programación pero es excesivamente sencillo de utilizar en el lenguaje JavaScript.
2. GraphApi: Es la interfaz más utilizada y más universal. Permite acceder a los datos de un usuario por medio de objetos o entidades que pueden ser tratadas posteriormente a través de la aplicación.
3. FQL(Developers, 2002): Son las siglas de Facebook Query Language o, dicho de otra manera, es el lenguaje de SQL de Facebook para el acceso a datos de usuario.
4. Legacy REST API: Es el conjunto de servicios REST que Facebook puso en disposición para el acceso a datos mediante peticiones HTTP. Es el más antiguo de todos y está próximo a la desaparición.

2.6.2. RestFB

En la actualidad, el creciente volumen de desarrolladores que trabajan en proyectos de aplicaciones de Facebook ha desencadenado una oleada de apariciones de arquitecturas o proyectos que encapsulan el acceso a las interfaces ofrecidas por Facebook y hacen más fácil trabajar con esta. Con estas nuevas arquitecturas, se ofrece el acceso a toda la funcionalidad de Facebook mediante clases de cualquier lenguaje de programación. RestFB es una de estas arquitecturas.

RestFB(RestFB, 2009) engloba las APIs de Facebook Graph API y Old REST API mediante un cliente escrito en el lenguaje de programación JAVA. Tiene las siguientes características:

1. Es totalmente transparente a la interfaz que se utiliza para el acceso a datos de Facebook. El programador no conoce con certeza qué interfaz se está utilizando.
2. Su api es mínima, ya que engloba una serie de métodos públicos que son configurables utilizando las sentencias FQL proporcionadas por Facebook, por lo que los métodos definidos en sus interfaces son realmente pocos.
3. Los cambios que se pueden producir por parte de Facebook en su interfaz son automáticamente absorbidos, haciendo que el usuario no cambie su forma de desarrollar la aplicación.
4. Para configurarlo, no hay más que introducir su dependencia Maven correspondiente en el pom.xml del proyecto.
5. No guarda dependencias con ningún otro módulo.
6. El transporte de datos se realiza mediante lenguaje XML o JSON, pudiendo ser parseado al llegar a la aplicación dependiendo de las necesidades funcionales del módulo.

Por todas estas razones, esta arquitectura ha sido la elegida para llevar a cabo las interacciones con Facebook desde SNSAngelGuard.

2.7. Servicios Web

2.7.1. ¿Qué es un Servicio Web?

El consorcio W3C¹ define a los Servicios Web como sistemas software diseñados para soportar una interacción interoperable maquina a máquina sobre la red. Dichos servicios suelen ser presentados en forma de APIs Web que pueden ser accedidas en una red (principalmente Internet) por medio de una aplicación y se suelen ejecutar en la máquina que los aloja.

¹World Wide Web Consortium: Comunidad internacional que desarrolla estándares que aseguran el crecimiento de la Web a largo plazo. Fuente www.w3c.es

Hay muchas definiciones para éstos servicios pero la idea general se refiere a clientes y servidores que se comunican entre sí mediante mensajes XML² los cuales siguen el estándar SOAP³.

En estos últimos años, se ha popularizado una arquitectura de Servicio Web denominada REST⁴. Esta arquitectura se ha convertido en una nueva opción para decantarse por tecnología basada en Servicios Web. En éste caso, tenemos tres opciones:

1. Llamadas a Procedimientos Remotos (RPC, Remote Procedure Call): Estos servicios presentan una interfaz de llamada a procedimientos y funciones distribuidas, lo cual es muy familiar a la mayoría de los desarrolladores. Normalmente, la unidad básica de uso de éste servicio es la operación WSDL⁵.

Las primeras herramientas para Servicios Web estaban centradas en éste tipo de enfoque, es por esta razón que son denominados Servicios Web de primera generación y su uso está muy extendido. Sin embargo, ha sido ampliamente criticado y muchos expertos creen que debería desaparecer, ya que su implementación se basa en el mapeo de servicios directamente a funciones específicas del lenguaje o llamadas a métodos.

2. Arquitectura Orientada a Servicios SOA⁶: Según la arquitectura de llamadas a Procedimientos Remotos, la unidad básica de operación es la Operación. En el caso de los Servicios SOA, se trata de implementar Servicios Web a través del mensaje, que en éste caso será la unidad básica de operación. Esta arquitectura también es conocida como Servicio Orientado a Mensajes.
3. Servicios REST: Son Servicios Web que intentan basarse en protocolos HTTP o similares. Su objetivo es restringir el uso de una interfaz a un conjunto de operaciones estándar universalmente conocidas (GET, PUT, POST, DELETE). Por lo tanto, esta arquitectura se basa en interactuar con recursos con estado en vez de mensajes u operaciones.

²Son las siglas de Extensible Markup Language, una especificación/lenguaje de programación desarrollada por W3C. XML es una versión de SGML, diseñado especialmente para documentos de la red. Permite que los desarrolladores creen sus propias etiquetas, permitiendo la definición, transmisión, validación e interpretación de datos entre aplicaciones y entre organizaciones. Fuente: <http://www.w3.org/XML/>

³Son las siglas de Simple Object Access Protocol. Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Fuente: <http://www.w3.org/TR/soap/>

⁴Son las siglas de Representational State Transfer. Fuente http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

⁵Son las siglas de Web Service Description Language, un formato XML que se utiliza para describir Servicios Web. Describe la interfaz pública de un Servicio Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de mensaje necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje. Fuente: <http://www.w3.org/TR/wsdl>

⁶Son las siglas de Service Oriented Architecture. Fuente: http://en.wikipedia.org/wiki/Service-oriented_architecture

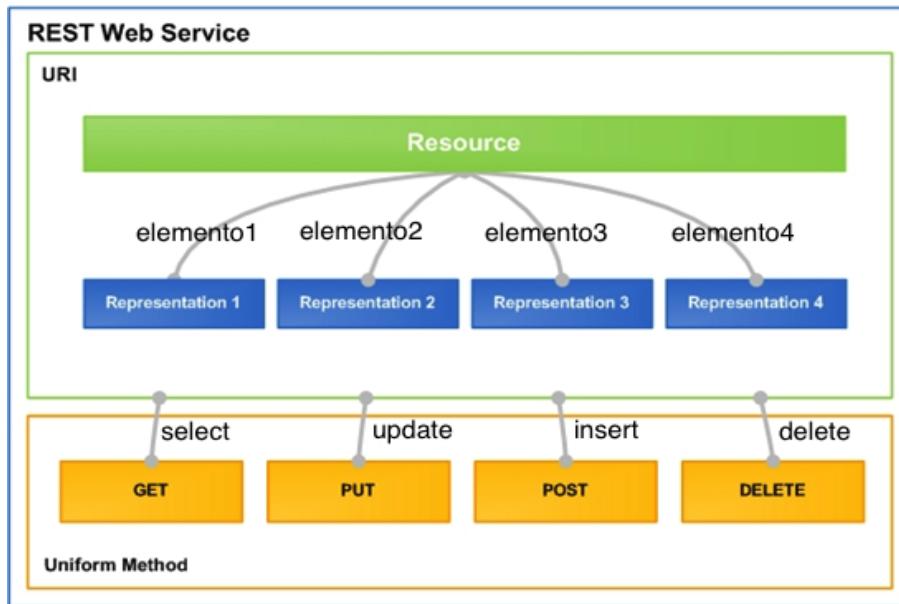


Figura 2.1: Arquitectura de la tecnología REST

2.7.2. Arquitectura REST

2.7.2.1. Introducción

REST(Fielding, 2000) es un estilo de arquitectura Software diseñado para el intercambio de datos en sistemas distribuidos. Fue introducido en la tesis doctoral de Roy Fielding en el año 2000, el cual es uno de los principales autores de la especificación HTTP.

Los servicios REST se basan en el concepto de “Todo recurso (información) debe tener y ser accesible mediante una URI única”. A partir de éste concepto, se usan los métodos de comunicación web sobre HTTP (GET, PUT, POST, DELETE) para definir diferentes acciones predefinidas sobre los recursos “URIs”. Las características básicas de los servicios REST son los siguientes:

1. Todo recurso debe tener una URI única.
2. Los recursos son enlazados entre sí.
3. Uso de tecnología estándar (HTTP, XML, JSON, ..., etc).
4. El recurso puede tener múltiples representaciones dependiendo de la solicitud.

La figura 1.1 representa la arquitectura de la tecnología REST:

2.7.2.2. Principios básicos

1. Escalabilidad de la interacción con los componentes: Un determinado sitio web puede crecer exponencialmente sin degradar su rendimiento. Varios sistemas pueden acceder y utilizar servicios REST simultáneamente sin que esto degrade el rendimiento de la aplicación.

2. Interfaces estándar: Gracias al protocolo HTTP, cualquier cliente puede interactuar con un servicio REST sin necesidad de una configuración especial, únicamente definiendo la URI característica del propio recurso
3. Funcionamiento independiente de la arquitectura: Al ser una arquitectura basada en protocolos HTTP y estándares de intercambio de mensajes, permite adaptarse a cualquier tipo de cliente, ya que HTTP permite la extensibilidad mediante el uso de cabeceras a través de las URIs.
4. Es compatible con sistemas intermedios: Pueden ser utilizados en combinación con proxys para Web, herramientas de mejora de la seguridad, como firewalls y con herramientas de encapsulado para la Web, como es el caso de los gateways.

REST logra conseguir estos objetivos aplicando las siguientes restricciones:

1. Identificación de recursos y manipulación de ellos a través de representaciones. Esto se consigue a partir de URIs. HTTP es un protocolo centrado en URIs. Los recursos son objetos lógicos que intercambian mensajes con las entidades que requieran sus servicios. No pueden ser directamente accedidos o modificados, más bien se trabaja con representaciones de ellos, es decir, cuando invocamos al método PUT para modificar los datos de un recurso, realmente estamos enviándole como mensaje una representación de lo que debería ser. Internamente, el recurso es totalmente transparente, es decir, puede desde un registro de una entidad de base de datos, un fichero plano...
2. Mensajes autodescriptivos. REST establece que los mensajes HTTP deben ser tan descriptivos como sea posible. Esto hace que los intermediarios interpreten los mensajes y ejecuten los servicios en nombre del usuario. HTTP logra este objetivo por medio de una definición de un método estándar (GET, PUT, POST, DELETE), muchas cabeceras y un método de direccionamiento. Por ejemplo, las cachés Web saben que, por defecto, el método GET es cacheable, sin embargo, el método POST no lo es. Además, saben cómo interpretar la información contenida en las cabeceras, es por esta razón por la cual, cuando una aplicación accede a un recurso de una base de datos por medio de un servicio REST, el navegador es capaz de obtener el usuario el cual intenta acceder y modificar dicha base de datos por medio de su cabecera.
3. Hipermedia como mecanismo de estado de la aplicación. El estado actual de una aplicación web es capturado mediante archivos de hipertexto que son alojados tanto en el cliente como en el servidor de la aplicación. El servidor conoce en todo momento el estado de los recursos pero no intenta seguir la pista a las sesiones individuales de cada cliente. Este trabajo es realizado por el navegador, quien sabe navegar de recurso en recurso, recogiendo la información que él necesita para cambiar de un estado a otro dependiendo del servicio REST que esté ejecutando en cada momento.

2.7.3. Funcionalidad

La funcionalidad básica de un servicio web se encuentra en el diseño de una interfaz que defina los métodos de acceso a los recursos para el resto de aplicaciones desde el servidor de la aplicación. Tras el diseño, esta interfaz se hará pública y las aplicaciones podrán hacer uso de todos sus recursos aplicando las restricciones que se han planteado anteriormente. Esta es precisamente, la arquitectura que se ha definido para SNSAngelGuard y que se procederá a describir en los capítulos posteriores.

Capítulo 3

Arquitectura de SNSAngelGuard

3.1. Introducción

SNSAngelGuard es una aplicación concebida como medio para realizar un ejemplo práctico sobre las investigaciones que pueden desarrollarse en una Red Social. En el presente documento, la red elegida es Facebook. Como se ha mencionado anteriormente, la base de datos de éstas investigaciones fue diseñada para albergar varias aplicaciones en las que estuvieran involucradas tambien redes sociales tales como Tuenti, Twitter, Open Social, ..., etc, por lo que la arquitectura de la base de datos no es exclusiva de SNSAngelGuard, pero sí que es totalmente compatible con Facebook.

3.2. Diseño estructural

La arquitectura se muestra en la figura 3.1.

Está conformada por una serie de módulos que se enumeran a continuación:

1. Base de Datos SocialNetwork: Contendrá todas las estructuras necesarias para acceder a cualquier funcionalidad de una Red Social. Guardará datos relacionados con su actividad, tales como datos a nivel personal, como pueden ser su fecha de nacimiento, su dirección, ..., etc., como datos a nivel de la propia Red Social, tales como sus amistades, sus relaciones personales con más miembros de dicha Red Social, sus comentarios en el muro, sus réplicas, enlaces a sus fotos o videos, ..., etc.
2. Interface RESTFUL: Este ente contendrá todas las estructuras necesarias para acceder a la base de datos por medio de Servicios Web. A través de éstas estructuras se creará una API que posteriormente se publicará para que todo aquel desarrollador que quiera utilizar esta base de datos pueda hacerlo como si se tratara de una API de una Red Social.
3. Módulo Servidor: Este módulo definirá una serie de estructuras que procesarán la información obtenida de los usuarios de la Red Social y, posteriormente, realizará las acciones propias de la aplicación, tales como el procesado de la información y la

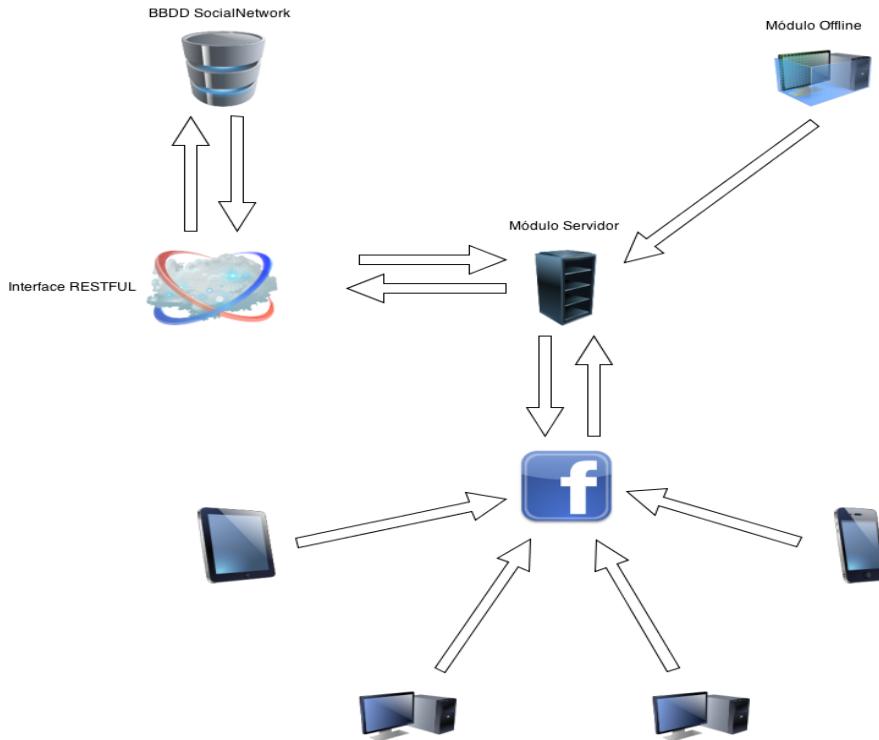


Figura 3.1: Arquitectura de la aplicación SNSAngelGuard

elaboración de informes para, posteriormente, ser enviados a los tutores del usuario en la aplicación.

4. Módulo Cliente: Está conformado por todos los dispositivos que pueden acceder a la Red Social y, a partir de ésta, ejecutar la aplicación SNSAngelGuard. Recordemos que ésta aplicación se ejecuta a través de Facebook, por lo que para acceder a ella, el usuario tendrá que tener un usuario válido dentro de ésta Red Social. Este módulo además, estará preparado para que cualquier dispositivo, sea cual fuere su naturaleza, pueda acceder a la aplicación y generar actividad a través de ella.
5. Módulo Offline: Este módulo, como su propio nombre indica, se ejecutará o podría ejecutarse, cuando un usuario no esté logueado dentro de Facebook. Estará conformado por una serie de procesos que realizan backups programados de la información del usuario en Facebook y elaborarán informes con la periodicidad indicada para cada tutor.

3.2.1. Estructura de la base de datos

El modelo Entidad Relación que seguirá la base de datos es la que muestran las figuras 3.2, 3.2 y 3.4.

La base de datos tiene tres estructuras bien definidas:

3.2. DISEÑO ESTRUCTURAL

37

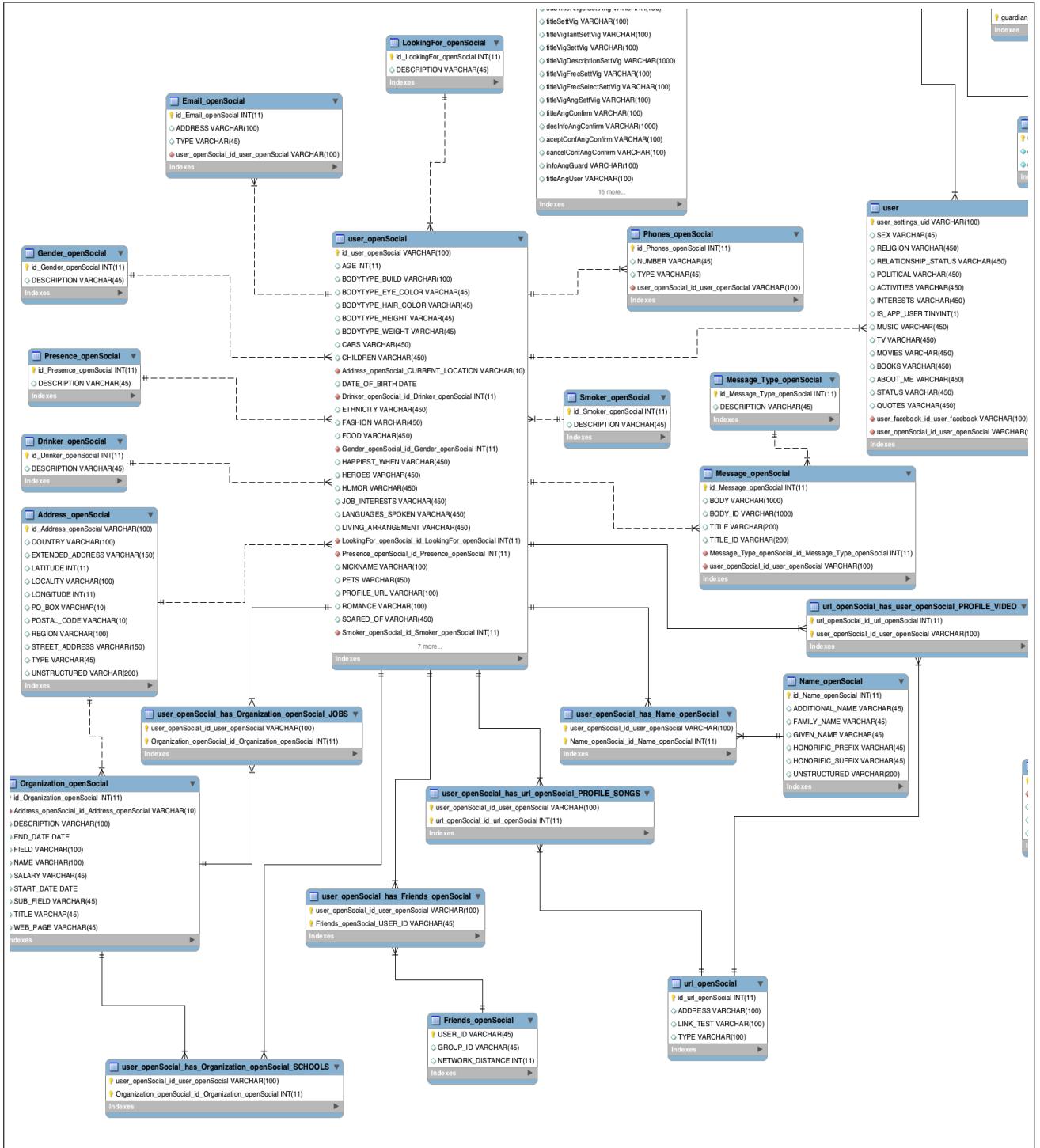


Figura 3.2: Modelo Entidad-Relación de la base de datos de SNSAngelGuard, módulo OpenSocial

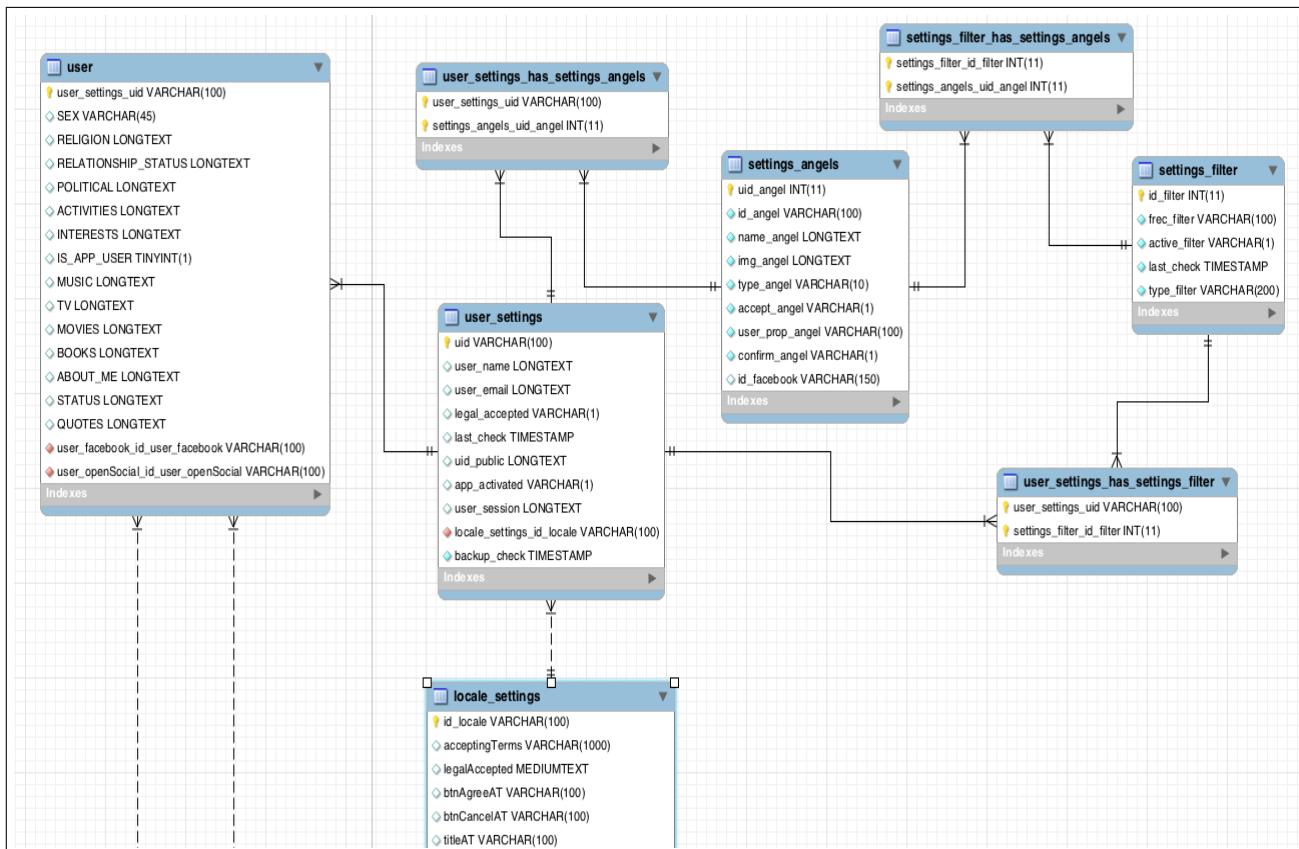


Figura 3.3: Modelo Entidad-Relación de la base de datos de SNSAngelGuard, módulo de configuración

3.2. DISEÑO ESTRUCTURAL

39

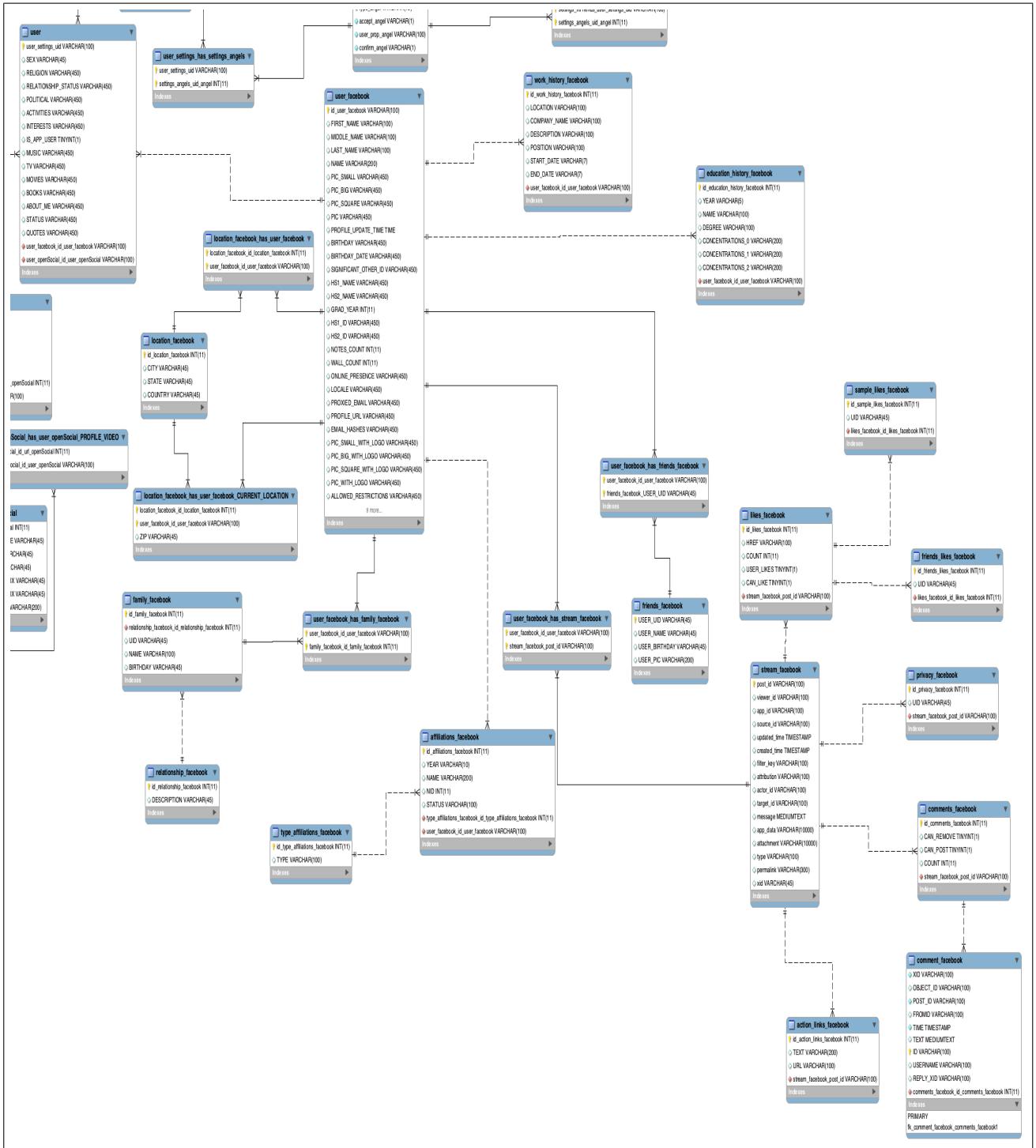


Figura 3.4: Modelo Entidad-Relación de la base de datos de SNSAngelGuard, módulo Facebook

1. Módulo de configuración de SNSAngelGuard: En este módulo se especificarán las estructuras necesarias para la configuración de la aplicación. Contendrá estructuras para guardar los filtros¹ que un usuario desee aplicar a su información, los ángeles² y la periodicidad con que éstos recibirán los informes generados por la aplicación.
2. Módulo propio de Facebook: Este módulo contendrá todas las estructuras necesarias para almacenar la información de un usuario que genera en Facebook y que será necesaria para el funcionamiento de la aplicación.
3. Módulo para otras Redes Sociales: Éste módulo contendrá las estructuras necesarias para almacenar información de otras redes sociales. Se diseñó utilizando el standar de Red Social proporcionado por OpenSocial y podrá ser adaptado a cualquier Red Social para la que se deseé realizar un estudio determinado.

El desarrollo de ésta base de datos ha sido realizado íntegramente bajo el entorno MySQL versión 5. Para la administración, se ha utilizado el software MySQLAdmin bajo plataforma UNIX, la cual puede ser utilizada tanto en sistemas Linux, como en sistemas de la familia Macintosh.

3.2.2. Módulo de comunicación con la base de datos: Interfaz RestFul

Como se ha indicado anteriormente, el módulo de comunicación con la que el Módulo Servidor se comunicará con la base de datos, será implementado a través de Servicios Web. Para ello, se describirán las siguientes estructuras:

1. Módulo de Servicios Web: Será un proyecto web independiente que dará soporte a la base de datos. En él se definirán todas las estructuras necesarias para dar soporte a los servicios web que posteriormente se utilizarán en la aplicación SNSAngelGuard.
2. API propia RESTFUL: Se definirá una clase que contendrá todos los accesos a los servicios web definidos en el Módulo anterior. La razón por la que se definirá esta estructura es para que cualquier usuario, desde cualquier punto del planeta, pueda utilizar la base de datos mediante una llamada a un método remoto que le dará los datos que desee, sin tener que construir una estructura propia para el acceso a la base de datos ni tener que montar en su equipo la base de datos. Con ésta decisión se garantiza el acceso a cualquier usuario que tenga permisos para ello y la consistencia y persistencia de los datos que en ésta se almacenan, ya que ningún usuario tendrá una copia de la base de datos en su equipo, sino que trabajará remotamente con la base de datos que se defina a nivel global y cuyos parámetros de acceso serán comunicados al usuario que la quiera utilizar.

¹Aplicaciones que analizarán la actividad social de un determinado usuario y generarán informes que serán enviados a sus ángeles.

²Tutores a quienes van dirigidos los informes de la actividad social de un determinado usuario.

3.2.3. Módulo servidor

El módulo servidor será la parte de la aplicación que se dedicará a procesar toda el negocio de la aplicación. Tendrá toda la responsabilidad de la aplicación y su funcionalidad. Sus objetivos serán los siguientes:

1. Controlar el acceso a la aplicación y gestionar la configuración de un usuario.
2. Gestionar los datos personales del usuario que se extraerán y que serán utilizados en la aplicación.
3. Analizar los datos extraídos por medio de filtros y elaborar informes.
4. Enviar los informes a los ángeles del usuario, quienes serán, en último término, los encargados de garantizar y supervisar la actividad social de un usuario en Facebook.

Este módulo además será el encargado de asumir las siguientes responsabilidades:

1. Garantizar el acceso único a la información de un usuario desde la propia aplicación.
2. Garantizar la privacidad de la información almacenada en la base de datos Social-Network, al igual que administrarla y gestionarla para garantizar la ejecución de los filtros definidos.
3. Analizar toda la actividad social de un usuario y comunicar cualquier anomalía definida en los filtros a sus ángeles.
4. Mantener en todo momento informado a los ángeles que se han configurado para un determinado usuario.
5. Garantizar la seguridad de la información que se envía desde la propia aplicación hasta servidores web externos a ella, tales como Gmail o Facebook.

Este módulo será diseñado y construido en su totalidad en el lenguaje de programación Java, combinando tanto el paradigma de Programación Orientada a Objetos, como la Programación Web en parte servidor, utilizando para ello las estructuras necesarias proporcionadas por dicho lenguaje de programación.

3.2.4. Módulo cliente

El módulo cliente será el encargado de la interfaz gráfica por el que el usuario realizará la configuración de la aplicación, seleccionando los ángeles que supervisarán su actividad social y los filtros que analizarán su información y enviarán informes de ello a dichos ángeles. Los objetivos de éste módulo serán los siguientes:

1. Mostrar al usuario una interfaz amigable y atractiva.
2. Integrar dicha interfaz en Facebook como una aplicación más que se ejecute desde el perfil de usuario.

3. Mostrar todos y cada uno de los amigos de Facebook para que puedan ser seleccionados por el usuario como sus ángeles.
4. Habilitar accesos desde el interfaz para poder seleccionar contactos desde cuentas de correo Gmail.
5. Habilitar las estructuras básicas para poder introducir manualmente un ángel, mediante cuadros de texto para su nombre y su dirección de correo.
6. Mostrar y controlar todos los filtros disponibles que puedan ser configurados por el usuario, mostrando además una pequeña descripción relativa a los mismos.

A parte de éstos objetivos, el módulo cliente se enfrentará a las siguientes responsabilidades:

1. Mostrar correctamente todas las estructuras necesarias para el correcto funcionamiento de la aplicación.
2. Llevar a cabo todas las validaciones de datos necesarias para que al almacenar la información de configuración, los datos estén preparados para ser procesados por el módulo servidor.
3. Controlar, en todo momento, que el usuario actual sea el único que puede acceder a su propia información de configuración.
4. Habilitar estructuras para el almacenado de la información.
5. Habilitar estructuras que informen visualmente al usuario sobre posibles errores o advertencias en el proceso de configuración.
6. Habilitar estructuras visuales de ayuda que sirvan para aclarar posibles dudas a la hora de guardar la configuración.

Todo este módulo será conformado por todas las herramientas web disponibles en la actualidad. Las principales que se han utilizado son las siguientes:

1. Lenguaje HTML¹: Será el encargado de realizar toda la parte visual y el almacenado de la información subyacente en la arquitectura.
2. Lenguaje JavaScript²: Será el encargado de realizar todas las validaciones necesarias anteriores al envío de la información y de controlar el contenido dinámico de la aplicación según las preferencias del usuario a la hora de configurar la aplicación.

¹HyperText Markup Language, hace referencia al lenguaje de marcado predominante para la elaboración de páginas web, que se utiliza para describir y traducir la estructura y la información en forma de texto, así como complementar el texto con objetos tales como imágenes o contenido multimedia. Fuente: <http://es.wikipedia.org/wiki/HTML>

²Es un lenguaje de programación interpretado, que se ejecuta en la parte de cliente en una arquitectura común de Cliente-Servidor. Se utiliza implementado mediante el navegador web, permitiendo mejoras en la interfaz de usuario y en páginas web dinámicas, cuyo contenido puede variar. Fuente: <http://es.wikipedia.org/wiki/JavaScript>

3. Lenguaje JSON³: Realizará la tarea de intercambiar información entre el módulo cliente y el módulo servidor. También será el encargado de transferir información desde la base de datos al módulo servidor y viceversa.
4. AJAX⁴: Será el encargado de realizar la navegación entre páginas web pertenecientes al módulo cliente y realizar las transferencias de información entre pestañas integradas en una única página web.
5. jQuery⁵: Será el encargado del acceso a los elementos html de las páginas web y agrupar funcionalidad JavaScript, haciendo el código más rápido y legible.
6. JSP⁶: Será el encargado de embeber las páginas html y controlar el flujo de ejecución con el servidor, realizando todas las validaciones y las transferencias de información de un módulo a otro.

3.2.5. Modulo Offline

El Módulo Offline será el encargado de realizar los procesos de carga, validación y análisis de datos de forma automática. Se ejecutará una o varias veces al día, como tarea programada en el servidor, y realizará las siguientes acciones:

1. Obtener, de la base de datos SocialNetwork, la información relativa a todos los usuarios de la aplicación.
2. Por cada usuario realizará las siguientes acciones:
 - a) Descargará la nueva información que haya generado en Facebook, tal como nuevos comentarios en el muro o nuevas amistades.
 - b) Dependiendo de la periodicidad con que tenga configurados los filtros, realizará un análisis de la nueva información obtenida.
 - c) Tras realizar el análisis, informará a los ángeles los resultados obtenidos.
 - d) Actualizará en base de datos la fecha y la hora del momento en el que se ha realizado el proceso de actualización de la información.

³JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML. Fuente: <http://es.wikipedia.org/wiki/JSON>

⁴Asynchronous JavaScript And XML, JavaScript anasíncrono y XML, es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones. Fuente: <http://es.wikipedia.org/wiki/AJAX>

⁵jQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Fuente: <http://es.wikipedia.org/wiki/Jquery>

⁶JavaServer Pages es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo, permitiendo la inclusión de código Java que se ejecutará directamente en servidor. Fuente: <http://es.wikipedia.org/wiki/JSP>

Este módulo será vital para mantener la base de datos actualizada con la última información del usuario y para realizar todas las tareas programadas por los filtros. Será desarrollado en su totalidad en Java y contendrá dos pequeños módulos:

1. Módulo Servidor: Éste módulo se ejecutará en el servidor web de aplicaciones mediante una conexión establecida por una tarea programada. Cuando le llega ésta llamada, pondrá en marcha todo el procedimiento descrito en el apartado anterior.
2. Tarea Programada: Dependerá del servidor o máquina física en el que se ejecute el servidor de aplicaciones. Se programará temporalmente una o varias veces diarias para poder realizar los backups de mantenimiento de la información. Será el desencadenante del proceso y sin él el Módulo Offline no se ejecutaría.

Capítulo 4

Base de Datos SocialNetwork

4.1. Introducción

Para la realización del proyecto SNSAngelGuard se ha requerido el diseño de una base de datos que persista todos los elementos necesarios para el análisis y el procesado de la información. Como se ha indicado en los capítulos anteriores, la base de datos se diseña para albergar datos no sólo de Facebook, sino también de otras redes sociales, como pueden ser Tuenti o Twitter, bajo el estándar de red social de Open Social.

Además, posteriormente, se requirió de una estructura que pudiera controlar la ejecución de la aplicación SNSAngelGuard, enlazando el concepto de usuario de la aplicación con el usuario de una red social, por lo que, un usuario que ejecuta la aplicación debe tener un usuario válido en la red social en la que lo esté ejecutando. Esta estructura es la que almacena toda la información referente a la configuración del usuario en la aplicación, abarcando desde los ángeles que define, hasta la periodicidad con que se ejecutarán sus filtros.

En el diseño de la base de datos se aprecian los siguientes módulos funcionales:

1. Módulo funcional de Configuración: Este módulo contendrá todas aquellas tablas necesarias para almacenar la configuración del usuario referente a filtros en ejecución, periodicidad de éstos y ángeles seleccionados para cada filtro. Esta información podrá cambiar cada vez que el usuario entre en la aplicación y guarde una información distinta a la que había originalmente.
2. Módulo funcional de Datos Comunes: Este módulo contendrá aquellas estructuras de información comunes a Facebook y OpenSocial reagrupadas, para dotar de mayor estabilidad a la base de datos reordenando aquella información que pueda ser simplificada en ambas redes.
3. Módulo funcional de OpenSocial: Este módulo contendrá todas aquellas tablas necesarias para guardar la información referente a OpenSocial. Contendrá funcionalmente las tablas para guardar la misma información que las estructuras definidas para almacenar información de Facebook. Aunque en éste desarrollo no se va a utilizar, se explicará el modelo de datos para sucesivos desarrollos.

4. Módulo funcional de Facebook: Contendrá todas aquellas tablas necesarias para albergar información personal del usuario de Facebook que actualmente utiliza la aplicación. Se podrán guardar datos personales de cualquier tipo, desde datos personales hasta referencias a los círculos familiares a los que pertenece o enlaces a las fotografías que cuelga en su perfil.

Enumerados los diferentes bloques funcionales, pasaremos a definirlos formalmente en los siguientes apartados.

4.2. Módulo funcional de Configuración

Está representado en la imagen 3.3. Este módulo, como bien se ha especificado anteriormente, contendrá todas las tablas relacionadas con la configuración de la propia aplicación. Será un modelo sencillo en el que inicialmente se puede guardar la siguiente información:

1. Datos del usuario: Será la tabla maestra que controlará toda la aplicación. Todo usuario de la aplicación deberá tener un registro persistido en ésta tabla, aunque en el resto de tablas no contenga datos. En ella se almacenarán datos tales como su correo electrónico, el indicador de si la aplicación está activa, la fecha de la última actualización de datos, ..., etc. Esta tabla estará relacionada con las tablas de ángeles, que almacenarán la relación entre un usuario y sus ángeles, y con la tabla de filtros, que almacenará la relación entre un usuario de la aplicación y sus filtros en ejecución.
2. Ángeles seleccionados: Será la tabla que almacene todos los datos de los ángeles que han sido definidos para controlar al usuario de la aplicación. Esta tabla estará relacionada con la tabla de filtros, siendo ésta última la que irá a buscar datos de los ángeles configurados en los filtros a la primera.
3. Filtros configurados: Actualmente existen cuatro filtros disponibles:
 - a) Filtro de control vocabulario ofensivo.
 - b) Filtro de control de amigos.
 - c) Filtro de control de privacidad.
 - d) Filtro de control de visitas.

Los filtros serán almacenados en una única tabla, distinguiéndolos por su tipo. A parte de ésto, también almacenarán para cada uno si el filtro está activo y la frecuencia con la que se desea aplicar sobre la información. Esta tabla estará relacionada con la tabla de ángeles y con la tabla de usuarios de la aplicación.

Funcionalmente, el módulo ya está definido. Pasemos ahora a explicar detalladamente el modelo físico que se ha diseñado para tal fin.

4.2.1. Tabla user_settings

Esta tabla albergará la información del usuario. Será la tabla maestra de la aplicación, por lo que todo usuario que pertenezca a la aplicación deberá tener un registro en ésta. Se relacionará con la tabla user en una relación 1 a 1, por lo que se cumple la anterior premisa, ya que ésta última contendrá datos maestros tanto de Facebook como de OpenSocial. Los campos de la tabla user_settings son los mostrados en la tabla B.1.

4.2.2. Tabla locale_settings

Esta tabla almacenará toda la información referente al idioma en que el usuario ejecuta la aplicación. Contendrá la información necesaria para poder traducir la aplicación(títulos de páginas, botones, ayuda, ..., etc.) al idioma en el que el usuario tenga configurado Facebook, es decir, si su perfil de dicha Red Social está en Castellano, la aplicación se mostrará en Castellano. Si por el contrario, estuviera en Inglés, la aplicación se mostraría en Inglés. Actualmente, la aplicación SNSAngelGuard cuenta con soporte para Castellano e Inglés.

Esta tabla estará relacionada mediante el campo locale_settings_id_locale perteneciente a la tabla user_settings. Éste campo contendrá el identificador del registro de la tabla locale_settings a la cual hace referencia, obteniendo a partir de éste todos los recursos de idioma de la aplicación. La tabla B.2 muestra toda la información que contiene ésta estructura.

4.2.3. Tabla settings_angels

Esta tabla contendrá la información de todos aquellos ángeles definidos para cada usuario. Estará relacionada con la tabla user_settings con una relación N a N, en la que muchos usuarios podrán tener muchos ángeles. De cada ángel se obtiene la siguiente información:

1. Su identificador. En éste caso, corresponderá con su correo electrónico para el envío de notificaciones.
2. Su dirección de correo electrónico.
3. Su nombre.
4. El tipo de angel, es decir, si pertenece a Facebook, Google u otros contactos.
5. Si ha aceptado ser el ángel de algún usuario de la aplicación.
6. El usuario al que pertenece dentro de la aplicación.
7. La imagen del ángel. Si pertenece a Facebook, se almacenará su foto de perfil en el sistema. En otro caso, el sistema almacenará la imagen de perfil por defecto.

Toda la información anterior puede verse reflejada en la tabla B.3.

4.2.4. Tabla settings_filter

Esta tabla contendrá la información necesaria para cada filtro configurado por el usuario. Estará relacionado con las tablas user_settings y settings_angels, con una relación N a N. Para cada usuario, se almacenará la siguiente información:

1. Identificador único en base de datos para cada filtro.
2. Frecuencia a la que se ejecutará el filtro de control de lenguaje. Este campo contendrá un valor entre el 0 y el 6 que indicará lo siguiente:
 - a) 0: El filtro se ejecutará diariamente.
 - b) 1: El filtro se ejecutará cada semana.
 - c) 2: El filtro se ejecutará cada dos semanas.
 - d) 3: El filtro se ejecutará mensualmente(valor por defecto).
 - e) 4: El filtro se ejecutará cada dos meses.
 - f) 5: El filtro se ejecutará cada seis meses.
 - g) 6: El filtro se ejecutará anualmente.
3. Si se encuentra activo.
4. La fecha de la última vez que se ejecutó el filtro.
5. Tipo de filtro.

Esta tabla, estará relacionada N a N con la tabla settings_angels, de la cual se obtendrán aquellos ángeles que estén configurados para cada filtro, a los cuales se les enviará, cuando la frecuencia indique, los informes de notificación correspondientes a la ejecución del filtro.

La estructura física de la tabla puede observarse en la tabla B.4.

4.3. Módulo funcional de Datos Comunes

En éste modulo se agrupará toda aquella información que pueda ser común entre las Redes Sociales a analizar, tales como Facebook y OpenSocial. Evidentemente, los datos más propensos a ser reagrupados serán aquellos que atañen directamente a la faceta personal del usuario, tales como sus datos personales, sus intereses, relaciones personales, estatus, citas sobre libros, etc. Por esta razón, se ha diseñado una tabla maestra denominada ‘user’ que será la que agrupe todos estos datos.

4.3.1. Tabla user

Esta tabla será maestra, es decir, estará relacionada directamente con el nivel superior, es decir, con la tabla `user_settings` y con las del nivel inferior, `user_facebook` y `user_opensocial`. Es evidente que un usuario de la aplicación debe tener una entrada en ésta tabla para que los datos sean consistentes. Los datos que contiene son de índole personal y estarán almacenados físicamente en la estructura mostrada en la tabla B.5.

4.4. Módulo funcional de OpenSocial

Está representado en la imagen 3.2. Aunque como se ha especificado anteriormente no se ha realizado funcionalidad alguna para la Red Social OpenSocial, si que se ha preparado la base de datos para un próximo desarrollo sobre dicha Red Social que conlleve el análisis y estudio de los datos que se puedan generar. Para ello, se pasará a describir las estructuras físicas diseñadas para ser utilizadas a corto-medio plazo.

4.4.1. Tabla `user_opensocial`

Es la tabla maestra de los datos de OpenSocial. Como se ha comentado anteriormente, desciende de la tabla de datos comunes `user`. En ella se centrarán todas las relaciones de datos e irán gestionadas desde esta tabla. Estas relaciones serán las siguientes:

1. Sus números de teléfono irán gestionados siguiendo una relación 1 a N con la tabla `Phones_opensocial`.
2. Si es fumador, la tabla almacenará un código que será una entrada a la tabla maestra `Smoker_opensocial`.
3. Sus preferencias en cuanto a tendencias sociales seguirán una relación 1 a N con la tabla `LookingFor_opensocial`.
4. Todas sus direcciones de email se almacenarán, mediante una relación 1 a N, con la tabla `Email_opensocial`.
5. Su sexo irá descrito mediante un código que referenciará una entrada a la tabla maestra `Gender_opensocial`.
6. Su disponibilidad en OpenSocial irá descrita mediante un código que referenciará una entrada a la tabla maestra `Presence_opensocial`.
7. Su grado de alcoholismo irá descrita mediante un código que referenciará una entrada a la tabla maestra `Drinker_opensocial`.
8. Sus direcciones particulares seguirán una relación 1 a N con la tabla `Address_opensocial` que a su vez estará relacionada con la tabla `Organization_opensocial`, la cual será capaz de almacenar los lugares de trabajo o sitios donde haya estudiado mediante relaciones N a N.

9. Sus amigos dentro de OpenSocial serán almacenados, mediante una relación N a N, con la tabla **Friends _openSocial**.
10. Las URLs sobre videos, clips de audio o páginas de interés serán almacenados, mediante una relación N a N, con la tabla **url _openSocial**.
11. Los diferentes nombres o apodos que el usuario vaya utilizando a lo largo de su ciclo de vida en OpenSocial se almacenarán, mediante una relación N a N, con la tabla **Name _openSocial**.
12. Los mensajes de muro que el usuario recibe en su perfil de OpenSocial se almacenarán, mediante una relación 1 a N, en la tabla **Message _openSocial**.

Su estructura física está representada en la tabla B.6.

4.4.2. Tabla Phones _openSocial

Como se ha especificado anteriormente, será la tabla que contenga todos los números de teléfono del usuario. Conformará una relación 1 a N con la tabla **user _openSocial**, ya que un usuario podrá tener N números de teléfono. Su estructura física estará representada en la tabla B.7.

4.4.3. Tabla Smoker _openSocial

Esta tabla contendrá todos los posibles valores para describir el grado de afición al tabaco por parte de un usuario. Será una tabla maestra que irá referenciada a la tabla **user _openSocial** mediante un código de entrada que podrá representar los siguientes valores:

1. Valor **vacío**: Indicará que el usuario no ha definido esta característica en su perfil.
2. Valor **HEAVILY**: Indica que es fumador en exceso.
3. Valor **NO**: Indica que el usuario no es fumador.
4. Valor **OCCASIONALLY**: Indica que el usuario no es fumador pero puede fumar ocasionalmente.
5. Valor **QUIT**: Indica que el usuario era fumador pero actualmente lo ha dejado.
6. Valor **QUITTING**: Indica que el usuario está dejando de fumar.
7. Valor **REGULARLY**: Indica que el usuario fuma de forma regular.
8. Valor **SOCIALLY**: Indica que el usuario fuma únicamente en reuniones sociales.
9. Valor **YES**: Indica que el usuario es fumador.

La tabla B.8 mostrará su organización física.

4.4.4. Tabla LookingFor_openSocial

Esta tabla contendrá todos los valores posibles que se encuadran en la búsqueda de actividades o perfiles sociales que el usuario busca para poder relacionarse. Al igual que la tabla **Smoker_openSocial**, será una tabla maestra cuyos valores son los siguientes:

1. Valor **vacío**: Indicará que el usuario no ha definido esta característica en su perfil.
2. Valor **ACTIVITY_PARTNERS**: Compañeros de actividad que comparten los mismos intereses sociales.
3. Valor **DATING**: Amigos para un servicio a corto plazo.
4. Valor **FRIENDS**: Amigos permanentes.
5. Valor **NETWORKING**: Compañeros de actividad en internet, como jugadores en red o compañeros de trabajo.
6. Valor **RANDOM**: Aleatorio, indiferente.
7. Valor **RELATIONSHIP**: Una relación, ya sea familiar o un vínculo sentimental.

La tabla B.9 mostrará su estructura física dentro del sistema.

4.4.5. Tabla Email_openSocial

Esta tabla contendrá todas las direcciones de correo electrónico que hayan sido introducidas por el usuario de OpenSocial en su perfil. Estará relacionada 1 a N con la tabla **user_openSocial** y su estructura física se muestra en la tabla B.10.

4.4.6. Tabla Gender_openSocial

Será una tabla maestra que definirá el sexo del usuario en OpenSocial. Sus valores serán los siguientes:

1. Valor **vacío**: Indicará que el usuario no ha definido esta característica en su perfil.
2. Valor **MALE**: Sexo masculino.
3. Valor **FEMALE**: Sexo femenino.

La tabla B.11 mostrará su organización física.

4.4.7. Tabla Presence_openSocial

Esta tabla indicará el estado actual del usuario en OpenSocial. Será también una tabla maestra que contendrá valores fijos que serán referenciados desde la tabla **user_openSocial**. Sus valores serán los siguientes:

1. Valor **vacío**: Indicará que el usuario no ha definido esta característica en su perfil.
2. Valor **AWAY**: El usuario está conectado pero se encuentra ausente.
3. Valor **CHAT**: El usuario se encuentra manteniendo una conversación por chat.
4. Valor **DND**: Son las siglas ‘Do Not Disturb’, por lo que el usuario estará en estado ‘No disponible’.
5. Valor **OFFLINE**: El usuario se encuentra desconectado actualmente.
6. Valor **ONLINE**: El usuario se encuentra conectado actualmente.
7. Valor **XA**: Son las siglas ‘Extended Away’, por lo que el usuario se encontrará en un rango temporal elevado de inactividad aunque se encuentre conectado.

La tabla B.12 mostrará su organización física.

4.4.8. Tabla Drinker_openSocial

Esta tabla contendrá todos los posibles valores que existen en OpenSocial para describir el posible grado de alcoholismo del usuario. Al igual que todas las tablas de valores anteriores, será una tabla maestra referenciada desde la tabla **user_openSocial**. Los valores definidos para ella serán los siguientes:

1. Valor **vacío**: Indicará que el usuario no ha definido esta característica en su perfil.
2. Valor **NO**: Indica que el usuario no bebe.
3. Valor **OCCASIONALLY**: Indica que el usuario no es bebedor pero puede beber ocasionalmente.
4. Valor **QUIT**: Indica que el usuario era bebedor pero actualmente lo ha dejado.
5. Valor **QUITTING**: Indica que el usuario está dejando de beber.
6. Valor **REGULARLY**: Indica que el usuario bebe de forma regular.
7. Valor **SOCIALLY**: Indica que el usuario bebe únicamente en reuniones sociales.
8. Valor **YES**: Indica que el usuario es bebedor.
9. Valor **HEAVILY**: Indica que es bebedor en exceso.

La tabla B.13 mostrará su organización física.

4.4.9. Tablas de direcciones Address_openSocial y Organization_openSocial

Estas tablas serán las encargadas de almacenar cualquier dirección de interés relacionada con el usuario.

La tabla **Address_openSocial** almacenará toda la información respecto a la dirección física de un usuario tal como la dirección donde vive o su lugar de trabajo. Estará relacionada con la tabla **user_openSocial** mediante una relación 1 a N y su estructura física puede verse en la tabla B.14.

La tabla **Organization_openSocial** será una extensión de la tabla anterior. Esta tabla almacenará información sobre lugares de trabajo o lugares de estudio del usuario y utilizará la tabla anterior para almacenar su dirección, mientras que la tabla **Organization_openSocial** almacenará datos propios del trabajo. Estará relacionada con la tabla **user_openSocial** de dos maneras:

1. Utilizará una relación N a N para almacenar todos los trabajos del usuario.
2. Utilizará también otra relación N a N para almacenar su historial académico sobre los lugares donde estudió.

Su estructura física podrá verse en la tabla B.15.

4.4.10. Tabla Friends_openSocial

Esta tabla contendrá todos los amigos del usuario en OpenSocial. Estará relacionada N a N con la tabla **user_openSocial** ya que un amigo puede tener N amigos y, además, coincidir con varios amigos de otro usuario de OpenSocial con M amigos. Su estructura física se muestra en la tabla B.16.

4.4.11. Tabla url_openSocial

Esta tabla contendrá todos los posibles links que sean de interés para el usuario. Todos aquellos enlaces de video o de cualquier otro tipo deberán tener una entrada en ésta tabla. Estará relacionada con la tabla **user_openSocial** de las siguientes maneras:

1. Una relación N a N para guardar todos los enlaces con referencias a videos.
2. Una relación N a N para guardar todos los enlaces con referencias a canciones de música.

La tabla B.17 mostrará su estructura física.

4.4.12. Tabla Name_openSocial

Esta tabla contendrá todo el historial de nombres u apodos que el usuario vaya usando en su perfil de OpenSocial. Estará relacionada con la tabla **user_openSocial** con una relación N a N. Su estructura física será la mostrada en la tabla B.18.

4.4.13. Tabla Message_openSocial

Esta tabla contendrá información sobre cualquier mensaje que el usuario reciba en su perfil de OpenSocial. La tabla **Message_Type_openSocial** determina, mediante un conjunto de valores referenciados por la tabla **Message_openSocial**, qué tipo de mensaje se está almacenando. Sus posibles valores serán los siguientes:

1. Valor **vacío**: Indicará que el tipo de mensaje no está definido.
2. Valor **EMAIL**: Indica que el mensaje es un email.
3. Valor **NOTIFICATION**: Indica que el mensaje es una notificación de alguna aplicación.
4. Valor **PRIVATE_MESSAGE**: Indica que el mensaje es un mensaje privado.
5. Valor **PUBLIC_MESSAGE**: Indica que el mensaje es un mensaje público, como puede ser un mensaje en el muro.

Su estructura física puede observarse en la tabla B.19.

Tras haber definido el tipo de mensaje, la tabla **Message_openSocial** estará relacionada con la tabla **user_openSocial** con una relación 1 a N. Su estructura física puede observarse en la tabla B.20.

4.5. Módulo funcional de Facebook

Está representado en la imagen 3.4. La máxima carga de estudio, análisis y diseño se ha desarrollado sobre la Red Social **Facebook**. Todas las decisiones que se han tomado han sido para emular, lo más exactamente posible, al entorno de procesamiento de datos de ésta Red Social. Todas las relaciones, al igual que en el diseño de OpenSocial, se centran sobre una única tabla, denominada **user_facebook**, la cual controlará todos los datos de un usuario y harán que todos ellos sean consistentes para ser objeto de estudio posteriormente.

4.5.1. Tabla user_facebook

Es la tabla maestra de datos de Facebook. Al igual que la tabla **user_openSocial**, descenderá directamente de la tabla **user** y aglutinará en ella todas las relaciones con el resto de tablas que conforman todos los datos de usuario de un perfil de Facebook. Estas relaciones serán las siguientes:

1. Su historial de trabajo estará gestionado mediante una relación 1 a N con la tabla **work_history_facebook**.
2. Todo su historial académico se gestionará mediante una relación 1 a N con la tabla **education_history_facebook**.
3. Sus amigos y sus datos de contacto serán gestionados mediante una relación N a N con la tabla **friends_facebook**.
4. Toda su actividad en el muro de Facebook será gestionado mediante una relación N a N con la tabla **stream_facebook**, que a su vez contará con más relaciones que ayudarán a recrear todo el muro de Facebook con todas sus características.
5. Todas sus vinculaciones e ideas políticas serán gestionadas mediante una relación 1 a N con la tabla **affiliations_facebook**.
6. Sus relaciones familiares serán gestionadas por una relación N a N con la tabla **family_facebook**.
7. Todas sus direcciones postales serán gestionadas por dos relaciones N a N con la tabla **location_facebook** que serán explicadas en el punto correspondiente.

Para gestionar todas estas relaciones, la estructura física necesaria será la mostrada en la tabla B.21.

4.5.2. Tabla work_history_facebook

Será la tabla que almacenará toda la vida laboral que el usuario introduce en su perfil de Facebook. Estará gestionada por una relación 1 a N con la tabla **user_facebook**. Su estructura física está representada en la tabla B.22.

4.5.3. Tabla education_history_facebook

Esta tabla contendrá los datos académicos universitarios del usuario de Facebook. Estará gestionada por una relación 1 a N con la tabla **user_facebook**. Su estructura física estará representada en la tabla B.23.

4.5.4. Tabla friends _facebook

Será la tabla donde se almacenarán todas las amistades de un determinado perfil. Estará gestionada con una relación N a N con la tabla **user _facebook**. Se almacenan datos básicos como la edad de un amigo para, posteriormente, ser analizado por el filtro de control de amistades. Su estructura física estará representada en la tabla B.24.

4.5.5. Tabla stream _facebook

En cuanto a diseño, la tabla **stream _facebook** es la más compleja de construir, no sólo estructuralmente, sino funcionalmente, ya que va a constituir uno de los pilares básicos de análisis para los filtros configurados actualmente y los que se diseñen en un futuro. El análisis de un comentario del muro de un usuario conlleva una serie de relaciones que analizamos a continuación:

1. Llevará una relación N a N con la tabla **user _facebook** para tener controlados todas las entradas en el muro del usuario en todo momento.
2. Estará relacionado 1 a N con la tabla **likes _facebook**, la cual contabilizará todos los comentarios positivos de los amigos que ven cada comentario.
3. El grado de privacidad del comentario estará gestionado por una relación 1 a N con la tabla **privacy _facebook**.
4. Los comentarios a la entrada del muro estarán almacenados mediante una relación 1 a N con la tabla **comments _facebook**.
5. Todos los links que se adjunten a la entrada del muro estarán almacenados mediante la relación 1 a N con la tabla **action _links _facebook**.

Por todo ello, la estructura física de la tabla **stream _facebook** seguirá la estructura de la tabla B.25.

4.5.5.1. Tabla likes _facebook

Esta tabla almacenará información acerca del numero de **likes** que recibe un post del muro de un usuario. Tendrá dos tipos de relaciones:

1. Una relación 1 a N con la tabla **stream _facebook** que controlará la información acerca de ese comentario en concreto.
2. Una relación 1 a N con la tabla **friends _likes _facebook** que controlará los datos de cada amigo al que le ha gustado la entrada de post.

Con todas estas referencias, la estructura física que conforma la tabla **likes _facebook** estará representada en la tabla B.26.

Para almacenar cada uno de los usuarios a los que les ha gustado el post, se ha diseñado la tabla **friends _likes _facebook**, cuya estructura física está representada en la tabla B.27.

4.5.5.2. Tabla **privacy_facebook**

Esta tabla controla los usuarios que pueden visualizar el post del usuario de Facebook. Estará gestionado por una relación 1 a N con la tabla **stream_facebook**. Su estructura física puede observarse en la tabla B.28.

4.5.5.3. Tabla **comments_facebook**

Esta tabla almacenará todas las respuestas a los post que se hayan generado en el muro del usuario. Tendrá las siguientes relaciones:

1. Seguirá una relación 1 a N con la tabla **stream_facebook** ya que un post podrá tener N respuestas.
2. Se relacionará 1 a N con la tabla **comment_facebook**, la cual almacenará todos los datos de cada una de las respuestas, cuando fue creado y cuando fue modificado.

Con estos datos, su estructura física será la representada en la tabla B.29.

Como se ha indicado anteriormente, la tabla **comments_facebook** se relacionará 1 a N con la tabla **comment_facebook**, la cual almacenará información de cada uno de los comentarios a los post y seguirá la estructura física de la tabla B.30.

4.5.5.4. Tabla **action_links_facebook**

La tabla **action_links_facebook** mostrará información de todos aquellos post que se hayan generado con cualquier tipo de enlace a videos o clips de audio dentro del texto. Estará relacionada con la tabla **stream_facebook** mediante una relación 1 a N. Su estructura física será la mostrada en la tabla B.31.

4.5.6. Tabla **affiliations_facebook**

Esta tabla almacenará cualquier tipo de relación del perfil de un usuario de Facebook, es decir, almacenará datos de la persona con la que tenga una relación de novios, maridos, etc. Se verá afectada por las siguientes relaciones:

1. Seguirá una relación 1 a N con la tabla **user_facebook**.
2. Se relacionará N a 1 con la tabla **type_affiliations_facebook**, la cual almacenará todos los tipos de relación que se hayan dado de alta en Facebook.

Con todos estos datos, su estructura física quedará de la forma indicada por la tabla B.32.

Para obtener el tipo de la relación, se diseñó la tabla **type_affiliations_facebook**, la cual indicará el tipo de relación entre ambas personas. Su estructura física se mostrará en la tabla B.33.

4.5.7. Tabla family_facebook

Esta tabla almacenará todas las relaciones familiares del usuario. Seguirá las siguientes relaciones:

1. Seguirá una relación N a N con la tabla **user_facebook**, ya que un usuario podrá tener N relaciones familiares.
2. Se relacionará N a 1 con la tabla **relationship_facebook**, la cual almacenará todos los tipos de relación familiar que se hayan dado de alta en Facebook.

Su estructura física puede observarse en la tabla B.34.

La tabla **relationship_facebook** mostrará los tipos de relaciones dados de alta por el usuario en Facebook. Su estructura física será la representada en la tabla B.35.

4.5.8. Tabla location_facebook

Esta tabla mostrará el lugar donde actualmente reside el usuario. Estará relacionado N a N con la tabla **user_facebook** y su estructura física seguirá la estructura de la tabla B.36.

Capítulo 5

Interface RESTFul

5.1. Introducción

En este capítulo se explicarán todas las estructuras que se han diseñado para los accesos a datos de la base de datos **SocialNetwork** a través de **Servicios Web**. Todas estas estructuras se han almacenado en una API pública para que, posteriormente a este desarrollo, cualquier programador que quiera hacer uso de dicha base de datos pueda hacerlo de forma transparente accediendo únicamente a los objetos que quiere referenciar, tal y como se han diseñado otras bases de datos, tales como la que utiliza Facebook para acceder al perfil de un usuario determinado.

En los primeros capítulos del presente documento se estableció la definición de **Servicios REST**, que no son más que Servicios Web que se basan en protocolos HTTP como medio de obtención de datos. Todo Servicio REST se basa en el concepto de “Todo recurso(información) debe tener y ser accesible mediante una URI única”. Para realizar este acceso, se usan las operaciones restringidas únicamente a los métodos GET, PUT, POST y DELETE, que pasaremos a explicar a continuación:

1. Método **GET**: Este método accede a un recurso determinado y devuelve un objeto como respuesta con todos los datos asociados al recurso. Su respuesta puede ser tratada mediante XML o JSON, a gusto del desarrollador.
2. Método **POST**: Este método introduce en la base de datos un recurso en la dirección URI que se le indique. Internamente, cuando la petición llega al proceso que gestiona internamente la base de datos remota, éste realiza una operación **INSERT** hacia todas las tablas que sean referenciadas por el objeto que se le ha pasado al argumento de llamada. Cuando éstos datos sean inconsistente, la base de datos no realizará operación alguna y se devolverá un código de error.
3. Método **PUT**: Análogamente al método POST, este método hará una actualización de un recurso en base de datos, realizando una operación **UPDATE** en la base de datos remota cuando ésta reciba la petición y el objeto que se desea actualizar. Al igual que el método anterior, si los datos no son consistentes se devolverá un error al método que realiza la llamada.

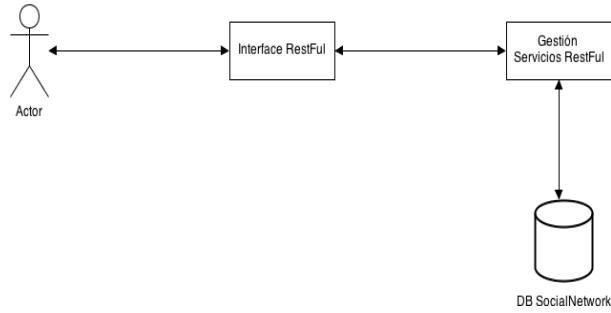


Figura 5.1: Arquitectura de la Interface RestFul

4. Método **DELETE**: Este método eliminará el recurso que apunta a la URI que se ha referenciado.

Todos los métodos anteriores devuelven códigos de error HTTP que serán tratados en capítulos posteriores.

Para realizar esta comunicación entre aplicación que demanda datos, Interfaz RestFul y base de datos, internamente vamos a necesitar las siguientes estructuras:

1. Una base de datos desarrollada bajo cualquier tecnología, en nuestro caso, sobre el entorno MySQL 5.
2. Un proyecto que gestione internamente todas las operaciones de base de datos y ofrezca servicios RestFul a los datos y que, a su vez, puedan ser accesibles mediante estos servicios.
3. Una interfaz RestFul definida para el acceso a datos que apunte directamente a todos los servicios definidos internamente por el proyecto que gestiona la base de datos.

En éste capítulo se pasará a explicar cada una de estas estructuras y a definir todos los servicios necesarios para esta arquitectura, mostrada en la imagen 5.1.

5.2. Casos de uso para la Interface RestFul

En éste apartado, vamos a estudiar, mediante el diagrama de casos de uso mostrado en la imagen 5.2, las posibles acciones a las que se enfrentará y para las que está diseñada la Inteface RestFul y el resto de componentes que necesita para interactuar.

El diagrama de casos de uso seguirá la siguiente secuencia:

1. La aplicación **SNSAngelGuardFB** generará una operación de datos, bien sea de petición, de actualización o de eliminación. A ésta petición irán encapsulados todos los datos necesarios para realizarla y la URI correspondiente al objeto que se quiere añadir, actualizar o eliminar de la base de datos.

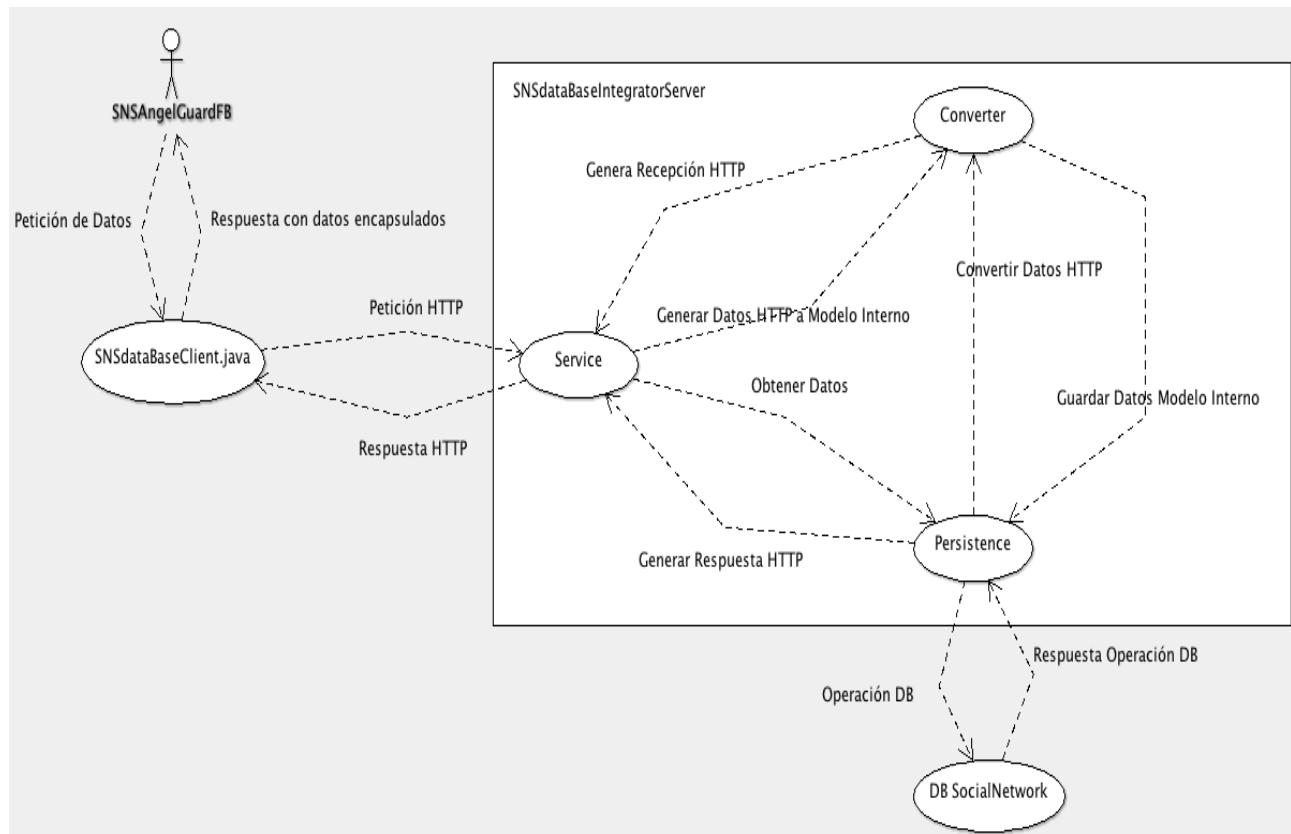


Figura 5.2: Diagrama de Casos de Uso de la Interface RestFul

2. La clase **SNSDataBaseClient.java** generará la operación HTTP necesaria para la operación y la enviará via HTTP al proyecto que se encargará de gestionar los Servicios RestFul junto con las operaciones con la base de datos, en última instancia. La clase SNS DataBase Client.java contendrá una serie de métodos a los que se invocará para acceder a dichos servicios. Esta será nuestra Interface RestFul en la práctica, ya que será la encargada de los envíos mediante las operaciones HTTP mencionadas anteriormente y de la recepción de las respuestas generadas encapsulando éstas en los tipos de datos necesarios para ser tratadas.
3. Cuando se invoca uno de los métodos de la clase SNS DataBase Client.java, se envia la petición HTTP al proyecto **SNS DataBase Integrator Server**, quien será el encargado de interactuar con la base de datos. Este proyecto se sustenta sobre tres pilares básicos:
 - a) El paquete **service** contendrá, para cada recurso, las clases necesarias que son las que ofrecen los Servicios RestFul que son invocados por la Interface, es decir, cada método que se invoca tiene una entrada en estas clases. Serán las encargadas de receptionar la petición, analizar mediante la URI recibida a qué objeto se refiere y se debe tratar y, finalmente, enviar la respuesta de la operación.

- b) Cuando llega una petición, bien sea de consulta o de actualización, se deberán parsear los datos recibidos de la operación al modelo interno de análisis. Estas operaciones serán realizadas por el paquete **converter** el cual, para cada recurso de la base de datos, tendrá una serie de estructuras que recibirán los datos de la petición HTTP y los convertirán a datos internos para que puedan ser procesados.
 - c) Tras realizar el parseo conveniente de los datos, éstos serán enviados a la clase que corresponda, según el recurso, del paquete **persistence**, el cual, para cada recurso de la base de datos, tendrá una serie de estructuras que serán, en último término, las encargadas de realizar la operación en base de datos y propagar la respuesta que a ésta se produzca, a su nivel superior, es decir, a la clase que corresponda del paquete **service**, el cual será capaz de propagarlo hasta la Interface RestFul.
4. Una vez que la respuesta ha llegado, el método de la clase SNSdataBaseClient.java parseará el objeto al tipo de datos que se le haya indicado en la petición y propagará la respuesta a su nivel superior, es decir, al método de la aplicación SNSAngel-GuardFB que lo haya invocado.
 5. Finalmente, al llegar la respuesta al método que originó toda la operación, se analizará el código de respuesta, se obtendrán los datos en caso afirmativo y se continuará con la ejecución de la aplicación.

5.3. Projecto SNSdataBaseIntegratorServer

SNSdataBaseIntegratorServer será el encargado de realizar toda la gestión de los servicios RestFul y, tras evaluar las peticiones, acceder a la base de datos SocialNetwork y realizar la operación que se ha enviado. Tras realizar la operación, generará una respuesta en la que irán encapsulados tanto los datos obtenidos como el código de respuesta HTTP.

Este proyecto está realizado bajo el lenguaje de programación **Java** y ha sido desarrollado automáticamente por las herramientas de que dispone el entorno de desarrollo **NetBeans**, las cuales, a partir de una base de datos, en nuestro caso SocialNetwork, generan todos los Servicios RestFul y todas las estructuras necesarias para realizar la gestión de dicha base de datos. Con ésto conseguimos una gestión de la base de datos totalmente independiente y, además, la base de datos puede no estar en la misma máquina que el servidor que ejecuta la operación pero, por el contrario, siempre se deberá tener ejecutando en un servidor web el proyecto SNSdataBaseIntegratorServer para que puedan llevarse a cabo estas operaciones.

Como bien se ha mencionado anteriormente, este proyecto cuenta con los siguientes paquetes:

1. Paquete **es.uah.cc.ie.service**: El cual contendrá todos los servicios RestFul de la base de datos.

2. Paquete **es.uah.cc.ie.converter**: El cual contendrá todas las estructuras necesarias para realizar la conversión de datos de modelo interno a operación HTTP y viceversa.
3. Paquete **es.uah.cc.ie.persistence**: El cual contendrá todas las estructuras necesarias para, en última instancia, resolver la operación de base de datos referenciada en el Servicio RestFul.

La estructura de paquetes puede verse reflejada en la imagen 5.3. Para cada recurso,

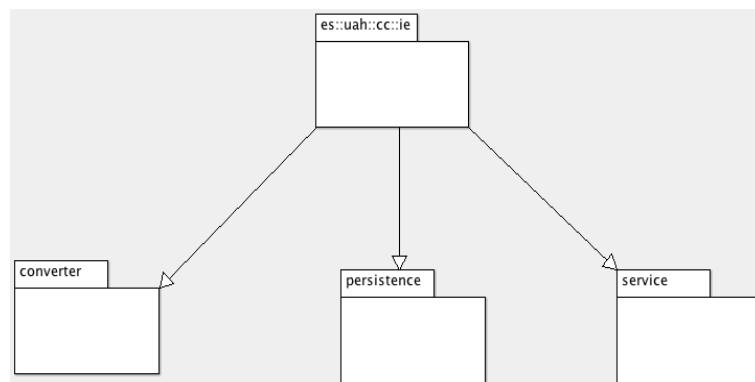


Figura 5.3: Diagrama de paquetes del proyecto SNSdataBaseIntegratorServer

cada paquete contendrá dos estructuras necesarias para su procesamiento(salvo el paquete **persistence**, que únicamente contendrá una clase por recurso), la estructura que procesa una unidad y la estructura que procesa un conjunto de unidades del mismo recurso. Por ejemplo, si queremos ver las estructuras necesarias para gestionar la tabla user, la imagen 5.4 nos despejará cualquier tipo de duda.

5.3.1. Paquete es.uah.cc.ie.service

Este paquete contendrá todo el soporte a una petición de un Servicio RestFul. En éste paquete se encontrará dos clases por cada recurso. La clase que gestiona una única entidad tendrá los siguientes componentes:

1. Atributo **ResourceContext**: Irá acompañado de la anotación **@Context**. Servirá para obtener el contexto en el que se está ejecutando el recurso y obtener su referencia física.

```
protected ResourceContext resourceContext;
```

2. Atributo **UriInfo**: También irá acompañado de la anotación **@Context**. En él se almacenará la URI absoluta de acceso al recurso que se está procesando.

```
protected UriInfo uriInfo;
```

3. Atributo **String id**: Almacenará el ID del objeto al que hace referencia dentro de la base de datos.

```
protected String id;
```

4. Método **get**: Irá precedido de la anotación **@GET**. Obtendrá un objeto de la base de datos indicado por el parámetro **ExpandLevel**, que por defecto valdrá 1. Se podrá indicar, mediante la anotación **@Produces**, en qué formato se van a devolver los datos que ésta llamada produzca, en nuestro caso los datos se enviarán tanto en XML como en JSON, siendo éste último el formato elegido para la recepción.
5. Método **put**: Irá precedido de la anotación **@PUT**. Actualizará un determinado objeto de la base de datos. En éste caso, irá acompañado de la anotación **@Consumes**, que indicará al método qué tipo de datos va a recibir. En nuestro caso, podrá recibir tanto en formato XML como en JSON.
6. Método **delete**: Irá precedido de la anotación **@DELETE**. Eliminará un objeto entero, con todas sus relaciones, de la base de datos. No recibirá ningún parámetro, sino que mediante el atributo **id** sabrá qué entidad es la que procederá a eliminar.

El resto de métodos que se definen en esta clase dependerá únicamente del tipo de recurso que se esté gestionando.

La clase que gestiona un conjunto de entidades se relacionará directamente con la clase anterior para cada una de ellas. Aun así, deberá definir los siguientes componentes:

1. Atributo **ResourceContext**: Irá acompañado de la anotación **@Context**. Servirá para obtener el contexto en el que se está ejecutando el recurso y obtener su referencia física.

```
protected ResourceContext resourceContext;
```

2. Atributo **UriInfo**: También irá acompañado de la anotación **@Context**. En él se almacenará la URI absoluta de acceso al recurso que se está procesando.

```
protected UriInfo uriInfo;
```

3. Método **get**: Irá precedido de la anotación **@GET**. Obtendrá un conjunto de objetos de la base de datos. Se podrá indicar, mediante la anotación **@Produces**, en qué formato se van a devolver los datos que ésta llamada produzca, en nuestro caso los datos se enviarán tanto en XML como en JSON, siendo éste último el formato elegido para la recepción. En éste caso, el método recibirá los siguientes parámetros siendo totalmente configurables desde la llamada al Servicio RestFul:
 - a) **int start**: Indicará el valor de inicio a la hora de tomar entidades en la consulta. Su valor por defecto será 0.

- b) **int max:** Indicará el número máximo de entidades a devolver en la consulta. Su valor por defecto será 10.
 - c) **int expandLevel:** Indicará los niveles hasta los que se puede realizar la consulta. Su valor por defecto será 1.
 - d) **String query:** Indicará la sentencia SQL que va a ejecutar.
4. Método **post:** Irá precedido de la anotación **@POST**. Actualizará un determinado objeto de la base de datos. En éste caso, irá acompañado de la anotación **@Consumes**, que indicará al método qué tipo de datos va a recibir. En nuestro caso, podrá recibir tanto en formato XML como en JSON. Recibirá una instancia del objeto que va a actualizar en la base de datos parseado al modelo interno de ésta.
5. Método de acceso a una entidad: Se diferencia del resto porque va precedido de la anotación **@Path** seguido del UID del objeto en cuestión. Devolverá el objeto de la base de datos referenciado por el parámetro UID de la anotación.

5.3.2. Paquete es.uah.cc.ie.converter

El paquete **es.uah.cc.ie.converter** será el encargado de transformar los tipos de datos que reciba de las peticiones HTTP de datos, en formato XML o JSON, y transformarlos al modelo interno de datos del paquete **es.uah.cc.ie.persistence** para que puedan ser tratados y almacenados en la base de datos SocialNetwork. Realizará el proceso también a la inversa, es decir, recibirá datos de la base de datos SocialNetwork y los transformará en un objeto de formato XML o JSON, dependiendo del valor indicado en la llamada al Servicio RestFul.

Al igual que el paquete **es.uah.cc.ie.service**, contará con dos clases para cada recurso, una para el tratamiento de la unidad y la otra para el tratamiento de un conjunto de unidades de un recurso determinado.

Los componentes comunes a todas las estructuras que gestionan una única unidad serán los siguientes:

1. Todas éstas clases, en su definición, irán precedidas de la anotación **@XmlRootElement**, perteneciente al paquete **javax.xml.bind.annotation**, seguido del nombre del recurso al que hace referencia. Para el recurso **userSettings**, su definición será la siguiente:

```
@XmlElement(name = "userSettings")
public class UserSettingsConverter {
    ...
    ... Definición de atributos y métodos de clase
    ...
}
```

2. Atributo **recurso-entidad** a la que hace referencia: Tendrá como atributo de clase la entidad a la que está asociada la clase del paquete **es.uah.cc.ie.persistence**. En nuestro caso, para el recurso **userSettings**, dispondrá del siguiente atributo:

```
private UserSettings entity;
```

3. Atributo **URI**: Este atributo pertenecerá al paquete de definición **java.net**. Indicará la URI propia del objeto al que hace referencia. En nuestro caso, para el recurso **userSettings**, su definición será la siguiente:

```
private URI uri;
```

4. Atributo **expandLevel**: Este atributo será de tipo **int** e indicará el nivel al que se debe acceder dentro de la jerarquía del recurso. Su valor por defecto será 1 y su definición, para el recurso **userSettings**, será la siguiente:

```
private int expandLevel;
```

5. Método **getEntity**: Este método devolverá un objeto de tipo **es.uah.cc.ie.persistence** asociado al recurso dependiendo del valor del atributo **recurso-entidad** del objeto. Su definición irá precedida de la anotación **@XmlTransient**, la cual pertenecerá al paquete de definición **javax.xml.bind.annotation**, y servirá para establecer un mapeo recurso-entidad y evitar conflictos con el resto de recursos. Para el recurso **userSettings**, su definición será la siguiente:

```
@XmlTransient
public UserSettings getEntity() {
    ...
    ... Definición del método
    ...
}
```

6. Método **resolveEntity**: Este método recibirá un objeto **EntityManager**, del paquete de definición **javax.persistence**, de la operación SQL correspondiente, y devolverá un objeto del paquete **es.uah.cc.ie.persistente** asociado al recurso. Será el encargado de obtener del resultado de una operación de base de datos la entidad y todos los datos a la que ésta hace referencia. Para el recurso **userSettings**, su definición será la siguiente:

```
public UserSettings resolveEntity(EntityManager em) {
    ...
    ... Definición del método
    ...
}
```

7. Conjunto de métodos para obtener los atributos de la entidad a la que se hace referencia: Estos métodos serán getters y setters asociados al recurso al que hace referencia. Los métodos getters irán precedidos de la anotación **@XmlElement**, perteneciente al paquete de definición **javax.xml.bind.annotation**, seguido del nombre del atributo del recurso. Estos métodos leerán directamente del objeto de intercambio XML o JSON el atributo al que hacen referencia y lo establecerán en el modelo interno necesario para el procesamiento de datos en la base de datos.

La definición de componentes para la clase que establece el conjunto de unidades de un recurso será la siguiente:

1. Este tipo de clases contará en su definición con la anotación **@XmlRootElement**, perteneciente al paquete **javax.xml.bind.annotation**, seguido del nombre del recurso al que hace referencia. Notesé que al tratarse de un conjunto de recursos, el recurso para este caso será tratado de manera diferente, por lo que el nombre será distinto. Todos los recursos que hagan referencia a un conjunto de recursos estarán denominados con la nomenclatura **NombreRecurso + s**, haciendo referencia a la pluralidad del recurso que se está tratando. Para el conjunto **userSettings**, su definición de clase será la siguiente:

```
@XmlRootElement(name = "userSettingss")
public class UserSettingssConverter {
    ...
    ... Definición de clase
    ...
}
```

2. Atributo **Collection<Recurso>**: Indicará la colección de recursos a la que hace referencia. Los recursos pertenecerán al paquete **es.uah.cc.ie.persistence**. Para el caso del recurso **userSettingss**, su definición será la siguiente:

```
private Collection<UserSettings> entities;
```

3. Atributo **Collection<RecursoConverter>**: Indicará la colección de recursos del paquete **es.uah.cc.ie.converter** que han sido parseados directamente del objeto de intercambio XML o JSON y están preparados para ser persistidos. En el caso del recurso **userSettingss**, su definición será la siguiente:

```
private Collection<UserSettingsConverter> items;
```

4. Atributo **URI**: Este atributo pertenecerá al paquete de definición **java.net**. Indicará la URI propia del objeto al que hace referencia. En nuestro caso, para el recurso **userSettingss**, su definición será la siguiente:

```
private URI uri;
```

5. Atributo **expandLevel**: Este atributo será de tipo **int** e indicará el nivel al que se debe acceder dentro de la jerarquía del recurso. Su valor por defecto será 1 y su definición, para el recurso **userSettingss**, será la siguiente:

```
private int expandLevel;
```

6. Método **getEntities**: Devolverá un objeto de tipo **Collection<Recurso>** asociado al recurso, del paquete **es.uah.cc.ie.persistence**, dependiendo del valor del atributo **recurso-entidad** del objeto. Su definición irá precedida de la anotación **@XmlTransient**, la cual pertenecerá al paquete **javax.xml.bind.annotation**, y servirá para establecer un mapeo recurso-entidad y evitar conflictos con el resto de recursos. Para el recurso **userSettingss**, su definición será la siguiente:

```
@XmlTransient
public Collection<UserSettings> getEntities() {
    ...
    ... Definición del método
    ...
}
```

5.3.3. Paquete es.uah.cc.ie.persistence

El paquete **es.uah.cc.ie.persistence** contendrá todas las estructuras necesarias para, en última instancia, persistir el objeto del recurso referenciado en base de datos. Por cada recurso existirá una única clase que contendrá todos los atributos de la tabla de la base de datos SocialNetwork a la que hace referencia junto con todas las definiciones de sus dependencias con otras tablas. Se tomará como ejemplo el recurso **userSettings**, que hará referencia a la tabla de base de datos **user_settings** con la que trabajará exclusivamente. Sus componentes serán los siguientes(que se podrán hacer extensibles al resto de recursos):

1. Definición de la clase: Su definición será un conjunto de parámetros que se explicarán a continuación:
 - a) Anotación **@Entity**: Pertenecerá al paquete de definición **javax.persistence**. Con esta anotación se indica que estamos tratando de una Entidad mapeada directamente en base de datos.
 - b) Anotación **@Table**: Pertenecerá al paquete de definición **javax.persistence**. Esta anotación irá seguida del nombre de la tabla con la que está mapeada.
 - c) Anotación **@NamedQueries**: Pertenecerá al paquete de definición de java **javax.persistence**. Ésta contendrá todas las definiciones de las posibles querys que pueden ser invocadas para cada uno de los atributos de la tabla. Para cada una de las definiciones, se utilizará la anotación **@NamedQuery**, que pertenecerá al mismo paquete de definición.
 - d) Implementará a la interface **Serializable**, perteneciente al paquete de definición **java.io**. Servirá para poder serializar un objeto y poder enviarlo a través del protocolo HTTP.

Con todas estas indicaciones, la definición de la entidad **userSettings** será de la siguiente forma:

```

@Entity
@Table(name = "user_settings")
@NamedQueries({
    @NamedQuery(name = "UserSettings.findAll", query = "SELECT u FROM
        UserSettings u"),
    @NamedQuery(name = "UserSettings.findByUid", query = "SELECT u FROM
        UserSettings u WHERE u.uid = :uid"),
    @NamedQuery(name = "UserSettings.findByName", query = "SELECT u
        FROM UserSettings u WHERE u.userName = :userName"),
    @NamedQuery(name = "UserSettings.findByUserEmail", query = "SELECT u
        FROM UserSettings u WHERE u.userEmail = :userEmail"),
    @NamedQuery(name = "UserSettings.findByLegalAccepted", query = "SELECT u
        FROM UserSettings u WHERE u.legalAccepted = :legalAccepted"),
    @NamedQuery(name = "UserSettings.findByLastCheck", query = "SELECT u
        FROM UserSettings u WHERE u.lastCheck = :lastCheck"),
    @NamedQuery(name = "UserSettings.findByUidPublic", query = "SELECT u
        FROM UserSettings u WHERE u.uidPublic = :uidPublic"),
    @NamedQuery(name = "UserSettings.findByAppActivated", query = "SELECT u
        FROM UserSettings u WHERE u.appActivated = :appActivated"),
    @NamedQuery(name = "UserSettings.findByUserSession", query = "SELECT u
        FROM UserSettings u WHERE u.userSession = :userSession"),
    @NamedQuery(name = "UserSettings.findByBackupCheck", query = "SELECT u
        FROM UserSettings u WHERE u.backupCheck = :backupCheck"))
public class UserSettings implements Serializable {
    ...
    ... Definición de la Entidad
    ...
}

```

2. Conjunto de métodos de acceso a cada uno de los atributos de la tabla: Para acceder al valor de cada uno de los atributos, en cada clase existirán métodos estructurales(get y set) para cada uno de los atributos.

5.4. Interface RESTful SNSdataBaseClient.java

Hasta este punto, hemos realizado el estudio, el análisis y el diseño de todos los servicios RestFul que van a poner en funcionamiento nuestra base de datos de una forma independiente y totalmente transparente al desarrollador. Mediante un servidor web pondremos en funcionamiento el proyecto SNSdataBaseIntegratorServer y tendremos a disposición todo el potencial de nuestra base de datos.

Ahora bien, el tema que nos ocupa, es el estudio de cómo poder llamar a los servicios RestFul definidos con anterioridad desde nuestra aplicación sin tener que hacer ninguna

operación de base de datos en ella. Esto lo conseguimos con la implementación de un fichero cliente que encapsulará, en métodos de llamada, todos los servicios RestFul que sean necesarios para el funcionamiento de nuestra aplicación. Mediante estos sencillos métodos se conseguirá enviar órdenes de base de datos hasta el proyecto SNSdataBaseIntegratorServer por medio del protocolo HTTP y nuestra aplicación será capaz tanto de realizar las llamadas como analizar las respuestas.

El cliente desarrollado en el fichero **SNSdataBaseClient.java** es una solución funcional encapsulada en el proyecto **SNSAngelGuardFB**, que contendrá toda la funcionalidad de la aplicación. Las ventajas de esta implementación radican en el simple hecho de que esta interface podría ser pública, por lo cual, cualquier desarrollador podría tomarla en su proyecto y utilizar todos sus métodos para la gestión de la base de datos SocialNetwork.

En las sucesivas secciones a ésta, se pasará a explicar la estructura de éste fichero y de cada uno de los métodos que han sido desarrollados para conseguir esta solución.

5.4.1. Atributos de la clase SNSdataBaseClient.java

En ésta sección se estudiarán y analizarán los diferentes atributos que contiene la clase **SNSdataBaseClient.java**, que serán los siguientes:

1. Atributo **WebResource**: Mediante este atributo se conseguirán construir recursos web que sean capaces de enviar y procesar respuestas HTTP. Pertenecerá al paquete **com.sun.jersey.api.client.Client**.
2. Atributo **Client**: Este atributo permite configurar las propiedades de conexión de un recurso web y crear una conexión, por lo que el atributo **WebResource** lo utilizará como conexión configurada para enviar peticiones HTTP. Pertenecerá, al igual que el atributo **WebResource**, al paquete **com.sun.jersey.api.client.Client**.
3. Constante **BASE_URI**: Será una constante de tipo **String** y contendrá la dirección en la que está instalado el proyecto que gestiona los servicios RestFul de la base de datos que, en nuestro caso, será el proyecto **SNSdataBaseIntegratorServer**. Para un entorno local de pruebas, el valor de éste atributo será el siguiente:

"`http://localhost:8080/SNSdataBaseIntegratorServer/resources/`"

5.4.2. Métodos de la clase SNSdataBaseClient.java

En ésta sección se documentará, a modo de API, todos los métodos organizados por recursos que formarán parte de ésta clase. Para más nivel de detalle en definición, se deberá ver el apéndice C.

5.4.2.1. Recurso userSettings

Este será el recurso que accederá a la tabla **user_settings**. Todos sus métodos están definidos en la tabla 5.1.

Tabla 5.1: Tabla de Servicios RESTFul del recurso userSettings

Recurso	Descripción
GET userSettings/userSettings_getEntities	Este método obtiene todas los registros de la tabla user_settings . Este método obtiene el usuario de la tabla user_settings que coincide con el parámetro uidPublic , el cual será su identificador publico , es decir, el identificador que puede observarse públicamente, obtenido de cifrar el identificador del usuario en Facebook con una clave privada proporcionada por el desarrollador de la aplicación.
GET userSettings/userSettings_getUserByUidPublic	Este método obtiene el usuario de la tabla user_settings que coincide con el parámetro uidPublic , el cual será su identificador publico , es decir, el identificador que puede observarse públicamente, obtenido de cifrar el identificador del usuario en Facebook con una clave privada proporcionada por el desarrollador de la aplicación.
PUT userSettings/userSettings_setUpdateTime	Este método actualiza la fecha lastCheck o backUpCheck a la fecha en la que se está realizando la operación y, además, actualiza la sesión del usuario a la última sesión de conexión de Facebook.
POST userSettings/userSettings_setNewEntityUserSettings	Este método introduce un nuevo usuario, con toda su información, en la tabla user_settings .
GET userSettings/userSettings_getUserSettingsByUid	Este método devuelve toda la información del usuario de la tabla user_settings al que pertenece el parámetro de entrada uid .
GET userSettings/userSettings_setNewAngelsCollectionByUid	Este método establece un nuevo ángel en la colección de un usuario de la tabla user_settings , al que pertenece el parámetro de entrada uidAngel .

5.4.2.2. Recurso localeSettings

Este será el recurso que accederá a la tabla **locale_settings**. Todos sus métodos están definidos en la tabla 5.2.

Recurso	Descripción
GET localeSettings/localeSettings_getLocaleSettingsByUid	Obtiene los recursos de idioma según el parámetro de entrada uid .

Tabla 5.2: Tabla de Servicios RESTFul del recurso localeSettings

Recurso	Descripción
GET settingsFilter/settingsFilter_getFiltersByIdFilter	Obtiene la configuración de un filtro identificado por el parámetro de entrada idFilter en la tabla settings_filter .
GET settingsFilter/settingsFilter_getFiltersByUserSettingsUid	Obtiene todos los filtros definidos para el usuario de la tabla user_settings identificado por el parámetro de entrada uid .
POST settingsFilter/settingsFilter_setNewFilter	Introduce un nuevo filtro en la tabla settings_filter identificado por el parámetro de entrada requestEntity .
PUT settingsFilter/settingsFilter_updateFilterByIdFilter	Actualiza la información de un filtro, identificado por el parámetro de entrada idFilter y por la información del filtro requestEntity , en base de datos. Puede actuar de dos modos, actualizando también la hora en la que se ejecutó el filtro o únicamente actualizando su configuración. Este comportamiento estará definido por el parámetro de entrada mode , el cual tomará los valores 0 , cuando no haya que actualizar la hora de ejecución del filtro, y 1 , cuando haya que actualizarla.
GET settingsFilter/settingsFilter_getAngelsCollectionByIdFilter	Obtiene la colección de ángeles definidos en el filtro que identifica el parámetro de entrada idFilter .

Tabla 5.3: Tabla de Servicios RESTFul del recurso settingsFilter

5.4.2.3. Recurso settingsFilter

Este será el recurso que accederá a la tabla **settings_filter**. Todos sus métodos están definidos en la tabla 5.3.

5.4.2.4. Recurso settingAngels

Este será el recurso que accederá a la tabla **settings_Angels**. Todos sus métodos están definidos en la tabla 5.4.

5.4.2.5. Recurso user

Este será el recurso que accederá a la tabla **user**. Todos sus métodos están definidos en la tabla 5.5.

Recurso	Descripción
PUT settingsAngels/settingsAngels_setAngelByUid	Actualiza la información del ángel en la tabla settings_angels que está identificado por el parámetro de entrada uid .
DELETE settingsAngels/settingsAngels_delAngelByUid	Borra la información del ángel en la tabla settings_angels que está identificado por el parámetro de entrada uid .
POST settingsAngels/settingsAngels_setNewAngel	Introduce un nuevo ángel en la tabla settings_angels .
GET settingsAngels/settingsAngels_getAngelsByPropUid	Obtiene la colección de ángeles definidos para el usuario de la tabla user_settings que está identificado por el parámetro de entrada uid .
GET settingsAngels/settingsAngels_getAngelsByUid	Obtiene un ángel de la tabla settings_angels que está identificado por el parámetro de entrada uid .
GET settingsAngels/settingsAngels_getAngelsByUidFacebook	Obtiene un ángel perteneciente a Facebook de la tabla settings_angels que está identificado por el parámetro de entrada uidFacebook .

Tabla 5.4: Tabla de Servicios RESTFul del recurso settingsAngels

Recurso	Descripción
GET user/user_getEntities	Obtendrá todos los usuarios de la tabla user .
GET user/user_getUserByUid	Obtiene el usuario de la tabla user que indica el identificador de entrada uid .
POST user/user_setNewUser	Almacena a un nuevo usuario en la tabla user .
POST user/user_setUserByUid	Actualiza la información del usuario de la tabla user que indica el identificador de entrada uid .

Tabla 5.5: Tabla de Servicios RESTFul del recurso user

Recurso	Descripción
GET userOpenSocial/userOpenSocial_getEntities	Obtiene todos los usuarios de la tabla user_opensocial .
POST userOpenSocial/userOpenSocial_setUserOpenSocialByUid	Actualiza la información de un usuario de la tabla user_opensocial .

Tabla 5.6: Tabla de Servicios RESTFul del recurso userOpenSocial

5.4.2.6. Recurso userFacebook

Este será el recurso que accederá a la tabla **userFacebook**. Todos sus métodos están definidos en la tabla 5.7.

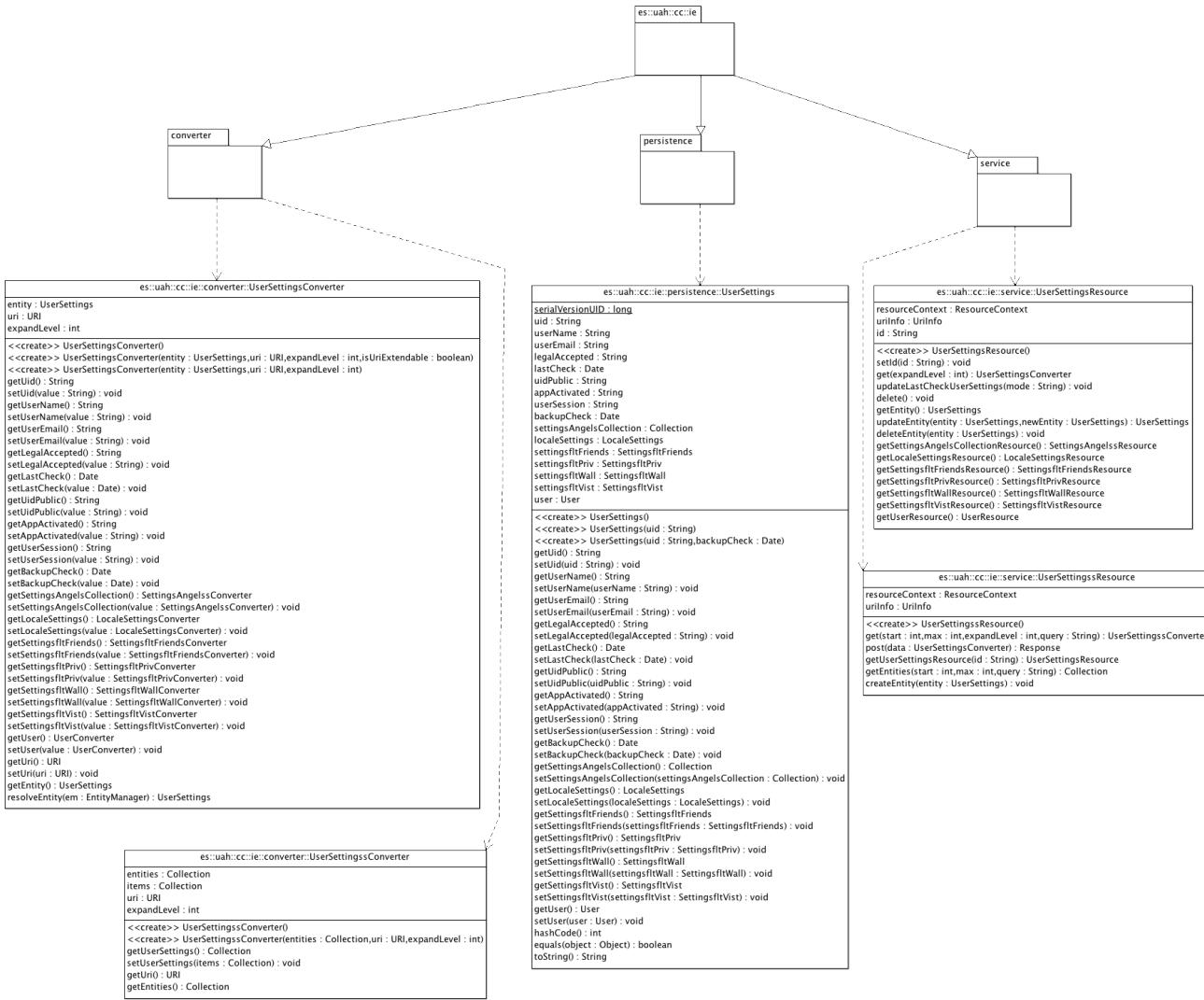
Tabla 5.7: Tabla de Servicios RESTFul del recurso userFacebook

Recurso	Descripción
GET userFacebook/userFacebook_getEntities	Obtendrá todos los usuarios de la tabla user_Facebook .
POST userFacebook/userFacebook_setNewUserFacebook	Almacena a un nuevo usuario en la tabla user_Facebook .
GET userFacebook/userFacebook_getUserFacebookByUid	Obtiene el usuario de la tabla user_Facebook que indica el identificador de entrada uid .
PUT userFacebook/userFacebook_setUserFacebookByUid	Actualiza la información del usuario de la tabla user_Facebook que indica el identificador de entrada uid .
POST userFacebook/userFacebook_setNewStreamComentsByUid	Inserta en la tabla stream_Facebook los nuevos comentarios en el muro de Facebook del usuario que indica el identificador de entrada uid .
GET userFacebook/userFacebook_getStreamComentByUid	Devuelve de la tabla stream_Facebook el comentario referenciado por el campo “postId”.
PUT userFacebook/userFacebook_setStreamComentsByUid	Actualiza un comentario de la tabla stream_Facebook con su nueva fecha de creación y de actualización.
POST userFacebook/userFacebook_setComentsPost	Establece un comentario a un post de muro de la tabla stream_Facebook con su nueva fecha de creación en la tabla comment_facebook .
GET userFacebook/userFacebook_getComentsPostById	Obtiene todos los comentarios a un post de muro de la tabla stream_facebook .
GET userFacebook/userFacebook_getComentsPostByTime	Obtiene todos los comentarios a un post de muro de la tabla stream_facebook que se han producido posteriormente a una fecha indicada por el parámetro time .

Recurso	Descripción
GET userFacebook/userFacebook_getStreamFacebookByUid	Obtiene todos los post del muro de un usuario cuyo identificador está contenido en el parámetro de entrada uid .
GET userFacebook/userFacebook_getStreamFacebookByUpdatedTime	Obtiene todos los post del muro de un usuario cuyo identificador está contenido en el parámetro de entrada uid y que se han producido a partir de la fecha contenida en el parámetro de entrada updatedTime .
GET userFacebook/userFacebook_getFriendsFacebookByUidCollection	Obtiene la colección de amigos en Facebook del usuario identificado por el parámetro de entrada uid .
GET userFacebook/userFacebook_isNewFriendsFacebookByUid	Indica si un amigo de Facebook, identificado por el parámetro de entrada friendUid , está o no registrado en la base de datos SocialNetwork.
GET userFacebook/userFacebook_getFriendsFacebookByUid	Devuelve toda la información de un amigo de Facebook, identificado por el parámetro de entrada friendUid .
POST userFacebook/userFacebook_setNewFriendFacebook	Almacena en la base de datos SocialNetwork la información de un nuevo amigo de Facebook del usuario.
PUT userFacebook/userFacebook_setFriendsFacebookByUid	Actualiza en la base de datos SocialNetwork la información de un amigo de Facebook del usuario.

5.4.2.7. Recurso userOpenSocial

Este será el recurso que accederá a la tabla **userOpenSocial**. Todos sus métodos están definidos en la tabla 5.6.

Figura 5.4: Diagrama de clases para el recurso `UserSettings`

Capítulo 6

Módulo Servidor

6.1. Introducción

El Módulo Servidor de la aplicación será la parte de mayor importancia. Como bien se indicó en los primeros capítulos, es el módulo que mayor responsabilidad tiene, ya que controla todos los accesos a los datos de los usuarios, gestionará su configuración, analizará sus datos y enviará informes con las anomalías detectadas en la actividad social del usuario.

A parte de todas estas funciones, deberá hacerse cargo de las siguientes responsabilidades funcionales:

1. **Garantizar el acceso único a la información de un usuario desde la propia configuración:** Se realizará un control exhaustivo sobre la información de un usuario, garantizando que un único usuario siempre acceda únicamente a su información.
2. **Garantizar la privacidad de la información almacenada en la base de datos SocialNetwork:** Se almacenarán datos personales de carácter privado y la aplicación deberá custodiarlos para que ningún otro usuario, de la aplicación o no, pueda acceder a ellos y realizar un uso fraudulento.
3. **Analizar toda la actividad social de un usuario y comunicar cualquier anomalía definida en los filtros a sus ángeles:** La aplicación será capaz de analizar cualquier dato almacenado en la base de datos para detectar anomalías que desemboquen en amenazas para el perfil del usuario, informando cada una de ellas a los ángeles definidos por el usuario.
4. **Mantener en todo momento informados a los ángeles que se han configurado para un determinado usuario:** Cuando se produzcan anomalías, será responsabilidad de la aplicación el envío, en forma de notificación, de la información relacionada a las anomalías detectadas a cada una de las personas o perfiles definidos como ángeles por parte del usuario, para que éstos puedan estar debidamente informados y tomen las medidas oportunas.

5. **Garantizar la seguridad de la información que se envía desde la propia aplicación hasta servidores web externos a ella, tales como Gmail o Facebook:** La aplicación se basa en llamadas a instrucciones públicas de acceso a diferentes redes sociales, en las cuales se accederá a datos personales del usuario, tanto de envío como de recepción. Por esta razón, se garantizará en todo momento la privacidad en los envíos de información, protegiendo ésta de los posibles ataques de otros usuarios.

Esta aplicación estará desarrollada íntegramente en Java y configurada como proyecto Maven¹ y, como tal, contendrá un archivo **pom.xml** en el que se encuentran todas las librerías externas a java que se han utilizado para el desarrollo de la aplicación. Éste archivo se encontrará en los archivos de configuración del proyecto, el cual, podrá ser modificado en función de las necesidades de nuevos desarrollos.

La aplicación estará jerarquizada en la estructura de paquetes que se muestran en la imagen 6.1. Todas estas estructuras serán explicadas en las siguientes secciones.

6.2. Paquete es.uah.cc.ie.snsangeguardfb

Este será el paquete raíz de la aplicación. A partir de éste se desplegará toda la funcionalidad del servidor de la aplicación. Contendrá las siguientes estructuras:

1. Clase Manager de la aplicación **SNSAngelGuardFBManager.java**: Será la clase raíz de la aplicación. Todas las operaciones irán gestionadas desde éste manager. Además, todas las estructuras de la aplicación contendrán un atributo de este tipo para mantener en todo momento un modelo de encapsulación de referencias.
2. Clase **ConfigurationManager**: Será la clase que lea los ficheros externos de configuración, necesarios para el funcionamiento de la aplicación, y proporcione sus ficheros al resto de estructuras de la aplicación.
3. Interface **ILifeCycleFilter**: Esta interface define el ciclo de vida de un filtro. Será necesaria su implementación para todos los filtros que se quieran utilizar dentro de la aplicación. La arquitectura de ésta está preparada para soportar todos los filtros que se quieran implementar, con tan sólo el requisito de implementar ésta interface.

¹Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

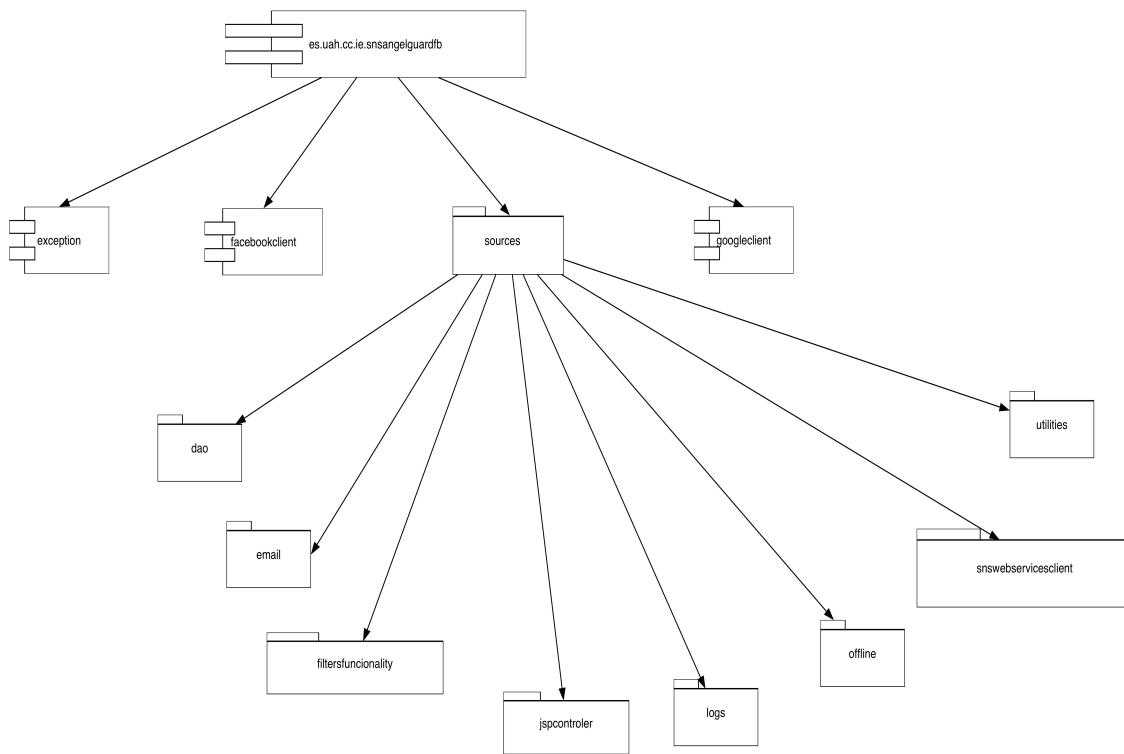


Figura 6.1: Diagrama de despliegue de la aplicación SNSAngelGuardFB

4. Paquete **facebookclient**: Contendrá todas las estructuras necesarias para realizar los accesos a los datos de Facebook de cada usuario.
5. Paquete **exception**: Contendrá todas las estructuras necesarias para el control de los errores en tiempo de ejecución que se puedan producir en la aplicación.
6. Paquete **sources**: Contendrá todas las clases y paquetes necesarios para realizar toda la lógica de ejecución de la aplicación, tales como funcionalidades para el acceso a base de datos, control offline de la aplicación, control de la parte cliente, ..., etc.
7. Paquete **googleclient**: Contendrá las estructuras necesarias para utilizar el API pública **Google Contacts**, para el acceso a los contactos de una cuenta de Google y poder seleccionar los contactos que el usuario de la aplicación crea conveniente.

En la figura 6.2 puede observarse la estructura de la aplicación, que se pasará a analizar en las siguientes secciones.

6.2.1. Clase SNSAngelGuardFBManager.java

Esta clase será la **clase Manager Principal de la aplicación**. Desde ésta se realizarán las siguientes acciones:

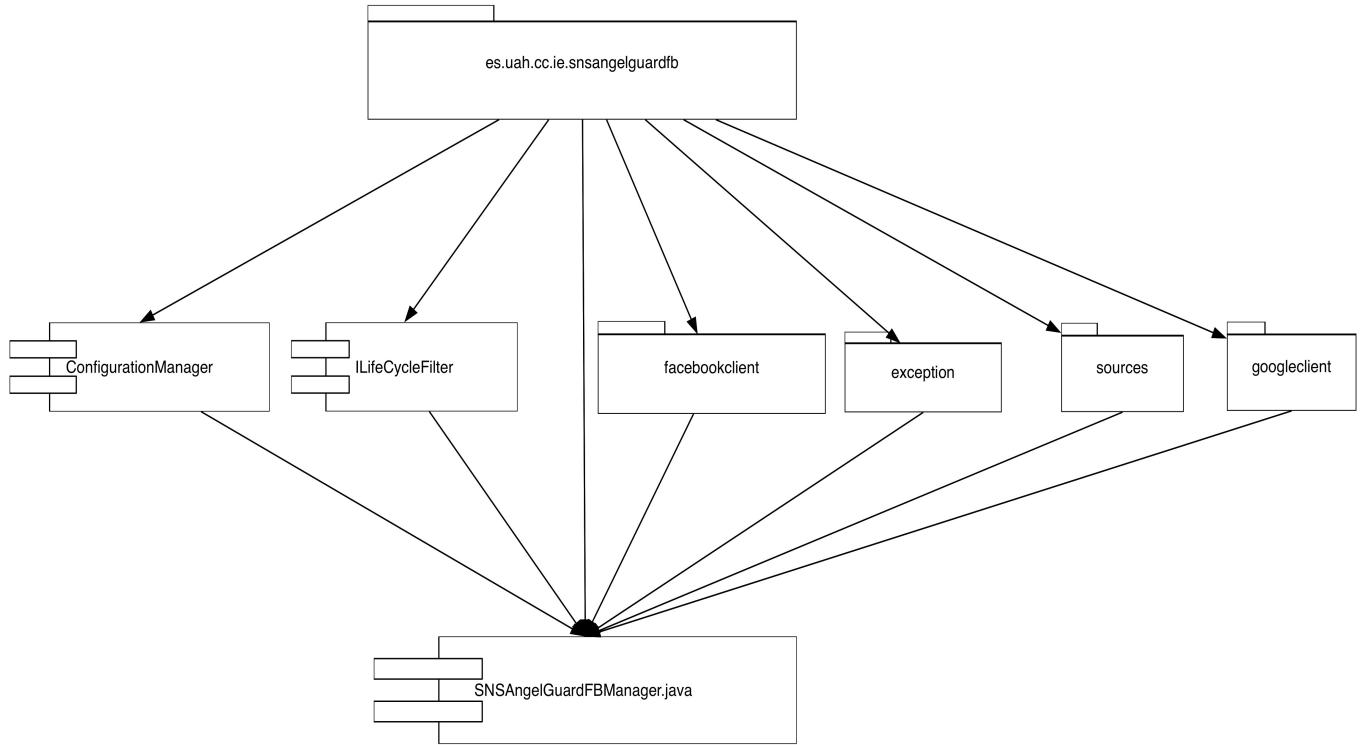


Figura 6.2: Diagrama del paquete es.uah.cc.ie.snsangelguardfb

1. Inicialización de la aplicación: Inicializará todos los módulos para comenzar a gestionar datos de la aplicación. Dentro de ésta clase existirán toda clase de estructuras que permitirán orquestar la correcta ejecución de la aplicación en todos sus niveles.
2. Gestionará las peticiones de conexión a Facebook de un usuario y será el encargado de orquestar todos sus elementos para el envío y la recepción de los datos que se quieren obtener de un determinado perfil de usuario.
3. Gestionará las peticiones de los distintos usuarios que se conecten a la aplicación mediante el módulo cliente de su propio navegador.
4. Gestionará todos los datos de cada usuario de forma restrictiva no permitiendo accesos no autorizados de información de otros usuarios de la aplicación.
5. Será el encargado de inicializar el proceso de actualización de datos offline que se ejecutará diariamente para garantizar la integridad de la información dentro de la base de datos SocialNetwork.

La tabla 6.1 representa los atributos que contendrá la clase **SNSAngelGuardFBManager.java**:

Tabla 6.1: Atributos de SNSAngelGuardFBManager.java

Campo	Tipo	Descripción
logger	Logger	Clase estática para la gestión de las trazas de información en la clase.
configurationManager	ConfigurationManager	Clase que gestiona la configuración de la aplicación.
facebookClient	FacebookClientLocal	Conjunto de APIs en formato FQL.
facebookRestClient	DefaultLegacyFacebookClient	API pública de Facebook, perteneciente al paquete com.restfb , encargado de realizar todas las llamadas directas para el acceso a datos de un determinado perfil de Facebook.
facebookQueryClient	FacebookClient	API pública de Facebook, perteneciente al paquete com.restfb , encargado de realizar todas las llamadas de lenguaje FQL ¹ .
userSettingsDaoManager	UserSettingsDaoManager	Objeto DAO de control de los datos del usuario.
localeSettingsDaoManager	LocaleSettingsDaoManager	Objeto DAO para los recursos de idioma del usuario actual .
client	SNSdataBaseClient	Cliente REST de base de datos SocialNetwork.
exceptionManager	ExceptionManager	Control de las excepciones que se produzcan en la aplicacion.
genericFilter	GenericFilterFuncionality	Objeto de Filtros Generico.
jsonUtilities	JSONUtilities	Operaciones comunes al tipo de datos JSON.
angelsUtilities	AngelsUtilities	Operaciones comunes para la definición y el manejo de ángeles.
dateTimeUtilities	DateTimeUtilities	Operaciones comunes para el manejo de fechas.
stringUtilities	StringUtilities	Operaciones comunes para el formateo de parámetros.
userUtilities	UserUtilities	Utilidades para operaciones del usuario de la aplicación.
emailObject	GenericEmailObject	Objeto para el envio de emails.
newConnection	boolean	Atributo de nueva conexion.
inicio	boolean	Atributo de inicio de la aplicacion.
context	String	Contexto de la aplicación.

¹Facebook Query Languaje, lenguaje proporcionado por Facebook para el acceso a datos de sus tablas de información.

Y sus métodos más importantes, a parte de sus métodos de acceso a atributos de clase, estarán representados en la tabla 6.2:

Método	Entrada	Descripción
getSessionInstance	HttpServletRequest request	Obtiene el objeto SNSAngelGuardFBManager por el cual se mantiene abierta la sesión para un mismo usuario durante toda la navegación de la aplicación.
getLoginFacebook	HttpServletRequest request, HttpServletResponse response	Realiza el login del usuario actual en Facebook.
logSession	HttpServletRequest request, HttpServletResponse response	Establece la sesión actual en Facebook para el usuario conectado.
cerrarConexionSNS		Cierra la conexión con la base de datos de SocialNetwork.
getLoginAppOffline	HttpServletRequest request, HttpServletResponse response	Obtiene la conexión Offline del usuario que se está tratando actualmente en la aplicación.

Tabla 6.2: Métodos de SNSAngelGuardFBManager.java

6.2.2. Clase ConfigurationManager.java

Será la clase que gestione la lectura de los ficheros externos de configuración y contendrá los parámetros de éstos para que el resto de estructuras de la aplicación puedan utilizarlos en cualquier momento.

La tabla 6.3 hace referencia a los atributos y la tabla 6.4 a sus métodos.

Tabla 6.3: Atributos de ConfigurationManager.java

Campo	Tipo	Descripción
PATH_FILE_CONFIG	final String	Ruta al fichero de configuración.
PATH_FILE_FILTERS	final String	Ruta al fichero de filtros activos.
NAME_CONTEXT_APPLICATION	final String	Clave al host de la aplicación en el fichero de configuración.
NAME_CONTEXT_APPLICATION_SSL	final String	Clave SSL al host de la aplicación en el fichero de configuración.

Campo	Tipo	Descripción
NAME_CONTEXT_RESTFULWS	final String	Clave al host de los servicios RESTful en el fichero de configuracion.
API_KEY	final String	Clave de aplicacion SNSAngelGuard en Facebook.
API_SECRET_KEY	final String	Clave secreta de la aplicacion SNSAngelGuard en Facebook.
PATH_KEYSTORE_SSL	final String	Clave del path del certificado de confianza SSL.
PASSWORD_KEYSTORE_SSL	final String	Clave del password del certificado de confianza SSL.
PATH_APPLICATION_FACEBOOK	final String	Clave de la direccion de la aplicacion en Facebook.
PATH_LEXICAL_FILES	final String	Clave de la dirección del servidor donde se encuentran los ficheros de idioma referentes al control del lenguaje.
XML_KEY_ID_FILTER	final String	Key para el identificador del filtro en el fichero xml.
XML_VALUE_CLASS	final String	Key para el valor de la clase del filtro.
GOOGLE_APP_NAME_KEY	final String	Key para el nombre de la aplicacion dada de alta en Google API Console.
GOOGLE_APP_CLIENT_ID_KEY	final String	Key para el identificador de la aplicacion generado automaticamente al dar de alta la apliacion en Google API Console.

Campo	Tipo	Descripción
GOOGLE_APP_CLIENT_SECRET_KEY	final String	Key para el identificador secreto de la aplicacion generado automaticamente al dar de alta la aplicacion en google API Console.
configHostApplication	String	Host de la aplicacion.
configHostApplicationSSL	String	Host SSL de la aplicacion.
configHostRESTFullWS	String	Host a los Servicios RESTFul.
apiKey	String	Identificador de Facebook de la aplicacion.
apiSecretKey	String	Identificador secreto de Facebook de la aplicacion.
pathKeyStoreSSL	String	Path al certificado de confianza SSL.
passwordKeyStoreSSL	String	Password del certificado de confianza SSL.
pathApplicationFacebook	String	Path de la aplicacion en Facebook.
pathLexicalFiles	String	Path a los ficheros de idioma propios del filtro de control de lenguaje.
listActiveFilters	List<String>	Lista de keys de los filtros activos.
keyValueClassFilter	Map<String, String>	Mapa que contiene los filtros a crear.
googleAppName	String	Nombre de la aplicacion dada de alta en Google API Console.
googleClientId	String	Identificador de la aplicacion generado automaticamente al dar de alta la aplicacion en Google API Console.

Campo	Tipo	Descripción
googleClientSecret	String	Identificador secreto de la aplicación generado automáticamente al dar de alta la aplicación en google API Console.

Método	Entrada	Salida	Descripción
loadConfigFile()			Carga el fichero de configuración principal con los parámetros necesarios.
loadActiveFiltersFile()			Carga el fichero de definición de los filtros activos.

Tabla 6.4: Métodos de la clase ConfigurationManager.java

6.2.3. Interface ILifeCycleFilter

Esta interfaz es el componente que definirá el ciclo de vida de los filtros. Cada filtro que se implemente en la aplicación deberá implementar la interfaz **ILifeCycleFilter** para conservar la arquitectura de la aplicación y sea totalmente configurable. Con ésta interfaz, los filtros aparecerán en pantalla automáticamente.

La tabla 6.5 hace referencia los métodos que se tendrán que implementar.

6.2.4. Paquete es.uah.cc.ie.snsangeguardfb.exception

Este paquete contendrá las estructuras necesarias para realizar todos los controles de excepciones que se produzcan durante la ejecución de la aplicación.

6.2.4.1. Enumerado CodeException.java

Cuando se capture una excepción, dependiendo de su tipología, se le asignará un código de error para que su tratamiento sea el más correcto posible. Estos códigos de error son dados de alta por los desarrolladores de la aplicación y cubren la mayor parte de los errores que se pueden producir en la misma.

El enumerado **CodeException** contendrá todos estos códigos que soporta la aplicación, los cuales podrán ser:

Método	Entrada	Salida	Descripción
init()	SNSAngelGuardFBManager snsObject, String id		Inicializa el filtro con la clase manager de la aplicacion..
executeFilter()	Map<String, Object>args	String	Ejecuta la funcionalidad del filtro pasándole por parámetro un mapa con todos los parámetros necesarios para su ejecución. Devolverá una cadena de texto en formato HTML con el resultado de la ejecución.
getInformationFacebook()			Obtiene toda la información interna de Facebook necesaria para el funcionamiento del filtro y la almacena en base de datos.
updateInformationFacebook()			Actualiza la información interna en base de datos para analizarla.
getId()		String	Obtiene el identificador del filtro.

Tabla 6.5: Métodos de la clase ILifeCycleFilter

```

ERROR_NO_AUTH_ACESSTOKEN,
ERROR_CLIENT_DATABASE,
ERROR_SERVER_DATABASE,
USER_NOT_FOUND_UID_PUBLIC,
UNSUPPORTED_ENCODING,
MYSQL_INTEGRITY_CONSTRAINT_VIOLATION_EXCEPTION,
PARSE_EXCEPTION,
DATA_LENGTH_EXCEPTION,
ILEGAL_STATE_EXCEPTION,
NO_SUCH_PROVIDER_EXCEPTION,
MESSAGING_EXCEPTION,
JSON_EXCEPTION,
IO_EXCEPTION,
UNIFORM_INTERFACE_EXCEPTION,
UNKNOWN_ERROR

```

6.2.4.2. Clase ExceptionManager.java

Será la clase que controle y gestione todas las excepciones que se producen en la aplicación. Recibirá como atributos el objeto manager de la aplicación y un CodeException indicando el error que se ha producido. Sus métodos están ilustrados en la tabla 6.6.

Método	Entrada	Salida	Descripción
initControlException()			Realiza el control de la excepción producida, lanzando excepciones de los tipos InterDataBaseException, InterProcessException o InterEmailException.
exceptionToString()	Exception e	String	Obtiene el contenido de una excepción y lo retorna en formato String, pudiendo lanzar excepciones del tipo UnsupportedEncodingException.

Tabla 6.6: Métodos de la clase ExceptionManager.java

6.2.4.3. Clase GenericException.java

Será una clase genérica que representa a todas las excepciones que pueden darse en la aplicación. Recibirá como parámetros la excepción capturada y un CodeException, que serán recibidos por el objeto en su constructor. Los métodos que esta clase pone a disposición serán los métodos get de acceso a sus dos atributos.

6.2.4.4. Clase InterDataBaseException.java

Esta clase manejará todas las excepciones que se produzcan a nivel de operaciones con la base de datos. Extenderá de la clase **GenericException**. No tendrá atributos propios, únicamente los de su clase padre. Tampoco dispondrá de métodos adicionales, únicamente cuando se ejecute su constructor, llamará a su instancia superior con la excepción y su CodeException.

6.2.4.5. Clase InterEmailException.java

Esta clase manejará todas las excepciones que se produzcan cuando se realizan operaciones de envío de emails. Extenderá de la clase **GenericException**. No tendrá atributos propios, únicamente los de su clase padre. Tampoco dispondrá de métodos adicionales, únicamente cuando se ejecute su constructor, llamará a su instancia superior con la excepción y su CodeException.

6.2.4.6. Clase InterProcessException.java

Esta clase manejará todas las excepciones que se produzcan en las operaciones de proceso de la aplicación. Extenderá de la clase **GenericException**. No tendrá atributos propios, únicamente los de su clase padre. Tampoco dispondrá de métodos adicionales, únicamente cuando se ejecute su constructor, llamará a su instancia superior con la excepción y su CodeException.

6.2.5. Paquete es.uah.cc.ie.snsangeguardfb.facebookclient

Este paquete contendrá todas las estructuras necesarias para dar soporte a los datos obtenidos de Facebook. Cuando se realiza una petición de datos en Facebook, la llamada parsea los datos directamente en estructuras diseñadas bajo la interface **com.restfb.Facebook** la cual, por medio de anotaciones tipo **@Facebook** obtendrá el parámetro deseado y lo almacenará en una variable marcada con ésta anotación.

La interface **com.restfb.Facebook** pertenecerá al paquete de datos **restfb**, adjuntado en el proyecto dentro del archivo **pom.xml** mediante la siguiente dependencia:

```
<dependency>
    <groupId>com.restfb</groupId>
    <artifactId>restfb</artifactId>
    <version>1.6.12</version>
</dependency>
```

En ésta dependencia Maven se encontrarán todas las estructuras necesarias para realizar llamadas a Facebook para obtener datos del usuario que posteriormente serán procesados por la aplicación, almacenados en la base de datos **socialNetwork** y, periódicamente, analizados para conformar los informes que serán enviados a los ángeles de cada usuario.

Para realizar una petición de datos a Facebook, tomaremos como ejemplo una llamada para obtener los datos básicos de un usuario, tales como su nombre y su email dentro de ésta Red Social:

```
List<InfoCurrentUser> jsonUserInfo = this.snsObject.getFacebookRestClient().executeForList("Users.getInfo",
    InfoCurrentUser.class,
    Parameter.with("uids", uidLong),
    Parameter.with("fields", "name,email"));
```

En ésta llamada, podemos diferenciar las siguientes secciones:

1. Objeto **this.snsObject.getFacebookRestClient()**: Ésta llamada hará referencia a nuestro objeto **Rest** inicializado en el manager general de la aplicación **snsObject**. La inicialización de éste objeto se lleva a cabo mediante la siguiente llamada:

```
this.facebookRestClient = new DefaultLegacyFacebookClient(accessToken);
```

En la cual necesitaremos la clave **accessToken**. Ésta clave es la que proporciona Facebook a cada usuario que se conecta a ella, por lo que será renovada en cada conexión que el usuario realice. La aplicación tendrá un mecanismo de actualización de dicha clave cada vez que el usuario acceda a ésta, por lo que será renovada automáticamente para no tener problemas en los accesos offline.

El cometido de dicho objeto será el de realizar todas las llamadas a Facebook para obtener datos de cada usuario y poder ser procesados por la aplicación.

2. Especificación de datos **Users.getInfo**: Mediante ésta parte de la llamada, estamos especificando a Facebook que los datos que queremos obtener son los referentes a la información del perfil del usuario que se conecta a la aplicación.
3. Objeto **InfoCurrentUser.class**: Este será el objeto que recepcionará los datos obtenidos de Facebook y los procesará dentro de él para que estén a disposición de la aplicación.
4. Identificador de usuario **uidLong**: Cada usuario que se conecta a Facebook tiene asignado un identificador único generado en el momento de la creación de la cuenta. En el momento en el que un usuario se conecta a Facebook, ésta nos va a devolver un mensaje de confirmación con su identificador y su accessToken, datos obligatorios para poder realizar llamadas para obtención de datos.
5. Parámetros de llamada **fields**: Esta etiqueta le indica a Facebook los datos que, dentro de la información de usuario indicada por el parámetro **Users.getInfo**, se quieren obtener con la llamada. En éste caso, se obtendrán el nombre y el email del usuario conectado a la aplicación.

Tras realizar la llamada, todos los datos serán almacenados en la variable de tipo **List** que estará compuesta de objetos tipo **InfoCurrentUser**, el cual tendrá la siguiente estructura:

```
public class InfoCurrentUser extends GenericDataFacebook {

    @Facebook("uid")
    String uid;

    @Facebook("name")
    String name;

    @Facebook("email")
    String email;
```

```

...
(Resto de métodos)
...
}

}

```

Como se puede observar en ésta clase, los parámetros que se quieren obtener de Facebook irán precedidos de la anotación **@Facebook** junto con el nombre del parámetro.

Éste proceso será generalizado para la obtención de cualquier dato almacenado en Facebook. Para realizar ésta llamada, únicamente deberemos definir en una estructura de éste estilo los datos que se quieren obtener y, a partir de ésta, realizar la llamada para que el parseo sea directo y tengamos los datos al instante para poder ser procesados.

El paquete que nos compete actualmente contendrá todas las estructuras necesarias para realizar estos accesos a la información de Facebook. Toda su estructura está mostrada en la figura 6.3.

6.2.5.1. Clase GenericDataFacebook.java

La clase **GenericDataFacebook.java** será una clase de definición **abstracta** de la cual extenderán todas aquellas clases del paquete **data** que se utilicen para realizar llamadas a Facebook para la obtención de datos de usuario. Como clase abstracta, tendrá únicamente el método de definición **toJson()** que parseará todos los datos de una clase y que se han obtenido de Facebook a un objeto JSON para poder ser tratados y procesados en la aplicación. El método tendrá la siguiente definición:

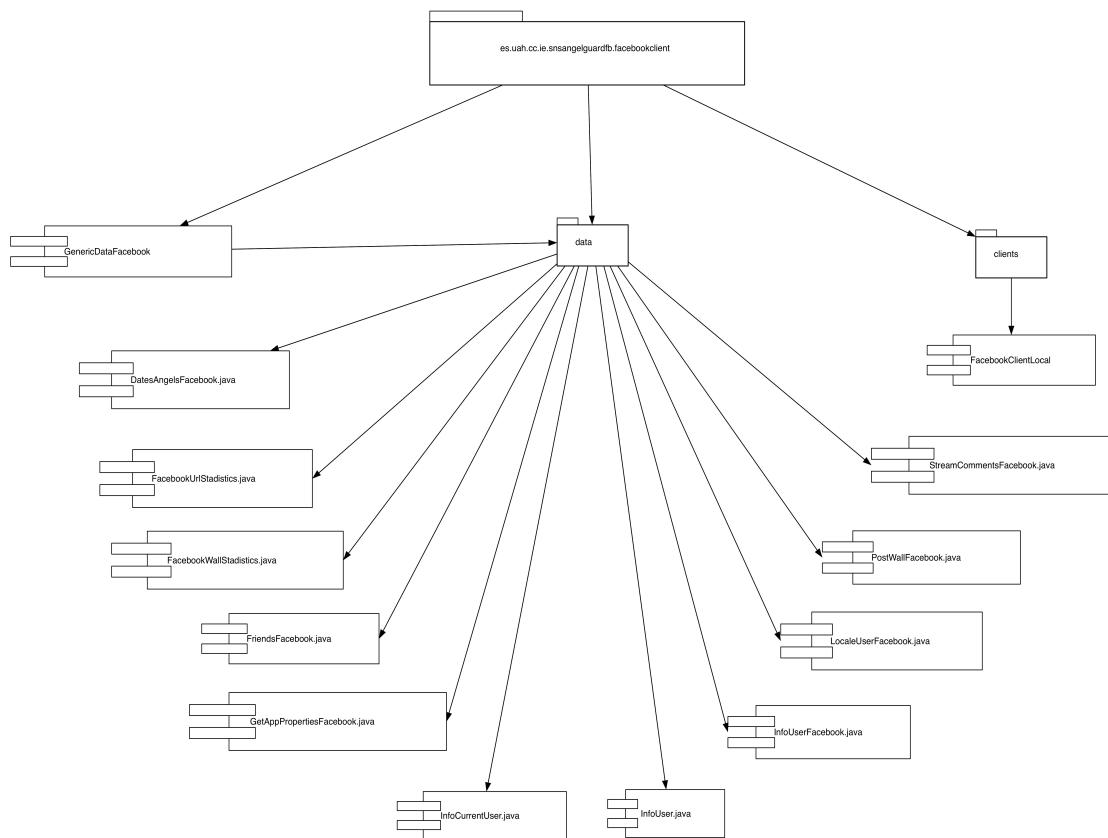
```
public abstract JSONObject toJson();
```

6.2.6. Paquete es.uah.cc.ie.snsangelguardfb.facebookclient.data

Contendrá todas las estructuras necesarias para obtener datos de Facebook. Todas estas clases extienden de la clase **GenericDataFacebook.java**, por lo que para acceder a todos sus datos bastará con definir un objeto JSON que contenga todos los datos de la clase. Los parámetros que necesitemos en cada una de las clases siguientes irán precedidos en su definición con la anotación **@Facebook** y entre paréntesis el nombre en Facebook del parámetro al que queremos acceder, tal y como se ha mostrado en el ejemplo de la clase **InfoCurrentUser**.

6.2.6.1. Clase DatesAngelsFacebook.java

Esta clase será la encargada de obtener los datos de un ángel de Facebook que posteriormente serán almacenados en la base de datos. Un ángel de Facebook no es más que un contacto de ésta red social que ha sido seleccionado por un usuario de la aplicación para que tutele su actividad social. Como tal, sólo necesitaremos los siguientes datos:

Figura 6.3: Diagrama del paquete `es.uah.cc.ie.snsangelguardfb.facebookclient`

1. Su identificador dentro de la red social.
2. Su dirección de correo electrónico, para poder enviarle notificaciones de información. En éste caso, sólo podremos elegir un ángel de Facebook si está dado de alta dentro de la aplicación.
3. Su nombre.
4. La dirección url de su fotografía en Facebook.

La tabla 6.7 hace referencia a los atributos y la tabla 6.8 a sus métodos.

Atributo	Tipo	Descripción
uid	String	Identificador del ángel en Facebook.
email	String	Email del ángel registrado en Facebook.
name	String	Nombre del contacto en Facebook.
picSquare	String	Ruta a la imagen de perfil en Facebook.

Tabla 6.7: Atributos de la clase DatesAngelsFacebook.java

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase DatesAngelsFacebook.

Tabla 6.8: Métodos de la clase DatesAngelsFacebook.java

6.2.6.2. Clase FacebookUrlStadistics.java

Esta clase muestra información sobre estadísticas de un perfil de Facebook tal como el número de amigos con los que cuenta o si tiene amigos en común con otros contactos. Los datos que almacenará serán los siguientes:

1. Su identificador dentro de la red social.
2. La url que apunta a su imagen de perfil dentro de Facebook.
3. El número de amigos del perfil en Facebook.
4. En caso de compararlo con otro perfil, el número de contactos que comparte con éste en Facebook.

La tabla 6.9 mostrará los atributos de ésta clase y la tabla 6.10 mostrará sus métodos.

Atributo	Tipo	Descripción
name	String	Nombre del contacto en Facebook.
picSquare	String	Ruta a la imagen de perfil en Facebook.
friendCount	String	Numero de contactos del perfil en Facebook.
mutualFriendCount	String	En caso de compararlo con otro perfil, mostrará los amigos que comparte con éste último.

Tabla 6.9: Atributos de la clase FacebookUrlStadistics.java

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase FacebookUrlStadistics.

Tabla 6.10: Métodos de la clase FacebookUrlStadistics.java

6.2.6.3. Clase FacebookWallStadistics.java

Esta clase almacenará la información adicional referente a una entrada-comentario en el muro de un usuario. Estos datos serán los siguientes:

1. El identificador del usuario de Facebook que realiza el comentario.
2. El usuario de Facebook al que va dirigido el comentario.
3. El número de 'Me gusta' que el comentario recibe.
4. Los identificadores de los comentarios que recibe la entrada del muro.

La tabla 6.11 mostrará los atributos de ésta clase y la tabla 6.12 mostrará sus métodos.

6.2.6.4. Clase FriendsFacebook.java

Esta clase contendrá la información básica de un amigo de Facebook. Esta información se almacenará tal cual de ésta clase a la tabla **friends_facebook** de la base de datos **SocialNetwork**, salvo la fecha de nacimiento, que será parseado a un formato propio entendible por el sistema. Como tal, contendrá la siguiente información:

1. El identificador del amigo en Facebook.
2. Su nombre.
3. Su fecha de nacimiento, que será de utilidad para el filtro de control de amistades.

Atributo	Tipo	Descripción
actorId	String	Identificador del usuario de Facebook que realiza el comentario.
targetId	String	Identificador del usuario de Facebook a quien va dirigido el comentario.
likes	String	Número de 'Me gusta' que el comentario recibe.
comments	String	Cadena de identificadores de los comentarios que recibe.

Tabla 6.11: Atributos de la clase FacebookUrlStadistics.java

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase FacebookWallStadistics.

Tabla 6.12: Métodos de la clase FacebookUrlStadistics.java

4. La url a su imagen principal de perfil.

La tabla 6.13 mostrará los atributos de ésta clase y la tabla 6.14 mostrará sus métodos.

Atributo	Tipo	Descripción
uid	String	Identificador del amigo en Facebook.
name	String	Nombre del amigo en Facebook.
birthdayDate	String	Fecha de nacimiento registrada en Facebook del amigo.
picSquare	String	Ruta a la imagen de perfil del amigo en Facebook.

Tabla 6.13: Atributos de la clase FriendsFacebook.java

6.2.6.5. Clase GetAppPropertiesFacebook.java

Esta clase contendrá información de la propia aplicación **SNSAngelGuardFB**, que pueden ser útil en algún futuro cercano para analizar y realizar estadísticas sobre su uso. Actualmente, se almacena la siguiente información:

1. El identificador interno de la aplicación.
2. El ámbito de acción de la aplicación.

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase FriendsFacebook.

Tabla 6.14: Métodos de la clase FriendsFacebook.java

3. La url de desinstalación de la aplicación.

La tabla 6.15 mostrará los atributos de ésta clase y la tabla 6.16 mostrará sus métodos.

Atributo	Tipo	Descripción
appId	String	Identificador de la aplicación en Facebook.
publishAction	String	Ámbito de acción de la aplicación.
uninstallUrl	String	Url de desinstalación de la aplicación.

Tabla 6.15: Atributos de la clase GetAppPropertiesFacebook.java

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase GetAppPropertiesFacebook.

Tabla 6.16: Métodos de la clase GetAppPropertiesFacebook.java

6.2.6.6. Clase InfoCurrentUser.java

Esta clase almacenará información del usuario que actualmente está conectado a la aplicación. Será de utilidad para contrastar posibles sesiones y no mezclar datos entre usuarios. La información de cada usuario de la aplicación que se almacenará será la siguiente:

1. Su identificador en Facebook.
2. Su nombre.
3. Su email.

La tabla 6.17 hace referencia a los atributos y la tabla 6.18 a sus métodos.

Atributo	Tipo	Descripción
uid	String	Identificador del usuario de Facebook.
name	String	Nombre del usuario en Facebook.
email	String	Email del usuario en Facebook.

Tabla 6.17: Atributos de la clase InfoCurrentUser.java

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase InfoCurrentUser.

Tabla 6.18: Métodos de la clase InfoCurrentUser.java

6.2.6.7. Clase InfoUser.java

Aunque ésta clase está diseñada para obtener los datos del usuario de Facebook, ésta funcionalidad se ha dividido en dos clases, tal y como está diseñada la base de datos, en la que tenemos una tabla para datos comunes del usuario, **user**, y dos tablas específicas para las dos redes sociales a las que la base de datos da soporte, **user_facebook** y **user_opensocial**. Esta clase almacenará toda la información común a ambas redes sociales, sólo que ésta será obtenida del perfil de Facebook y, mediante el objeto JSON obtenido mediante la implementación del método **toJson()**, devolverá un objeto que será persistido en la tabla **user** de la base de datos **SocialNetwork**. En ésta clase se almacenarán los siguientes datos:

1. Su identificador en Facebook.
2. Su sexo.
3. Su religión.
4. Su estado civil actual.
5. Sus tendencias políticas.
6. Sus preferencias a la hora de realizar actividades.
7. Sus intereses personales.
8. Si es usuario de la aplicación.
9. Sus gustos musicales.
10. Sus programas de televisión favoritos.
11. Sus películas favoritas.

12. Sus libros favoritos.
13. Descripción breve de la información acerca del usuario.
14. Su estatus actual.
15. Sus citas preferidas.

La tabla 6.19 hace referencia a los atributos y la tabla 6.20 a sus métodos.

Atributo	Tipo	Descripción
uid	String	Identificador del usuario de Facebook.
sex	String	Sexo del usuario.
religion	String	Email del usuario en Facebook.
relationshipStatus	String	Estado civil.
political	String	Tendencias políticas.
activities	String	Actividades.
interests	String	Intereses.
isAppUser	String	Indicador de si es o no usuario de la aplicación.
music	String	Gustos musicales.
tv	String	Programas de televisión.
movies	String	Películas favoritas.
books	String	Libros preferidos.
aboutMe	String	Descripción breve de la información acerca del usuario
status	String	Estatus social actual.
quotes	String	Citas de autor preferidas.

Tabla 6.19: Atributos de la clase InfoUser.java

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase InfoUser.

Tabla 6.20: Métodos de la clase InfoUser.java

6.2.6.8. Clase InfoUserFacebook.java

Como hemos especificado en el apartado anterior, ésta clase almacenará atributos de usuario propios del perfil de Facebook. La información que se obtiene de ésta clase es la siguiente:

1. Su identificador en Facebook.
2. Su nombre.
3. Su segundo nombre, propio de usuarios americanos.
4. Sus apellidos.
5. Las rutas a las diferentes fotos de perfil, dependiendo del tamaño que se elija.
6. Última fecha en la que se actualizó el perfil.
7. Su fecha de cumpleaños, en formato String.
8. Datos relativos a su formación.
9. Su email.
10. Restricciones del perfil.

La tabla 6.21 hace referencia a los atributos y la tabla 6.22 a sus métodos.

Tabla 6.21: Atributos de la clase InfoUserFacebook.java

Atributo	Tipo	Descripción
uid	String	Identificador del usuario de Facebook.
firstName	String	Nombre.
middleName	String	Segundo nombre.
lastName	String	Apellidos.
name	String	Nombre completo.
picSmall	String	Ruta a la imagen de perfil en tamaño pequeño.
picBig	String	Ruta a la imagen de perfil en tamaño grande.
picSquare	String	Ruta a la imagen de perfil en tamaño 50x50.
pic	String	Ruta a la imagen de perfil en tamaño normal.
profileUpdateTime	String	Última fecha de actualización del perfil.
birthday	String	Fecha de nacimiento.
birthdayDate	String	Fecha de nacimiento en formato Date.
signigicantOtherId	String	Identificador y nombre del usuario.
hsInfo	String	Información sobre la educación o el trabajo del usuario.
notesCount	String	Contador de notificaciones.
wallCount	String	Contador de mensajes del muro.
onlinePresence	String	Indicador de estado online/offline.

Atributo	Tipo	Descripción
locale	String	Idioma del usuario.
proxiedEmail	String	Email del usuario cifrado por Facebook para las notificaciones entre usuarios.
profileUrl	String	URL del perfil del usuario.
emailHashes	String	Encriptación del email.
picSmallWithLogo	String	Ruta a la imagen del perfil del usuario con el logo de Facebook en tamaño pequeño.
picBigWithLogo	String	Ruta a la imagen del perfil del usuario con el logo de Facebook en tamaño grande.
picSquareWithLogo	String	Ruta a la imagen del perfil del usuario con el logo de Facebook en tamaño 50x50.
picWithLogo	String	Ruta a la imagen del perfil del usuario con el logo de Facebook en tamaño normal.
allowedRestrictions	String	Restricciones permitidas en el perfil.
verified	String	Perfil confirmado.
profileBlurb	String	Indicador de si el perfil puede tener publicidad.
username	String	Nick online del usuario.
website	String	Página web del usuario.
isBlocked	String	Indicador de perfil bloqueado.
contactEmail	String	Email de contacto.
email	String	Email.
meetingFor	String	Información relativa a la búsqueda de relaciones sociales.
meetingSex	String	Sexo buscado para iniciar relaciones.

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase InfoUserFacebook.java

Tabla 6.22: Métodos de la clase InfoUserFacebook.java

6.2.6.9. Clase LocaleUserFacebook.java

Almacena información sobre el idioma que el usuario tiene configurado en Facebook. Este estará representado mediante un código alfanumérico formado por dos partes, el

idioma nativo y su posible variación, separado por un guión, tal que, el idioma castellano estará representado por el código 'es_ES'. Este código será procesado en la aplicación para reconocer qué idioma está usando el usuario. La información que se procesa en ésta clase es la siguiente:

1. El identificador del usuario en Facebook.
2. Su código de idioma.

La tabla 6.23 hace referencia a los atributos y la tabla 6.24 a sus métodos.

Atributo	Tipo	Descripción
uid	String	Identificador del usuario de Facebook.
locale	String	Código alfanumérico de idioma.

Tabla 6.23: Atributos de la clase LocaleUserFacebook.java

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase LocaleUserFacebook.

Tabla 6.24: Métodos de la clase LocaleUserFacebook.java

6.2.6.10. Clase PostWallFacebook

Esta clase representará la información referida a una entrada en el muro de Facebook del usuario, es decir, cada post que escriba el usuario o cualquiera de nuestros contactos, tendrá una estructura de información en la que se almacenarán los siguientes datos:

1. El identificador interno del post.
2. Identificador del usuario que ve el post.
3. El identificador de la aplicación en caso de que sea ésta la que escribe en nuestro muro.
4. El identificador a quien va dirigido el post.
5. Hora de actualización del post.
6. Hora de creación del post.
7. Mensaje que se ha escrito en el post.

La tabla 6.25 hace referencia a los atributos y la tabla 6.26 a sus métodos.

Atributo	Tipo	Descripción
postId	String	Identificador del post.
viewerId	String	Identificador del usuario que ve el post.
appId	String	Identificador de la aplicación.
sourceId	String	El identificador a quien va dirigido el post.
updatedTime	String	Hora de actualización.
createdTime	String	Hora de creación.
filterKey	String	Clave para filtrado.
attribution	String	Campo propio para post publicados por apps. Contiene el nombre completo de la aplicación.
actorId	String	Usuario, grupo, evento o aplicación que publica el post.
targetId	String	Usuario, grupo, evento o aplicación a quien va dirigido.
message	String	Mensaje de texto del post
appData	String	Array con parámetros específicos de la app que creó el post.
attachment	String	Array con información adicional del post.
type	String	Tipo del post. Se identificará con un entero de 32 bits con un código propio registrado en Facebook.
permalink	String	Url del post.
xid	String	Información asociada al chat de Facebook.

Tabla 6.25: Atributos de la clase PostWallFacebook.java

6.2.6.11. Clase StreamCommentsFacebook

Esta clase contendrá información de cada comentario a un post de entrada en el muro de usuario. Contendrá los siguientes datos:

1. Identificador interno del comentario.
2. Tipo de comentario, puede ser un video, una foto, un link, etc.
3. Identificador del usuario, grupo, evento o app que crea el comentario.
4. Hora en la que se escribió.
5. Texto del comentario.
6. Identificador único dado por el chat de Facebook.

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase PostWallFacebook.

Tabla 6.26: Métodos de la clase PostWallFacebook.java

7. Identificador del chat de Facebook.

La tabla 6.27 hace referencia a los atributos y la tabla 6.28 a sus métodos.

Atributo	Tipo	Descripción
xid	String	Identificador del usuario en el chat de Facebook.
objectId	String	Tipo del comentario.
postId	String	Identificador interno del comentario.
fromId	String	Identificador del usuario que escribió el comentario.
time	String	Hora en la que se registró el comentario.
text	String	Mensaje de texto del comentario.
id	String	Identificador creado por el chat de Facebook.
username	String	Nombre del usuario propietario.
replyXid	String	Identificador del usuario que respondió en el chat de Facebook.

Tabla 6.27: Atributos de la clase PostWallFacebook.java

Método	Entrada	Salida	Descripción
toJson()		JSONObject	Devuelve un objeto JSONObject() con los atributos de la clase StreamCommentsFacebook.

Tabla 6.28: Métodos de la clase StreamCommentsFacebook.java

6.2.7. Paquete es.uah.cc.ie.snsangelguardfb.facebookclient.clients

Este paquete contendrá las estructuras necesarias para la comunicación con Facebook. Anteriormente a la aparición de la arquitectura desarrollada por el paquete **com.restfb**, la comunicación con dicha Red Social se realizaba mediante la clase **FacebookClientLocal.java** contenida en éste paquete. Debido a las continuas actualizaciones de las APIs

de Facebook, se tomó la decisión de utilizar estas estructuras únicamente para realizar la conexión inicial con cada usuario y dejar el resto de operaciones bajo la responsabilidad del paquete **com.restfb**.

Este paquete, como hemos comentado anteriormente, contendrá únicamente la clase **FacebookClientLocal.java**, que será la que pasaremos a describir a continuación.

6.2.7.1. Clase FacebookClientLocal.java

Será la clase que realizará las conexiones a Facebook de cada usuario. Comprobará si un usuario tiene sesión en Facebook, mediante el método **login()** y, mediante la sesión de usuario almacenada en la base de datos, utilizará el método **loginOffline()** para conectarse a la cuenta de un usuario en los procesos de actualización diarios y acceder a la información, siempre que el usuario haya dado permiso a la aplicación para que así lo haga.

Esta clase tendrá arquitectura de servicio RESTFul, por lo que tendrá un atributo **WebResource** y una **URL** de base, en la que se especificará la dirección incial del API de Facebook. A parte de estas estructuras, contendrá dos Servlets estáticos que realizarán las siguientes funciones:

1. **FacebookLoginServlet**: Será el servlet de conexión a Facebook. Recibirá como parámetros los objetos **HttpServletRequest** y **HttpServletResponse** y se conectará directamente a Facebook mediante una URL con los siguientes parámetros:
 - a) Una URL de base: En éste caso será la siguiente url:
https://www.facebook.com/dialog/oauth/
 - b) El identificador interno de la aplicación, proporcionado por Facebook al dar de alta la aplicación en el espacio de desarrolladores.
 - c) La URL de vuelta a la aplicación, en éste caso:
https://apps.facebook.com/snsangelguard/FacebookCallbackServlet/
 - d) Los permisos necesarios que deben ser confirmados por el usuario para que la aplicación acceda a su información esencial que necesite.
 - e) El tipo de respuesta, que en éste caso se indica que el formato que queremos recibir es el token de sesión del usuario.
2. **FacebookCallbackServlet**: Será el encargado de determinar si un usuario se ha conectado satisfactoriamente a Facebook y lo redireccionará al inicio de la aplicación. Para ello recibirá los siguientes parámetros:
 - a) El identificador del usuario que se ha conectado.
 - b) El token de sesión del usuario.

Si éstos parámetros se han recibido correctamente, la aplicación iniciará su ejecución. Si la conexión no se ha realizado correctamente, no se podrá acceder a ninguna funcionalidad de la aplicación.

Una parte esencial en la aplicación es la aceptación de permisos por parte del usuario para que SNSAngelGuardFB sea capaz de acceder a toda la información necesaria de éste en el perfil de usuario, tales como su información personal, sus amistades, sus comentarios en el muro, ..., etc. Estos permisos serán los siguientes:

1. Permiso **read_stream**: Con éste permiso se accederá a todos sus comentarios en el muro.
2. Permiso **offline_access**: Con éste permiso estamos proporcionando acceso a la información del usuario cuando éste no esté físicamente conectado a Facebook. Es esencial para los procesos de actualización diaria de información.
3. Permiso **friends_birthday**: Permite acceder a la fecha de nacimiento de cada uno de nuestros contactos. Es esencial para la ejecución del filtro de control de amigos.
4. Permiso **user_about_me**: Permite acceder a toda la información personal del perfil de usuario.
5. Permiso **user_relationship**: Permite acceder a la información del perfil del usuario concerniente a las relaciones personales con otros contactos.
6. Permiso **user_religion_politics**: Permite acceder a las tendencias políticas y religiosas del usuario.
7. Permiso **user_birthday**: Permite acceder a la fecha de nacimiento del usuario.
8. Permiso **user_work_history**: Permite acceder al historial laboral del usuario.
9. Permiso **user_education_history**: Permite acceder al historial de formación personal del usuario.
10. Permiso **user_website**: Permite acceder a los sitios web que el usuario haya definido de interés para él.
11. Permiso **user_hometown**: Permite acceder al lugar de residencia del usuario.
12. Permiso **user_location**: Permite acceder a la información concerniente a su localización.
13. Permiso **user_relationship_details**: Permite acceder a la información adicional de las relaciones personales del usuario.
14. Permiso **email**: Permite acceder a la dirección de email del usuario para que, cuando alguno de sus amigos acceda a la aplicación, pueda seleccionarle como ángel de Facebook y así poder enviarle notificaciones a su correo electrónico.

La tabla 6.29 contendrá la información de los atributos de clase y la tabla 6.30 mostrará la definición de los métodos utilizados por la aplicación para acceder a Facebook.

Atributo	Tipo	Descripción
BASE_URI	String	Constante con la dirección del API de Facebook. En éste caso se encontrará en la dirección http://api.facebook.com .
webResource	WebResource	Objeto que encapsula la funcionalidad del Servicio Web.
client	Client	Objeto perteneciente al Servicio Web para indicar que nos encontramos en la parte Cliente de éste.

Tabla 6.29: Atributos de la clase FacebookClientLocal.java

6.2.8. Paquete es.uah.cc.ie.snsangelguardfb.googleclient

Una de las funcionalidades de la aplicación será dar soporte al usuario para que pueda importar contactos de su cuenta de **Google** y pueda asignarlos como **Ángeles** de la aplicación.

Para realizar ésta funcionalidad será necesario construir estructuras que puedan acceder a la API de Google para acceder a su funcionalidad. En nuestro caso, se utilizará dos versiones de la API **Google Contacts** de la siguiente forma:

1. **Versión 2.x:** Se utilizará ésta versión para proporcionar el logueo en Google al usuario, de forma independiente a nuestra aplicación y proporcionandole a Google toda la responsabilidad sobre sus datos. Esta versión se utilizará en la parte cliente de la aplicación.
2. **Versión 3.x:** Se utilizará ésta versión para tratar los datos de retorno del logueo y poder obtener los contactos de la cuenta del usuario. Ésta versión es la que se utilizará en la parte servidor de la aplicación porque dicha versión proporciona herramientas para ser tratadas por lenguajes de alto nivel como Java.

Este paquete contendrá todas las estructuras necesarias para trabajar con la versión **3.x** de Google, que pasamos a describir a continuación.

6.2.8.1. Servlet GoogleContactsServlet

Como hemos especificado anteriormente, la clase **GoogleContactsServlet** será la estructura que trate con la API de Google **v 3.x** utilizando un **Servlet** de conexión que recibirá los datos de la petición que previamente se haya realizado desde cliente.

El **objetivo fundamental** de ésta clase será recibir una cadena de texto de la petición HTTP y convertirla en un objeto **JSON** con los datos de cada contacto del usuario.

La tabla 6.31 hace referencia a los atributos y la tabla 6.32 a sus métodos.

Método	Entrada	Salida	Descripción
FacebookClientLocal()			Constructor de clase. Conectará con el API de Facebook mediante la inicialización de los atributos webResource y client .
login()	HttpServletRequest, HttpServletResponse, String apiKey, String applicationSecret		Realiza el login del usuario en Facebook.
loginOffline()	HttpServletRequest, HttpServletResponse, String apiKey, String applicationSecret		Realiza el login offline del usuario en Facebook.
dispatch()	HttpServletRequest, HttpServletResponse		Distribuye hacia los servlet de conexión el flujo de la aplicación, dependiendo de si el usuario se encuentra o no conectado.

Tabla 6.30: Métodos de la clase FacebookClientLocal.java

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
URL_GET_CONTACTS	final String	URL de petición de datos a Google.
snsObject	SNSAngelGuardFBManager	Manager principal de la aplicación.

Tabla 6.31: Atributos de la clase GoogleContactsServlet.java

6.3. Paquete es.uah.cc.ie.snsangeguardfb.sources

Este paquete contendrá toda la funcionalidad propia de la aplicación, es decir, las anteriores estructuras han descrito toda la funcionalidad externa de la aplicación (llamadas a Facebook y sus tipos de datos).

A partir de éste momento, se pasará a describir todas las estructuras que se han diseñado para llevar a cabo todo el negocio de la aplicación y para que, en última instancia, se ejecute en Facebook. Las estructuras de las que hablamos son las siguientes:

1. Paquete **dao**: Contendrá todas las estructuras necesarias para el acceso a base de datos de datos de configuración del usuario.
2. Paquete **email**: Contendrá las clases necesarias para que la aplicación pueda enviar emails como medio de notificación.
3. Paquete **filterfuncionality**: Contendrá todas las estructuras necesarias para ejecutar los filtros de control de la aplicación.

Método	Entrada	Salida	Descripción
processRequest()	HttpServletRequest request, HttpServletResponse response		Procesador de la petición de entrada.
sortJSONArray()	JSONArray contacts	JSONArray	Ordena un JSONArray por nombre.
getJSONArrayAllContactsUser()	ContactsService myService, String hdAngelsGoogleSelected	JSONArray	Obtiene un JSONArray con todos los datos de todos los contactos del usuario autenticado en la aplicacion.
getJSONContact()	String name, String email, String linkPhoto	JSONObject	Devuelve un objeto JSON agrupando los datos de un contacto de Google. Si se produce alguna excepcion en el proceso, el metodo devolvera el valor null.
isSelectedAngel()	String email, String hdAngelsGoogleSelected	boolean	Comprueba si el email de un contacto ya esta seleccionado como angel en la aplicacion.

Tabla 6.32: Métodos de la clase GoogleContactsServlet.java

4. Paquete **jspcontroler**: Contendrá todas las clases de control de recursos y ejecución de las páginas jsp que mostrarán la parte cliente de la aplicación.
5. Paquete **logs**: Contendrá las estructuras de configuración de los logs de la aplicación.
6. Paquete **offline**: Contendrá todas las clases necesarias para ejecutar el proceso harvested de obtención y actualización de información diaria.
7. Paquete **snswebservicescliente**: Contendrá la clase definida como API pública para el acceso mediante servicios RESTFul a la base de datos SocialNetwork.
8. Paquete **utilities**: Contendrá las clases estáticas de acceso a funcionalidades comunes para tipos de operaciones determinadas.

El diagrama 6.4 describe estas estructuras.

6.3.1. Paquete es.uah.cc.ie.snsangeguardfb.sources.dao

Este paquete contendrá todas las estructuras necesarias para el almacenamiento y la gestión de los datos de configuración en la base de datos **SocialNetwork**. Estarán bajo la nomenclatura **DAO**, es decir, Data Base Object, y serán las únicas estructuras de la aplicación que accederán a los servicios RESTFul que acceden a funcionalidades de la base de datos en su módulo de configuración. El diagrama 6.5 mostrará todas las estructuras contenidas en éste paquete y que serán expuestas a continuación.

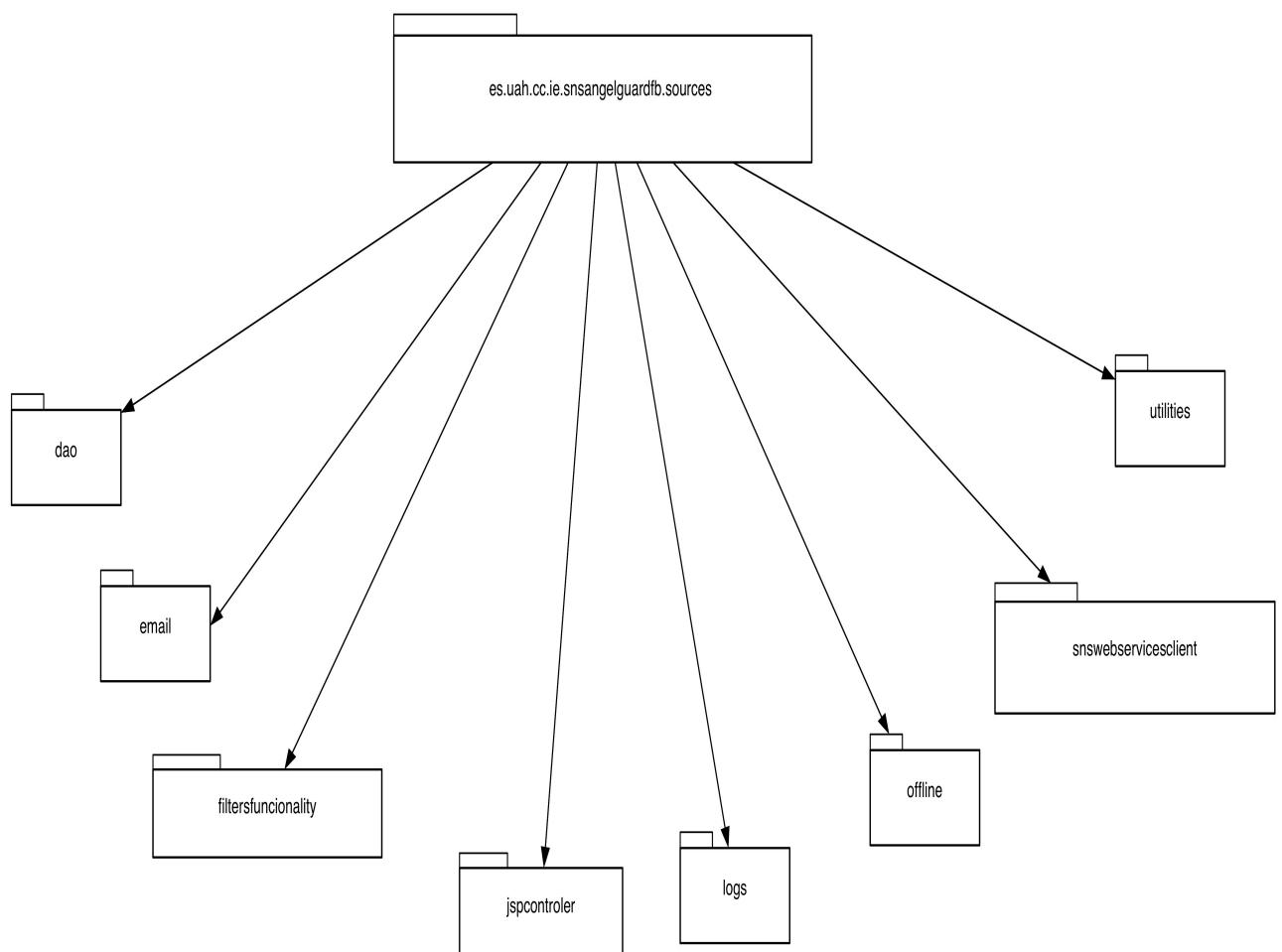
6.3.1.1. Clase LocaleSettingsDaoManager.java

Clase Manager que controla todos los recursos de idioma que se utilizan en la aplicación, realizando para ello todas las operaciones de base de datos necesarias. Almacenará un atributo de clase **SNSAngelGuardFBManager**, asegurando la integridad de la sesión y el acceso a todas las estructuras de la aplicación.

Esta clase tendrá las siguientes responsabilidades:

1. Cargar los recursos de idioma de la aplicación cliente de Facebook dependiendo del idioma configurado por cada usuario de ésta Red Social. El idioma por defecto será el inglés, siendo únicamente configurada en castellano cuando un usuario tiene éste idioma en su perfil de Facebook.
2. Cargar los recursos de idioma de la aplicación en los procesos offline y en las notificaciones: En éste caso, el idioma se cargará en función del campo **Locale** de la tabla **user_facebook**.

La tabla 6.33 mostrará los atributos de ésta clase y la tabla 6.34 mostrará sus métodos.

Figura 6.4: Paquete `es.uah.cc.ie.snsangelguardfb.sources`

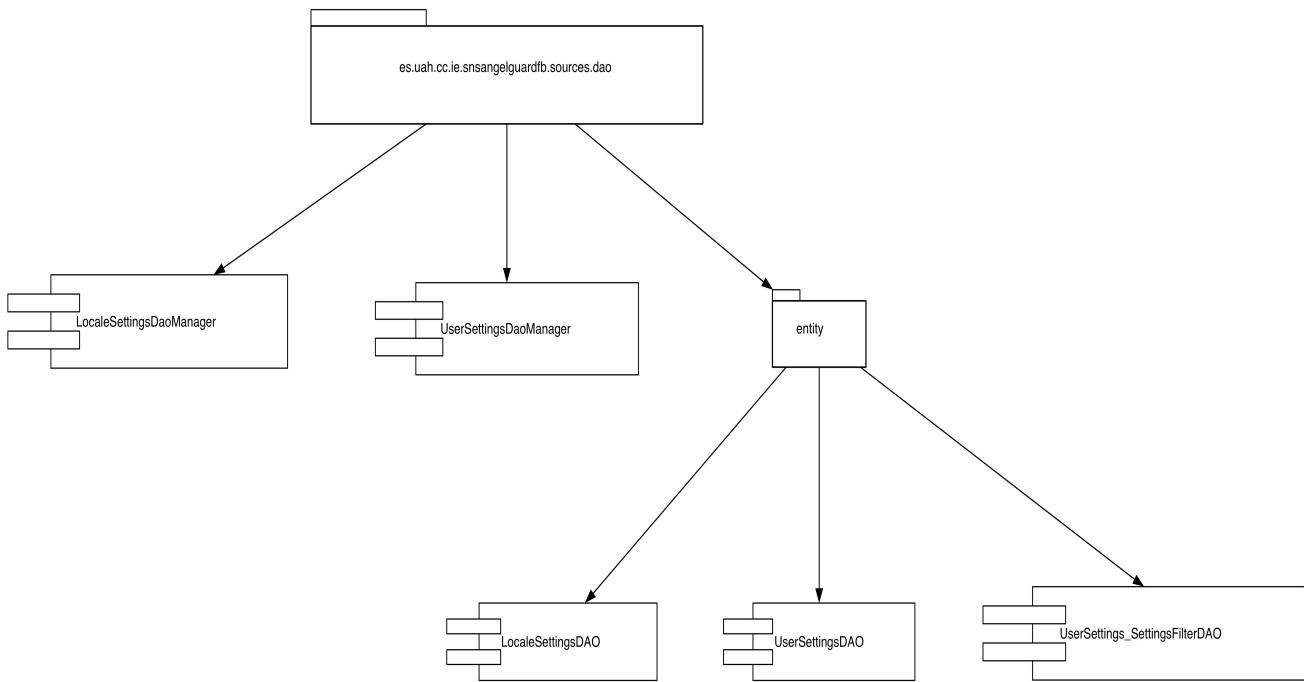


Figura 6.5: Paquete es.uah.cc.ie.snsangelguardfb.sources.dao

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.
localeSettingsDao	LocaleSettingsDAO	Clase entidad de los recursos de idioma.

Tabla 6.33: Atributos de la clase LocaleSettingsDaoManager.java

Tabla 6.34: Métodos de la clase LocaleSettingsDaoManager.java

Método	Entrada	Salida	Descripción
LocaleSettingsDaoManager	SNSAngelGuardFBManager		Constructor de clase.Inicializará el objeto manager de la aplicación.
getSnsObject		SNSAngelGuardFBManager	Obtiene el objeto manager de la aplicación.
setSnsObject	SNSAngelGuardFBManager		Establece el objeto manager de la aplicación.
getLocaleSettingsDao		LocaleSettingsDAO	Obtiene la clase donde se almacenan los recursos de idioma.

Método	Entrada	Salida	Descripción
setLocaleSettingsDao	LocaleSettingsDAO		Establece la clase donde se almacenan los recursos de idioma.
getJsonLocaleSettingsOffLine		JSONObject	Obtiene los recursos de idioma para un usuario en modo offline. Llamará al servicio de base de datos correspondiente para recuperar los recursos de la tabla locale_settings. Puede lanzar excepciones del tipo JSONException.
loadLocaleSettingsOffLine			Carga los recursos de idioma en modo offline en el atributo localeSettings-Dao.
getJsonLocaleSettings		JSONObject	Obtiene los recursos de idioma en modo online para un usuario. Llamará al servicio de base de datos correspondiente para recuperar los recursos de la tabla locale_settings. Puede lanzar excepciones del tipo JSONException, UniformInterfaceException o IOException.
loadLocaleSettings			Carga los recursos de idioma en modo online en el atributo localeSettings-Dao.

6.3.1.2. Clase UserSettingsDaoManager.java

Será la clase Manager que controlará toda la gestión de base de datos referente a las operaciones de un usuario en la aplicación. Realiza todas las posibles operaciones con información de la base de datos concernientes a un usuario de la aplicación. Será el encargado de determinar si es o no el usuario nuevo en la aplicación y, en caso afirmativo, almacenará toda la información necesaria para la aplicación. En caso negativo, actualizará la información existente por la posible nueva información del usuario en Facebook.

Los atributos que tendrá están representados en la tabla 6.35 y sus métodos mostrados en la tabla 6.36.

Tabla 6.36: Métodos de la clase UserSettingsDaoManager.java

Método	Entrada	Salida	Descripción
--------	---------	--------	-------------

Método	Entrada	Salida	Descripción
UserSettingsDaoManager()	SNSAngelGuardFBManager		Constructor de clase. Inicializa los datos del Manager de la aplicacion y de los datos de usuario.
getSnsObject()		SNSAngelGuard-FBManager	Obtiene el Manager principal de la aplicacion.
setSnsObject()	SNSAngelGuardFBManager		Establece el Manager principal de la aplicacion en la instancia de la clase.
getUserSettingsDAO()		UserSettingsDAO	Obtiene la instancia de la clase que almacena los datos del usuario de la aplicacion.
setUserSettingsDAO()	UserSettingsDAO		Establece la instancia de la clase que almacena los datos del usuario de la aplicacion.
loadUser()	JSONObject		Carga los datos de un usuario en el atributo userSettingsDAO a partir de un objeto json. Podrá lanzar excepciones del tipo JSONException, ParseException o bsh.ParseException.
loadSettings()			Carga los datos de un usuario de la aplicacion, tanto los datos registrados como sus preferencias a la hora de la ejecucion. Si no está registrado, llamará al metodo setNewUserConnected() para almacenar sus datos en la base de datos. Podrá lanzar excepciones del tipo ParseException, JSONException o bsh.ParseException.

Método	Entrada	Salida	Descripción
loadUserConnected()	JSONObject		Carga el usuario conectado con su información almacenada en el objeto JSONObject en el atributo userSettingsDAO. Llamará al método loadUser e inicializará el atributo del objeto de la clase SNSAngelGuardFBManager newConnection al valor false. Mostrará en el log toda la información del usuario conectado. Podrá lanzar excepciones del tipo JSONException, ParseException, bsh.ParseException.
setNewUserConnected()			Obtiene la información del nuevo usuario de la aplicación conectado a Facebook y la carga en el objeto userSettingsDAO. Podrá lanzar excepciones del tipo ParseException.
putNewInstanceUS()	HttpServletRequest, HttpServletResponse		Almacena una nueva instancia del usuario en la base de datos accediendo al método getInstance del atributo userSettingsDAO. Podrá lanzar excepciones del tipo JSONException, ParseException, UniformInterfaceException, IOException, DataLengthException, IllegalStateException, InvalidCipherTextException, UnsupportedEncodingException, NoSuchProviderException o MessagingException.

Método	Entrada	Salida	Descripción
getUserInfoSimple()			Obtiene los datos de un usuario de la aplicacion de la base de datos mediante el metodo userSettings_getUserSettings ByUID del cliente de la base de datos y los carga en el atributo userSettingsDAO mediante el método loadUser. Podrá lanzar excepciones del tipo JSONException, ParseException o bsh.ParseException.
checkAngelConfirmation()	JSONObject		Recibe un objeto JSONObject con la información del usuario y actualiza la fecha y la hora de la ultima vez que el usuario se conecto a la aplicacion en el campo lastCheck de la tabla user_settings cuando se realiza una confirmación de un ángel. Llamará al método updateLastCheckUS del atributo userSettingsDAO. Puede lanzar excepciones del tipo ParseException, JSONException, MySQLIntegrityConstraintViolationException, NoSuchProviderException, FileNotFoundException, MessagingException o bsh.ParseException.
updateFacebookUserInfo()	JSONObject, String	JSONObject	Actualiza la informacion de Facebook de un usuario ya existente en la aplicacion en la base de datos. Recibirá en un objeto JSONObject la nueva información del usuario y su identificador dentro de la base de datos. Devolverá un objeto JSONObject con la información ya persistida en la base de datos.

Método	Entrada	Salida	Descripción
getUserInfo()	boolean		Almacena en base de datos informacion conjunta del perfil de Facebook y el perfil de Open Social del usuario de la aplicacion, junto con su informacion comun, que sera almacenada en la tabla User. Recibe por parametro un objeto boolean que indica si es la primera vez que accede a la aplicacion. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException, JSONException o MySQLIntegrityConstraintViolationException.
setUserFacebook_UserSettings()	JSONObject, String		Actualiza la informacion de Facebook de un usuario de la aplicacion. Recibirá un objeto JSONObject con la informacion obtenida de Facebook para ser almacenada en la tabla user_facebook y su identificador interno de base de datos.
setUser_UserSettings	JSONObject, String		Actualiza la informacion personal del usuario de la aplicacion, es decir, la informacion comun a Facebook y OpenSocial y la informacion especifica de los anteriores. Recibirá un objeto JSONObject con la informacion que va a ser almacenada en la tabla user y su identificador interno de base de datos.
getUserFacebook()		JSONObject	Obtiene toda la informacion de Facebook asociada al perfil del usuario. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException o JSONException.

Método	Entrada	Salida	Descripción
getEmptyUserOpenSocial()		JSONObject	Genera un objeto JSON vacío con los campos de información necesarios para generar un objeto OpenSocial nulo. Podrá lanzar una excepción del tipo JSONException.
getUser()	JSONObject userAux, JSONObject userFacebook, JSONObject userOpenSocial	JSONObject	Obtiene en un objeto JSON todos los datos comunes a los perfiles de Facebook y OpenSocial. Lo genera a partir de un objeto JSONObject con la información de la tabla user, un JSONObject con la información de la tabla user_facebook y otro JSONObject con la información de la tabla user_openSocial. Podrá lanzar una excepción del tipo JSONException.
getFacebookFriendsDB()	String	JSONArray	Obtiene todos los amigos de Facebook almacenados en la base de datos para un determinado usuario de la aplicación a partir del identificador de un usuario indicado en el parámetro de entrada de tipo String.
updateInformationFilters()			Actualiza en base de datos la información necesaria para la ejecución de cada filtro.
getInformationFilters()			Obtiene la información necesaria para cada filtro y la almacena en base de datos.

6.3.2. Paquete es.uah.cc.ie.snsangeguardfb.sources.dao.entity

Contendrá las entidades contenedoras de datos con las que las clases managers realizarán operaciones. Contendrá las siguientes clases:

1. Clase **LocaleSettingsDAO.java**: Contendrá todos los datos y recursos de idioma del usuario.
2. Clase **UserSettingsDAO.java**: Gestiona toda la información de la tabla userSettings para la configuración de la aplicación para un determinado usuario.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.
userSettingsDAO	UserSettingsDAO	Objeto DAO que almacena los datos de un usuario de la aplicación.

Tabla 6.35: Atributos de la clase UserSettingsDaoManager.java

3. Clase **UserSettings _ SettingsFilterDAO.java**: Clase que realiza todas las operaciones DAO necesarias para un filtro.

6.3.2.1. Clase LocaleSettingsDAO.java

Esta clase contendrá todos los recursos de idioma cargados directamente de la tabla **locale _ settings**. Como tal, no ejecutará ninguna operación de base de datos, ya que éstas se realizarán en la clase manager **LocaleSettingsDaoManager**.

Tendrá un atributo por cada campo de dicha tabla, un constructor que recibirá todos los parámetros de la clase y métodos de acceso **get** y establecimiento **set** de datos. Nótese que algunos de estos campos pueden contener más de una cadena de texto, es decir, algunos campos de la base de datos, por tener cadenas de texto de temas relacionados, estarán formados por arrays de Strings separados por el carácter ';', de tal manera que a la hora de cargar estos campos bastará con realizar un split y se obtendrán todas las cadenas que forman parte del campo.

Los atributos que se representarán en ésta clase se muestran en la tabla 6.37.

Tabla 6.37: Atributos de la clase LocaleSettingsDao.java

Campo	Tipo	Descripción
idLocale	String	Identificador interno de la tabla.
acceptingTerms	String	Contendrá la descripción de cabecera de la página de aceptación del acuerdo legal.
legalAccepted	String	Contendrá el disclaimer del acuerdo legal.
btnAgreeAT	String	Título del botón Aceptar en la página de aceptación del acuerdo legal.
btnCancelAT	String	Título del botón Cancelar en la página de aceptación del acuerdo legal.
titleAT	String	Título de la página de Acuerdo Legal.
titleSettings	String	Título de la página principal de configuración.
titleMenSettings	String	Titulo de las pestañas de la página de configuración(Array).
btnSaveCheckSettings	String	Título del botón Guardar de la página de configuración.

Campo	Tipo	Descripción
titleSettAng	String	Título de la página de selección de ángeles.
titleFriendsContentSettAng	String	Título del contenedor de la página de ángeles para los contactos de Facebook.
titleFriendsImportSettAng	String	Títulos de los contenedores de la página de ángeles para contactos de Google y de otros tipos(Array).
txtNameTutorSettAng	String	Título para el campo de texto del nombre del tutor de la página de ángeles.
txtEmailTutorSettAng	String	Título para el campo de texto del email del tutor de la página de ángeles.
btnImportSettAng	String	Título del botón de importar contactos de Google.
titleFbListSettAng	String	Títulos para las pestañas del contenedor de los contactos de Facebook(Array).
subTitleAngelSettAng	String	Título para cada contacto de Facebook.
titleSettVig	String	Título para la página de configuración de filtros.
titleVigilantSettVig	String	Título del contenedor de filtros de la página de configuración de filtros.
titleVigSettVig	JSONObject	Títulos de los filtros disponibles de la página de configuración de filtros.
titleVigDescriptionSettVig	JSONObject	Descripción de cada filtro de la página de configuración de filtros.
titleVigFrecSettVig	String	Título del contenedor de selección de la frecuencia de la página de configuración de filtros.
titleVigFrecSelectSettVig	String	Títulos de las frecuencias(Array).
titleVigAngSettVig	String	Título del contenedor de ángeles de la página de configuración de filtros..
titleAngConfirm	String	Título del mail de confirmación de un nuevo ángel.
desInfoAngConfirm	String	Descripción para el mail de confirmación de un nuevo ángel.
acceptConfAngConfirm	String	Título del enlace para confirmar al nuevo ángel.
cancelConfAngConfirm	String	Título del enlace para cancelar al nuevo ángel.
infoAngGuard	String	Descripción de pie de página de cada mail de notificaciones.
titleAngUser	String	Título de la página de identificación de un usuario de la aplicación.
nameUserAngUser	String	Título para el nombre del usuario en la página de identificación.
btnCloseAngUser	String	Título del botón Cerrar de la página de identificación de usuarios.
titleGoogleCont	String	Título de la página de importación de contactos de Google.

Campo	Tipo	Descripción
titleContGoogleCont	String	Descripción de cabecera de la página de importación de contactos de Google
btnLogGoogleCont	String	Título del botón de Login en Google.
titleNameContactGoogleCont	String	Título para el campo nombre del contacto de Google de la tabla de contactos de la página de importación de contactos de Google.
titleEmailContactGoogleCont	String	Título para el campo email del contacto de Google de la tabla de contactos de la página de importación de contactos de Google.
btnAcceptGoogleCont	String	Botón Aceptar de la página de importación de contactos de Google.
btnCancelGoogleCont	String	Botón Cancelar de la página de importación de contactos de Google.
helpMe	String	Descripciones de la página de Ayuda(Array).
warnings	String	Mensajes de aviso de la aplicación(Array).
titleInformationMessage	String	Título de la ventana de información de la aplicación.
informationMessage	String	Mensajes de información de la aplicación(Array).
mailDelete	String	Mensajes de información del mail de eliminación de un ángel(Array).
mailNotification	String	Mensajes de los mail de notificación(Array).
altContactsAngelsEd	String	Títulos del contenedor de importación de otros contactos de la página de selección de ángeles.
titleVisitsFilterOptions	String	Títulos para las notificaciones del filtro de visitas(Array).

6.3.2.2. Clase UserSettingsDAO.java

Esta será la clase entidad de la clase **UserSettingsDaoManager**. Gestiona toda la información de la tabla **user_settings** para la configuración de la aplicación para un determinado usuario.

Esta clase será de vital importancia ya que, en última instancia, será quien realice todas las llamadas al API de servicios RESTFul del cliente de base de datos. Será su propia responsabilidad ser la única estructura que acceda a estos servicios para almacenar o acceder a datos del módulo de configuración de la base de datos **SocialNetwork**. Como atributos contendrá cada uno de los campos de la tabla **user_settings** y sus métodos podrán relacionar todos sus atributos con todas las estructuras con las que se relaciona esta tabla.

Una de las particularidades de ésta clase es que contendrá las estructuras necesarias para realizar el cifrado del identificador único de cada usuario mediante una clave privada única que está indicada en la clase. Este identificador público se utilizará para poder reconocer al usuario referenciado en accesos públicos de los ángeles, es decir, será el identificador que cada página del cliente llevará como parámetro para poder acceder a los datos de un usuario en concreto.

Para poder realizar éste cifrado, se ha utilizado el algoritmo AES¹ de 64 bits. Además, se ha introducido una librería externa al proyecto que realiza ésta tarea. Estas clases se encuentran en el paquete **org.bouncycastle.crypto** y su dependencia maven contenida en el archivo **pom.xml** será la siguiente:

```
<dependency>
    <groupId>bouncycastle</groupId>
    <artifactId>bcpprov-jdk16</artifactId>
    <version>140</version>
</dependency>
```

Sus atributos y sus métodos se muestran en las tablas 6.38 y 6.39. A parte de los mostrados en ésta tabla, también contendrá métodos de acceso y establecimiento de todos los atributos privados de la clase.

Tabla 6.38: Atributos de la clase UserSettingsDAO.java

Atributo	Tipo	Descripción
FILTER_DEFAULT_ACTIVE_VALUE	final String	Constante. Valor por defecto para el activado del filtro. Por defecto contendrá el valor 0.
FILTER_DEFAULT_FREC_VALUE	final String	Constante. Valor por defecto para la frecuencia de un filtro. Por defecto contendrá el valor 3.
CF_64	final String	Constante. Clave única privada para la generación del cifrado de la clave pública de acceso.

¹Advanced Encryption Standard (AES), también conocido como Rijndael (pronunciado Rain Doll.^{en} inglés), es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. El AES fue anunciado por el Instituto Nacional de Estándares y Tecnología (NIST) como FIPS PUB 197 de los Estados Unidos (FIPS 197) el 26 de noviembre de 2001 después de un proceso de estandarización que duró 5 años. Se transformó en un estándar efectivo el 26 de mayo de 2002. Desde 2006, el AES es uno de los algoritmos más populares usados en criptografía simétrica. El cifrado fue desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen, ambos estudiantes de la Katholieke Universiteit Leuven, y enviado al proceso de selección AES bajo el nombre Rijndael".

Atributo	Tipo	Descripción
IMG_STANDAR_ANGEL	final String	Constante. Path a la imagen estandar para contactos no pertenecientes a Facebook.
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.
manager	UserSettingsDaoManager	Manager del objeto UserSettingsDAO.
uid	String	Identificador único en base de datos.
userName	String	Nombre del usuario.
userEmail	String	Email del usuario.
legalAccepted	Boolean	Indicador de aceptacion del acuerdo legal.
lastCheck	Date	Fecha que muestra el ultimo acceso del usuario a la aplicacion.
uidPublic	String	Identificador publico cifrado de acceso a la aplicacion.
appActivated	Boolean	Indicador de activacion de la aplicacion.
userSession	String	Token de acceso a Facebook.
localeSettings	String	Identificador del idioma en base de datos. Relacionado con la tabla locale_settings.
backupCheck	Date	Fecha del ultimo backup de informacion offline.
filterDaoMap	Map	Mapa que contiene los filtros configurados en la aplicación.
angelsSelected	String	Campo que almacena los angeles seleccionados en la aplicacion separados por el caracter ;. Todos los ángeles, al ser recepcionados en el módulo servidor, tendrán este formato que habrá que parsear.

Tabla 6.39: Métodos de la clase UserSettingsDAO.java

Método	Entrada	Salida	Descripción
getIdLocale()	String locale	String	Obtiene el identificador de idioma de la tabla locale_settings dependiendo del idioma del perfil de Facebook del usuario. Podrá lanzar excepciones del tipo JSONException.
getJSONLocale()	String codLocale	JSONObject	Obtiene en un objeto JSON todos los recursos de idioma para un perfil de Facebook.
static cifrar()	String textoEnClaro	String	Obtiene una cadena de texto cifrada en Base64 utilizando el algoritmo AES a partir de la cadena de entrada. Podrá lanzar excepciones del tipo org.bouncycastle.crypto.DataLengthException, java.lang.IllegalStateException o org.bouncycastle.crypto.InvalidCipherTextException.
static descifrar()	String criptogramaB64	String	Descifra una cadena de texto cifrada en Base64 utilizando el algoritmo AES. Podrá lanzar excepciones del tipo org.bouncycastle.crypto.DataLengthException, java.lang.IllegalStateException o org.bouncycastle.crypto.InvalidCipherTextException.
initNewFilters()			Inicializa y enlaza en base de datos, una instancia para cada filtro definido al usuario de la aplicación.

Método	Entrada	Salida	Descripción
getNewInstanceUS()	HttpServletRequest, HttpServletResponse	JSONObject	Crea un objeto JSONObject a partir de los atributos de la clase y lo persiste en base de datos mediante el servicio web userSettings_setNewEntityUserSettings, después lo retorna como resultado. Puede lanzar excepciones de los tipos JSONException, DataLengthException, IllegalStateException, InvalidCipherTextException, UnsupportedEncodingException, UniformInterfaceException, IOException, NoSuchProviderException, MessagingException, InterDataBaseException, InterProcessException o InterEmailException.
getArrayActiveFilter()	String activeFilter	String[]	Devuelve un array de un String.
setAngelsUserSettings()	String activeFilter, boolean newConnection		Persiste en base de datos los angeles definidos por un usuario. Puede lanzar excepciones del tipo throws JSONException, NoSuchProviderException, MessagingException, UniformInterfaceException o IOException.
setAngelsUserSettingsByFilter()	String desFilter, String activeFilter		Actualiza la informacion de un determinado filtro. Puede lanzar excepciones del tipo JSONException, UnsupportedEncodingException, UniformInterfaceException, IOException, NoSuchProviderException o MessagingException.
getJsonAngelWithUserSettingsRel	JSONObject jsonAngel	JSONObject	Incluye en un angel la relacion con el usuario de la aplicacion que le ha dado de alta.
setCollectionAngels()	JSONObject jsonAngel, int modo, String des		Actualiza los angeles para un determinado filtro. Puede lanzar excepciones del tipo JSONException.

Método	Entrada	Salida	Descripción
getNewAngels()			Obtiene los nuevos angeles para un determinado filtro. Puede lanzar excepciones del tipo JSONException.
deleteAngelsRelationship()			Elimina las relaciones existentes entre un angel eliminado de la configuracion del usuario y los diferentes filtros existentes. Puede lanzar excepciones del tipo JSONException.
deleteAngelFiltersRelationship()	JSONObject jsonAngel		Elimina las relaciones de un angel con respecto a los filtros de la aplicacion.
deleteAngelFromUserSettingsAngelJSONObject()	JSONObject jsonAngel	JSONObject	Borra del objeto angel la relacion que le unia a un usuario de la aplicacion.
setNotActiveAngels()			Inicializa como no activo los angeles seleccionados nuevos en base de datos. Puede lanzar excepciones del tipo JSONException.
isActiveAngel()	JSONObject jsonAngel	boolean	Comprueba si un angel esta activo para empezar a enviarle informes. Si esta activo retornará el valor true, y false en cualquier otro caso. Puede lanzar excepciones del tipo JSONException.
isSelectedArray()	String angelsSelected, String angel	boolean	Comprueba si un angel esta seleccionado por un usuario. Devolverá true en caso de encontrarse seleccionado, false en caso contrario.
getAngels()	String[][] arrayAngels	JSONArray	Obtiene un JSONArray con los datos de los angeles definidos en la aplicacion. Puede lanzar excepciones del tipo JSONException.
getNewAngelNotFacebook()	String nameAngel, String emailAngel, String typeAngel	JSONObject	Obtiene, en formato JSON, los datos de un nuevo angel no perteneciente a Facebook para ser insertado en base de datos.

Método	Entrada	Salida	Descripción
getNewAngelFacebook()	String[] datesAngel	JSONObject	Obtiene, en formato JSON, los datos de un nuevo angel perteneciente a Facebook para ser insertado en base de datos.
getAngelsUserFilter()	String[][] arrayAngels, String angelsFilter		Obtiene los angeles seleccionados para cada filtro definido.
putNewAngelsNewUser()	JSONArray angels		Almacena en base de datos los nuevos angeles para un nuevo usuario de la aplicacion.
putNewAngelFacebook()	JSONObject jsonNewAngel	JSONObject	Establece un nuevo angel de tipo Facebook.
putNewAngelsUser()	JSONObject jsonRespuesta, JSONArray angels		Inserta los nuevos angeles en la base de datos para un usuario ya existente.
putNewAngels()	boolean inicio		Inserta los nuevos angeles en la base de datos y los devuelve como retorno en un JSONArray. Puede lanzar excepciones del tipo JSONException, UnsupportedEncodingException, UniformInterfaceException, IOException, NoSuchProviderException o MessagingException.
findAngel()	JSONObject jsonAngel	JSONObject	Indica si un angel esta almacenado en base de datos, devolviendo el JSONObject con la informacion de la base de datos o un nuevo JSONObject vacio que indica que el usuario no ha sido encontrado. Puede lanzar excepciones del tipo JSONException.
setToDeleteOlderAngels()	JSONArray angels		Prepara la lista de angeles no seleccionados para ser borrados en base de datos. Puede lanzar excepciones del tipo JSONException.
deleteOlderAngels()			Elimina los angeles eliminados desde la aplicacion. Puede lanzar excepciones del tipo NoSuchProviderException, MessagingException, UniformInterfaceException o IOException.

Método	Entrada	Salida	Descripción
putAngelsFilter()	JSONArray angelsFilter, String des		Almacena los angeles seleccionado para un determinado filtro. Puede lanzar excepciones del tipo JSONException.
updateOfflineAngelsFilter()	JSONArray jsonArrayAngels, String des		Actualiza las relaciones entre un filtro y los angeles definidos.
getAngelInformationWithFilters()	JSONObject jsonAngel	JSONObject	Obtiene un angel con todas sus relaciones actualizadas.
updateFilters()		JSONObject	Actualiza en base de datos los filtros almacenados.
formatJsonLstAngel()	JSONArray jsonAngels, String desTypeAngel, String des	String	Formatea en un String los angeles seleccionados en la aplicacion.
existAnyAngel()	JSONObject jsonAngels, String typeTable	boolean	Indica si hay algun angel definido para los diferentes tipos de angeles de la aplicacion. Enviará el valor true en caso afirmativo y false en caso contrario.
getAngelsUser()	String desTypeAngel	String	Obtiene todos los angeles definidos por un usuario para sus diferentes filtros. Puede lanzar excepciones del tipo JSONException.
getJSONArray()	String strValue	JSONArray	Obtiene de un String de base de datos un JSONArray de objetos JSON. Puede lanzar excepciones del tipo JSONException.
initFiltersMap()			Inicializa en el objeto el mapa de filtros disponibles para el usuario.
getFiltersUserFromDB()		JSONArray	Obtiene de base de datos los filtros definidos para un usuario.
loadFullFuntionalityFilter()			Carga toda la funcionalidad de la aplicacion para los filtros definidos.
loadFilter()	String des, JSONArray arrayFilters		Carga desde base de datos todas las características de un determinado filtro.
updateFilter()	String desFilter	JSONObject	Actualiza un filtro con la nueva informacion y lo retorna al sistema. Puede lanzar excepciones del tipo JSONException.
initDBnewFilter()	String des		Inicializa la base de datos para un nuevo filtro no seleccionado.

Método	Entrada	Salida	Descripción
putNewInstanceFilter()	String des, String activeFilter		Almacena en base de datos un nuevo filtro definido. Puede lanzar excepciones del tipo JSONException.
formatJsonLstAngel()	JSONArray jsonAngels, String desTypeAngel, String des	String	Formatea en un String los angeles seleccionados en la aplicacion. Puede lanzar excepciones del tipo JSONException.
existAnyAngel()	JSONObject jsonAngels, String typeTable	boolean	Indica si hay algun angel definido para los diferentes tipos de angeles de la aplicacion. Enviará el valor true en caso afirmativo y false en caso contrario.
getAngelsUser()	String desTypeAngel	String	Obtiene todos los angeles definidos por un usuario para sus diferentes filtros. Puede lanzar excepciones del tipo JSONException.
getJSONArray()	String strValue	JSONArray	Obtiene de un String de base de datos un JSONArray de objetos JSON. Puede lanzar excepciones del tipo JSONException.
getAngelsFilter()	String des	String	Obtiene los angeles para un filtro determinado.
private loadFullFuntionalityFilter()			Carga toda la funcionalidad de la aplicacion para los filtros definidos.
loadFilter()	String des		Carga desde base de datos todas las caracteristicas de un determinado filtro.
updateFilter()	String desFilter	JSONObject	Actualiza un filtro con la nueva informacion y lo retorna al sistema. Puede lanzar excepciones del tipo JSONException.
updateLastCheckUS()			Actualiza la fecha en la que un usuario se conecta a la aplicacion y realiza un guardado de informacion. Puede lanzar excepciones del tipo UniformInterfaceException, IOException o JSONException.
updateBackUpCheckUS()			Actualiza la fecha en la que se realiza el ultimo backup de informacion del usuario.

Método	Entrada	Salida	Descripción
getEntitiesUserSettings()	SNSAngelGuardFBManager snsObjectManager	JSONArray	Obtiene toda la información de la base de datos de todos los usuarios de la aplicación y la retorna en un objeto JSONArray. Su utilidad básica radica en los procesos offline de información. Puede lanzar excepciones del tipo JSONException.
getJSONUserByUidPublic()	SNSAngelGuardFBManager snsObjectOff, String uidPublic	JSONObject	Obtiene una entidad de la tabla UserSettings a partir de su uidPublica y lo retorna como un JSONObject. Puede lanzar excepciones del tipo InterDataBaseException, InterProcessException o InterEmailException.
isActiveFilterForAngel()	String desFilter, String currentAngel	boolean	Encuentra de entre los ángeles definidos para un filtro, si está definido para este el angel actual.
getUserFriends()			Obtiene todos los amigos de un usuario en Facebook y los almacena en la base de datos SocialNetwork.
getNewFriend()	String friendUid		Obtiene de Facebook todos los datos necesarios para almacenar un nuevo amigo en la base de datos.
isNewFriend()	JSONObject friend	boolean	Comprueba si un amigo de un usuario ya está almacenado en la colección de amigos de la base de datos SocialNetwork.
isNewInFriendsFacebook()	JSONObject friend	boolean	Comprueba si un amigo del usuario es nuevo dentro de la base de datos.
updateRelationshipNewFriend()	JSONObject jsonFriendFacebook		Relaciona un nuevo amigo con el conjunto de amigos del usuario en la base de datos SocialNetwork.

Método	Entrada	Salida	Descripción
getRelationshipFriendsFacebook()	JSONObject jsonFriend	JSONArray	Obtiene un array con todas las relaciones de un usuario de Facebook con los usuarios de la aplicación. Devolverá un array con todas las URIs de los usuarios de la aplicación ya preparadas para ser insertadas en la base de datos.
setCollectionFriendsFacebook()	JSONObject jsonFriend		Incluye a un amigo en la relación con un usuario de Facebook.
updateCollectionFriendsFacebook()	JSONObject jsonFriend, JSONArray jsonArrayRelationship		Establece las relaciones con otros usuarios de la aplicación de un contacto de Facebook.
updateFriend()	JSONObject jsonFriend		Actualiza los datos de un amigo del usuario en base de datos.
updateDatesFriend()	JSONObject jsonFriend		Actualiza los datos de un amigo del usuario en la base de datos SocialNetwork.
convertURIServerToURIUserFacebook()	JSONArray jsonArrayUserFacebook, String strUidFriend	JSONArray	Convierte un array de objetos JSON con URIs obtenidas desde servidor en un array de objetos JSON con URIs preparadas para volver a ser relacionadas.
getURIConverted()	String strURI, String strUidFriend	String	Convierte una URI obtenida para un friendFacebook desde servidor al formato correcto para volver a ser introducida en la relación.

6.3.2.3. Clase UserSettings_SettingsFilterDAO.java

Esta clase será una entidad del manager **UserSettingsDaoManager**. Realizará todas las operaciones DAO necesarias para un filtro. Como atributos tendrá todos aquellos campos que se almacenarán en la tabla **settings_filter**.

Esta clase estará relacionada en la clase anterior mediante un mapa que relacione al usuario con todos sus filtros, siendo éstos de la clase **UserSettings_SettingsFilterDAO**.

Las tablas 6.40 y 6.41 mostrarán los atributos y los métodos definidos para ésta estructura.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
manager	UserSettingsDaoManager	Manager del objeto UserSettings-DAO.
typeFilter	String	Tipo del filtro en cuestión.
uid	Long	Identificador en base de datos del filtro.
frec	String	Frecuencia de ejecucion del filtro.
angels	String	Angeles separados por el caracter ;.
active	String	Indicador de actividad del filtro.
uidProp	Long	Identificador propietario del filtro.
lastCheck	Date	Ultimo chequeo realizado por el filtro.

Tabla 6.40: Atributos de la clase UserSettings_SettingsFilterDAO.java

Tabla 6.41: Métodos de la clase UserSettings_SettingsFilterDAO.java

Método	Entrada	Salida	Descripción
formatTime()	String strTime	Date	Formatea un String a un Date con formato 'yyyy-MM-dd HH:mm:ss'. Podrá lanzar excepciones del tipo ParseException o java.text.ParseException.
getObjectFilter()	String desFiltro	JSONObject	Obtiene el filtro de la base de datos indicado por el parámetro de entrada desFiltro. Podrá lanzar excepciones del tipo JSONException.
getAngelsFilter()	String[][] arrayAngels	JSONArray	Obtiene un JSONArray con todos los datos de los angeles de la aplicacion. Podrá lanzar excepciones del tipo JSONException.
loadSettingsFilter()	JSONObject jsonFilter, JSONArray jsonArrayAngels, String des		Carga en la aplicacion un filtro seleccionado. Podrá lanzar excepciones del tipo JSONException, ParseException o java.text.ParseException.
loadAngelsFilter()	JSONArray jsonArrayAngels		Carga los angeles definidos en la aplicacion para presentarlos por pantalla. Podrá lanzar excepciones del tipo JSONException.

Método	Entrada	Salida	Descripción
saveNewFilter()			Inicializa la base de datos para un nuevo filtro no seleccionado.
getFilterWithRelationship WithUserSettings()	JSONObject instanceFilter, String uidUserSettings	JSONObject	Devuelve un objeto JSON del filtro preparado para ser enlazado por primera vez con un usuario de la aplicación.
resetAngelsToFilter()	JSONObject instanceFilter	JSONObject	Elimina las relaciones entre un filtro y sus ángeles.
getJsonAngelWithFilterRelationship()	JSONObject jsonAngel	JSONObject	Enlaza un ángel al filtro.
updateInformationAngelForFilter()	JSONObject jsonUpdateAngel, String desFilter	JSONObject	Obtiene la información de un ángel actualizada con el filtro marcado por el parámetro desFilter.
deleteAngelFromFilterAngelCollection()	JSONObject jsonAngel	JSONObject	Borra del objeto angel la relación que le unía a un filtro.
putAngelInCollectionFilter()	JSONObject jsonAngel		Actualiza los ángeles de un usuario. Puede lanzar excepciones del tipo JSONException.

6.3.3. Paquete es.uah.cc.ie.snsangeguardfb.sources.email

Este paquete contendrá todas las estructuras necesarias para que la aplicación pueda, por medio de emails, enviar todas las notificaciones que se produzcan a cada ángel de la aplicación. Para ello, se utilizará el paquete **javax.mail**, para el cual habrá que definir una cuenta de correo electrónico que será la que envíe, en última instancia, el mail a los ángeles.

La forma de enviar notificaciones ha ido cambiando en éstos últimos meses debido a la política de Facebook. En el año 2011, desde Facebook podíamos enviar notificaciones a nuestros amigos por medio de su API, pero ésta funcionalidad se eliminó porque se producían muchísimos ataques SPAM a las cuentas de correo. Esta decisión perjudicó al diseño de la aplicación, porque a partir de ese momento se tuvo que cambiar el desarrollo para utilizar una cuenta de correo electrónico externa, es decir, se dió de alta una nueva cuenta de correo electrónico en Gmail y se adecuó el desarrollo de la aplicación para, mediante sus credenciales, enviar mails a través de ella.

Para llevar a cabo estas acciones, se han diseñado las estructuras que se describen a continuación.

6.3.3.1. Clase GenericEmailObject.java

Esta clase se ocupará de distribuir y enviar las notificaciones a los ángeles. Tendrá estructuras para generar el cuerpo del mensaje y utilizará la clase **EmailObject** para

enviar los mails.

Como notificaciones disponibles en la aplicación tenemos las siguientes:

1. Notificación de confirmación a un ángel: Esta notificación se enviará cuando un usuario de la aplicación elige un nuevo ángel, entonces éste recibirá un mail preguntándole si desea recibir las notificaciones de actividad del usuario.
2. Notificación de eliminación de un ángel: Esta notificación se enviará cuando la aplicación detecte que un ángel que antes estaba definido por el usuario ahora éste lo ha desactivado como tal. En ese momento, la aplicación manda un mail informativo sobre esta nueva situación.
3. Notificaciones de actividad: Este mail será el medio de comunicación sobre los resultados obtenidos al ejecutar los filtros. Enviará un mail cada vez que se ejecuten los filtros, ya sea online u offline.

La tabla 6.42 mostrará los atributos de ésta tabla y la tabla 6.43 sus atributos.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager principal de la aplicación.
facebookClient	FacebookClientLocal	Cliente de Facebook (Deprecado).
email	EmailObject	Objeto con las credenciales del servidor de correo electrónico.

Tabla 6.42: Atributos de la clase GenericEmailObject.java

Tabla 6.43: Métodos de la clase GenericEmailObject.java

Método	Entrada	Salida	Descripción
GenericEmailObject()	UserSettingsDaoManager manager, FacebookClientLocal		Constructor de clase.
sendMailConfirmationAngel()	JSONObject jsonAngel, String uidPublic		Envio de la notificación de confirmación a un nuevo ángel. Podrá lanzar excepciones del tipo JSONException, UnsupportedEncodingException, UniformInterfaceException, IOException, NoSuchProviderException o MessagingException.

Método	Entrada	Salida	Descripción
sendMailDeleteAngel()	JSONObject jsonAngel, String uidPublic		Envio de la notificacion de borrado de un angel. Podrá lanzar excepciones del tipo JSONException, NoSuchProviderException, MessagingException, UniformInterfaceException o IOException.
getBodyMailConfirmation()	String uidPublic, String idAngel	String	Obtiene el cuerpo del email de confirmacion para un usuario y su angel. Podrá lanzar excepciones del tipo UnsupportedEncodingException.
getBodyMailDeleteAngel()	String uidPublic	String	Obtiene el cuerpo del email de borrado de un angel.
sendEmailCheck()	String resultFltWall, String resultFltFriends, String resultFltPriv, String resultFltVist, JSONObject jsonAngel		Envia las notificaciones en funcion del resultado obtenido del chequeo de los filtros. Podrá lanzar excepciones del tipo JSONException, NoSuchProviderException, MessagingException, UniformInterfaceException o IOException.

6.3.3.2. Clase EmailObject.java

Esta clase se ocupará sólo y exclusivamente del envío de mail. Tendrá un único método, **sendMail** que enviará, en última instancia, el mail de notificación.

La tabla 6.44 contendrá los atributos y la tabla 6.45 contendrá los atributos de ésta clase.

6.3.4. Paquete es.uah.cc.ie.snsangelguardfb.sources.filtersfuncionality

Este paquete contendrá todas las estructuras necesarias para ejecutar los filtros de chequeo de la actividad de un determinado usuario dentro de Facebook. Toda la lógica de negocio de la aplicación se encontrará en éste paquete.

Cada filtro será una implementación distinta de la clase **ILifeCycleFilter**, la cual controlará el ciclo de vida para cada filtro, consiguiendo así un tratamiento homogéneo en toda la aplicación para cada filtro diferente.

Los filtros para los que está preparado el programa son los siguientes:

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
SMTP_HOST_NAME	String	Constante: Host del servidor de correo electronico.
SMTP_HOST_PORT	String	Constante: Puerto del servidor de correo electronico.
SMTP_AUTH_USER	String	Constante: Direccion de la cuenta de correo electronico.
SMTP_AUTH_PWD	String	Constante: Contraseña de la cuenta de correo electronico.

Tabla 6.44: Atributos de la clase EmailObject.java

Metodo	Entrada	Salida	Descripción
EmailObject()			Constructor sin argumentos.
sendEmail()	String subject, String body, String sendTo		Envia un email a la direccion de correo electronico del atributo sentTo. Podrá lanzar excepciones del tipo NoSuchProviderException, MessagingException o MalformedURLException.

Tabla 6.45: Métodos de la clase EmailObject.java

1. **Filtro de control de lenguaje:** Será el filtro que controle todos los comentarios del muro del usuario detectando posibles muestras de lenguaje malintencionado.
2. **Filtro de control de amistades:** Será el filtro que controle las edades de nuestros contactos en Facebook y determine si son o no demasiado mayores.
3. **Filtro de control de privacidad:** Será el filtro que controle la configuración de privacidad de nuestro perfil en Facebook determinando si es demasiado permisivo.
4. **Filtro de control de visitas:** Será el filtro que controle los amigos que tenemos en común, aquellos contactos con los que compartimos más amigos y aquellos con los que compartimos menos.

La imagen 6.6 muestra la estructura del paquete.

6.3.5. Interface IKeyArgsFilter

Cada filtro implementado necesita una serie de parámetros de entrada para su correcta ejecución. Estos parámetros serán implementados como un mapa clave-valor que el método de ejecución de un filtro sabrá interpretar correctamente en función de sus necesidades.

La interface **IKeyArgsFilter** es una interface de definición que cuenta con las claves necesarias que serán leidas por cada filtro. La interface está definida de la siguiente forma:

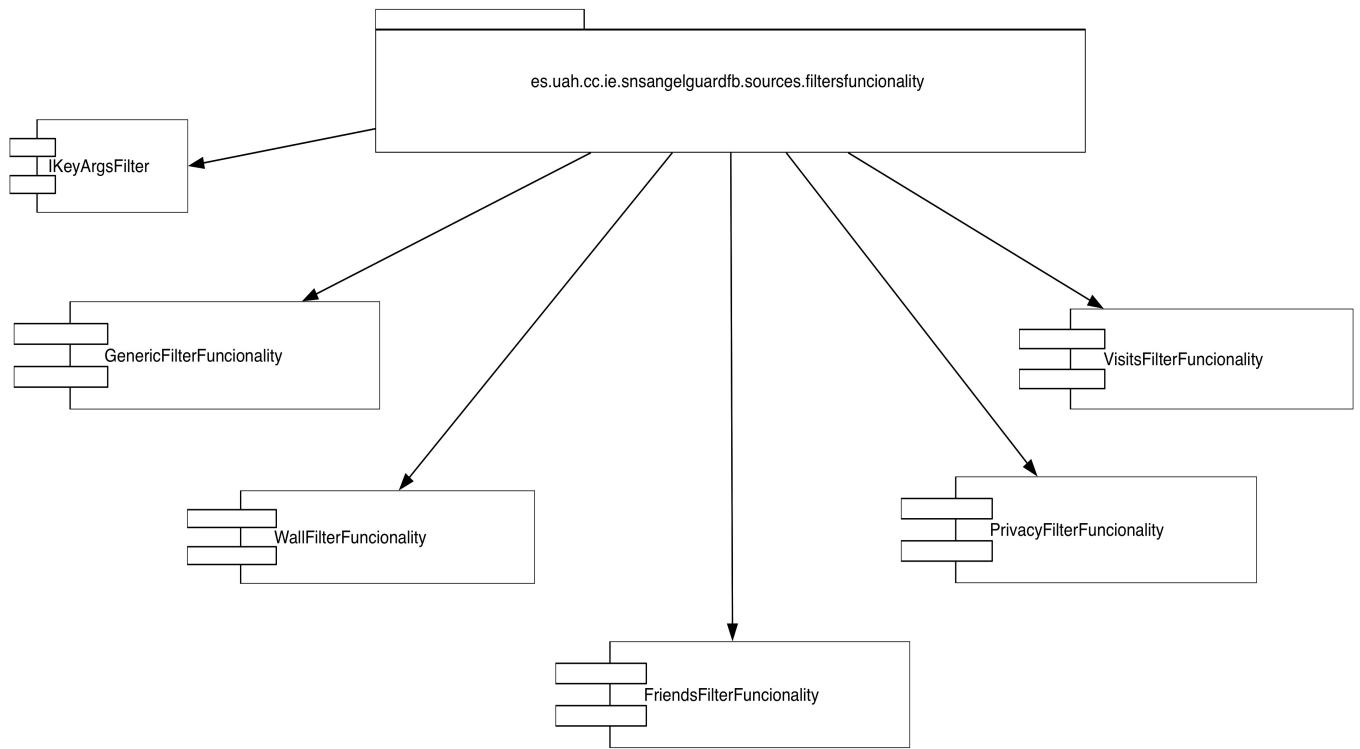


Figura 6.6: Paquete es.uah.cc.ie.snsangelguardfb.sources.filtersfuncionality

```

public interface IKeyArgsFilter {

    /** Key de la request de entrada */
    public static final String ARGS_KEY_REQUEST = "request";

    /** Key de los parametros del filtro en formato JSON */
    public static final String ARGS_KEY_JSONFILTER = "jsonFilter";

    /** Key de la definicion del filtro */
    public static final String ARGS_KEY_DESFILTER = "desFilter";

    /** Key de la fecha del primer chequeo */
    public static final String ARGS_KEY_FIRSTCHECK = "firstCheck";

    /** Key de la fecha del ultimo chequeo */
    public static final String ARGS_KEY_LASTCHECK = "lastCheck";
}
  
```

6.3.5.1. Clase GenericFilterFuncionality.java

Esta clase controlará la ejecución de todos los filtros definidos en la aplicación. A la hora de la ejecución, será su responsabilidad llamar a cada uno de los filtros para que se ejecuten. Implementará la interface **IkeyArgsFilter** para ser capaz de leer los parámetros de entrada necesarios para cada filtro.

Para controlar la ejecución de los filtros, al implementar éstos la interface **ILifeCycleFilter**, contendrá un mapa de filtros activos que tratará de forma genérica para realizar todas las operaciones necesarias con ellos.

Implementará todas las funcionalidades necesarias para realizar las comprobaciones de información, tanto en operaciones online como en offline. Las tablas 6.46 y 6.47 mostrarán sus atributos y sus métodos respectivamente.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
filterActiveMap	Map<String, ILifeCycleFilter>	Mapa que contiene los filtros activos.

Tabla 6.46: Atributos de la clase GenericFilterFuncionality.java

Tabla 6.47: Métodos de la clase GenericFilterFuncionality.java

Método	Entrada	Salida	Descripción
GenericFilterFuncionality()	SNSAngelGuardFBManager snsObject		Constructor de clase.
isActiveFilter()	String desFilter	boolean	Comprueba si el usuario tiene activo el filtro desFilter.
checkFilter()	HttpServletRequest request, String desFilter, boolean isTimeToCheck, boolean firstCheck, JSONObject jsonAngel, Date lastCheck	String	Pasa el filtro desFilter a cada uno de los angeles definidos para desFilter por el usuario si cumplen las siguientes condiciones: - El filtro esta activo - El filtro tiene definido algun angel para su actividad. - El tiempo transcurrido desde la ultima comprobacion es superior al tiempo definido por el usuario. Puede lanzar excepciones del tipo Exception.
getArgsFilter()	HttpServletRequest request, String desFilter, boolean firstCheck, JSONObject jsonFilter, Date lastCheck	Map<String, Object>	Obtiene un mapa con todos los parametros de la ejecucion de un filtro.

Método	Entrada	Salida	Descripción
getResultCheckFilter()	HttpServletRequest request, String desFilter, boolean firstCheck, JSONObject jsonFilter, Date lastCheck	String	Realiza el chequeo del filtro desFilter.
firstCheckAngelConfirmation()	HttpServletRequest request, JSONObject jsonAngel		Realiza el chequeo inicial de todos los filtros disponibles en la aplicacion, mandando un email al angel correspondiente con el resultado del chequeo. Puede lanzar excepciones del tipo InterDataBaseException, InterProcessException o InterEmailException.
isTimeToCheck()	String desFilter, Date lastCheck	boolean	Comprueba si el filtro debe pasarse en la ejecución presente.
checkUserSettingsOffLine()	HttpServletRequest request, JSONArray angelsUser		Metodo que realiza los chequeos correspondientes fuera de sesion. Es el metodo utilizado para hacer los chequeos en el momento del backup diario. Puede lanzar excepciones del tipo InterDataBaseException, InterProcessException o InterEmailException.
checkFilterToday()	String desFilter	boolean	Indica si el filtro desFilter se debe pasar hoy. Puede lanzar excepciones del tipo Exception.
getAngelsFilterUpdated()	String idNewAngel, String idOlderAngel, String olderAngels	String	Obtiene, en una cadena de texto separados por el caracter ;, todos los angeles actualizados.
updateAngelForFilter()	String idOlderAngel, String idNewAngel		Actualizamos el angel referenciado para los filtros.
loadMapFilters()			Carga en el map interno los objetos que representan los filtros.
cifrarUIDFriend()	String uidFriend	String	Realiza el cifrado del identificador de un amigo para ser mandado por email y así respetar su intimidad.

6.3.5.2. Clase WallFilterFuncionality.java

Implementará la interface **ILifeCycleFilter**. Esta clase controlará la ejecución del filtro de control del lenguaje. Controlará todos los comentarios que se hagan en el muro del usuario, detectando posible lenguaje malintencionado. Para ello, se valdrá de ficheros de recursos dentro del proyecto que contendrán palabras consideradas de acoso, que serán contrastadas con el vocabulario existente en el muro del usuario.

El control del filtro dependerá del lenguaje del perfil de Facebook del usuario. Si el perfil está configurado en inglés, se aplicará un fichero y si está en castellano se aplicará otro. Estos son los dos únicos lenguajes en los que, por ahora, está configurada la aplicación.

Las tablas 6.48 y 6.49 mostrarán sus atributos y sus métodos respectivamente.

Atributo	Tipo	Descripción
PATH_LEXICAL_FILE_EN	String	Constante: Ruta al fichero de control de lenguaje en inglés.
PATH_LEXICAL_FILE_ES	String	Constante: Ruta al fichero de control de lenguaje en castellano.
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.

Tabla 6.48: Atributos de la clase WallFilterFuncionality.java

Tabla 6.49: Métodos de la clase WallFilterFuncionality.java

Método	Entrada	Salida	Descripción
WallFilterFuncionality()	SNSAngelGuardFBManager snsObject		Constructor de clase. Recibe el objeto Manager de la aplicación.
getPostWall()			Obtiene todos los post del muro de Facebook del usuario y los almacena en base de datos. Puede lanzar excepciones del tipo UniformInterfaceException, MySQLIntegrityConstraintViolationException, IOException o JSONException.
getStreamComments()	String postId, String modo	String	Obtiene de la base de datos los comentarios a un post del muro del usuario. Tendrá dos modos de funcionamiento: - modo = newComment: Obtendrá todos los comentarios almacenados para un post en la base de datos. - modo = updateComment: Obtendrá todos los comentarios almacenados para un post hasta la fecha indicada por el campo de la tabla user_settings backupCheck. Puede lanzar excepciones del tipo JSONException.

Método	Entrada	Salida	Descripción
updatePostWall()			Actualiza la base de datos con los nuevos comentarios y post que se han realizado en Facebook tras el ultimo chequeo de informacion. Puede lanzar excepciones del tipo JSONException o ParseException.
isNewStreamPost()	JSONObject comentario	boolean	Comprueba si un post obtenido de Facebook esta registrado en la base de datos.
getPathFileBadWords()	String localeSettings	String	Obtiene el path del fichero badWords correspondiente a cada idioma configurado.
loadFileBadWords()		JSONArray	Carga el fichero badWords correspondiente al idioma configurado del usuario.
checkPostWall()	HttpServletRequest request, boolean firstCheck, JSONObject jsonFilter	String	Realiza la ejecucion del filtro de control del lenguaje a partir de los datos almacenados en la base de datos. Puede lanzar excepciones del tipo FileNotFoundException, IOException, JSONException, NoSuchProviderException, MessagingException, ParseException o bsh.ParseException.
checkCommentPost()	JSONObject streamComment, JSONArray badWordsArray, String informe, boolean firstCheck, JSONObject jsonFilter	String	Determina si un comentario contiene lenguaje ofensivo e introduce en el campo 'informe' las anomalias detectadas. Puede lanzar excepciones del tipo JSONException, ParseException o bsh.ParseException.
cargarFichero()	BufferedReader buffer	JSONArray	Carga el fichero que contiene las expresiones consideradas como lenguaje ofensivo. Puede lanzar excepciones del tipo JSONException o IOException.
isBadWords()	String comentario, JSONArray buffer	boolean	Analiza para una comentario si existe lenguaje ofensivo, devolviendo true en caso afirmativo. Puede lanzar excepciones del tipo IOException o JSONException.

6.3.5.3. Clase FriendsFilterFuncionality.java

Implementará la interface **ILifeCycleFilter**. Esta clase controla la ejecución del filtro de amistades. Recorrerá la lista de contactos de un usuario de Facebook y lanzará una notificación cuando alguno de ellos no cumpla alguna de las siguientes condiciones:

1. Su edad tiene que estar especificada en su perfil de usuario.
2. Su edad no puede ser mayor a una diferencia especificada por configuración(en este caso, no podrá tener una diferencia de edad mayor a cinco años).

La diferencia de edad estará controlada por la constante **MAX_AGE_LIMITED**, la cual puede tomar cualquier valor entero que se le quiera configurar.

La tabla 6.50 especificará los atributos de ésta clase y la tabla 6.51 sus métodos.

Atributo	Tipo	Descripción
MAX_AGE_LIMITED	int	Constante: Maxima diferencia de edad permitida para los contactos.
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.

Tabla 6.50: Atributos de la clase FriendsFilterFuncionality.java

Tabla 6.51: Métodos de la clase FriendsFilterFuncionality.java

Método	Entrada	Salida	Descripción
FriendsFilterFuncionality()	SNSAngelGuard-FBManager snsObject		Constructor de clase. Inicializa su atributo manager de la aplicacion.
getURIConverted()	String strURI, String strUidFriend	String	Convierte una URI obtenida para un friendFacebook desde servidor al formato correcto para volver a ser introducida en la relacion.
convertURIServerToURIUserFacebook()	JSONArray jsonArrayUserFacebook, String strUidFriend	JSONArray	Convierte un array de objetos JSON con URIs obtenidas desde servidor en un array de objetos JSON con URIs preparadas para volver a ser relacionadas. Podrá lanzar excepciones del tipo JSONException.
setCollectionFriendsFacebook()	JSONObject jsonFriend		Incluye a un amigo en la relacion con un usuario de Facebook.
updateRelationshipNewFriend()	JSONObject jsonFriendFacebook		Relaciona un nuevo amigo con el conjunto de amigos del usuario en la base de datos SocialNetwork. Podrá lanzar excepciones del tipo JSONException, InterDataBaseException, InterProcessException o InterEmailException.

Método	Entrada	Salida	Descripción
getNewFriend()	String friendUid		Obtiene de Facebook todos los datos necesarios para almacenar un nuevo amigo en la base de datos. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException, JSONException, InterDataBaseException, InterProcessException o InterEmailException.
getUserFriends()			Obtiene todos los amigos de un usuario en Facebook y los almacena en la base de datos SocialNetwork. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException, InterDataBaseException, InterProcessException o InterEmailException.
isNewFriend()	JSONObject friend	boolean	Comprueba si un amigo de un usuario ya esta almacenado en la colección de amigos de la base de datos SocialNetwork. Podrá lanzar excepciones del tipo JSONException.
isNewInFriendsFacebook()	JSONObject friend	boolean	Comprueba si un amigo del usuario es nuevo dentro de la base de datos. Podrá lanzar excepciones del tipo JSONException.
checkFriends()		String	Realiza el chequeo del filtro. Manda una lista, en formato HTML, con todos los amigos que tengan una edad mayor a cinco años de diferencia con el usuario o no hayan especificado su edad. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException, NoSuchProviderException, MessagingException o JSONException.
isAgeLimit()	int birthdayUser, String anio	boolean	Comprueba si la edad de un amigo de Facebook tiene una diferencia mayor de cinco años a la del usuario o no ha especificado su edad. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException o JSONException.

6.3.5.4. Clase PrivacyFilterFuncionality.java

Implementará la interface **ILifeCycleFilter**. Esta clase comprobará la privacidad de un determinado perfil de Facebook. Al no estar implementada en ésta parte del proyecto, no es responsabilidad de éste documento explicar su posible implementación. Cuando, en futuros desarrollos, se decida implementar éste filtro, bastará con implementar la funcionalidad en ésta clase, ya que la arquitectura de la aplicación está preparada para soportar un filtro de privacidad en un futuro.

6.3.5.5. Clase VisitsFilterFuncionality.java

Implementará la interface **ILifeCycleFilter**. Esta clase controlará la ejecución del filtro de visitas. Este filtro implementará las siguientes funcionalidades:

1. Mostrará un ranking con la lista de entradas al muro de usuario hecha por cada amigo. Mostrará los diez amigos que más post hayan escrito.
2. Mostrará un ranking de amigos con los que el usuario comparte más contactos. Al igual que en el caso anterior, mostrará hasta diez contactos en la lista.
3. Mostrará un ranking de amigos con los que el usuario comparte menos contactos. La lista estará compuesta por los diez amigos con los que menos comparte contactos.

La tabla 6.52 mostrará los atributos de la clase y la tabla 6.53 mostrará sus métodos.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.
titleOptions	String[]	Array que contiene todos los títulos utilizados por el filtro para incluir en los correos de informes.

Tabla 6.52: Atributos de la clase VisitsFilterFuncionality.java

Tabla 6.53: Métodos de la clase VisitsFilterFuncionality.java

Método	Entrada	Salida	Descripción
VisitsFilterFuncionality()	SNSAngelGuardFBManager snsObject		Constructor de clase.

Método	Entrada	Salida	Descripción
checkVisitFilter()	boolean firstCheck	String	Realiza el chequeo del filtro de visitas devolviendo el informe generado por los siguientes metodos privados: - getMutualFriends: Devolvera una lista con los primeros diez contactos con los que mas contactos compartimos en Facebook. - getNotMutualFriends: Devolvera una lista con los primeros diez contactos con los que menos contactos compartimos en Facebook. - getRankingPostFriends: Devolvera una lista de los amigos que mas post, incluyendo entradas en el muro y comentarios, han escrito en nuestro muro. Podrá lanzar excepciones del tipo JSONException.
getMutualFriends()		String	Devolvera una lista con los primeros diez contactos con los que mas contactos compartimos en Facebook. Cada entrada de la lista contendrá la foto del contacto actual en Facebook, su nombre y el numero de amigos en comun.
getNotMutualFriends()		String	Devolvera una lista con los primeros diez contactos con los que menos contactos compartimos en Facebook. Cada entrada de la lista contendrá la foto del contacto actual en Facebook, su nombre y el numero de amigos en comun.
getCommentsNumberPost()	String uidFriend, String postId, boolean firstCheck	int	Devuelve el numero de comentarios a un determinado post indicado por el parametro de entrada postId que ha hecho un determinado amigo, indicado por el parametro de entrada uidFriend.
isInArrayPost()	JSONArray friendsWithPostArray, String actorId	boolean	Comprueba si el identificador de un amigo esta contenido en el array de amigos que han escrito en el muro del usuario. Podrá lanzar excepciones del tipo JSONException.

Método	Entrada	Salida	Descripción
getFriendsWithPost()	JSONArray postStreamArray	JSONArray	Obtiene un JSONArray con todos los amigos que han escrito en el muro del usuario. Podrá lanzar excepciones del tipo JSONException.
getRankingPostFriends()	boolean firstCheck	String	Obtiene una lista con los diez amigos que mas comentarios y entradas en el muro del usuario han realizado hasta una fecha determinada. Si el chequeo se realiza por primera vez, tomara todos los comentarios del muro del usuario. Si no es así, tomara todos los comentarios realizados hasta la fecha indicada por el filtro de visitas. Podrá lanzar excepciones del tipo JSONException.
parseJsonArrayToMap()	JSONArray jsonArray	HashMap<String, Integer>	Convierte un JSONArray en un HashMap para proceder a su posterior ordenacion.
shortMap()	HashMap<String, Integer> map	HashMap<String, Integer>	Ordena un HashMap en orden descendente de mayor a menor.

6.3.6. Paquete es.uah.cc.ie.snsangelguardfb.sources.logs

Este paquete se encargará de configurar los mensajes de consola y redireccionarlos a un fichero .log. Este paquete contendrá una única clase que será la que, al inicio de la aplicación, se encargará de leer la configuración de un fichero **properties** y poner en disposición de todas las estructuras del programa las funcionalidades necesarias para que todas las trazas sean redireccionadas a su correspondiente fichero.

Para realizar ésta funcionalidad, se ha recurrido al tipo de traza **log4j**¹ que, al ser una librería externa, será necesario obtenerla por medio de su dependencia maven definida en el fichero **pom.xml**:

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.8</version>
</dependency>
```

¹Log4j es una biblioteca open source desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o “logs” (data logging) a tiempo de ejecución y no a tiempo de compilación como es comúnmente realizado. La configuración de salida y granularidad de los mensajes es realizada a tiempo de ejecución mediante el uso de archivos de configuración externos.

Para llevar a cabo esta funcionalidad, serán necesarias dos estructuras:

1. Un fichero de propiedades, donde se definirá el modo de funcionamiento del log, sus ficheros donde se redireccionarán las trazas, su nivel de granulidad, etc.
2. Un servlet de conexión que inicialice el fichero de propiedades y ponga toda la funcionalidad en disposición de todas las estructuras de la aplicación.

6.3.6.1. Fichero log4j.properties

Este fichero se encontrará en la carpeta **WEB-INF**, al mismo nivel que el fichero **web.xml**. Se encargará de definir la configuración que utilizará log4j para representar las trazas dentro del programa. Su configuración será la siguiente:

1. **Raiz desde donde se iniciarán las trazas:** Será necesario definir desde qué estructura dentro del programa se van a captar trazas. En nuestro caso, el paquete de inicio será **es.uah.cc.ie.snsangeguardfb**.
2. **Granulidad de la traza:** Se podrá definir hasta qué nivel de detalle se quiere mostrar. Por defecto, se mostrarán todos los mensajes indicados como **info**, pero si se quiere mostrar un detalle mayor se han especificado estos tres niveles más:
 - a) **debug**: Este nivel contendrá aquellas trazas más detalladas sobre operaciones.
 - b) **error**: Este nivel se hará cargo de aquellas trazas de error que lancen alguna excepción pero que no comprometan el funcionamiento del programa.
 - c) **fatal**: Este nivel se hará cargo de las trazas que se generen por errores graves del sistema y comprometan su funcionamiento.
3. **Niveles de salida de las trazas:** Se configurará desde éste fichero properties hacia qué ficheros se redireccionarán las trazas. Aquí se especificará también la ubicación exacta de los ficheros, en nuestro caso, se ubicarán dentro del servidor Web en la carpeta **logs**. Actualmente hay configurados cuatro niveles:
 - a) Nivel de consola **stdout**: Este nivel mostrará las trazas con su nivel de configuración generadas por el programa en la propia consola del IDE que estemos utilizando para su desarrollo. Muy útil para pruebas locales a la hora de desarrollar.
 - b) Nivel de fichero **file**: A éste nivel se redireccionarán todas aquellas trazas marcadas como **info** o **debug**. Se configurará también el nombre del fichero, el número de ficheros parciales y el máximo tamaño de éstos. En nuestro caso, nuestro fichero se llamará **SNSAngelGuardFB.log**, podrá tener un máximo de 10 ficheros parciales y su tamaño máximo serán 300 Kb.
 - c) Nivel de fichero **error**: En éste fichero se almacenarán, como hemos comentado antes, aquellas trazas que se generen por errores que no comprometan el funcionamiento de la aplicación. El fichero que los contendrá se denominará **SNSAngelGuardFB_error.log**, sólo habrá uno y su tamaño máximo será de 500 Kb.

- d) Nivel de fichero **fatal**: En éste fichero se almacenarán aquellos errores que han comprometido la estabilidad del sistema y han provocado su caída. Sólo habrá un fichero por este tipo de error, se denominará **SNSAngelGuardFB_fatal.log**, y su tamaño máximo será de 500Kb.

6.3.6.2. Clase Log4Init.java

Esta clase será un servlet de conexión disponible en el fichero **web.xml**. Al iniciarse la aplicación, éste carga todas las propiedades del fichero **log4j.properties** y las pondrá a disposición del resto de estructuras de la aplicación.

Cada clase que desee representar sus trazas en éste sistema, deberá tener un atributo estático **logger** que las represente. Su definición será la siguiente:

```
private static Logger logger = Logger.getLogger(NombreClase.class);
```

Tras esto, ya podrá escribir trazas en todos los niveles y ficheros donde estén configuradas. Para sacar un mensaje en el log, se seguirá la siguiente nomenclatura para cualquier nivel de los definidos anteriormente:

```
logger.info(Mensaje....)
```

6.3.7. Paquete es.uah.cc.ie.snsangeguardfb.sources.offline

Este paquete contendrá aquellas estructuras necesarias para llevar a cabo los procesos offline de actualización de información de la aplicación. Estos procesos se lanzarán desde el servidor donde esté instalada la aplicación mediante una orden del sistema que desencadenará la ejecución de las clases que se encuentran en éste paquete. Por el momento, sólo habrá definida una clase, **harvestedSNS.java**, que pasaremos a describir a continuación.

6.3.7.1. Clase HarvestedSNS.java

Realizará las operaciones de backup de la información de la base de datos y realizará los chequeos de información cuando sea necesario para cada uno de los filtros activos definidos para cada usuario. Extenderá de la clase **HttpServlet** y su definición se encontrará en el fichero de configuración **web.xml**. Al ser un servlet, se encontrará siempre a la espera de notificaciones hasta que el proceso offline comunique con él e inicie el proceso de backup.

Las tablas 6.54 y 6.55 mostrarán los atributos y los métodos, respectivamente, de ésta clase.

Tabla 6.55: Métodos de la clase HarvestedSNS.java

Método	Entrada	Salida	Descripción
VisitsFilterFuncionality()	SNSAngelGuardFBManager snsObject		Constructor de clase.

Método	Entrada	Salida	Descripción
updateUsers()	HttpServletRequest request, HttpServletResponse response		Realiza la actualizacion de las ultimas novedades de la actividad social en Facebook de todos los usuarios de la aplicacion y realiza los chequeos de informacion para los angeles definidos por cada usuario. Podra lanzar excepciones del tipo ParseException, FileNotFoundException, NoSuchProviderException, MessagingException, MySQLIntegrityConstraintViolationException, bsh.ParseException, java.security.NoSuchProviderException, InterDataBaseException, InterProcessException o InterEmailException.
openConectionOffLine()	HttpSession session	HttpSession	Abre la conexion off line del usuario actual que se va a analizar.
closeConectionOffLine()	HttpSession session	HttpSession	Cierra la conexion off line del usuario que se estaba analizando.
doGet()	HttpServletRequest request, HttpServletResponse response		Realiza la recepción de peticiones desde el servlet y desencadena las operaciones offline.

6.3.8. Paquete es.uah.cc.ie.snsangeguardfb.sources.snswebservicesclient

Este paquete contendrá todas las estructuras necesarias para comunicar, por medio de servicios **RESTFul**, la aplicación con el servidor de base de datos, contenido en la aplicación **SNSdataBaseIntegratorServer**, que será la que realice toda la lógica de negocio para persistir los datos en la base de datos **SocialNetwork**.

Como tal, éste paquete sólo contiene una clase, **SNSdataBaseClient.java**, la cual será un API de conexión con la base de datos por medio de servicios RESTFul. Toda la estructura de dicha clase está especificada en el capítulo 5.4, por lo que la especificación de su funcionalidad no es el objeto de ésta sección.

6.3.9. Paquete es.uah.cc.ie.snsangeguardfb.sources.utilities

Este paquete contendrá todas aquellas estructuras de utilidad que trabajan con datos de otras entidades y realizan operaciones comunes. El diagrama 6.7 nos muestra las clases definidas en éste paquete y que pasaremos a detallar a continuación.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.

Tabla 6.54: Atributos de la clase HarvestedSNS.java

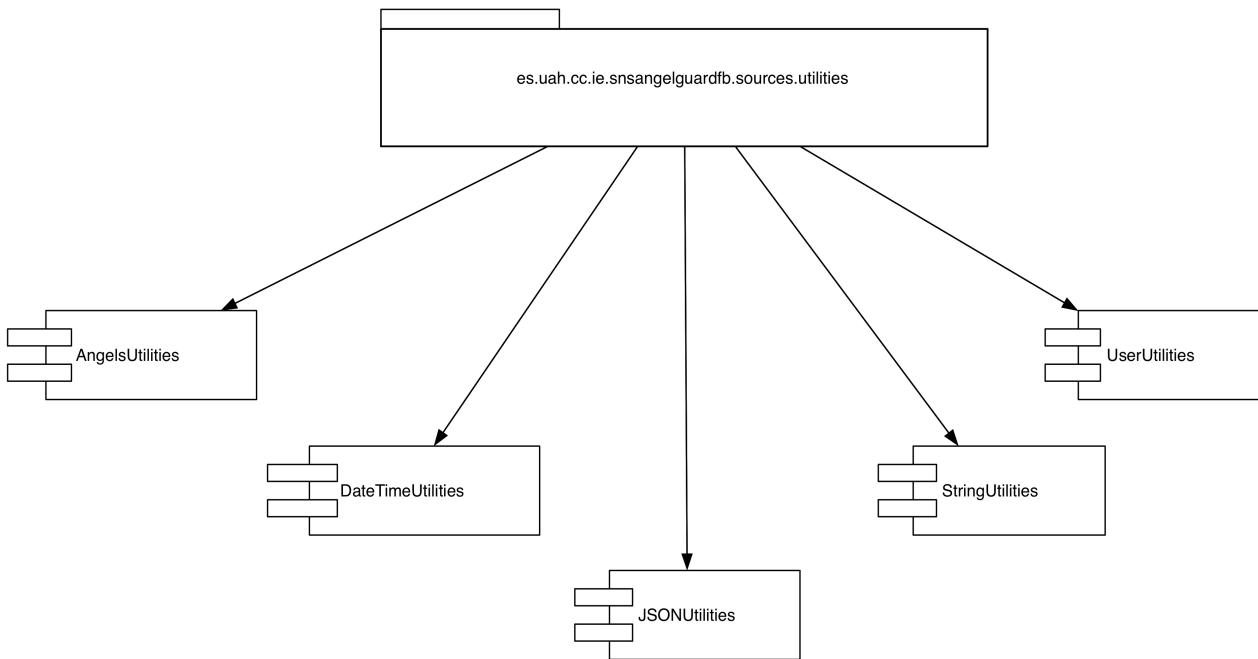


Figura 6.7: Paquete es.uah.cc.ie.snsangelguardfb.sources.utilities

6.3.9.1. Clase AngelsUtilities.java

Esta clase realizará operaciones propias con los ángeles de un usuario, tanto los seleccionados como los no seleccionados. Las tablas 6.56 y 6.57 mostrarán respectivamente los atributos y los métodos de ésta clase.

Tabla 6.57: Métodos de la clase AngelsUtilities.java

Método	Entrada	Salida	Descripción
VisitsFilterFuncionality()	SNSAngelGuardFBManager snsObject		Constructor de clase.
AngelsUtilities()	SNSAngelGuardFBManager snsObject		Constructor de la clase.
getAngelDates()	String angelUid	String[]	Obtiene de Facebook todos los datos de un angel.

Método	Entrada	Salida	Descripción
getEmailUserSNSAngelGuard()	String uid	String	Obtiene el email de un amigo de Facebook únicamente si este está dado de alta como usuario de la aplicación.
getAngelsSelected()	String angelsSelected		Obtiene de un String separado por ; todos los ángeles seleccionados y los almacena en la variable arrayAngelsSelected.
copyAngel()	int longArray	String[][]	Devuelve una lista con el número de ángeles seleccionados indicado por el parámetro de entrada longArray.
genericJoinListAngel()	String newAngels, String des, String Type		Une en una sola lista los ángeles de diferente topología, tales como Ed o Google.
joinAngels()	String angelsGoogle, String angelsEd		Une los ángeles Ed y Google en una sola lista.
getAngels()		String[][]	Obtiene de Facebook todos los amigos para mostrarlos por pantalla como ángeles a seleccionar. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException o JSONException.
getAngelDatesDB()	JSONObject jsonAngel	String[]	Obtiene todos los datos de un ángel almacenado en la base de datos y los almacena a un array de String desde un JSONObject.
getAngelsDB()	JSONArray jsonFriendsFacebook	String[][]	Obtiene todos los amigos de Facebook almacenados en la base de datos y los formatea a un array de String[][] para ser mostrados por pantalla. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException o JSONException.
isAngelSelect()	String[] angels, String idAngel	boolean	Indica si un amigo de Facebook está dentro de los seleccionados como ángeles del usuario.
loadAngelFile()	String[][] angels, String lstAngels, String txtNameAngels	JSONArray	Obtiene un JSONArray con la lista de los ángeles elegidos. Puede lanzar excepciones del tipo JSONException o IOException.

Método	Entrada	Salida	Descripción
getJSONAngel()	String idAngel, String uid	JSONObject	Obtiene los datos de un angel en un objeto JSON de la base de datos en el caso en el que existiera. Podra lanzar excepciones del tipo JSONException.
setJsonAngel()	JSONObject jsonAngel		Actualiza en base de datos la informacion de un angel. Puede lanzar excepciones del tipo JSONException.

6.3.9.2. Clase DateTimeUtilities.java

Sera la encargada de gestionar todas las operaciones con fechas que se realizan en la aplicacion. Las tablas 6.58 y 6.59 mostraran respectivamente los atributos y los métodos de ésta clase.

Tabla 6.59: Métodos de la clase DateTimeUtilities.java

Método	Entrada	Salida	Descripción
DateTimeUtilities()	SNSAngelGuardFBManager snsObject		Constructor de la clase.
formatTime()	String strTime	Date	Convierte un String a un objeto Date. Puede lanzar excepciones del tipo ParseException.
formatearFecha()	String fecha	String	Formatea una fecha recibida como String a otro String con formato YYYY-MM-DD.
getNextCheckTime()	int freqFilter, Date lastCheck	long	Obtiene la proxima fecha en la que se debera hacer un chequeo de informacion.

6.3.9.3. Clase JSONUtilities.java

Sera la encargada de realizar los parseos entre objetos String y JSONObject. Las tablas 6.60 y 6.61 mostraran respectivamente los atributos y los métodos de ésta clase.

Tabla 6.61: Métodos de la clase JSONUtilities.java

Método	Entrada	Salida	Descripción
JSONUtilities()	SNSAngelGuardFBManager snsObject		Constructor de la clase.

Método	Entrada	Salida	Descripción
getJSONArray()	String strValue	JSONArray	Convierte una cadena de texto a JSONArray. Puede lanzar excepciones del tipo JSONException.
getJSONCollection()	String strValue	JSONObject	Convierte una cadena de texto en la que hay en formato JSONObject varios objetos y devuelve un JSONObject con todos ellos. Puede lanzar excepciones del tipo JSONException.
getJSONObject()	String strValue	JSONObject	Convierte una cadena de texto sin formatear en un objeto JSONObject. Puede lanzar excepciones del tipo JSONException.
completStrToJSONObject()	String strValue	JSONObject	Convierte una cadena de texto formateada a JSON en un objeto JSONObject. Puede lanzar excepciones del tipo JSONException.

6.3.9.4. Clase StringUtilities.java

Clase de utilidad para cadenas de texto de tipo String. En muchas ocasiones, necesitaremos pasar cadenas de texto a objetos JSONObject con formato (clave, valor) más fáciles de manejar. En la mayoría de los casos, las cadenas de texto ya vendrán formateadas a un formato que hará más fácil los parseos hacia objetos JSONObject ya que ésta clase tendrá un constructor que recibirá un String y, si este tiene el formato correcto, devolverá un objeto JSONObject con el contenido del String. Las tablas 6.62 y 6.63 mostrarán respectivamente los atributos y los métodos de ésta clase.

Tabla 6.63: Métodos de la clase StringUtilities.java

Método	Entrada	Salida	Descripción
StringUtilities()	SNSAngelGuardFBManager snsObject		Constructor de la clase.
formatParams()	String order, String params	String	Formatea una cadena de parámetros separada por el carácter ; en una cadena de parámetros con el formato (...,...).
arrayToString()	String[][] angels	String	Convierte un array de dos dimensiones en una cadena de texto con formato de JSONArray.

Método	Entrada	Salida	Descripción
stringToArray()	String cadena	String[]	Convierte una cadena de parametros separados por el caracter ; en un array de una dimension.

6.3.9.5. Clase UserUtilities.java

Clases de utilidad para un usuario de la aplicacion.. Las tablas 6.64 y 6.65 mostraran respectivamente los atributos y los métodos de ésta clase.

Tabla 6.65: Métodos de la clase UserUtilities.java

Método	Entrada	Salida	Descripción
UserUtilities()	SNSAngelGuardFBManager snsObject		Constructor de la clase.
getImgUser()	String uid	String	Obtiene la url de la imagen de un contacto de Facebook. Podra lanzar excepciones del tipo JSONException.

6.4. Paquete es.uah.cc.ie.snsangelguardfb.sources.jspcontroler

La explicación de éste paquete se ha dejado en una sección a parte, ya que las estructuras en él contenidas van a hacer de nexo entre la parte cliente y la parte servidor de la aplicación.

Según la arquitectura de la aplicación, por cada página que interaccione con el usuario y envíe datos, debe haber una estructura en servidor que procese estos datos y sea capaz de interactuar con la página en cuestión. En nuestro caso, habrá un controlador por cada pantalla del cliente que, además de procesar los datos, dote a la página de todos los recursos de idioma necesarios para poder ejecutarse.

Este paquete contendrá las siguientes estructuras:

1. Clase **GenericJSPControler.java**: De ésta clase extenderán todas las estructuras que controlen las páginas de la parte cliente en la parte servidor.
2. Paquete **entity**: Contendrá todas las estructuras que extiendan de la clase GenericJSPControler.java.
3. Paquete **resources**: Contendrá todas las estructuras necesarias para dotar a las páginas del cliente de los recursos de idioma precargados anteriormente a su ejecución.

La imagen 6.8 mostrará la organización de éste paquete que pasará a explicarse a continuación.

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER153

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.
arrayAngels	String[][]	Array con todos los ángeles del usuario.
arrayAngelsSelected	String[][]	Array con los ángeles seleccionados de un usuario.

Tabla 6.56: Atributos de la clase AngelsUtilities.java

Atributo	Tipo	Descripción
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.

Tabla 6.58: Atributos de la clase DateTimeUtilities.java

6.4.1. Clase GenericJSPControler.java

Clase general a implementar por todas las clases JSPControler. Será una clase abstracta sin ningún atributo definido y únicamente un método que todas las clases que extiendan de ella deberán sobreescribir, el método **process()**, que será el que lleve a cabo toda la ejecución de la parte servidor y cargará todos los recursos necesarios para que la página cliente pueda usarlos.

6.4.2. Paquete entity

Este paquete contendrá todas las estructuras necesarias que extenderán de la clase **GenericJSPControler.java**. Habrá una clase por cada página de navegación del módulo cliente, es decir, una clase por cada flujo de información. Serán las siguientes:

1. Clase **CheckNowJSPControler**: Controlará la página **checkNow.jsp** que recibirá los datos elegidos de la página de configuración y los procesará para persistirlos en base de datos.
2. Clase **DeleteAngelSelectedJSPControler**: Controlará la página **deleteAngelSelected.jsp** que recibirá los datos necesarios para borrar un ángel.
3. Clase **DoOperationWithGoogleContactsJSPControler**: Controlará la página **doOperationWithGoogleContacts.jsp** que recibirá los datos necesarios para realizar todas las operaciones con ángeles de Google.
4. Clase **DoOperationWithOtherContactsJSPControler**: Controlará la página **doOperationWithOtherContacts.jsp** que recibirá los datos necesarios para realizar todas las operaciones con ángeles de otra tipología diferente a Facebook o Google.
5. Clase **IndexJSPControler**: Controlará la página **index.jsp** que será la página de inicio de la aplicación.

Atributo	Tipo	Descripción
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.

Tabla 6.60: Atributos de la clase JSONUtilities.java

Atributo	Tipo	Descripción
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.

Tabla 6.62: Atributos de la clase StringUtilities.java

6. Clase **LegalAcceptedJSPController**: Controlará la página **legalAccepted.jsp** que será la página de aceptación del acuerdo legal.
7. Clase **SaveEmailFacebookContactJSPController**: Controlará la página **saveEmailFacebookContact.jsp** que recibirá los datos necesarios para almacenar un ángel seleccionado de Facebook.
8. Clase **SaveNewAngelJSPController**: Controlará la página **saveNewAngel.jsp** que almacenará un nuevo ángel dado de alta en la aplicación.
9. Clase **SchedulerUserLoggedFacebookJSPController**: Controlará la página **schedulerUserLoggedFacebook.jsp** que enviará el flujo de ejecución a la página del acuerdo legal o a la página principal de configuración de la aplicación, dependiendo si el usuario es nuevo o no.
10. Clase **SettingsSNSAngelGuardJSPController**: Controlará la página **settingsSNSAngelGuard.jsp** que será la página de configuración principal.
11. Clase **SettingsSNSAngelGuardJSPController_Angels**: Controlará la selección de ángeles en la página **settingsSNSAngelGuard_Angels.jsp** que será una de las pestañas de la pagina principal.
12. Clase **SettingsSNSAngelGuardJSPController_Vigilants**: Controlará la selección de vigilantes en la página **settingsSNSAngelGuard_Vigilants.jsp** que será otra de las pestañas de la página de configuración principal.
13. Clase **TutorialInicioJSPController**: Controlará la ejecución de la página **tutorialInicio.jsp** que mostrará el tutorial al inicio de la aplicación.
14. Clase **UpdateFacebookFriendsOnlineJSPController**: Controlará la página **updateFacebookFriendsOnline.jsp** la cual actualizará los contactos de Facebook del usuario de la aplicación.

La imagen 6.9 mostrará el diagrama de despliegue de éste paquete.

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER155

Atributo	Tipo	Descripción
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.

Tabla 6.64: Atributos de la clase UserUtilities.java

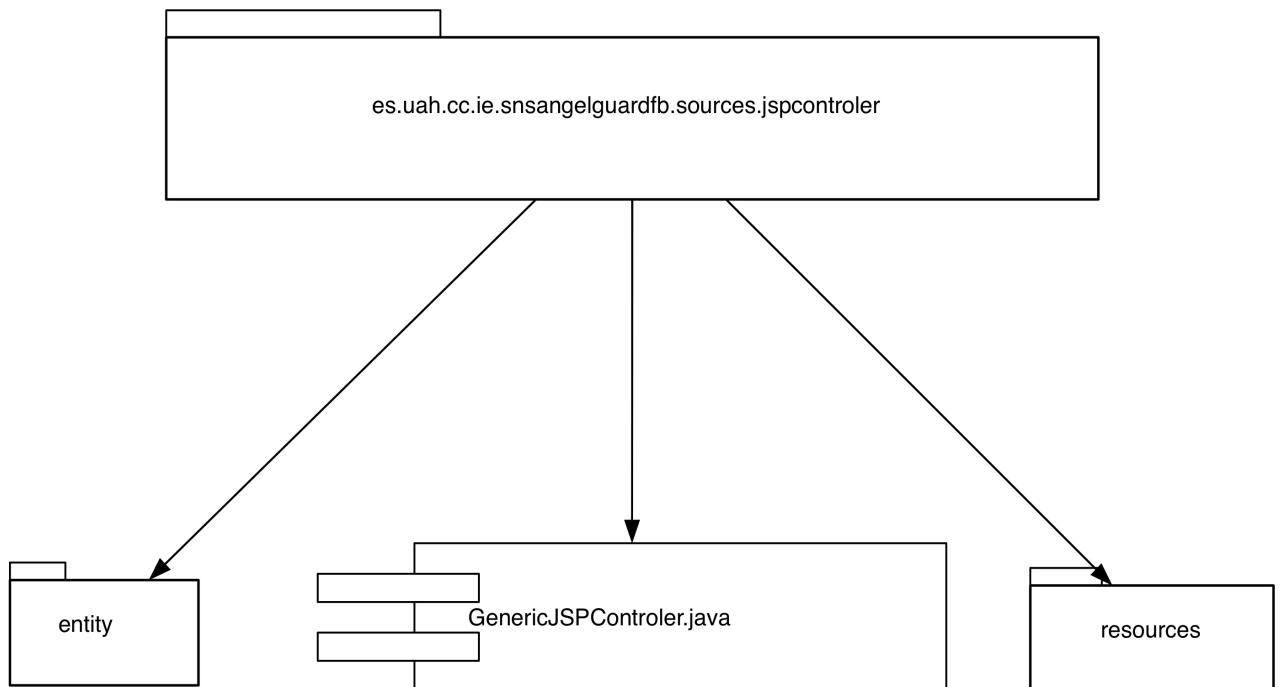


Figura 6.8: Paquete es.uah.cc.ie.snsangelguardfb.jspcontroler

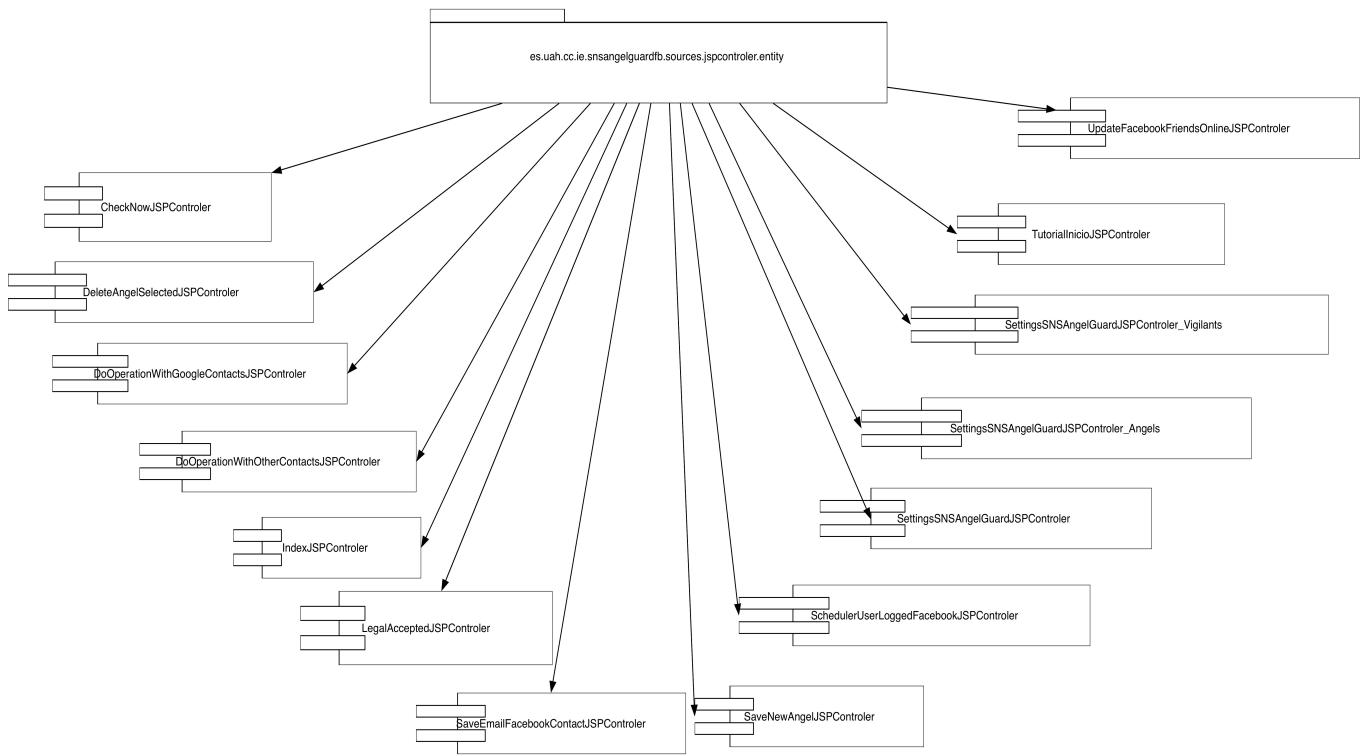
6.4.2.1. Clase CheckNowJSPControler.jsp

Esta clase será la encargada de controlar el flujo de ejecución de la página **checkNow.jsp**. No tendrá que cargar ningún recurso de idioma, únicamente realizará el procesado de información y la persistirá en base de datos. Si todo ha salido correctamente, volverá a la página de configuración mostrando un mensaje de confirmación. Si la operación no se ha ejecutado correctamente, lanzará un mensaje de error. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**.

Sus atributos se muestran en la tabla 6.66 y sus métodos en la tabla 6.67.

6.4.2.2. Clase DeleteAngelSelectedJSPControler.java

Esta clase será la encargada de gestionar a la página **deleteAngelSelected.jsp**. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Sus atributos se muestran en la tabla 6.68 y sus métodos en la tabla 6.69.

Figura 6.9: Paquete `es.uah.cc.ie.snsangelguardfb.jspcontroler.entity`

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.
request	HttpServletRequest	Manager de la aplicación.
response	HttpServletResponse	Atributo response de la sesión.

Tabla 6.66: Atributos de la clase `CheckNowJSPControler.jsp`

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER157

Método	Entrada	Salida	Descripción
CheckNowJSPControler()	HttpServletRequest request, HttpServletResponse response		Constructor de clase.
process()			Carga todos los filtros de información, actualiza los datos del usuario y retorna a la aplicación con el resultado obtenido.
loadFilters()			Carga la informacion de los filtros obtenida de la pagina de configuracion para, posteriormente, ser guardada en la base de datos. Podrá lanzar excepciones del tipo UnsupportedEncodingException.
updateInformationAndReturn()			Actualiza la informacion del usuario y vuelve a la pagina de configuracion de la aplicacion. Podrá lanzar excepciones del tipo UniformInterfaceException, IOException, JSONException, NoSuchProviderException, MessagingException, MySQLIntegrityConstraintViolationException, InterDataBaseException, InterProcessException o InterEmailException.
getStrFrecuencyFilter()	String freq	String	Obtiene la descripción de la frecuencia definida para un filtro.

Tabla 6.67: Métodos de la clase CheckNowJSPControler.jsp

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.
request	HttpServletRequest	Atributo request de la sesión.
response	HttpServletResponse	Atributo response de la sesión.
typeAngel	String	Tipo de ángel.
idAngel	String	Identificador en base de datos del ángel.
errorPostingWallAngel	String	Indica si se borrar el ángel por haberse cometido un error al escribir en el muro del usuario.

Tabla 6.68: Atributos de la clase DeleteAngelSelectedJSPControler.jsp

Método	Entrada	Salida	Descripción
DeleteAngelSelectedJSPControler()	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
process()			Ejecutará toda la funcionalidad de borrado de base de datos de un ángel.

Tabla 6.69: Metodos de la clase DeleteAngelSelectedJSPControler.jsp

6.4.2.3. Clase DoOperationWithGoogleContactsJSPControler.java

Esta clase será la encargada de gestionar a la página **doOperationWithGoogleContacts.jsp**. Realizará operaciones de inserción, actualización o borrado de los contactos de Google. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Sus atributos se muestran en la tabla 6.70 y sus métodos en la tabla 6.71.

6.4.2.4. Clase DoOperationWithOtherContactsJSPControler.java

Esta clase será la encargada de gestionar a la página **doOperationWithOtherContacts.jsp**. Realizará operaciones de inserción, actualización o borrado de los contactos que no pertenecen a Google ni a Facebook. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Sus atributos se muestran en la tabla 6.72 y sus métodos en la tabla 6.73.

6.4.2.5. Clase IndexJSPControler.java

Esta clase será la encargada de gestionar a la página **index.jsp**, que será la página inicial de la aplicación. Su función básica será un control de flujo. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Sus atributos se muestran en la tabla 6.74 y sus métodos en la tabla 6.75.

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER159

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
TYPE_ANGEL	final String	Tipo de angeles a tratar.
KEY_NAME_ANGEL_GOOGLE	final String	Key para el acceso al nombre del angel.
KEY_EMAIL_ANGEL_GOOGLE	final String	Key para el acceso al email del angel.
KEY_ID_ANGEL_GOOGLE_DEL	final String	Key para el acceso al id del angel.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.
typeOperation	TypeOperationContact	Tipo de operacion a realizar con el contacto.
jsonGoogleAngels	JSONArray	Array de objetos JSON que almacenara los angeles que van a ser procesados.

Tabla 6.70: Atributos de la clase DoOperationWithGoogleContactsJSPControler.java

6.4.2.6. Clase LegalAcceptedJSPControler.java

Esta clase será la encargada de controlar el flujo de ejecución de la página **legalAccepted.jsp**. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Esta clase como tal no realizará ninguna operación, únicamente cargará los recursos de idioma necesarios llamando a la clase **LegalAcceptedJSPControlerResources**.

Sus atributos se muestran en la tabla 6.76 y sus métodos en la tabla 6.77.

6.4.2.7. Clase SaveEmailFacebookContactsJSPControler.java

Esta clase será la encargada de controlar el flujo de ejecución de la página **saveEmailFacebookContacts.jsp**. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Esta clase proporcionará los mecanismos básicos para que un contacto de Facebook pueda registrar el email donde quiera recibir las notificaciones del usuario.

Sus atributos se muestran en la tabla 6.78 y sus métodos en la tabla 6.79.

Método	Entrada	Salida	Descripción
DoOperationWithGoogle..	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
loadTypeOperation()	String typeOperationRequest	TypeOperationContact	Establece el tipo de operacion a realizar con el/los contactos.
loadDetailOperation()			Obtiene los datos necesarios para realizar la operacion obtenidos del objeto request de la peticion.
loadArrayAngels()			Carga en el atributo de clase los angeles seleccionados para realizar esta operacion.
process()			Ejecutará todas las operaciones, dependiendo del tipo de operación, con los contactos de Google.

Tabla 6.71: Metodos de la clase DoOperationWithGoogleContactsJSPControler.java

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
TYPE_ANGEL	final String	Tipo de angeles a tratar.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.
typeOperation	TypeOperationContact	Tipo de operacion a realizar con el contacto.
idContact	String	Identificador en base de datos del ángel.
nameContact	String	Nombre del ángel.
emailContact	String	Email del ángel.

Tabla 6.72: Atributos de la clase DoOperationWithOtherContactsJSPControler.java

6.4.2.8. Clase SaveNewAngelJSPControler.java

Esta clase será la encargada de controlar el flujo de ejecución de la página **saveNewAngel.jsp**. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Esta clase almacenará un nuevo ángel de cualquier tipología y retornará a la página de configuración de la aplicación.

Sus atributos se muestran en la tabla 6.80 y sus métodos en la tabla 6.81.

6.4.2.9. Clase SchedulerUserLoggedFacebookJSPControler.java

Esta clase será la encargada de controlar el flujo de ejecución de la página **schedulerUserLoggedFacebook.jsp**. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Esta clase obtiene los datos de conexión del usuario

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER161

Método	Entrada	Salida	Descripción
DoOperationWithOther..	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
loadTypeOperation()	String typeOperationRequest	TypeOperationContact	Establece el tipo de operacion a realizar con el contacto.
loadDetailOperation()			Obtiene los datos necesarios para realizar la operacion con el contacto.
process()			Ejecutará todas las operaciones, dependiendo del tipo de operación, con el contacto.

Tabla 6.73: Metodos de la clase DoOperationWithOtherContactsJSPControler.java

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.

Tabla 6.74: Atributos de la clase IndexJSPControler.java

y reparte el flujo de ejecucion de la aplicacion en funcion de si el usuario es nuevo en la aplicacion o no.

Sus atributos se muestran el la tabla 6.82 y sus métodos en la tabla 6.83.

6.4.2.10. Clase SettingsSNSAngelGuardJSPControler.jsp

Esta clase será la encargada de controlar el flujo de ejecución de la página **settingsSNSAngelGuard.jsp**. Esta página además controlará, por medio de pestañas, la ejecución de las páginas de ángeles y vigilantes definidas en la parte cliente, por lo que también será necesario que cargue sus recursos de idioma. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**.

Sus atributos se muestran el la tabla 6.84 y sus métodos en la tabla 6.85.

Método	Entrada	Salida	Descripción
IndexJSPControler()	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
process()			Ejecutará el control de flujo de la aplicación.

Tabla 6.75: Metodos de la clase IndexJSPControler.java

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
jspResources	LegalAcceptedJSPControlerResources	Recursos de idioma para la pagina legalAccepted.jsp.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.

Tabla 6.76: Atributos de la clase LegalAcceptedJSPControler.java

Método	Entrada	Salida	Descripción
LegalAcceptedJSPControler()	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
process()			Este metodo carga los recursos de idioma y prepara la página para la aceptación o negación del acuerdo legal.

Tabla 6.77: Metodos de la clase LegalAcceptedJSPControler.java

6.4.2.11. Clase SettingsSNSAngelGuardJSPControler _ Angels.jsp

Esta clase será la encargada de controlar el flujo de ejecución de la página **settingsSN-SAngelGuard_Angels.jsp**. Esta página ejecutará todo el contenido de la pestaña de selección de ángeles y cargará todos sus recursos de idioma. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**.

Sus atributos se muestran en la tabla 6.86 y sus métodos en la tabla 6.87.

6.4.2.12. Clase SettingsSNSAngelGuardJSPControler _ Vigilants.jsp

Esta clase será la encargada de controlar el flujo de ejecución de la página **settingsSN-SAngelGuard_Vigilants.jsp**. Esta página ejecutará todo el contenido de la pestaña de configuración de vigilantes y cargará todos sus recursos de idioma. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**.

Sus atributos se muestran en la tabla 6.88 y sus métodos en la tabla 6.89.

6.4.2.13. Clase TutorialInicioJSPControler.java

Esta clase será la encargada de gestionar a la página **tutorialInicio.jsp**, que será la página que muestre el tutorial de inicio. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Sus atributos se muestran en la tabla 6.90 y sus métodos en la tabla 6.91.

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER163

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
PATH_IMAGE_LOADING	final String	Imagen de la carga entre páginas.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.
uidAngel	String	Identificación del ángel.
uidPublicUser	String	Identificador del usuario a quien pertenece el angel (Inicialmente vendra cifrada).
jsonAngel	JSONObject	Informacion en formato JSON del angel cargada de base de datos.
nameAngelValue	String	Nombre del angel que saldra por pantalla.
notificationMsg	String	Mensaje de autorizacion para el envio de notificaciones.
emptyEmailMsg	String	Mensaje de email vacio.
notValidEmailMsg	String	Mensaje de email no válido.
loaderSave	String	Mensaje de carga para el loader.
loaderWait	String	Mensaje de espera para el loader.

Tabla 6.78: Atributos de la clase SaveEmailFacebookContactsJSPControler.java

6.4.2.14. Clase UpdateFacebookFriendsOnlineJSPControler.java

Esta clase será la encargada de gestionar a la página **updateFacebookFriendsOnline.jsp**, que actualizará todos los amigos de Facebook de un usuario y los recargará en la pantalla de configuracion de angeles. Extenderá de la clase **GenericJSPControler.java** e implementará su método **process()**. Sus atributos se muestran en la tabla 6.92 y sus métodos en la tabla 6.93.

Método	Entrada	Salida	Descripción
SaveEmailFacebookContact..	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
process()			Este método guarda el email introducido por el usuario de Facebook en base de datos y muestra el mensaje de confirmación.

Tabla 6.79: Metodos de la clase SaveEmailFacebookContactsJSPControler.java

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicación.
request	HttpServletRequest	Atributo request de la sesión.
response	HttpServletResponse	Atributo response de la sesión.
typeAngel	String	Tipo de ángel.
idAngel	String	Identificador en base de datos del ángel.

Tabla 6.80: Atributos de la clase SaveNewAngelJSPControler.java

Método	Entrada	Salida	Descripción
SaveNewAngelJSPControler()	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
process()			Almacena el ángel seleccionado por el usuario.

Tabla 6.81: Métodos de la clase SaveNewAngelJSPControler.java

6.4.3. Paquete data

Este paquete contendrá todas las clases de datos internos que necesite el paquete anterior para su funcionamiento.

6.4.3.1. Enumerado TypeOperationContact

Este enumerado contendrá los tipos de operaciones a realizar cuando se ejecuta una operación con los ángeles. Contendrá únicamente tres valores:

1. **NEW_CONTACT**: Para la inserción de un nuevo ángel.
2. **UPDATE_CONTACT**: Para la actualización de un ángel ya existente.
3. **DELETE_CONTACT**: Para el borrado de un ángel ya existente.

6.4.4. Paquete threads

Este paquete contendrá las estructuras necesarias para realizar operaciones pesadas en la parte cliente en hilos independientes de ejecución. Así conseguimos flexibilizar la parte cliente con operaciones que se ejecutan en segundo plano sin afectar a la ejecución de la aplicación.

6.4.4.1. Clase ThProcessCheckFilter.java

Este hilo realiza el primer chequeo de información para un filtro. Cuando un ángel acepta el acuerdo de seguimiento de un usuario de Facebook, se guarda este consentimiento en base de datos y, acto seguido, se crea un hilo de ejecución con ésta clase, que extenderá

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER165

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.
pathDestino	String	URL de destino de la pagina.
loaderSave	String	Mensaje de carga para el loader.
loaderWait	String	Mensaje de espera para el loader.

Tabla 6.82: Atributos de la clase SchedulerUserLoggedFacebookJSPControler.java

Método	Entrada	Salida	Descripción
SchedulerUserLoggedFa..	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
readURL()	URL url	String	Lee la url que obtiene la accessToken ampliada.
getExtendedTokenFacebook()	String accessToken	String	Metodo privado que amplia la accessToken a un rango temporal mayor. Necesario para accesos offline.
process()			Almacena el ángel seleccionado por el usuario.

Tabla 6.83: Metodos de la clase SchedulerUserLoggedFacebookJSPControler.java

de la clase **Thread**, para realizar la operación pesada del primer chequeo de información, generando un informe que será enviado a su email.

Sus atributos se muestran en la tabla 6.94 y sus métodos en la tabla 6.95.

6.4.4.2. Clase ThUpdateInformationUser.java

Este hilo actualiza la información del usuario de Facebook tras guardar su configuración de la aplicación. Cuando un usuario de la aplicación guarda la configuración, los datos de ésta se guardan y, acto seguido, se crea un hilo de ejecución con ésta clase, que extenderá de la clase **Thread**, para realizar la operación pesada de la actualización de todos sus datos del perfil de Facebook en nuestra base de datos.

Sus atributos se muestran en la tabla 6.96 y sus métodos en la tabla 6.97.

6.4.5. Paquete resources

Este paquete contendrá todas las estructuras necesarias para cargar los recursos de idioma de cada una de las páginas a las que hace referencia. Al igual que las estructuras contenidas en el paquete anterior, habrá una estructura de este tipo por cada página de cliente que necesite recursos de idioma y genere un flujo de información de respuesta contra la parte servidora. Las estructuras de este paquete serán las siguientes:

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
jspResources	SettingsSNSAngelGuardJSP- ControlerResources	Recursos de idioma para la pagina settingsSNSAngelGuard.jsp.
request	HttpServletRequest	Manager de la aplicacion.
response	HttpServletResponse	Atributo response de la sesion.
newConection	String	Indicador de nueva conexion.
resultSave	String	Resultado de la operacion.
angels	String[][]	Angeles del usuario.

Tabla 6.84: Atributos de la clase SettingsSNSAngelGuardJSPControler.jsp

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSPControler()	HttpServletRequest request, HttpServletResponse response		Constructor de clase. Al obtener parámetros del objeto request, podrá lanzar excepciones del tipo InterDataBaseException, InterProcessException o InterEmailException.
process()			Este metodo carga los recursos de idioma y, dependiendo si es un usuario nuevo, inicializa las estructuras para almacenar informacion o carga los datos de usuario si es un usuario existente.
loadResources()			Carga los recursos de idioma.

Tabla 6.85: Métodos de la clase SettingsSNSAngelGuardJSPControler.jsp

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER167

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
jspResources	SettingsSNSAngelGuardJSP- ControlerResourcesAngels	Recursos de idioma para la pagina settingsSNSAngel- Guard_Angels.jsp.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.
hdAngels	String	Angeles elegidos.
hdAngelsEd	String	Angeles elegidos de Otros Contac- tos.
hdAngelsGoogleSelected	String	Angeles elegidos de contactos de Google.
hdLstAngelsFltWall	String	Lista de angeles para el filtro de con- trol de lenguaje.
hdLstAngelsFltFriends	String	Lista de angeles para el filtro de con- trol de amistades.
hdLstAngelsFltPriv	String	Lista de angeles para el filtro de con- trol de privacidad.
hdLstAngelsFltVist	String	Lista de angeles para el filtro de con- trol de visitas.
hdActiveFltWall	String	Indicador de si esta activo el filtro de control de lenguaje
hdActiveFltFriends	String	Indicador de si esta activo el filtro de control de amistades.
hdActiveFltPriv	String	Indicador de si esta activo el filtro de control de privacidad.
hdActiveFltVist	String	Indicador de si esta activo el filtro de control de visitas.
hdFrecFltWall	String	Frecuencia elegida para el filtro de control de lenguaje.
hdFrecFltFriends	String	Frecuencia elegida para el filtro de control de amistades.
hdFrecFltPriv	String	Frecuencia elegida para el filtro de control de privacidad.
hdFrecFltVist	String	Frecuencia elegida para el filtro de control de visitas.
hdAngelsAux	String	Lista de angeles auxiliar.

Tabla 6.86: Atributos de la clase SettingsSNSAngelGuardJSPControler_Angels.jsp

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSPController_Angels()	HttpServletRequest request, HttpServletResponse response		Constructor de clase. Podrá lanzar excepciones del tipo InterDataBaseException, InterProcessException o InterEmailException.
process()			Este método carga los recursos de idioma e inicializa la página de selección de ángeles.
loadResources()			Carga los recursos de idioma.

Tabla 6.87: Métodos de la clase SettingsSNSAngelGuardJSPController_Angels.jsp

1. Recurso **LegalAcceptedJSPControllerResources.java**: Contendrá todos los recursos de idioma de la página **legalAccepted.jsp**, así como algunas rutas a recursos de imágenes o páginas de destino.
2. Recurso **SettingsSNSAngelGuardJSPControllerResources.java**: Contendrá todos los recursos de idioma de la página **settingsSNSAngelGuard.jsp**, que será la página principal de configuración de la aplicación.
3. Recurso **SettingsSNSAngelGuardJSPControllerResourcesAngels.java**: Contendrá todos los recursos de idioma de la página **settingsSNSAngelGuard_Angels.jsp**, la cual será una de las dos pestañas que se carguen dentro de la página principal de configuración.
4. Recurso **SettingsSNSAngelGuardJSPControllerResourcesAngelsMenu.java**: Contendrá los recursos de idioma para cargar el menú de las pestañas que hacen mención a la selección de ángeles dentro de la página de configuración.
5. Recurso **SettingsSNSAngelGuardJSPControllerResourcesJQueryMenuResources.java**: Contendrá los recursos de idioma para cargar el plugin de visualización de los contactos de Facebook y los ángeles dentro de la página de configuración de la aplicación.
6. Recurso **SettingsSNSAngelGuardJSPControllerResourcesVigilants.java**: Contendrá todos los recursos de idioma de la página **settingsSNSAngelGuard_Vigilants.jsp**, la cual será una de las dos pestañas dentro de la página principal de configuración.
7. Recurso **SettingsSNSAngelGuardJSPControllerResourcesVigilantsMenu.java**: Contendrá los recursos de idioma para cargar el menú de las pestañas que hacen mención a la selección de vigilantes dentro de la página de configuración.
8. **TutorialInicioJSPControllerResources.java**: Contendrá los recursos de idioma necesarios para cargar el tutorial de inicio de la aplicación.

La imagen 6.10 hará mención a la arquitectura del paquete indicada hasta ahora.

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER169

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
jspResources	SettingsSNSAngelGuardJSP- ControlerResourcesVigilants	Recursos de idioma para la pagina settingsSNSAngel- Guard_Angels.jsp.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.
hdAngels	String	Angeles elegidos.
hdAngelsEd	String	Angeles elegidos de Otros Contactos.
hdAngelsGoogleSelected	String	Angeles elegidos de contactos de Google.
hdLstAngelsFltWall	String	Lista de angeles para el filtro de control de lenguaje.
hdLstAngelsFltFriends	String	Lista de angeles para el filtro de control de amistades.
hdLstAngelsFltPriv	String	Lista de angeles para el filtro de control de privacidad.
hdLstAngelsFltVist	String	Lista de angeles para el filtro de control de visitas.
hdActiveFltWall	String	Indicador de si esta activo el filtro de control de lenguaje
hdActiveFltFriends	String	Indicador de si esta activo el filtro de control de amistades.
hdActiveFltPriv	String	Indicador de si esta activo el filtro de control de privacidad.
hdActiveFltVist	String	Indicador de si esta activo el filtro de control de visitas.
hdFrecFltWall	String	Frecuencia elegida para el filtro de control de lenguaje.
hdFrecFltFriends	String	Frecuencia elegida para el filtro de control de amistades.
hdFrecFltPriv	String	Frecuencia elegida para el filtro de control de privacidad.
hdFrecFltVist	String	Frecuencia elegida para el filtro de control de visitas.
hdAngelsAux	String	Lista de angeles auxiliar.
arrayAngels	String[][]	Array bidimensional de angeles.

Tabla 6.88: Atributos de la clase SettingsSNSAngelGuardJSPControler_Vigilants.jsp

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSPController_Vigilants()	HttpServletRequest request, HttpServletResponse response		Constructor de clase. Podrá lanzar excepciones del tipo InterDataBaseException, InterProcessException o InterEmailException.
process()			Este método carga los recursos de idioma e inicializa la página de configuración de los vigilantes.
loadResources()			Carga los recursos de idioma.

Tabla 6.89: Métodos de la clase SettingsSNSAngelGuardJSPController_Vigilants.jsp

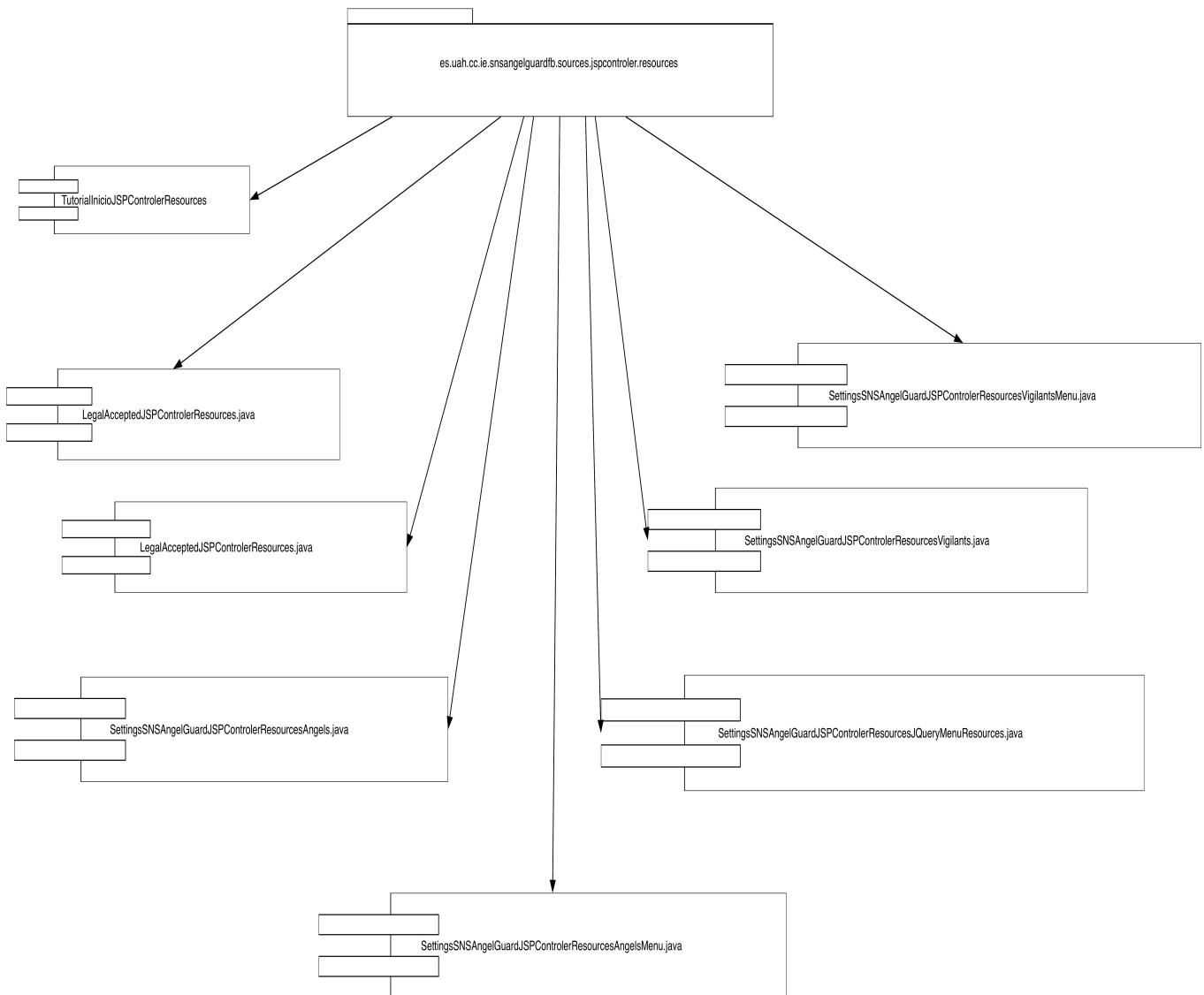


Figura 6.10: Paquete es.uah.cc.ie.snsangelguardfb.sources.jspcontroler.resources

6.4. PAQUETE ES.UAH.CC.IE.SNSANGELGUARFB.SOURCES.JSPCONTROLER171

Atributo	Tipo	Descripción
COD_LOCALE_SP	final String	Codigo interno de locale para el idioma castellano .
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.
locale	String	Logger del sistema.
resources	TutorialInicioJSPControlerResources	Recursos de idioma.

Tabla 6.90: Atributos de la clase TutorialInicioJSPControler.java

Método	Entrada	Salida	Descripción
TutorialInicioJSPControler()	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
getCodLocale()		String	Método que obtiene el código del idioma del usuario para cargar sus descripciones. Por defecto, siempre devolverá el código de locale en inglés. Si el usuario tiene idioma castellano, devolverá el locale de España.
loadResources()			Método que carga todos los recursos de idioma necesarios para el tutorial.
process()			Ejecuta el método loadResources().

Tabla 6.91: Metodos de la clase TutorialInicioJSPControler.java

6.4.5.1. Clase LegalAcceptedJSPControlerResources.java

Esta clase contendrá todos los recursos de idioma, almacenados en variables de cadenas de texto y constantes, necesarias para representar la página de aceptación del acuerdo legal. Sus atributos están reflejados en la tabla 6.98 y sus métodos en la tabla 6.99.

6.4.5.2. Clase SettingsSNSAngelGuardJSPControlerResources.java

Esta clase contendrá todos los recursos de idioma necesarios para la página principal de configuración de la aplicación. Contendrá además, todos los recursos necesarios para cargar el menú de pestañas con las que se navega desde la parte de selección de ángeles hasta la parte de configuración de los vigilantes. Sus atributos están reflejados en la tabla 6.100 y sus métodos en la tabla 6.101.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
request	HttpServletRequest	Atributo request de la sesion.
response	HttpServletResponse	Atributo response de la sesion.

Tabla 6.92: Atributos de la clase UpdateFacebookFriendsOnlineJSPControler.java

Método	Entrada	Salida	Descripción
UpdateFacebookFriends..	HttpServletRequest request, HttpServletResponse response		Constructor de la clase.
process()			Inicia la operación de actualización de los contactos de Facebook del usuario.

Tabla 6.93: Metodos de la clase UpdateFacebookFriendsOnlineJSPControler.java

6.4.5.3. Clase SettingsSNSAngelGuardJSPControlerResourcesAngels.java

Esta clase contendrá todos los recursos de idioma necesarios para la página de selección de ángeles de la página de configuración. A ésta página se accederá por medio de las pestañas de selección, pinchando sobre **Ángeles**. Sus atributos están reflejados en la tabla 6.102 y sus métodos en la tabla 6.103.

6.4.5.4. Clase SettingsSNSAngelGuardJSPControlerResourcesAngelsMenu.java

Esta clase contendrá todos los recursos de idioma para la pestaña de selección de la página de ángeles. Sus atributos están reflejados en la tabla 6.104 y sus métodos en la tabla 6.105.

6.4.5.5. Clase SettingsSNSAngelGuardJSPControlerResourcesJQueryMenu-Resources.java

Esta clase controlará toda la ejecución y los recursos de la selección de pestañas. Sus atributos están reflejados en la tabla 6.106 y sus métodos en la tabla 6.107.

6.4.5.6. Clase SettingsSNSAngelGuardJSPControlerResourcesVigilants.java

Esta clase contendrá todos los recursos de idioma necesarios para la página de configuración de vigilantes de la página de configuración. A ésta página se accederá por medio de las pestañas de selección, pinchando sobre **Vigilantes**. Sus atributos están reflejados en la tabla 6.108 y sus métodos en la tabla 6.109.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.
request	HttpServletRequest	Atributo request de la sesion.
jsonAngel	JSONObject	Datos del ángel.
isAcceptApp	boolean	Flag que indica si el angel ha aceptado la aplicacion.
typeAngel	String	Tipo de angel.
jsonUser	JSONObject	JSON con los datos del usuario de la aplicacion instanciado .

Tabla 6.94: Atributos de la clase ThProcessCheckFilter.java

Método	Entrada	Salida	Descripción
ThProcessCheckFilter	SNSAngelGuardFBManager snsObject, HttpServletRequest request, JSONObject jsonAngel, boolean isAcceptApp, String typeAngel, JSONObject jsonUser		Constructor de la clase.
run()			Método heredado de la clase Thread . Ejecuta el hilo.

Tabla 6.95: Metodos de la clase ThProcessCheckFilter.java

6.4.5.7. Clase SettingsSNSAngelGuardJSPControlerResourcesVigilantsMenu.java

Esta clase contendrá todos los recursos de idioma para la pestaña de selección de la página de vigilantes. Sus atributos están reflejados en la tabla 6.110 y sus métodos en la tabla 6.111.

6.4.5.8. Clase TutorialInicioJSPControlerResources.java

Esta clase contendrá todos los recursos de idioma necesarios para la ejecución del tutorial de inicio.

6.5. Operaciones

En éste capítulo se llevará a cabo una explicación de las operaciones más importantes que lleva a cabo el módulo de servidor, que estarán relacionadas con el flujo de información entre las pantallas de la aplicación.

Atributo	Tipo	Descripción
logger	Logger	Logger del sistema.
snsObject	SNSAngelGuardFBManager	Manager de la aplicacion.

Tabla 6.96: Atributos de la clase ThUpdateInformationUser.java

Método	Entrada	Salida	Descripción
ThUpdateInformationUser	SNSAngelGuardFBManager snsObject		Constructor de la clase.
run()			Método heredado de la clase Thread . Ejecuta el hilo.

Tabla 6.97: Metodos de la clase ThUpdateInformationUser.java

6.5.1. Login en la aplicación

Esta operación mostrará las acciones a realizar para que un usuario se logue en Facebook para acceder a la aplicación y los pasos previos para poder ejecutarla. Para realizar el inicio de la aplicación será necesario que el usuario que la vaya a utilizar tenga una cuenta válida en Facebook. A partir del momento en el que el usuario lanza la operación, se producirán las siguientes acciones:

1. El módulo servidor lanzará la página **index.jsp**. Esta comprobará si el usuario está conectado a Facebook:
 - a) Si no está conectado, el módulo servidor lanzará una url que conectaría exteriormente con Facebook para que el usuario pueda introducir sus datos directamente en la página oficial de Facebook, la cual, después del logueo, enviará los datos de vuelta a nuestra aplicación.
2. Cuando el usuario esté logueado, la aplicación comprobará si tiene los permisos necesarios para ejecutarse. Estos son permisos ofrecidos por Facebook para que las aplicaciones puedan acceder a datos del usuario siempre que éste haya dado su consentimiento previamente. Los permisos que necesitamos están enumerados en el capítulo 6.2.7.1.
 - a) Si no tiene los permisos, se lanzará una URL desde el módulo servidor para obtener el consentimiento de los permisos por parte del usuario. En el caso en el que el usuario no desee autorizar a la aplicación, esta dejará de ejecutarse.
3. En el caso en el que el usuario cuente con los permisos necesarios o los haya confirmado en éste momento, se iniciará la aplicación.

Todas estas operaciones pueden verse en el diagrama de actividad 6.11.

Atributo	Tipo	Descripción
DESTINY_REDIRECT_JPS	String	Constante: Pagina de destino.
PATH_IMAGE_ICONO_LEFT	String	Constante: URL al icono de la parte izquierda superior.
PATH_IMAGE_ICONO_RIGHT	String	Constante: URL al icono de la parte derecha superior.
PATH_IMAGE_LOADING	String	URL a la imagen de carga entre páginas .
snsObject	SNSAngelGuardFBManager	Clase manager de la aplicacion.
title	String	Titulo de la pagina.
acceptingTerms	String	Introducion al acuerdo legal.
legalAccepted	String	Texto informativo del acuerdo legal.
loaderSave	String	Menaje de guardado de informacion.
loaderWait	String	Mensaje de espera de la carga entre páginas.
titleBtnCancel	String	Titulo del boton cancelar.
titleBtnIAgree	String	Titulo del boton aceptar.

Tabla 6.98: Atributos de la clase LegalAcceptedJSPControlerResources.java

Método	Entrada	Salida	Descripción
LegalAcceptedJSPControlerResources()	SNSAngelGuardFBManager snsObject		Constructor de clase.
loadResources()			Carga los recursos de idioma.

Tabla 6.99: Métodos de la clase LegalAcceptedJSPControlerResources.java

Atributo	Tipo	Descripción
DESTINY_JSP_HELP	String	Constante: URL a la pagina de ayuda.
DESTINY_JSP_INIT	String	Constante: URL a la pestaña inicial que se cargara por defecto.
snsObject	SNSAngelGuardFBManager	Clase manager de la aplicacion.
angels	String	Angeles.
jspResourcesJQuery-Menu	SettingsSNSAngelGuardJSP-ControllerResourcesJQuery-MenuResources	Recursos de idioma para el menu de las pestañas.
jspResourcesAngels-Menu	SettingsSNSAngelGuardJSP-ControllerResourcesAngelsMenu	Recursos de idioma para la pestaña de angeles.
jspResourcesVigilants-Menu	SettingsSNSAngelGuardJSP-ControllerResourcesVigilants-Menu	Recursos de idioma para la pestaña de vigilantes.
menLoader	String[]	Mensajes de carga entre paginas.
titleSettings	String	Titulo de la pagina.
localeMenu	String[]	Titulos de las pestañas de la pagina.
titleHelp	String[]	Titulo para el enlace a la ayuda.
titleFbList	String	Titulo para el contenedor de angeles de Facebook.
titleAngelSettAng	String	Titulo para los contenidos de angeles de Google y otros contactos.
nameTutor	String	Titulo del nombre del angel en el contenedor de otros contactos.
emailTutor	String	Titulo para el email del angel en el contenedor de otros contactos.
mensaje	String	Mensaje de confirmacion.
menSave	String	Mensaje de guardado.
menWait	String	Mensaje de espera entre cargas de paginas.
btnSaveSettings	String	Titulo para el boton guardar.

Tabla 6.100: Atributos de la clase SettingsSNSAngelGuardJSPControllerResources.java

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSP-ControllerResources()	SNSAngelGuardFBManager snsObject, String angels		Constructor de clase.
loadResourcesSettings()			Carga los recursos de idioma.

Tabla 6.101: Métodos de la clase SettingsSNSAngelGuardJSPControllerResources.java

Atributo	Tipo	Descripción
snsObject	SNSAngelGuardFBManager	Clase manager de la aplicacion.
arrayContacts	String[]	Titulos para el contenedor de amigos de Facebook.
arrayAltAngels	String	Titulos para los contenedores de Google y Otros Contactos.
nameContact	String	Titulo del nombre del contenedor de Otros Contactos.
emailContact	String	Titulo para el email del contenedor de Otros Contactos.
warnings	String[]	Mensajes de aviso para la pagina.
confirm	String	Mensaje de confirmacion de angel.
delete	String	Mensaje de borrado de un angel.

Tabla 6.102: Atributos de la clase SettingsSNSAngelGuardJSPControlerResources.java

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSPControlerResourcesAngels()	SNSAngelGuardFBManager snsObject		Constructor de clase.

Tabla 6.103: Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesAngels.java

Atributo	Tipo	Descripción
ON_OUT_ICON	String	Constante: URL al icono de la pestaña cuando se pasa el puntero del raton.
ON_CLICK_ICON	String	Constante: URL al icono de la pestaña cuando se pulsa sobre esta.
HTML_STATUS_CONTENT	String	Constante: Estatus HTML.
DESTINY_URL_JSP	String	Constante: URL de destino.
title	String	Titulo de la pestaña.
dataAngels	String	Datos de los angeles.

Tabla 6.104: Atributos de la clase SettingsSNSAngelGuardJSPControlerResourcesAngelsMenu.java

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSPControlerResourcesAngelsMenu()	String title, String angels		Constructor de clase.

Tabla 6.105: Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesAngelsMenu.java

Atributo	Tipo	Descripción
CAPTION_CLASS	String	Constante: Caption class.
ON_OUT_CLASS	String	Constante: Clase cuando no está seleccionada la pestaña.
ON_OVER_CLASS	String	Constante: Clase seleccionada cuando el puntero se situa sobre la pestaña.
ON_CLICK_CLASS	String	Constante: Clase seleccionada al clicar sobre la pestaña.
HS_OUT_CLASS	String	Constante: HS out class.
HS_CLICK_CLASS	String	Constante: UHS click class.
PATH_IMAGE_LOADING_MENU	String	Constante: Path de la imagen loading que sale al clicar sobre una pestaña.
AJAX_CONTENT	String	Constante: Contenedor ajax de la pagina.
title	String	Titulo del menu.

Tabla 6.106: Atributos de la clase SettingsSNSAngelGuardJSPControlerResourcesJQueryMenuResources.java

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSP- ControlerResourcesJQueryMenuResources()	String title		Constructor de clase.

Tabla 6.107: Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesJQueryMenuResources.java

Atributo	Tipo	Descripción
snsObject	SNSAngelGuardFBManager	Clase manager de la aplicacion.
titleSettVig	String	Titulo de la pagina.
titleVigilantSettVig	String	Titulo del contenedor de filtros.
titleVigFrecSettVig	String	Titulo del contenedor de angeles para seleccion de frecuencia.
titleFbList	String	Titulos para las pestañas del contenedor de los contactos de Facebook.
titleAngelSettAng	String	Titulo para cada contacto de Facebook.
arrayVig	String[]	Titulos de los filtros disponibles.
arrayDes	String[]	Descripcion para cada filtro.
arrayFrc	String[]	Titulos del contenedor de seleccion de frecuencia.
titleVigAngSettVig	String	Titulo del contenedor de angeles.

Tabla 6.108: Atributos de la clase SettingsSNSAngelGuardJSPControlerResources.java

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSPControlerResourcesVigilants()	SNSAngelGuardFBManager snsObject		Constructor de clase.

Tabla 6.109: Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesAngeles.java

Atributo	Tipo	Descripción
ON_OUT_ICON	String	Constante: Icono entrada/salida.
ON_CLICK_ICON	String	Constante: URL al icono de la pestaña cuando se pulsa sobre esta.
HTML_STATUS_CONTENT	String	Constante: Estatus HTML.
DESTINY_URL_JSP	String	Constante: URL de destino.
DATA	String	Constante: Datos de llamada.
title	String	Titulo de la pestaña.

Tabla 6.110: Atributos de la clase SettingsSNSAngelGuardJSPControlerResourcesVigilantsMenu.java

Método	Entrada	Salida	Descripción
SettingsSNSAngelGuardJSPControlerResourcesVigilantsMenu()	String title		Constructor de clase.

Tabla 6.111: Métodos de la clase SettingsSNSAngelGuardJSPControlerResourcesVigilantsMenu.java

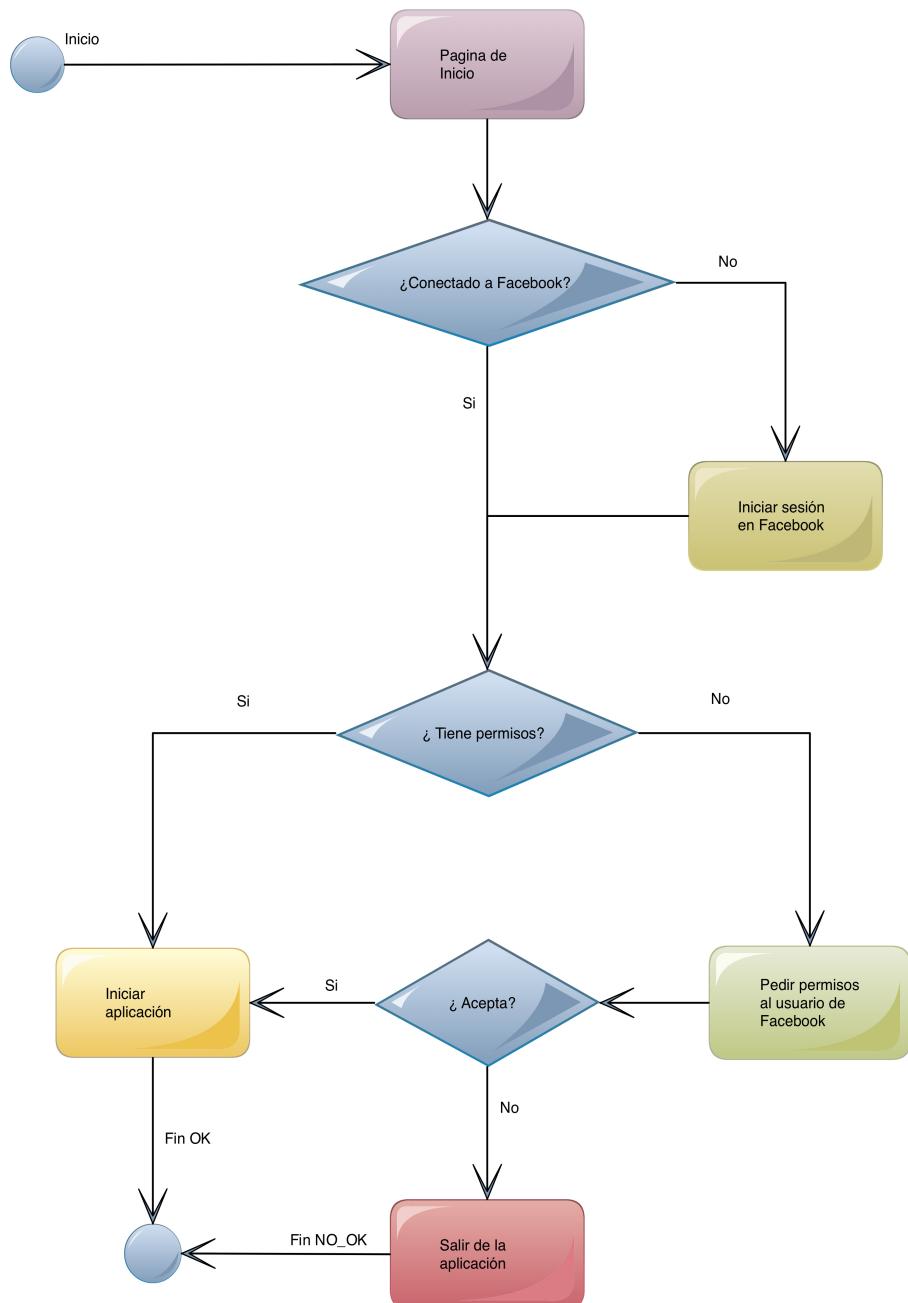


Figura 6.11: Diagrama de operación de login

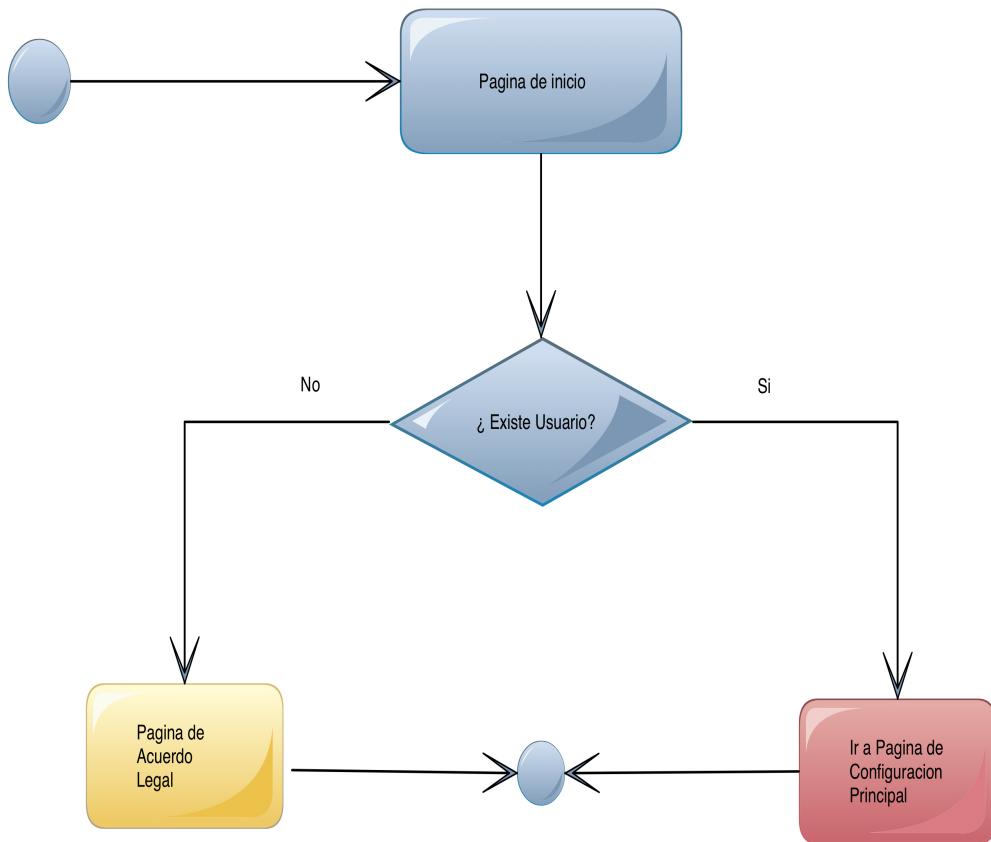


Figura 6.12: Diagrama de operación del inicio de la aplicación

6.5.2. Comprobación en la página de inicio

Una vez logueado en Facebook y aceptado los permisos necesarios, la página de inicio realizará las siguientes acciones:

1. Comprobará si el usuario **está dado de alta en la aplicación**: Para ello, consultará por medio del identificador de usuario que Facebook ha proporcionado al usuario y que ha devuelto a la aplicación si éste existe ya en la tabla `user_settings`.
 - a) Si éste existe y está dado de alta en la aplicación, el módulo servidor nos redirecciónará a la página de configuración principal de la aplicación.
 - b) Si el usuario no está dado de alta en la aplicación, el módulo servidor nos redirecciónará a la página de aceptación del acuerdo legal.

Estas operaciones pueden verse en el diagrama de actividad 6.12.

6.5.3. Aceptación del acuerdo legal

En ésta operación se expondrá al usuario la página del acuerdo legal para que sea aceptado. Esta página es de vital importancia, ya que si éste acuerdo no está aceptado, la aplicación no podrá descargar datos, por lo que no podrá realizar su cometido. Notesé que hasta éste momento, **la aplicación no ha almacenado dato alguno del usuario**, será en el momento de la aceptación cuando comenzará a descargar sus datos y a persistirlos en base de datos. Las operaciones que se realizarán serán las siguientes:

1. Se mostrará la pantalla de acuerdo legal, contenido en el fichero **legalAccepted.jsp**.
2. Tras leer detenidamente el acuerdo, el usuario podrá aceptarlo o cancelarlo:
 - a) En el caso de cancelarlo, la aplicación no realizará ninguna acción. Saldrá del programa, volviendo a la página principal de Facebook.
3. En el caso de aceptarlo, la aplicación realizará las siguientes acciones:
 - a) Obtendrá de Facebook todos los datos personales correspondientes al perfil, es decir, obtendrá sus datos personales, sus comentarios en el muro y los amigos que tiene por contactos.
 - b) Tras haber obtenido esta información, la persistirá en base de datos como un nuevo usuario de la aplicación, inicializando las estructuras de ésta que sean oportunas.
 - c) Tras haber realizado éste proceso, el módulo servidor redireccionará a la aplicación a la página principal de configuración.

Estas operaciones pueden verse en el diagrama de actividad 6.13.

6.5.4. Guardar datos

Esta operativa almacenará todos los datos de configuración de la aplicación. Esta acción se desencadenará a partir del pulsado del botón **Guardar**. En ése momento, el módulo servidor redireccionará la aplicación a la página **checkNow.jsp**. Es en ésta en la que se realizarán las siguientes acciones:

1. **Guardar la configuración del usuario:** Lo primero que se realizará es almacenar la configuración de filtros y de ángeles que el usuario ha elegido en la página de configuración, esto es, filtros elegidos para su ejecución, frecuencia de notificaciones para cada filtro y angeles seleccionados para cada filtro. Si es la primera vez que se guarda información de éste tipo, al ser un usuario nuevo, el módulo servidor enviará un correo de confirmación a cada ángel para que pueda aceptar el tutelado de la información de éste. Este proceso sólo se realizará en el caso de que el usuario seleccione nuevos ángeles o borre alguno que tuviera seleccionado, ya que en ésta parte no se produce verificación alguna de los datos del usuario.

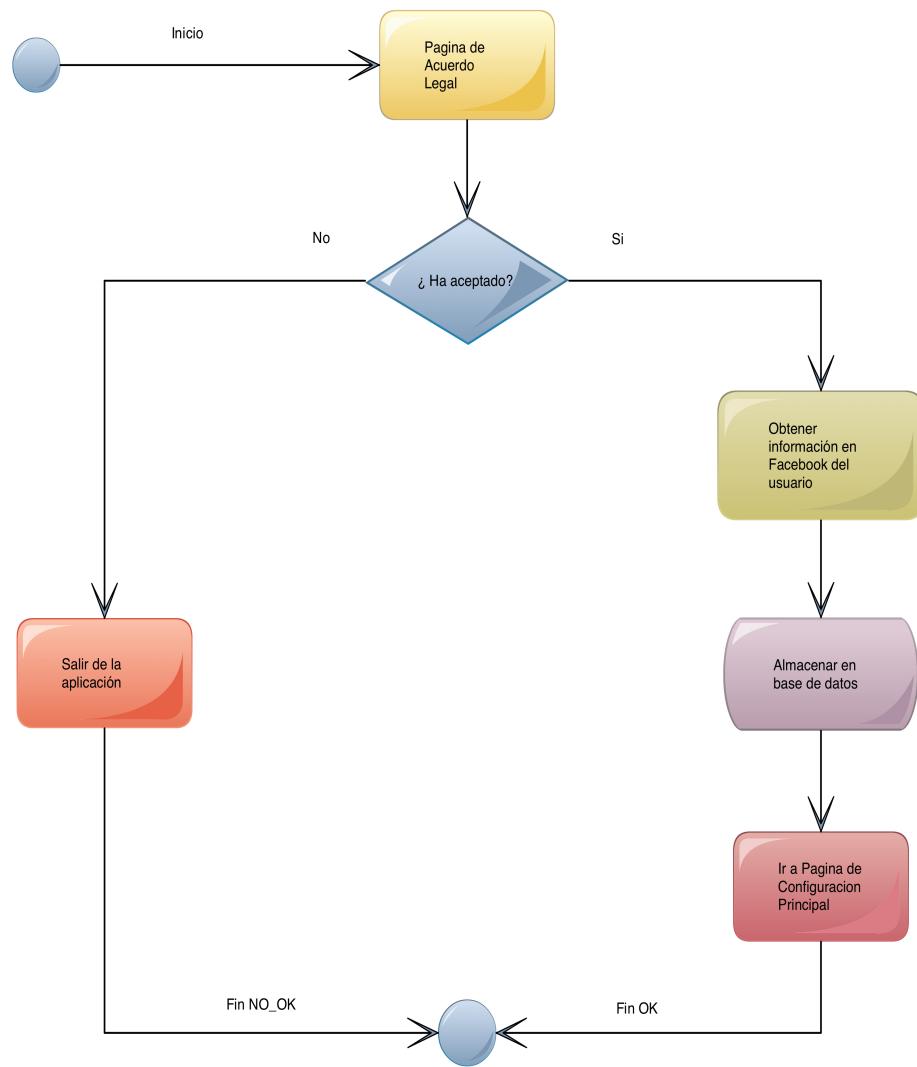


Figura 6.13: Diagrama de operación de aceptación del acuerdo legal

2. **Actualizar los datos de Facebook y almacenarlos en base de datos:** En éste paso, se actualizará toda la información existente en base de datos añadiendo la posible nueva información que haya podido producirse desde el último chequeo de información o la última vez que el usuario accedió a la aplicación.
3. **Vuelta a la página de configuración:** Si todo ha salido correctamente, se volverá a la página de configuración principal de la aplicación y se mostrará un mensaje OK. Si, por el contrario, en alguno de los pasos anteriores se ha producido un error, se lanzará una excepción que, al tratarla, el módulo servidor redireccionará a la aplicación hasta una pantalla de error con el detalle de la excepción producida.

Estas operaciones pueden verse en el diagrama de actividad 6.14.

6.5.5. Confirmación de un ángel

Esta operación se desencadenará cada vez que un usuario de la aplicación elija un nuevo ángel. Este deberá confirmar la solicitud para empezar a recibir informes de la actividad social del usuario. El enlace que se le enviará le conectará directamente con la página **angelConfirmation.jsp**, la cual realizará toda la lógica de la operación. Ésta se llevará a cabo de la siguiente manera:

1. **Envío de correo de confirmación a un ángel:** Cada vez que se detecte a un nuevo ángel, la aplicación le enviará un correo para pedirle confirmación de si puede o no enviarle notificaciones de información.
 - a) En caso de no aceptar el envío de notificaciones, se eliminará toda la información en base de datos del ángel y se mostrará un mensaje de confirmación por pantalla.
2. En caso de aceptación, **se almacenará en su información la confirmación**, para posteriormente, enviarle notificaciones sin ninguna restricción.
3. Una vez guardada ésta información, se procederá a realizar el primer chequeo de información con los filtros que el usuario haya elegido.
4. Una vez terminados los chequeos, se procederá al **envío de la notificación** con el resultado del chequeo. A partir de éste momento, al ángel le llegarán notificaciones con la frecuencia que el usuario haya elegido para los filtros en los que se encuentre definido como ángel.
5. Se mostrará **un mensaje de confirmación por pantalla** en caso de que todo haya salido correctamente. Si se ha producido algún error en el proceso, se mostrará un mensaje de error y se saldrá de la operación.

Estas operaciones pueden verse en el diagrama de actividad 6.15.

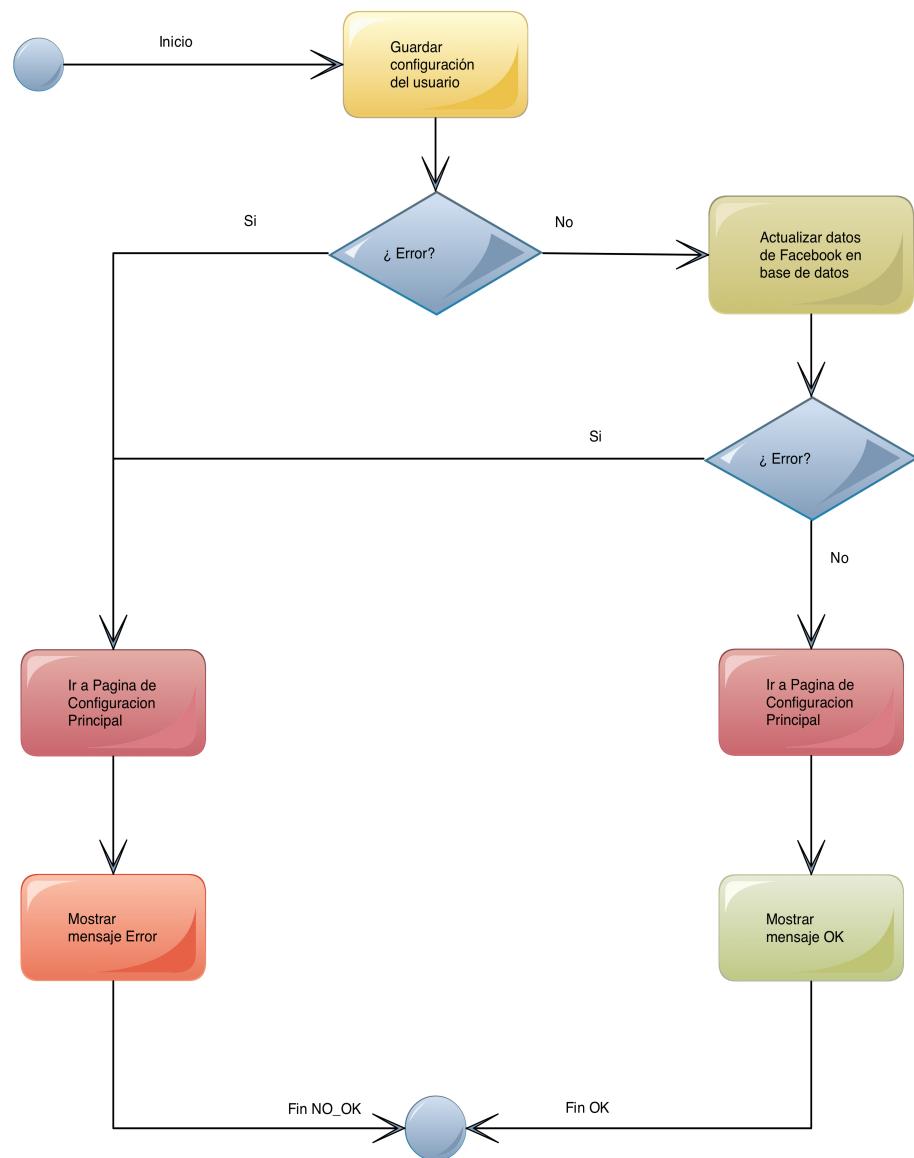


Figura 6.14: Diagrama de operación de guardado datos

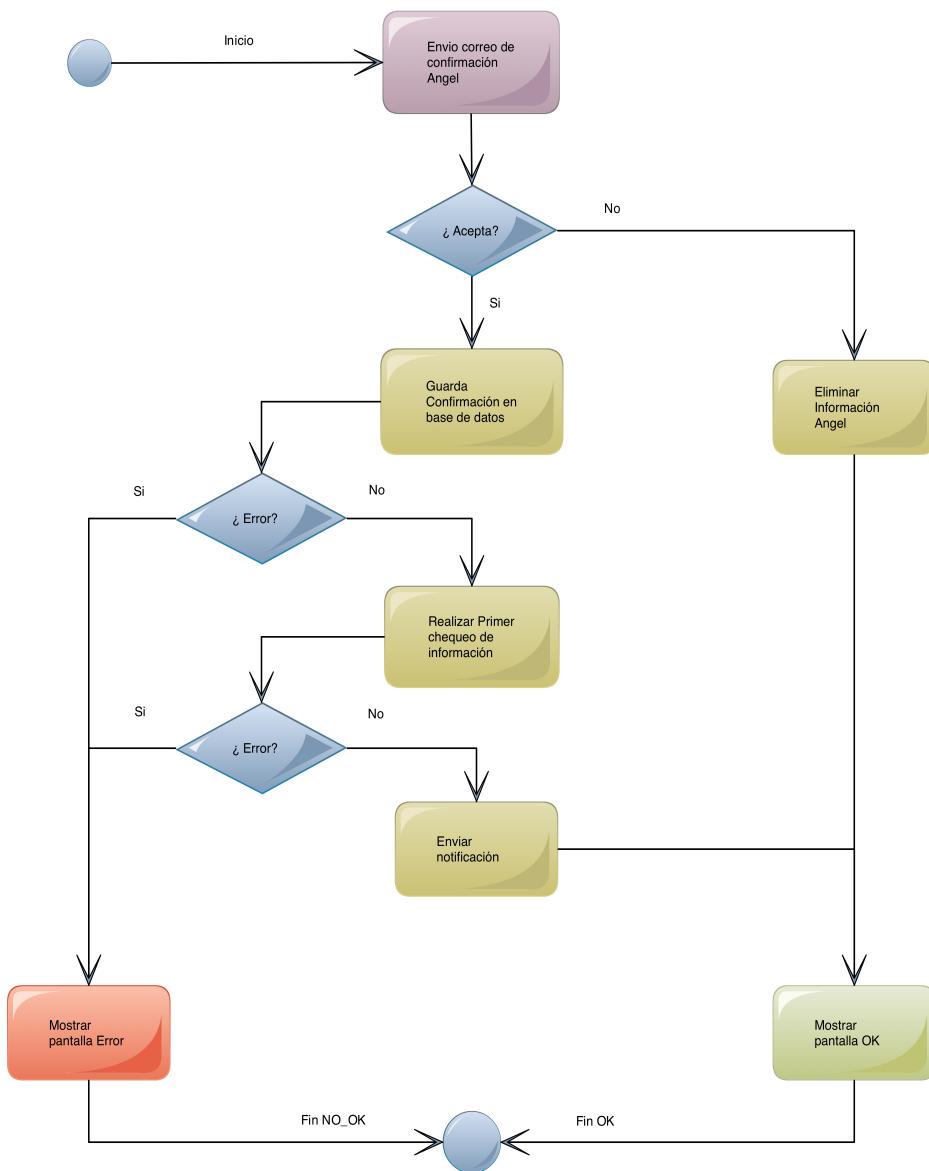


Figura 6.15: Diagrama de operación de la confirmación de un ángel

Capítulo 7

Módulo Cliente

7.1. Introducción

El módulo cliente será el encargado de la interfaz gráfica por el que el usuario realizará la configuración de la aplicación, seleccionando los ángeles que supervisarán su actividad social y los filtros que analizarán su información y enviarán informes de ello a dichos ángeles. Los objetivos de éste módulo serán los siguientes:

1. Mostrar al usuario una interfaz amigable y atractiva.
2. Integrar dicha interfaz en Facebook como una aplicación más que se ejecute desde el perfil de usuario.
3. Mostrar todos y cada uno de los amigos de Facebook para que puedan ser seleccionados por el usuario como sus ángeles.
4. Habilitar accesos desde el interfaz para poder seleccionar contactos desde cuentas de correo Gmail.
5. Habilitar las estructuras básicas para poder introducir manualmente un ángel, mediante cuadros de texto para su nombre y su dirección de correo.
6. Mostrar y controlar todos los filtros disponibles que puedan ser configurados por el usuario, mostrando además una pequeña descripción relativa a los mismos.

A parte de éstos objetivos, el módulo cliente se enfrentará a las siguientes responsabilidades:

1. Mostrar correctamente todas las estructuras necesarias para el correcto funcionamiento de la aplicación.
2. Llevar a cabo todas las validaciones de datos necesarias para que al almacenar la información de configuración, los datos estén preparados para ser procesados por el módulo servidor.



Figura 7.1: Estructura de organización del Módulo Cliente

3. Controlar, en todo momento, que el usuario actual sea el único que puede acceder a su propia información de configuración.
4. Habilitar estructuras para el almacenado de la información.
5. Habilitar estructuras que informen visualmente al usuario sobre posibles errores o advertencias en el proceso de configuración.
6. Habilitar estructuras visuales de ayuda que sirvan para aclarar posibles dudas a la hora de guardar la configuración.

Todo este módulo será conformado por todas las herramientas web disponibles en la actualidad, tales como HTLM, AJAX, jQuery y el formato JSON para la serialización de datos, todo ello encapsulado en las clases JSP que se comunicarán con el servidor y pintarán todo el código embebido en ellas como parte cliente.

7.2. Organización del módulo

El módulo está encuadrado dentro de la aplicación **SNSAngelGuardFB**, dentro de la carpeta **Web Pages**, como muestra la imagen 7.1. Todas las estructuras pasarán a explicarse a continuación.

7.2.1. Hojas de estilos

Las hojas de estilo CSS¹ describirán la presentación semántica de las pantallas de la aplicación **SNSAngelGuardFB**. Estarán contenidas en la carpeta **Styles** y las más importantes son las siguientes:

1. **Style.css**: Contendrá toda la estructura de representación de la parte contenedora del menú **AJAX** necesario para cargar las pestañas que permitirán pasar de una pantalla a otra sin cargar la pantalla contenedora de la aplicación.
2. **facebook.css**: Será la que determine el aspecto general de toda la aplicación. Ha sido completamente desarrollado en su totalidad para la aplicación. Todas las pantallas de la parte cliente llevarán encapsulado en su código HTML ésta hoja de estilo, que tendrá todas las estructuras definidas para pintar cada una de las pantallas.
3. **fcblistselection.css**: Será la hoja de estilo que utilizarán todos los contenedores que tenga que pintar usuarios con fotografía y nombre. Se utilizará principalmente en los contenedores de selección de contactos de Facebook, dentro de la página de selección de ángeles.
4. **fcblistselectionVig.css**: Será la hoja de estilo que utilizarán todos los contenedores que tengan que pintar usuarios con fotografía y nombre en el área de los vigilantes. Guarda la misma estructura que la hoja **fcblistselection.css** pero con algunos métodos propios del control de vigilantes.
5. **jquery.loader.css**: Será la hoja de estilo que marque la presentación de las pantallas de espera y precarga utilizadas en la aplicación, para controlar su flujo entre transiciones de pantallas.
6. **styleSwitchVig.css**: Será la hoja de estilo que pinte los botones de activación/desactivación de los vigilantes.

7.2.2. Imágenes e iconos

Todas los iconos que se utilicen en la aplicación se almacenarán en dos carpetas determinadas:

1. Carpeta **images**: Contendrá los iconos necesarios para pintar los listados de usuarios utilizados en la hoja de estilo **fcblistselection.css**.
2. Carpeta **resources**: Contendrá el resto de iconos de la aplicación. Todos los iconos diseñados específicamente para la aplicación y que no pertenecen a ningún plugin externo se ubicarán en ésta carpeta.

¹Las hojas de estilo en cascada o (Cascading Style Sheets, o sus siglas CSS) hacen referencia a un lenguaje de hojas de estilos usado para describir la presentación semántica (el aspecto y formato) de un documento escrito en lenguaje de marcas. Su aplicación más común es dar estilo a páginas webs escritas en lenguaje HTML y XHTML, pero también puede ser aplicado a cualquier tipo de documentos XML, incluyendo SVG y XUL.

7.2.3. Ficheros de contenido JavaScript

Toda la funcionalidad del módulo cliente depende en gran medida de JavaScript. La carga de pestañas, realizada en **AJAX**, los diferentes pluggins de visualización, desarrollados en **jQuery** y las herramientas de carga entre transiciones de pantallas están desarrollados en lenguaje JavaScript. La carpeta **js** contendrá todas estas estructuras que pasaremos a detallar a continuación.

1. Archivo **AjaxFlagMenu-1.0.2.min.js**: Contendrá todas las estructuras necesarias para controlar la funcionalidad de la parte de carga entre pestañas de la pantalla principal de la aplicación. Estará sustentado en la hoja de estilo **Style.css**.
2. Archivo **fcbklistselection.js**: Será el encargado de representar el listado de amigos de Facebook en la pantalla de selección de ángeles. Formará tandem con la hoja de estilo **fcbklistselection.css**.
3. Archivo **fcbklistselectionVig.js**: Controlará la ejecución del listado de ángeles para un determinado filtro en la pantalla de selección de vigilantes. Se sustentará también en la hoja de estilo **fcbklistselection.css**.
4. Archivo **jQuery-1.8.2.js**: Contendrá todas las instrucciones para utilizar la tecnología **jQuery**. Todas las pantallas de la aplicación contendrán éste archivo embebido en su código.
5. Archivo **jquery-ui-1.8.12.custom.min.js**: Contendrá aquellas instrucciones pertenecientes al juego de instrucciones de **jQuery** específicas para el control de la parte visual de la aplicación.
6. Archivo **jquery.loader.js**: Será el encargado de realizar la funcionalidad de la pantalla de transición entre la aplicación. Será quien cargue el **loader** visual.
7. Archivo **utilidadesForumularios.js**: Será el fichero que controle todo el flujo de datos de la aplicación. Está desarrollado en **jQuery** y **JavaScript** nativo. Controlará todas las estructuras específicamente desarrolladas para la aplicación.
8. Archivo **settingsSNSAngelGuard_Vigilants.js**: Será el fichero que se encargue de controlar todo el flujo de datos en la página de selección de vigilantes.

7.2.4. Ficheros de vocabulario

Estos ficheros serán aquellos que controlen el lenguaje malintencionado del filtro de control del lenguaje. Existirá uno por cada lenguaje en el que esté traducida la aplicación. Se encontrarán en la carpeta **lexicalFiles** y actualmente hay dos:

1. Fichero **badWords_en.txt**: Contendrá todo el vocabulario malintencionado para el idioma inglés.
2. Fichero **badWords_es.txt**: Contendrá todo el vocabulario malintencionado para el idioma castellano.

7.3. Pantallas de la aplicación SNSAngelGuardFB

En ésta sección se explicarán visual y técnicamente, las pantallas que formarán parte de la navegación de la aplicación **SNSAngelGuardFB**. La sección estará divida en dos subsecciones:

1. **Pantallas Principales:** Serán aquellas indispensables para la ejecución de la información. En ellas se producirán flujos de información y se representarán todos los datos necesarios para que cada usuario pueda configurar la aplicación con total normalidad.
2. **Pantallas Secundarias:** Serán aquellas que informen de eventualidades, tanto a los usuario de la aplicación como a sus ángeles. A ésta división pertenecerán las pantallas que avisan al usuario del resultado de la aplicación o al ángel cuando confirma la solicitud del usuario.

7.3.1. Pantallas Principales

Serán aquellas en las que el flujo de información que se traslada entre ellas será de vital importancia para la aplicación. Darán lugar a la navegación de la aplicación y serán las responsables de presentar sus datos para que el usuario pueda configurar su aplicación. Forman parte de ésta división las siguientes pantallas:

1. Pantalla de **Acuerdo Legal:** Será la pantalla que se presentará al usuario de la aplicación para que acepte los términos legales de la aplicación.
2. Pantalla de **Configuración General:** Será la pantalla principal de la aplicación. Contendrá las pestañas que darán acceso a las páginas de **Selección de Ángeles** y **Configuración de Vigilantes**.
3. Pantalla de **Selección de Ángeles:** Será la pantalla que representará todos los datos para que un usuario pueda seleccionar todos los contactos que desee como ángeles.
4. Pantalla de **Configuración de Vigilantes:** Será la pantalla que representará todas las estructuras y datos necesarios para que un usuario pueda configurar los vigilantes, es decir, los filtros que se ejecutarán con su información.
5. Pantalla de **Selección de contactos de Google:** Será la pantalla que representará a todos los contactos de una cuenta de **Gmail** para que el usuario pueda seleccionar de entre ellos a sus ángeles.

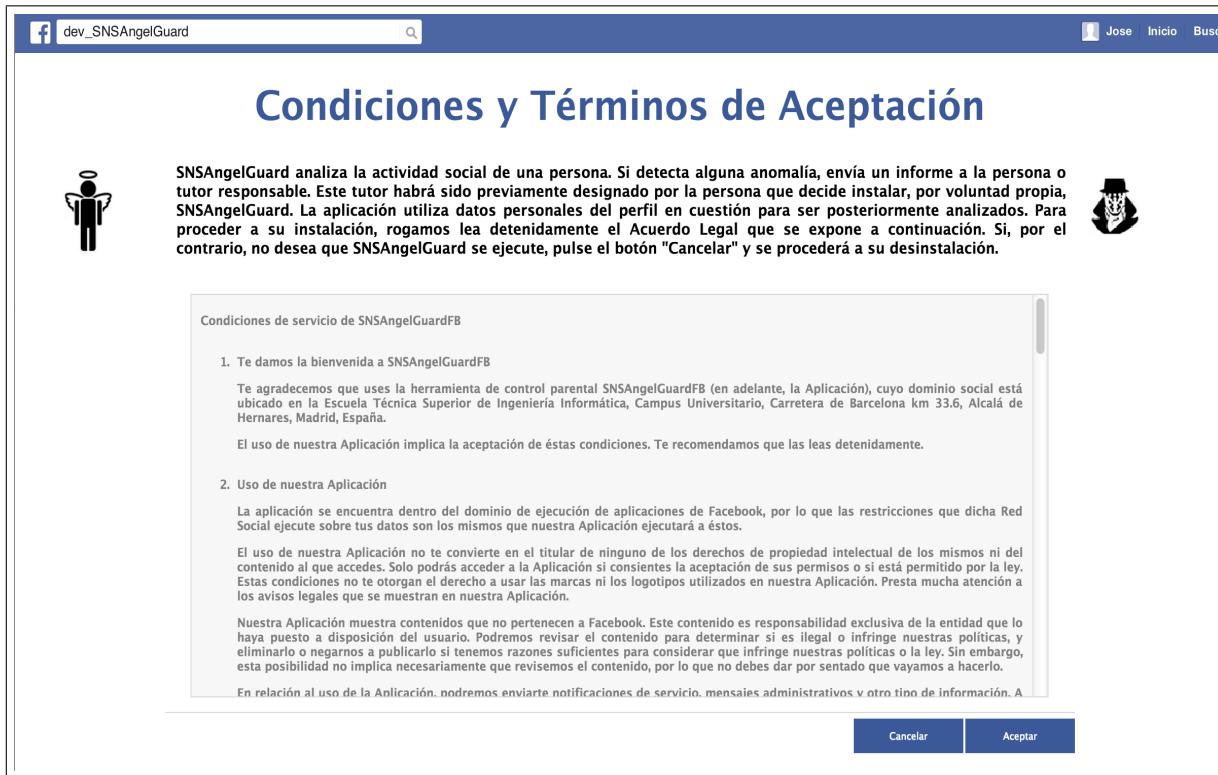


Figura 7.2: Pantalla de Acuerdo Legal

7.3.1.1. Pantalla de Acuerdo Legal

Será la pantalla que se presentará al usuario de la aplicación para que acepte los términos legales de la aplicación. Estará representada en la imagen A.1. Esta pantalla está representada por el archivo **legalAccepted.jsp** y tendrá asociados a su ejecución los siguientes recursos:

1. Hoja de estilo **facebook.css**: Hoja de estilo definida para todas las pantallas de la aplicación.
2. Hoja de estilo **jquery.loader.css**: Controlará la representación del objeto loader de carga entre pantallas.
3. Fichero JavaScript **jquery.js**: Fichero que carga la funcionalidad básica para utilizar la tecnología **jQuery**.
4. Fichero JavaScript **utilidadesFormularios.js**: Fichero que contiene todas las operaciones necesarias para la parte cliente de la aplicación.
5. Fichero JavaScript **jquery.loader.js**: Fichero que controla la acción del objeto loader de carga entre pantallas.

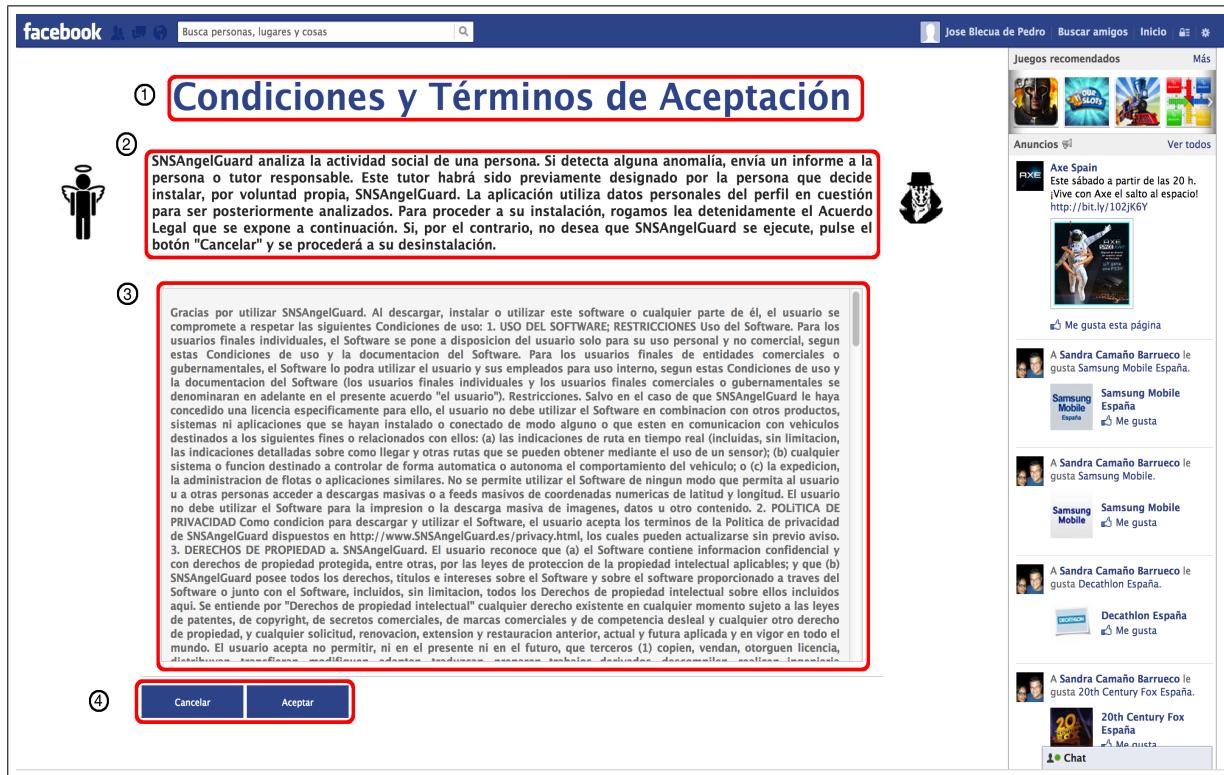


Figura 7.3: Pantalla de Acuerdo Legal, por partes

En la imagen 7.3 pueden verse desglosadas las partes de la pantalla, que serán las siguientes:

- 1. Título de la página**
- 2. Texto informativo del Acuerdo Legal:** En este contenedor se mostrará una pequeña información sobre la actividad de la aplicación y el acuerdo legal que se debe aceptar.
- 3. Acuerdo Legal:** En éste contenedor se mostrará la información relativa al acuerdo legal principal de la aplicación.
- 4. Botonera inferior:** Contendrá los siguientes botones:
 - a) Botón **Cancelar**: Mostrará este botón, el cual, al ser pulsado, saldrá de la aplicación y volverá a la página principal de Facebook.
 - b) Botón **Aceptar**: Mostrará este botón que, al ser pulsado, guardará la información del usuario en Facebook y pasará a la página de configuración principal de la aplicación.

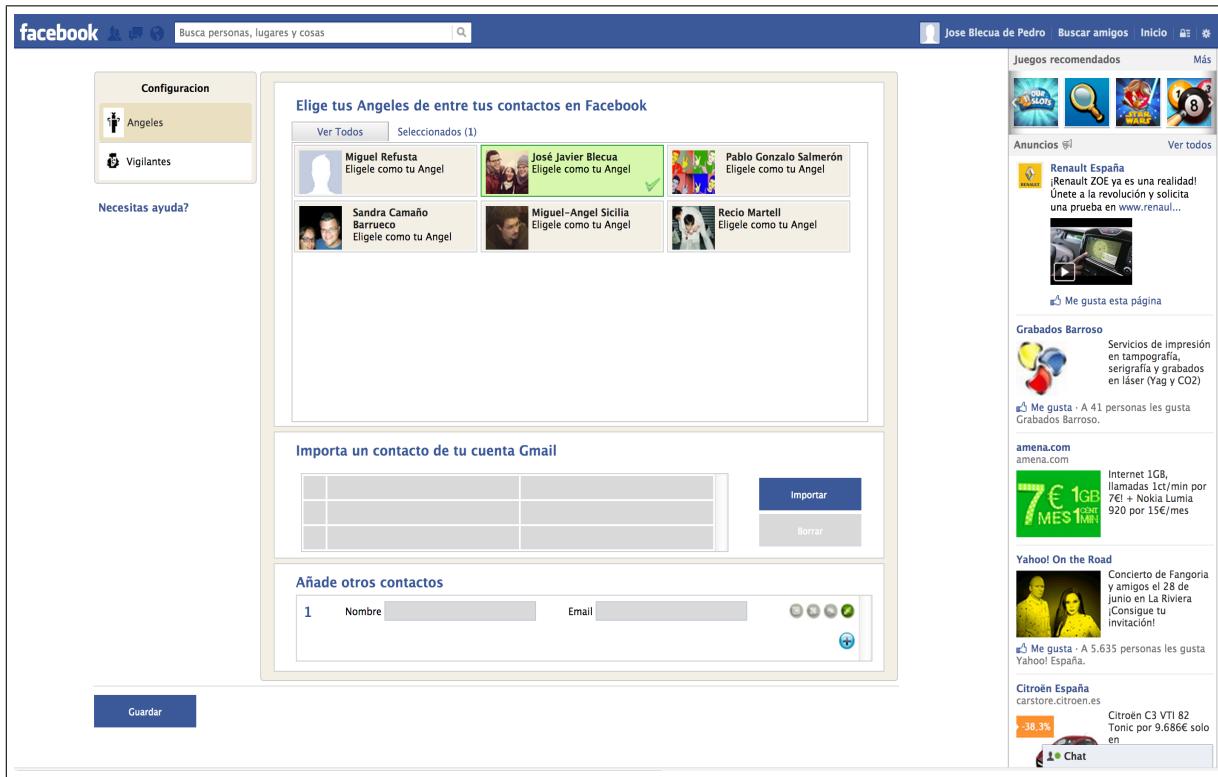


Figura 7.4: Pantalla de Configuración General

7.3.1.2. Pantalla de Configuración General

Esta pantalla será la pantalla principal de la aplicación y está representada en la pantalla 7.4. Desde ella se proporcionará el acceso a las páginas de **Selección de Ángeles** y **Configuración de Vigilantes**. Será la que mostrará el botón de guardado de la información seleccionada y, por tanto, será, en última instancia, la que almacene la información obtenida de las dos anteriores pantallas. Esta pantalla está representada por el archivo **settingsSNSAngelGuard.jsp** y tiene incluidos los siguientes ficheros:

1. Hojas de estilos: Contendrán los siguientes ficheros:
 - a) Archivo **Style.css**: Hoja de estilo definida para todas las pantallas de la aplicación.
 - b) Archivo **facebook.css**: Hoja de estilo definida para todas las pantallas de la aplicación.
 - c) Archivo **jquery.loader.css**: Controlará la representación del objeto loader de carga entre pantallas.
2. Archivos **JavaScript**, que serán los siguientes:
 - a) Fichero **jquery.js**: Fichero que carga la funcionalidad básica para utilizar la tecnología **jQuery**.

7.3. PANTALLAS DE LA APLICACIÓN SNSANGELGUARDFB

195

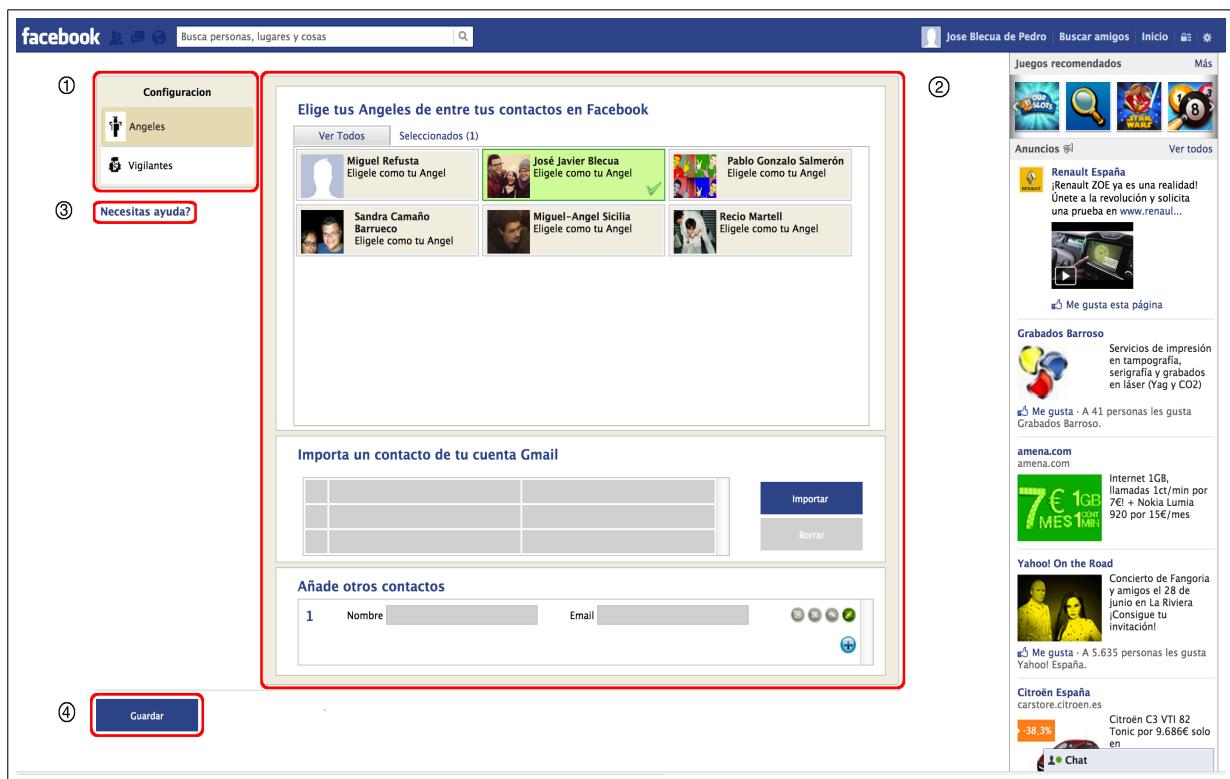


Figura 7.5: Pantalla de Configuración General, por partes

- b) Ficheros **jquery.corner.js** y **AjaxFlagMenu-1.0.2.min.js**: Controlará el contenedor **Ajax** que representará a las pestañas que darán acceso a las páginas de ángeles y vigilantes.
- c) Fichero **utilidadesFormularios.js**: Fichero que contiene todas las operaciones necesarias para la parte cliente de la aplicación.
- d) Fichero **jquery.loader.js**: Fichero que controla la acción del objeto loader de carga entre pantallas.

En la imagen 7.5 pueden verse desglosadas las partes de la pantalla, que serán las siguientes:

1. **Menú de selección de página**: En ésta parte se representará el menú donde estarán las pestañas de selección de página, tanto para ángeles como para vigilantes.
2. **Contenedor de página**: En éste contenedor se cargarán las páginas que muestre el menú de selección.
3. **Ayuda**: Mostrará el enlace a la ayuda de la aplicación.
4. **Botonera inferior**: Contendrá el botón **Guardar**, el cual almacenará toda la información seleccionada en la página.



Figura 7.6: Pantalla de Selección de Ángeles

7.3.1.3. Pantalla de Selección de Ángeles

Esta será la que represente a todos los ángeles que el usuario puede seleccionar. Está representada por la imagen 7.6. En ella se podrán almacenar ángeles de Facebook, Google y cualquier otra cuenta de correo electrónico. Esta pantalla está representada por el archivo **settingsSNSAngelGuard_Angels.jsp** y tiene incluidos los siguientes ficheros:

1. Hojas de estilos: Contendrán los siguientes ficheros:
 - a) Archivo **Style.css**: Hoja de estilo definida para todas las pantallas de la aplicación.
 - b) Archivo **fcbklistselection.css**: Hoja de estilo definida para mostrar, bajo un formato de nombre y fotografía, a todos los contactos de Facebook de un usuario.
 - c) Archivo **facebook.css**: Hoja de estilo definida para todas las pantallas de la aplicación.
 - d) Archivo **jquery.loader.css**: Controlará la representación del objeto loader de carga entre pantallas.
2. Archivos **JavaScript**, que serán los siguientes:
 - a) Fichero **jquery.js**: Fichero que carga la funcionalidad básica para utilizar la tecnología **jQuery**.

- b) Fichero **fcbklistselection.js**: Controlará el contenedor de selección de contactos de Facebook.
- c) Fichero **utilidadesFormularios.js**: Fichero que contiene todas las operaciones necesarias para la parte cliente de la aplicación.
- d) Fichero **jquery.loader.js**: Fichero que controla la acción del objeto loader de carga entre pantallas.

En la imagen 7.7 pueden verse desglosadas las partes de la pantalla, que serán las siguientes:

1. **Contenedor de contactos de Facebook**: En éste contenedor aparecerán todos los contactos de Facebook. Contendrá dos pestañas:
 - a) Pestaña **Ver Todos**: Esta pestaña contendrá todos los contactos de Facebook del usuario.
 - b) Pestaña **Seleccionados**: Esta pestaña contendrá, en el mismo formato que la anterior, únicamente aquellos contactos de Facebook que hayan sido seleccionados como ángeles del usuario. En el título de la pantalla puede verse entre paréntesis cuantos de ellos han sido seleccionados.
 - c) Botón **Actualizar contactos de Facebook**: Este botón lanzará la actualización de los contactos de facebook y recargará la página con los nuevos contactos que no se hayan registrado en la aplicación.
2. **Contenedor de contactos de Gmail**: En éste contenedor se cargarán todos los ángeles seleccionados de una cuenta de **Gmail**. Está estructurado en las siguientes partes:
 - a) **Tabla de contactos**: En ésta aparecerán los ángeles de Gmail seleccionados.
 - b) Botón **Importar**: Al pulsar éste botón, aparecerá la pantalla de importación de contactos de Google, desde la cual se podrá acceder a todos los contactos de Gmail de un usuario y elegir aquellos que se quieran como ángeles.
 - c) Botón **Borrar**: Eliminará el/los contactos seleccionados de la tabla de contactos.
3. **Contenedor de otros contactos**: Mostrará aquellos contactos que no pertenecen ni a Facebook ni a Google. Serán introducidos manualmente por el usuario y sólo necesitará, como datos, su nombre y un email válido. Contendrá las siguientes estructuras:
 - a) **Número de contacto**: Se mostrará en la parte izquierda.
 - b) **Nombre**: Será un campo de texto que, por defecto, aparecerá deshabilitado.
 - c) **Email**: Será un campo de texto que, por defecto, aparecerá deshabilitado.
 - d) Botón **Guardar**: Almacenará un ángel tras introducir correctamente sus datos. Estará por defecto deshabilitado hasta que se introduzcan correctamente los datos de un ángel.



Figura 7.7: Pantalla de Selección de Ángeles, por partes

- e) Botón **Eliminar**: Eliminará un ángel anteriormente introducido. Aparecerá por defecto deshabilitado y sólo se deshabilitará al pulsar el botón **Editar**.
- f) Botón **Cancelar**: Limpiará y deshabilitará los campos. Aparecerá por defecto deshabilitado.
- g) Botón **Editar**: Habilitará los campos **Nombre** y **Email** para su edición. Por defecto aparecerá habilitado.
- h) Botón **Añadir Nuevo Angel**: Estará siempre habilitado y al ser pulsado añadirá la estructura siguiente para poder introducir un nuevo ángel.

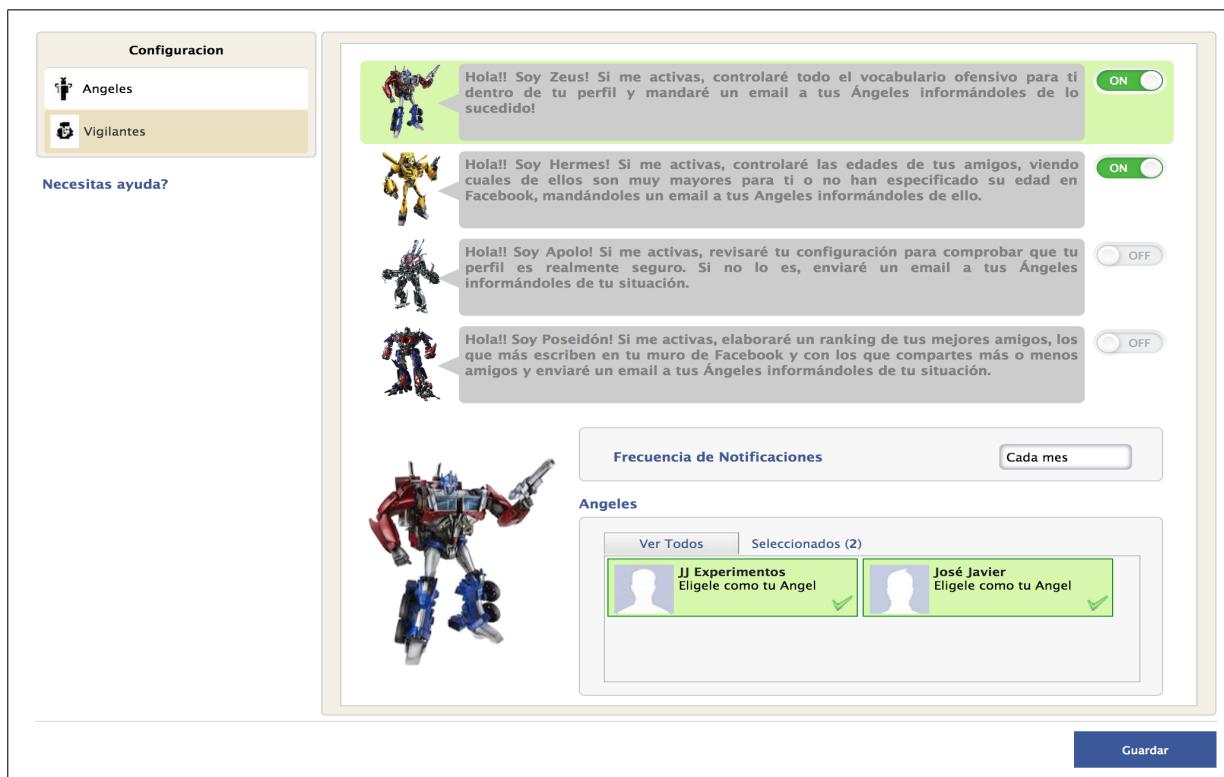


Figura 7.8: Pantalla de Configuración de Vigilantes

7.3.1.4. Pantalla de Configuración de Vigilantes

Esta pantalla será la que represente a todos los vigilantes disponibles en la aplicación. Está representada por la imagen 7.8. En ella se podrán configurar todos los vigilantes con su frecuencia de ejecución y los ángeles a los que se enviarán las notificaciones. Esta pantalla está representada por el archivo **settingsSNSAngelGuard_Vigilants.jsp** y tiene incluidos los siguientes ficheros:

1. Hojas de estilos: Contendrán los siguientes ficheros:
 - a) Archivo **fcbklistselectionVig.css**: Hoja de estilo definida para mostrar, bajo un formato de nombre y fotografía, a todos los ángeles de cada vigilante.
 - b) Archivo **facebook.css**: Hoja de estilo definida para todas las pantallas de la aplicación.
 - c) Archivo **jquery.loader.css**: Controlará la representación del objeto loader de carga entre pantallas.
 - d) Archivo **jquery.loader.css**: Controlará la representación del objeto loader de carga entre pantallas.
 - e) Archivo **styleSwitchVig.css**: Controlará la representación del botón de activado/desactivado de un vigilante.

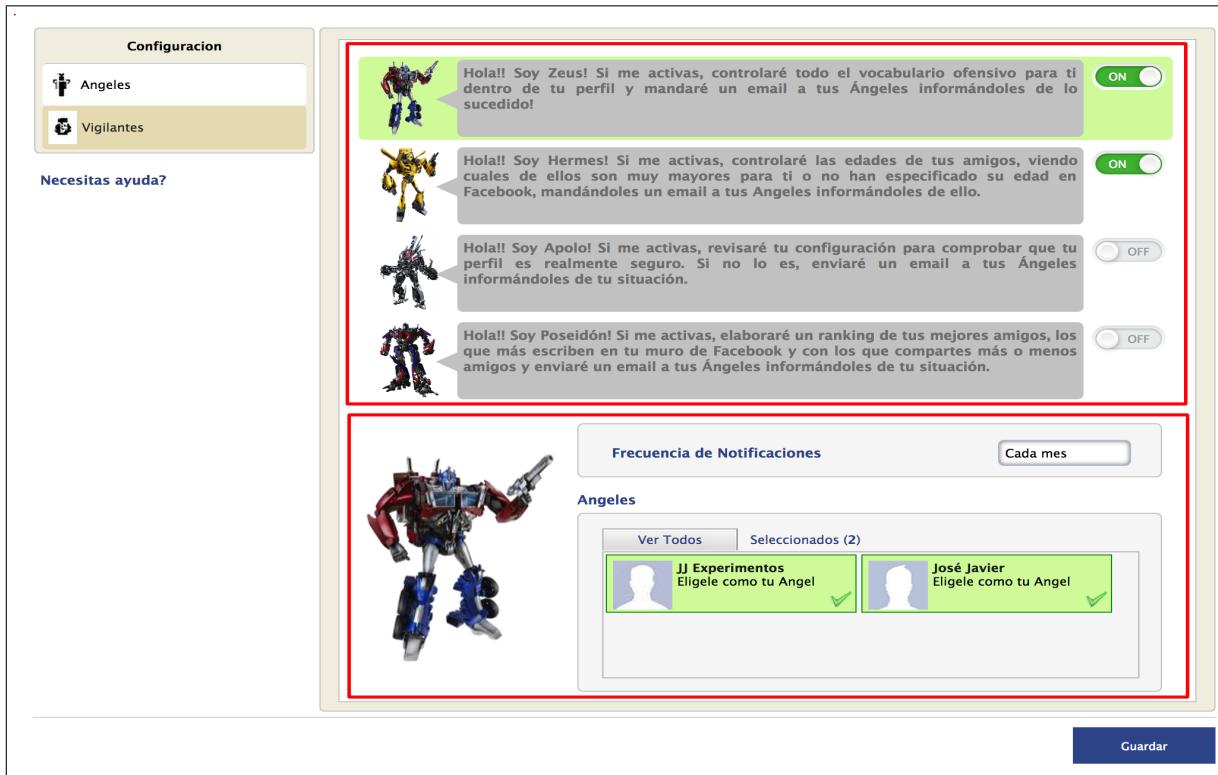


Figura 7.9: Pantalla de Configuración de Vigilantes, por partes

2. Archivos **JavaScript**, que serán los siguientes:

- Fichero **jquery.js**: Fichero que carga la funcionalidad básica para utilizar la tecnología **jQuery**.
- Fichero **fcbklistselectionVig.js**: Controlará el contenedor de selección de ángeles para cada vigilante.
- Fichero **utilidadesFormularios.js**: Fichero que contiene todas las operaciones comunes a la aplicación.
- Fichero **settingsSNSAngelGuard_Vigilants.js**: Fichero que controla todas las acciones de la página.

En la imagen 7.9 pueden verse desglosadas las partes de la pantalla, que serán las siguientes:

- Contenedor de filtros disponibles**: Contendrá una serie de contenedores con todos los filtros que estén disponibles en la aplicación. Este contenedor tiene un tamaño fijo dentro de la página, pero internamente está controlado por un scroll invisible que hace que se puedan cargar más filtros de los que se pueden ver. Cada filtro contendrá un switch de activación que indicará si el filtro está o no activo. Cada filtro contendrá una imagen en miniatura, su descripción y el switch.

2. **Contenedor de configuración del filtro:** Este contenedor solo se activará si el filtro está activo. Contendrá una imagen a tamaño grande del filtro, el combo de selección de frecuencia y la lista de ángeles seleccionados para el filtro.



Figura 7.10: Pantalla de Selección de contactos de Google

7.3.1.5. Pantalla de Selección de contactos de Google

Esta pantalla será la que represente a todos los contactos de **Gmail** de una cuenta válida. Está representada por la imagen 7.10. En ella se podrá importar todos los contactos de una cuenta de **Gmail** válida y seleccionar los contactos deseados como ángeles de la aplicación. Estos ángeles se cargarán en la pantalla de **Selección de Ángeles** en el contenedor de los contactos de Google. Esta pantalla está representada por el archivo **modalContacts.jsp** y tiene incluidos los siguientes ficheros:

1. Hojas de estilos: Contendrán los siguientes ficheros:
 - a) Archivo **facebook.css**: Hoja de estilo definida para todas las pantallas de la aplicación.
2. Archivos **JavaScript**, que serán los siguientes:
 - a) Fichero **jQuery-1.8.2.js**: Fichero que carga la funcionalidad básica para utilizar la tecnología **jQuery**.
 - b) Fichero **utilidadesFormularios.js**: Fichero que contiene todas las operaciones necesarias para la parte cliente de la aplicación.
 - c) Fichero **https://apis.google.com/js/client.js**: Fichero que contendrá toda la funcionalidad del API Google Contacts¹, necesario para importar los contactos de Google.

¹Google Contacts API version 3.0, permite a cualquier aplicación Web acceder y modificar los contactos de una cuenta Google. Los contactos de Google serán actualizados automáticamente por medio de una aplicación cualquiera a partir de esta API.

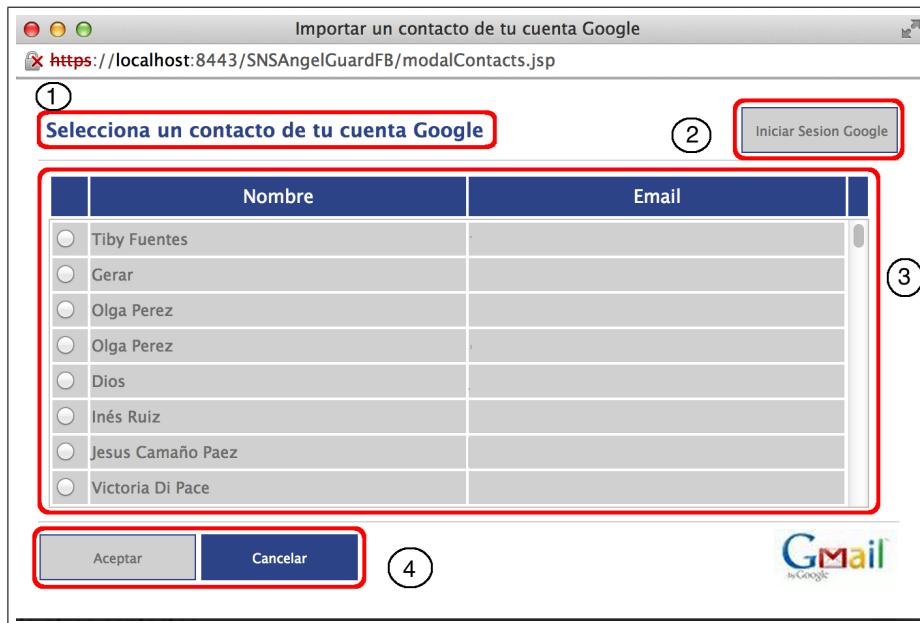


Figura 7.11: Pantalla de Selección de contactos de Google, por partes

En la imagen 7.11 pueden verse desglosadas las partes de la pantalla, que serán las siguientes:

1. **Título de la página.**
2. Botón **Iniciar Sesión Google**: Al pulsar éste botón, se abrirá una pantalla de conexión perteneciente a Google en la que se iniciará sesión en Google y ésta devolverá todos los contactos de la cuenta.
3. **Tabla de contactos**: Contendrá tres columnas:
 - a) Columna **Radiobutton de selección**: Al pulsar éste, se seleccionará directamente el contacto referenciado.
 - b) Columna **Nombre**: Contendrá el nombre del contacto.
 - c) Columna **Email**: Contendrá el email del contacto. La actual imagen del presente documento se ha modificado en ésta columna para no mostrar datos de terceros.
4. **Botonera inferior**: Contendrá dos botones:
 - a) Botón **Aceptar**: Al pulsarlo, se cerrará la ventana y se volverá a la selección de ángeles con los contactos que se hayan seleccionado.
 - b) Botón **Cancelar**: Al pulsarlo, no se seleccionará ningún contacto y se cerrará la ventana, volviendo a la selección de ángeles.

7.3.2. Pantallas Secundarias

Serán aquellas cuyo propósito sea meramente informativo. Informarán al usuario o a sus ángeles del resultado de sus operaciones, o mostrarán los errores que puedan ocurrir en los procesos de la aplicación. Estas pantallas serán las siguientes:

1. Pantalla de **Ayuda**: Informará, mediante pequeñas explicaciones, las entidades que forman la aplicación y su utilidad.
2. Pantalla de **Resultado a la confirmación de un ángel**: Informará del resultado de la confirmación de un usuario.
3. Pantalla de **Resultado de la operación**: Informará el resultado del proceso de guardado de la configuración en la pantalla general de la aplicación.
4. Pantalla de **Información de Error**: Informará al usuario del error que se ha producido en el proceso.
5. Pantalla de **Identificación del usuario**: Mostrará información del usuario que ha enviado la solicitud a un ángel.



Figura 7.12: Pantalla de Ayuda

7.3.2.1. Pantala de Ayuda

Mostrará una pequeña información de las funcionalidades de la aplicación. Será una pantalla modal que se mostrará tras pulsar el link de la pantalla de Selección de Ángeles **Necesitas Ayuda?**, que estará implementada por la página **helpMe.jsp**. La información que mostrará será la siguiente:

1. **Información referente a un ángel:** Informará sobre qué es un ángel y la funcionalidad que éste tiene dentro de la aplicación.
2. **Información referente a un vigilante:** Al igual que en el apartado anterior, mostrará información sobre qué es un vigilante y los tipos que nos podemos encontrar en la aplicación.
3. **Información sobre la configuración de un vigilante.**

La imagen 7.12 mostrará ésta pantalla.

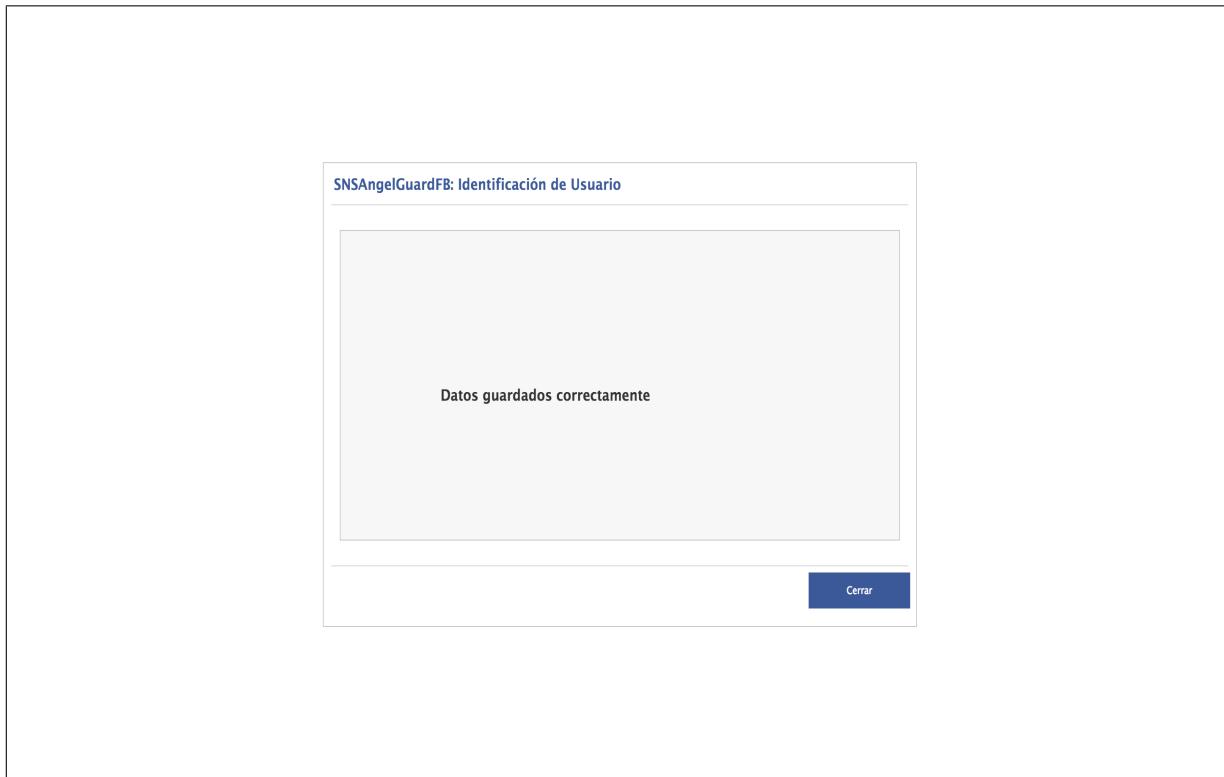


Figura 7.13: Pantalla de Resultado a la confirmación de un ángel

7.3.2.2. Pantalla de Resultado a la confirmación de un ángel

Esta pantalla se mostará como información cuando un ángel confirme o cancele a un usuario. Estará implementada por la página **informationMessage.jsp** y su representación está en la imagen A.20. Podrá contener tres tipos de mensajes:

1. **Datos guardados correctamente:** Aparecerá cuando la confirmación se haya producido satisfactoriamente y no se haya producido ningún error.
2. **Usuario ya confirmado anteriormente:** Aparecerá cuando el ángel ya haya confirmado al usuario y pulse de nuevo el enlace de confirmación.
3. **Angel no existente:** Aparecerá cuando se intente confirmar un ángel que haya sido borrado anteriormente.

La imagen 7.14 mostrará las partes de la pantalla, que serán las siguientes:

1. **Título de la página.**
2. **Contenedor de mensaje:** En él se podrá leer el resultado de la operación.
3. **Botonera Inferior:** Mostrará el botón **Cerrar**, el cual, al ser pulsado, cerrará la página.

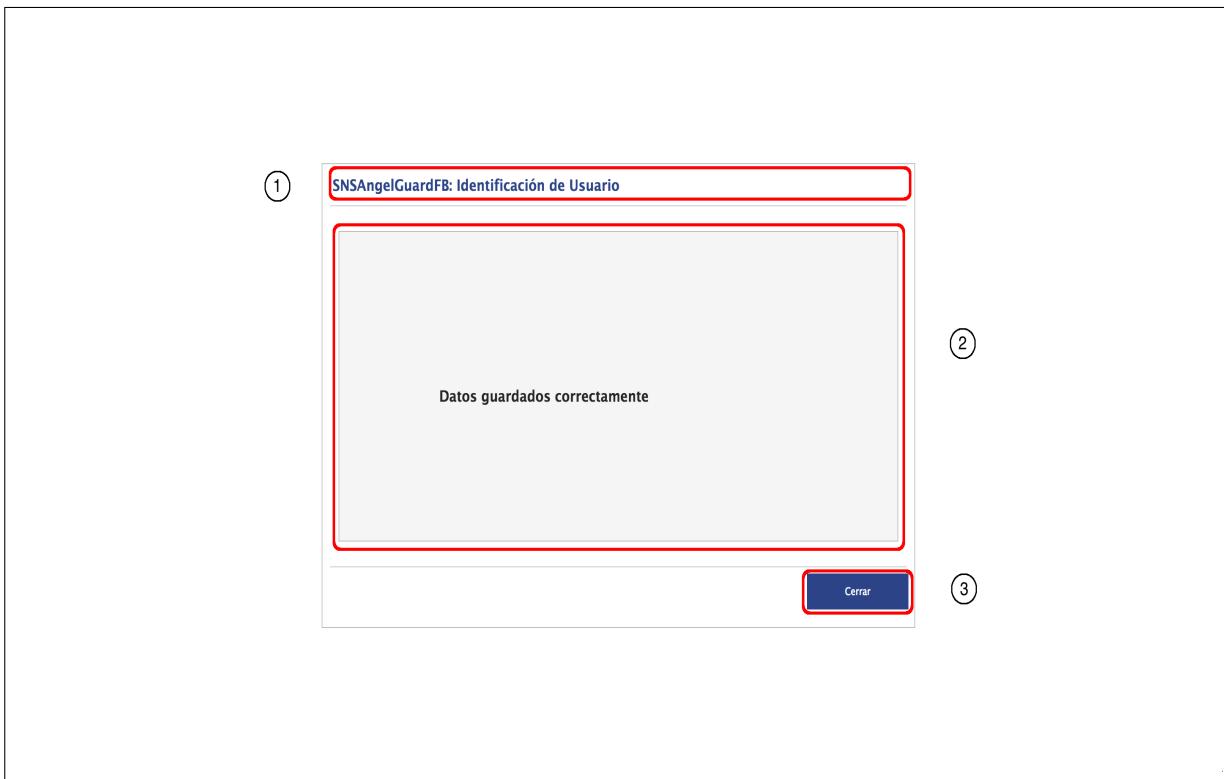


Figura 7.14: Pantalla de Resultado a la confirmación de un ángel, por partes.



Figura 7.15: Pantalla de Resultado de la operación

7.3.2.3. Pantalla de Resultado de la operación

Será una pantalla modal que aparecerá cada vez que se produzca un guardado de datos desde la pantalla de principal de la aplicación. Estará implementada por la página **infoMessage.jsp**, mostrada en la imagen 7.15 y podrá representar los siguientes tipos de información:

1. **Mensaje OK:** Se mostrará con su ícono correspondiente indicando que todo ha salido correctamente.
2. **Mensaje de Alerta:** Se mostrará con su ícono correspondiente e indicará al usuario de que se ha producido un error que no ha sido suficiente para parar la ejecución de la operación.
3. **Mensaje de Error:** Se mostrará con su ícono de error correspondiente e indicará al usuario de que se ha producido un error insalvable que ha obligado a la aplicación a salir de la operación.

La imagen 7.16 mostrará la estructura de la ventana, que tendrá las siguientes partes:

1. **Título de la página.**
2. **Contenedor del mensaje:** Estará dividido en dos partes:
 - a) **Icono:** Mostrará un ícono distinto dependiendo del resultado de la operación.
 - b) **Mensaje del resultado de la operación.**

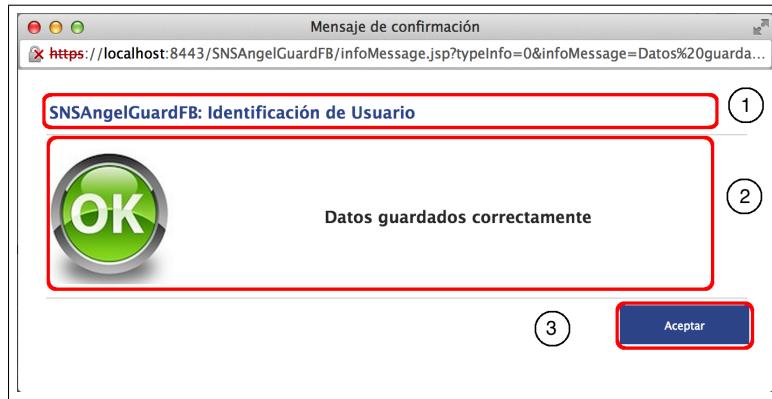


Figura 7.16: Pantalla de Resultado de la operación, por partes.



Figura 7.17: Pantalla de Información de Error.

7.3.2.4. Pantalla de Información de Error

Esta ventana mostrará información de todos los errores que se produzcan en tiempo de ejecución en la aplicación. Estará implementada por la página **infoError.jsp** y se puede observar en la imagen 7.17. Cuando aparezca este tipo de pantallas se deducirá que se ha producido un error en la aplicación que nos obliga a reiniciar el proceso completo, por lo que al pulsar el botón **Aceptar** se reiniciará de nuevo la aplicación en busca de subsanar el posible error.

La imagen 7.18 mostrará las partes de ésta ventana, que serán las siguientes:

1. **Título de la página.**
2. **Contenedor general del mensaje:** Contendrá las siguientes partes:
 - a) **Icono de error:** Se mostrará en la parte izquierda del contenedor a tamaño grande.

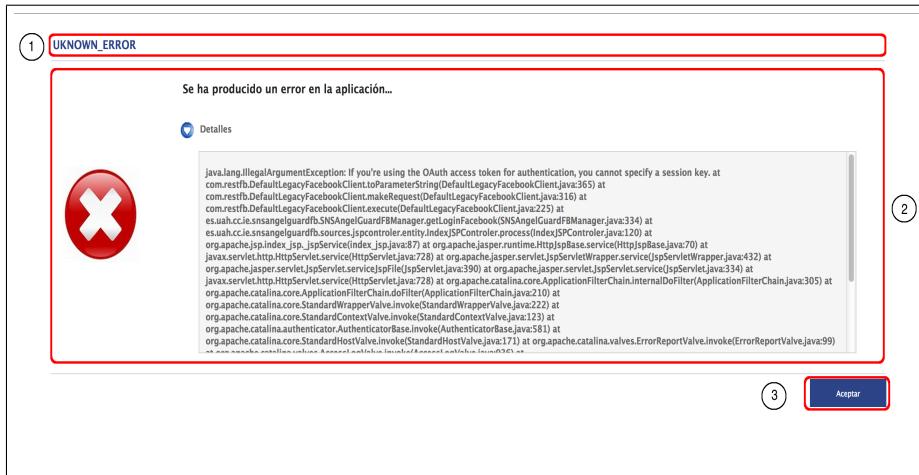


Figura 7.18: Pantalla de Información de Error, por partes.

- b) **Subtítulo del contenedor**, que mostrará siempre el literal **Se ha producido un error en la aplicación....**
 - c) **Botón y título de detalles**: Este botón, al ser pulsado, desplegará o contraerá el detalle del error, mostrado en el **Contenedor de error**.
 - d) **Contenedor de error**: Este contenedor aparecerá o desaparecerá de la página según la posición del botón de detalles. Mostrará información detallada sobre el error que se ha producido en la aplicación, es decir, mostrará la traza completa de la excepción capturada.
3. **Botonera inferior**: Contendrá al botón **Aceptar**, el cual, al ser pulsado, reiniciará de nuevo la aplicación para subsanar el error.

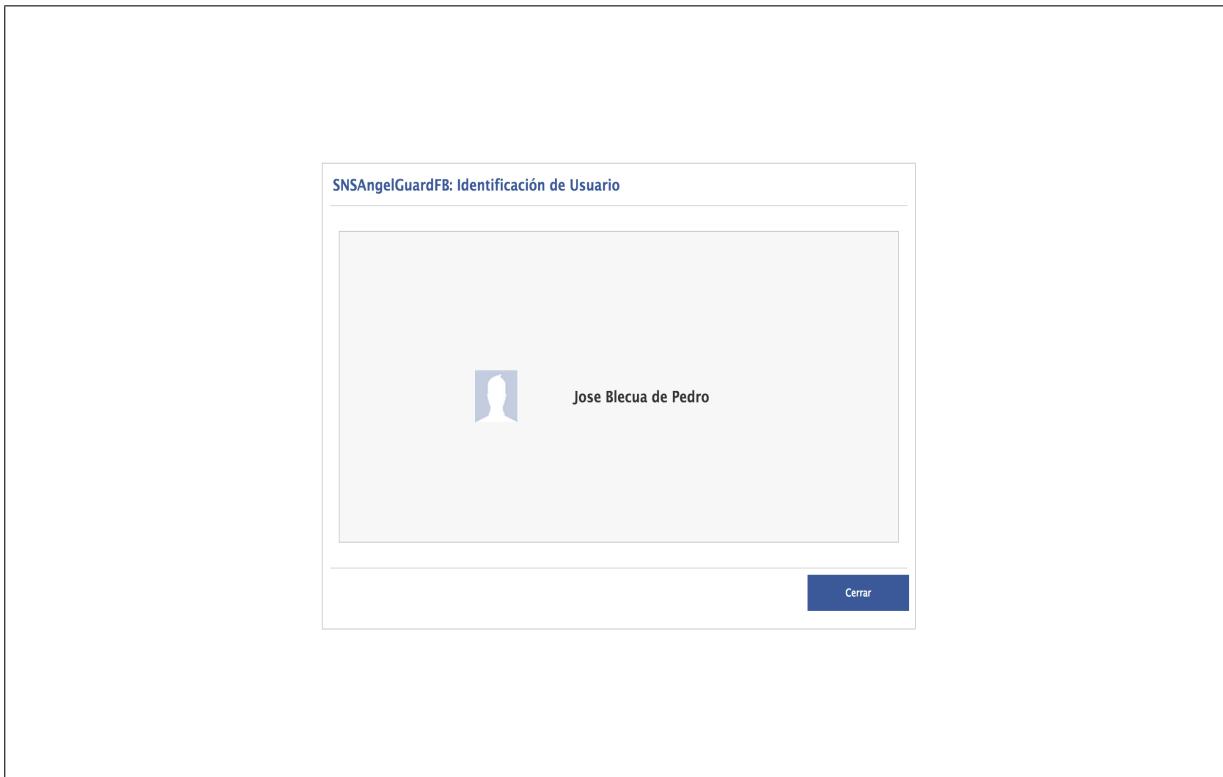


Figura 7.19: Pantalla de Identificación del usuario

7.3.2.5. Pantalla de Identificación del usuario

Será la que muestre información sobre el usuario que ha enviado una solicitud para que el ángel tutele su actividad social en Facebook y estará implementada por la página **angelUser.jsp**. En cada correo se enviará un enlace para que el ángel pueda ver en todo momento de qué usuario se trata. Estará representada en la figura A.18 y en la imagen 7.20 podrán diferenciarse las siguientes partes:

1. **Título de la página.**
2. **Contenedor principal:** Mostrará la información del usuario de la aplicación. Contendrá las siguientes partes:
 - a) **Foto de perfil:** Mostrará la foto de perfil del usuario en Facebook. Si no tiene definida ninguna, mostrará una foto estandar por defecto.
 - b) **Nombre del usuario de la aplicación**
3. **Botonera Inferior:** Mostrará el botón **Cerrar**, el cual, al ser pulsado, cerrará la página.

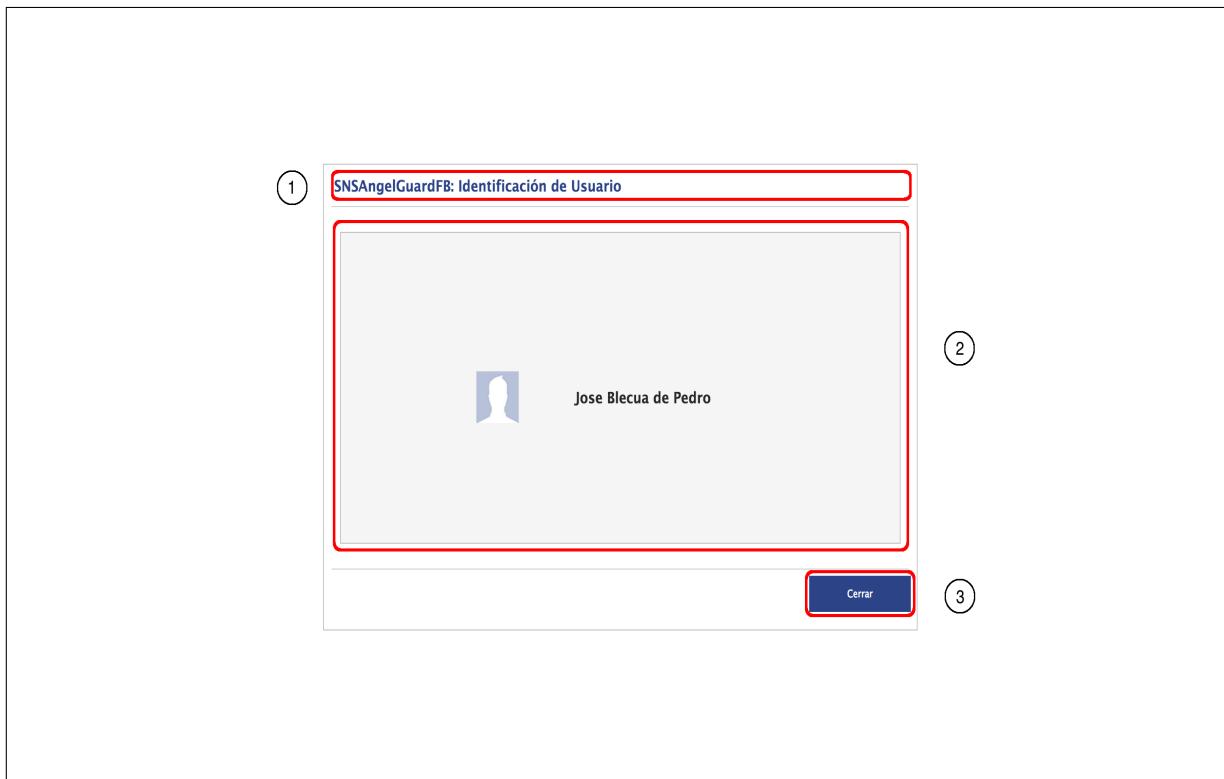


Figura 7.20: Pantalla de Identificación del usuario, por partes.



Figura 7.21: Notificación de confirmación de ángel.

7.4. Formato de notificaciones de correo

En ésta sección se presentarán los formatos de notificaciones que están presentes en la aplicación. En éste caso tendremos tres posibles tipos de notificaciones:

1. **Notificación de confirmación de ángel:** Será un mail que recibirá cada ángel cada vez que le demos de alta en la aplicación.
2. **Notificación de actividad:** Será un mail que recibirá cada ángel cuando se le envíe desde la aplicación los resultados de la ejecución de los filtros que tiene configurados.
3. **Notificación de eliminación:** Será un mail que reciba un ángel cuando el usuario de la aplicación decida no enviar más notificaciones a éste ángel.

7.4.1. Notificación de confirmación de ángel

Esta notificación será enviada a cada ángel que se elija desde la aplicación. En la imagen A.17 puede verse un ejemplo de éste tipo de notificación. Desde ella podrá realizar las siguientes acciones:

1. **Confirmar la solicitud:** Desde el enlace **Aceptar** podrá confirmar la solicitud y empezar a recibir notificaciones de actividad desde ese preciso momento.
2. **Rechazar la solicitud:** Desde el enlace **Cancelar** podrá rechazar la solicitud y no recibirá notificaciones.
3. **Consultar la información del usuario que le envía la solicitud:** Desde el enlace contenido en el nombre del usuario podrá consultar quién le está enviando la solicitud. Para ello bastará con pulsar éste enlace y se le abrirá en el navegador de internet una página con los datos del usuario.



Figura 7.22: Notificación de confirmación de ángel, por partes.

En la imagen 7.22 puede verse la estructura del correo, que tendrá las siguientes partes:

1. **Título del correo.**
2. **Cuerpo del mensaje:** Contendrá las siguientes partes:
 - a) **Texto informativo de la notificación:** Contendrá una pequeña explicación de la notificación, que además tendrá un enlace para acceder a la información del usuario.
 - b) **Enlace Aceptar:** Será el enlace con el que se confirmará la solicitud y se recibirá la primera notificación de actividad.
 - c) **Enlace Cancelar:** Será el enlace con el que se rechazará la solicitud del usuario.
3. **Texto informativo de la aplicación:** Contendrá un enlace a la aplicación y la información del propietario de la misma.

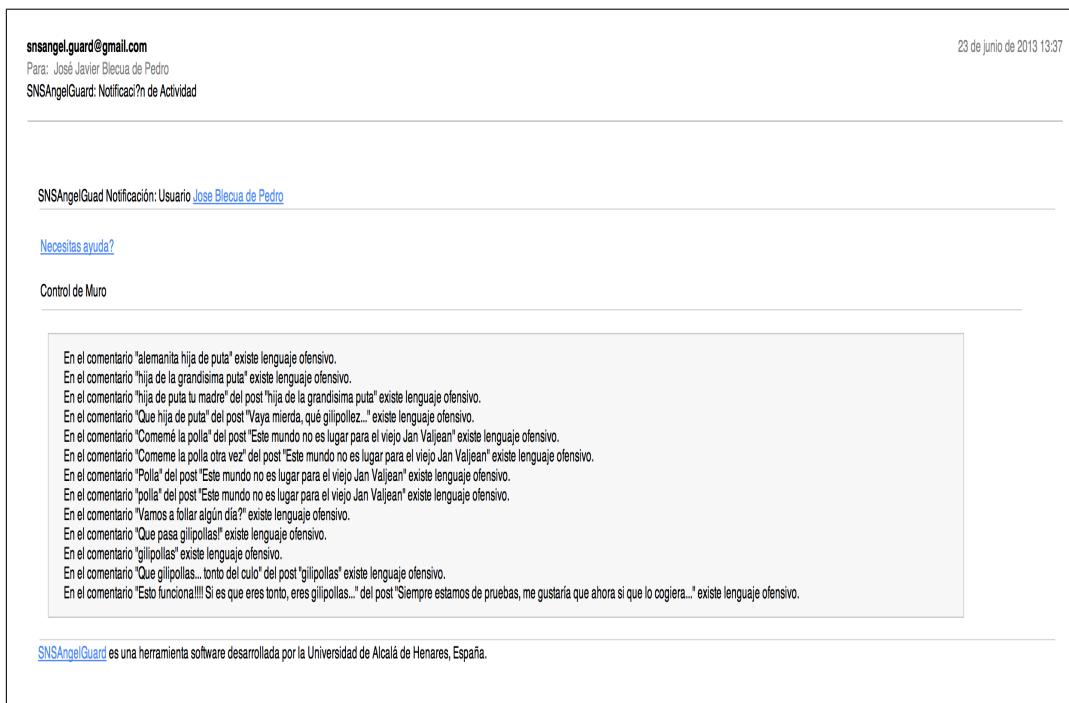


Figura 7.23: Notificación de actividad.

7.4.2. Notificación de actividad

Será el correo normal de notificación. Contendrá un informe por cada filtro que tengamos configurado. Está representado por la figura A.21. En la imagen 7.24 puede verse la estructura de la notificación que pasaremos a detallar:

- Título de la notificación.**
- Enlace a la Página de Ayuda:** Mediante éste enlace se mostrará en una pantalla del navegador la página de ayuda de la aplicación por si el ángel tiene dudas sobre el funcionamiento de la aplicación.
- Cuerpo de la notificación:** Esta parte contendrá todas las anomalías referidas a los filtros que tenga configurado el usuario.
- Texto informativo de la aplicación:** Contendrá un enlace a la aplicación y la información del propietario de la misma.

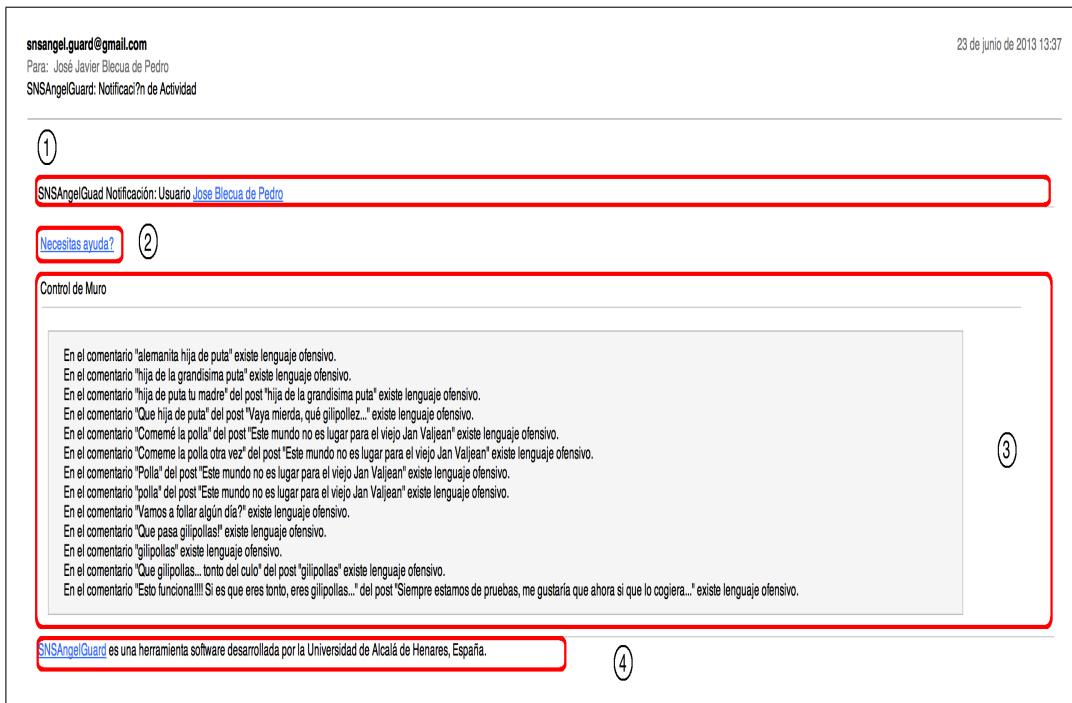


Figura 7.24: Notificación de actividad, por partes.

7.4.3. Notificación de eliminación

Esta notificación será enviada a cada ángel que sea borrado por el usuario, por lo que dejarán de recibir notificaciones de su actividad. En la imagen 7.25 puede verse un ejemplo de éste tipo de notificación.

En la imagen 7.26 puede verse la estructura del correo, que tendrá las siguientes partes:

1. **Título del correo.**
2. **Cuerpo del mensaje:** Contendrá un enlace al usuario que le ha borrado como ángel y el mensaje asociado.
3. **Texto informativo de la aplicación:** Contendrá un enlace a la aplicación y la información del propietario de la misma.

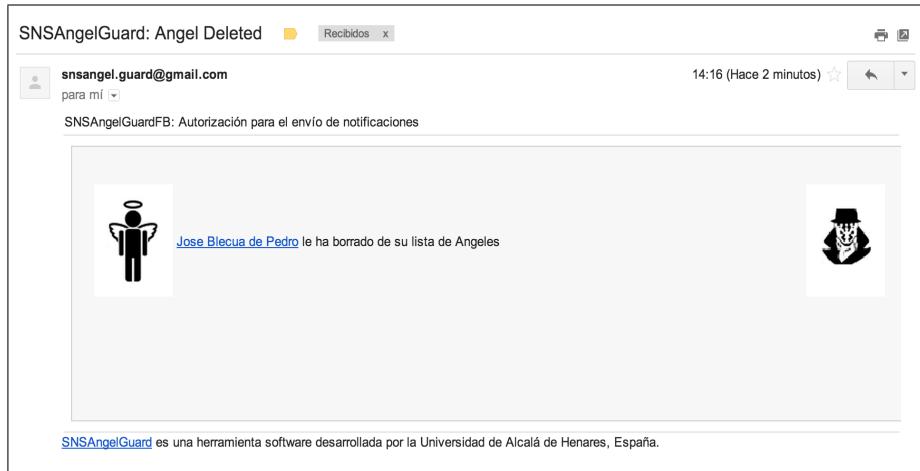


Figura 7.25: Notificación de eliminación de ángel.

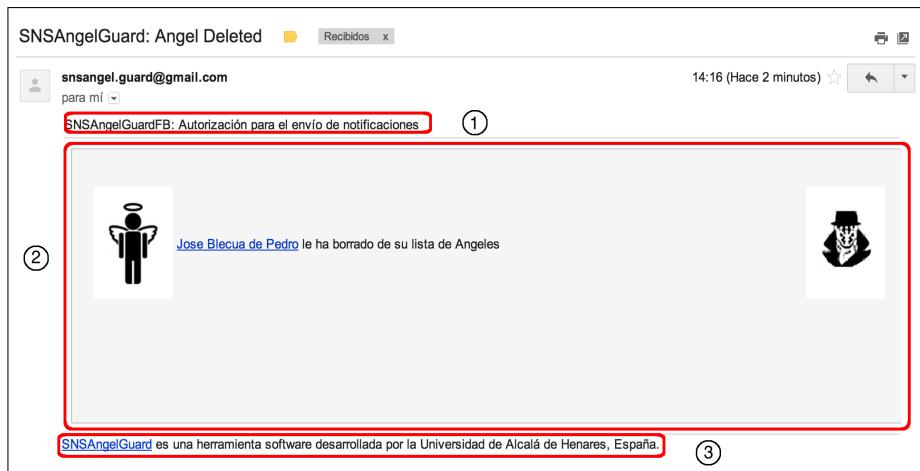


Figura 7.26: Notificación de eliminación de ángel, por partes.

Capítulo 8

Módulo Offline

8.1. Introducción

Para realizar el cometido de la aplicación, será necesaria una herramienta que realice backups offline de información de todos los usuarios y realice la ejecución del filtro. El Módulo Offline será el encargado de realizar los procesos de carga, validación y análisis de datos de forma automática. Se ejecutará una o varias veces al día, como tarea programada en el servidor, y realizará las siguientes acciones:

1. Obtener, de la base de datos SocialNetwork, la información relativa a todos los usuarios de la aplicación.
2. Por cada usuario realizará las siguientes acciones:
 - a) Descargará la nueva información que haya generado en Facebook, tal como nuevos comentarios en el muro o nuevas amistades.
 - b) Dependiendo de la periodicidad con que tenga configurados los filtros, realizará un análisis de la nueva información obtenida.
 - c) Tras realizar el análisis, informará a los ángeles los resultados obtenidos.
 - d) Actualizará en base de datos la fecha y la hora del momento en el que se ha realizado el proceso de actualización de la información.

Éste módulo será vital para mantener la base de datos actualizada con la última información del usuario y ejecutar los filtros configurados por éste. Su arquitectura será la siguiente:

1. **Script o tarea programada:** Dependerá del servidor o máquina física en el que se ejecute el servidor de aplicaciones. Se programará temporalmente una o varias veces diarias para poder realizar los backups de mantenimiento de la información. Será el desencadenante del proceso y sin él el Módulo Offline no se ejecutaría.
2. **Clase de enlace:** El script lanzará la aplicación **harvestedSNSAngelGuard** que se encargará de enlazar éste con el servlet de peticiones offline **HarvestedSNS**

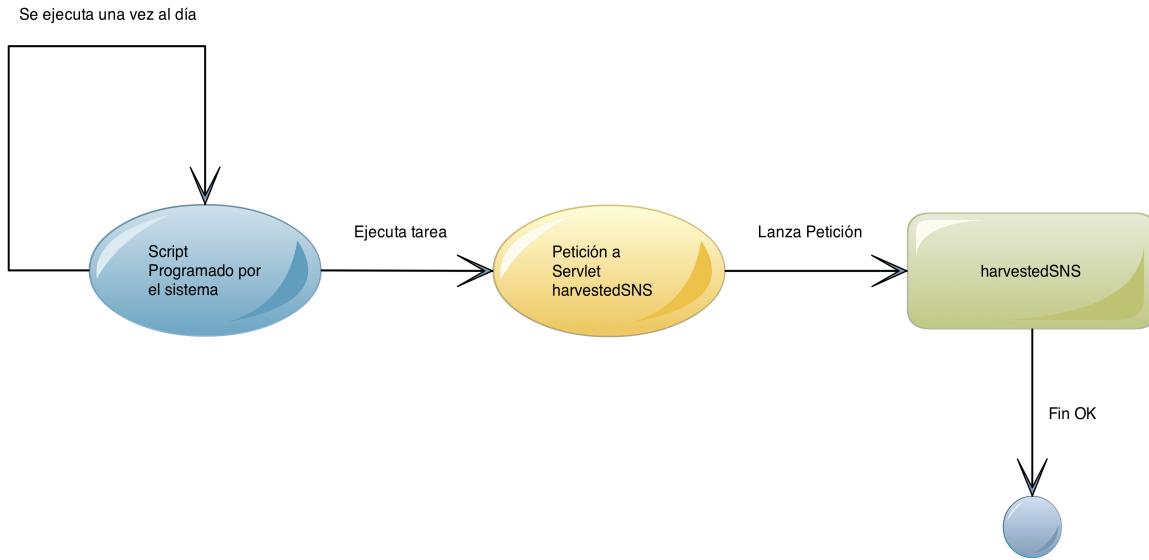


Figura 8.1: Diagrama de ejecución del Módulo Offline

contenido en el proyecto web **SNSAngelGuardFB**, el cual, estará disponible y ejecutándose en un servidor web. Esta aplicación deberá estar instalada en el servidor físico pero no será un proyecto Web.

3. Servlet **HarvestedSNS**: Se encontrará disponible en el servidor web de aplicaciones a través de la aplicación **SNSAngelGuardFB**. Será el encargado de realizar las siguientes acciones:

- Obtendrá una lista con todos los usuarios de la aplicación.
- Irá uno a uno actualizando la información: Obtendrá la nueva información que el usuario haya producido en Facebook y la almacenará en la base de datos.
- Tras almacenar la nueva información, realizará sobre ésta los chequeos de los filtros correspondientes, es decir, los chequeos se realizarán únicamente sobre la nueva información que se produzca en el intervalo de tiempo entre chequeos, no sobre toda la información del usuario. El chequeo general se produce cuando un ángel realiza la confirmación oficial sobre un usuario. Entonces, el chequeo de cada filtro se realizará sobre toda la información disponible hasta ese momento. Después, se realizará por la información producida en el periodo temporal entre cada ejecución de los filtros.

La imagen 8.1 mostrará el diagrama de despliegue y ejecución de éste módulo que pasaremos a explicar a continuación.

8.2. Configuración de la tarea programada

Para lanzar el Módulo Offline, será necesario modificar aquellas estructuras del servidor físico en el que se vaya a lanzar dicho módulo que lancen tareas programadas para el mismo y para otras aplicaciones. Será la parte más dependiente de la arquitectura física de ejecución, es decir, configurar la tarea programada dependerá de la arquitectura del servidor, más concretamente de su **Sistema Operativo**. En nuestro caso tomaremos el ejemplo de un servidor de una máquina con un Sistema Operativo **Linux**.

La configuración de una tarea programada en Linux se realizará desde los ficheros **crontab**. Este tipo de ficheros son la entrada del programa **cron**, el cual permite a los usuarios de Linux/Unix ejecutar automáticamente comandos o scripts a una hora, fecha o día específico. Esta aplicación será un demonio, es decir, sólo requiere ser iniciado una vez, que generalmente será en el mismo arranque del sistema. A partir de ése momento, todas las tareas marcadas dentro del fichero crontab se ejecutarán en la fecha indicada para ello.

8.2.1. Fichero crontab

Éste fichero será el que se modificará para introducir nuestro script. Se encontrará en la dirección **/etc/crontab** y para modificarlo se introducirá la siguiente sentencia:

```
vi /etc/crontab
```

En éste momento, el editor **vi** nos mostrará por pantalla el contenido del fichero, que mostrará una serie de instrucciones con el siguiente formato:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

En éste fichero podemos observar las primeras líneas de configuración del fichero, que serán las siguientes:

1. Parámetro **SHELL**: Indicará la **shell** bajo la cual se ejecutará el cron.
2. Parámetro **PATH**: Indica la ruta a los directorios en los cuales cron buscará el comando a ejecutar.
3. Parámetro **MAILTO**: Es el usuario al que se le enviará la salida del comando que se ejecute. Para ello, cron enviará un correo al usuario que esté indicado en éste parámetro.

4. Parámetro **HOME**: Es el directorio raíz o principal del comando cron.

Dentro del fichero también podemos observar las tareas que se van a ejecutar precedidas de un código alfanumérico de cinco partes, que indicarán al cron cuando se ejecutará la tarea a la que están asociados. Estos caracteres indican lo siguiente:

1. La primera posición indicará el **minuto** en el que se ejecutará la tarea. Podrá tomar valores entre 0 y 59.
2. La segunda posición indicará la **hora**, que podrá tomar valores de 0 a 23.
3. La tercera posición indicará el **día del mes** en el que se ejecutará la tarea.
4. La cuarta posición indicará el **mes** en el que se ejecutará la tarea, por lo que tomará valores de 1 a 12.
5. La quinta posición indicará el **día de la semana** en el que se ejecutará la tarea. Puede ser un campo numérico o textual, con el día de la semana en inglés.

El carácter * indicará inicio-fin del campo, es decir, todo. Un * en el campo minutos indicará que la tarea se ejecutará en todos los minutos. Después del código temporal de ejecución, se definirá el usuario que realiza la operación y la ruta exacta al fichero script que se ejecutará.

8.2.2. Modificación del fichero crontab

Para modificar éste fichero, bastará con que elijamos una fecha y hora a la que se ejecutará nuestro script y ya estaremos en disposición de iniciar la modificación. En nuestro caso, añadiremos al fichero la siguiente linea:

```
00 5 * * * root /usr/local/script/harvestedSNSAngelGuardFB.sh
```

La cual indicará que el usuario root iniciará el script **harvestedSNSAngelGuardFB.sh** contenido en la ruta **/usr/local/script** a las 5 en punto de la mañana. En ese momento, se iniciará el Módulo Offline de la aplicación. Decir que ésto es un ejemplo, el responsable de la aplicación podrá decidir en todo momento en qué fecha y hora se ejecutará este módulo y, para ello, sólo tendrá que modificar éste fichero de la forma anteriormente indicada.

8.3. Script harvestedSNSAngelGuardFB.sh

Este script forma parte de la arquitectura del servidor en el que se instale, ya sea Windows o Linux. Siguiendo con el apartado anterior, se tomará el ejemplo de Linux. Contendrá una serie de líneas que realizarán las siguientes acciones:

1. Configuración del fichero **logFile**: En ésta variable se indicará el fichero que contendrá todas las trazas de la ejecución de la tarea. Por definición, éstas se guardarán en un fichero del tipo:

```
execute_harvesting_SNSAngelGuardFB_YYYY_MM_DD.log
```

En el que los últimos caracteres indicarán el año, el mes y el día de la ejecución. En ésta variable se podrá elegir la ruta en la que se guardará éste fichero.

2. Posicionamiento en el directorio en el que se encontrará el fichero a ejecutar: En nuestro caso, queremos ejecutar la aplicación **harvestedSNSAngelGuard**, por lo que, mediante el comando **cd** nos posicionaremos en el directorio de ésta aplicación instalada en el sistema.
3. Mediante **maven**, obtendremos todas las dependencias que necesitemos para la ejecución de la tarea, las copiaremos a la dirección de ejecución y redireccionaremos la salida al fichero de logs.
4. Por último, **ejecutaremos la tarea** por medio de la máquina virtual de java y maven: En éste caso, indicaremos la clase exacta dentro de la estructura de paquetes de la aplicación **harvestedSNSAngelGuard**, en la cual sólo existirá la clase **Main** que será la que, finalmente, se ejecute, y redireccionaremos todas las trazas que se generen en la ejecución al fichero de logs. La linea que define este caso será la siguiente:

```
mvn exec:java  
-Dexec.mainClass="es.uah.cc.ie.harvestedsnsangelguard.Main" >> \${logFile}
```

Tras la definición de éste fichero, se podrá comenzar con la parte menos dependiente de la arquitectura del servidor.

8.4. Aplicación harvestedSNSAngelGuard

Esta será una aplicación No Web que estará instalada en el espacio de ficheros del servidor. Únicamente contendrá la clase **Main** dentro del paquete:

`es.uah.cc.ie.harvestedsnsangelguard`

Su funcionalidad es la siguiente:

1. Abrirá una conexión a la dirección en la que esté instalada la aplicación: Para un ámbito local, abrirá una conexión en la siguiente dirección:

`http://localhost:8080/SNSAngelGuardFB/HarvestedSNS`

2. Esta conexión conectará directamente con el servlet **HarvestedSNS** y, en ese momento, se iniciará la tarea de ejecución y actualización de información del Módulo Offline.

8.5. Servlet HarvestedSNS

Esta clase estará contenida en el paquete `es.uah.cc.ie.snsangeguardfb.sources.offline` y su estructura está especificada en la sección 6.3.7.1. Su funcionamiento es el siguiente:

1. Método de llegada **doGet()**: Este método recepcionará la petición, conectará la aplicación offline con Facebook y lanzará el método **updateUsers**.
2. Método **updateUsers()**: Obtendrá todos los usuarios de la base de datos y por cada uno de ellos actualizará su información de Facebook en la base de datos y chequeará sus filtros configurados. Por último, si se han producido anomalías, enviará las correspondientes notificaciones a los ángeles por medio de emails.

En las siguientes secciones se entrará más a fondo en la explicación de estos métodos.

8.5.1. Método doGet()

Este método será quien recepcione las peticiones offline y las procese. Tendrá dos parámetros de entrada:

1. **HttpServletRequest request**: Contendrá la petición de entrada al servlet con todos los datos de la sesión.
2. **HttpServletResponse response**: Contendrá la respuesta a la petición de entrada.

A partir de estos parámetros, este método realizará las siguientes acciones:

1. Por medio del parámetro de entrada **request**, inicializará un objeto **SNSAngelGuardFBManager** con los datos de la sesión y lo almacenará como atributo del objeto.
2. Mediante el objeto SNSAngelGuardFBManager, iniciará la sesión offline en la aplicación mediante su método **getLogginAppOffline**, ya que para que una aplicación se ejecute offline deberá antes identificarse en Facebook.
3. Por último, **updateUsers()** actualizará la información de los usuarios de la aplicación y ejecutará sus filtros.

El diagrama de actividad 8.2 indicará las acciones explicadas.

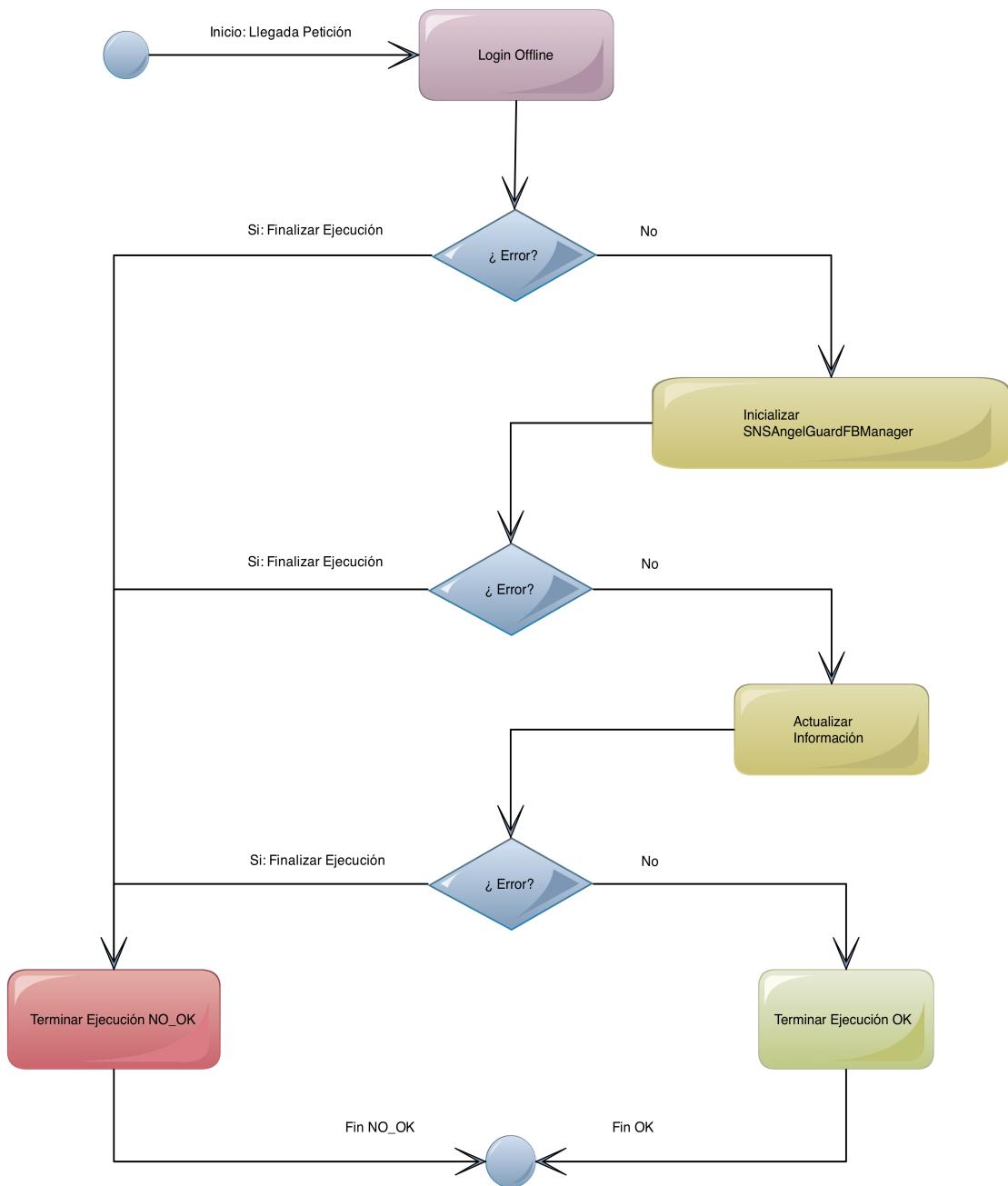


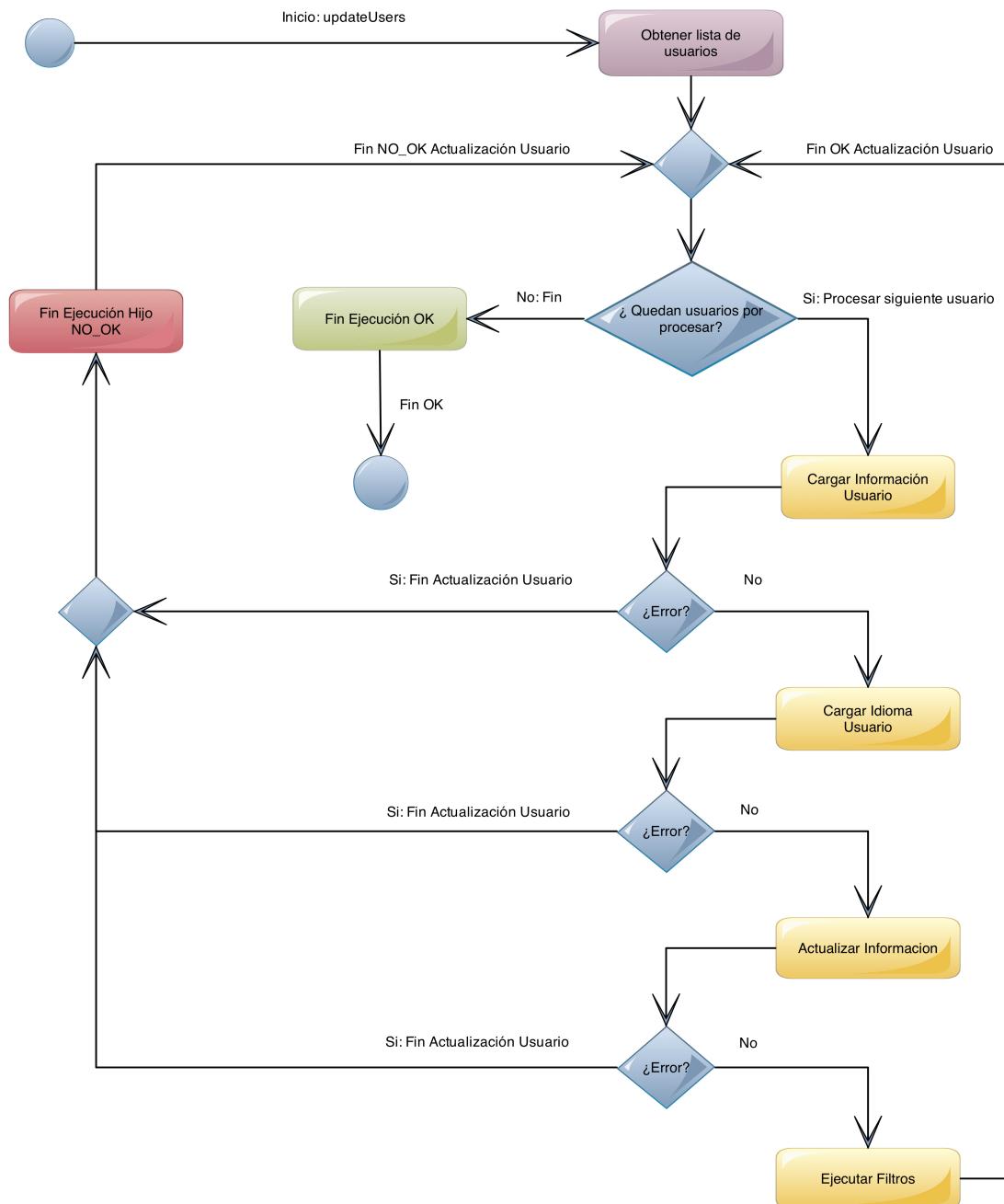
Figura 8.2: Diagrama de actividad del método doGet()

8.5.2. Método updateUsers()

Este método será, en última instancia el que lleve a cabo todo el negocio del Módulo Offline. Será el que actualice toda la base de datos para cada usuario y, por último, ejecute sus filtros y mande las notificaciones pertinentes a sus ángeles. Para ello, realizará las siguientes acciones:

1. **Obtendrá todos los usuarios de la aplicación**, contenidos todos ellos en la tabla `user_settings`: Para ello ejecutará de la clase `UserSettingsDAO` el método `getEntitiesUserSettings`, el cual devolverá, en un objeto `JSONArray`, la lista con todos los usuarios de la aplicación y sus datos de configuración.
2. Para **cada uno de los elementos de la lista** se realizarán las siguientes acciones:
 - a) Se cargará toda su información de configuración de usuario en la aplicación.
 - b) Se obtendrán todos los recursos de idioma para el usuario actual de forma offline.
 - c) Si la aplicación está activada:
 - 1) Obtendrá toda la información de Facebook, incluyendo información personal de su perfil público, sus nuevos comentarios y los datos actualizados de sus contactos.
 - 2) Almacenará esta información en la base de datos.
 - 3) Ejecutará sus filtros y mandará, en caso afirmativo, las notificaciones a sus ángeles que corresponda por filtro.
3. Si en algún momento del proceso ocurre un error durante las operaciones con algún usuario, se lanzará una excepción, se pintará en el fichero de trazas y se pasará al siguiente usuario sin salir de la aplicación general.
4. Cuando se termine de recorrer la lista de usuarios, se terminará el proceso.

El diagrama de actividad de la imagen 8.3 mostrará el algoritmo a seguir para realizar ésta tarea.

Figura 8.3: Diagrama de actividad del método `updateUsers()`

Capítulo 9

Cómo extender SNSAngelGuardFB

9.1. Introducción

En ésta sección se especificará cómo se puede ampliar la funcionalidad de la aplicación **SNSAngelGuardFB** para, en futuro próximo, satisfacer las nuevas necesidades de las redes sociales. Estos cambios van desde una ampliación en los filtros de análisis de la información hasta posibles nuevas traducciones de la aplicación ya que, al ser una aplicación multiidioma, se podrá traducir a todos los idiomas oportunos siguiendo una serie de directrices que en éste apartado pasaremos a detallar.

9.2. Añadir nuevos filtros a SNSAngelGuardFB

Una de las funciones más potentes de la aplicación es su extensibilidad y flexibilidad a la hora de añadir nuevas funcionalidades. Éste es el caso a la hora de añadir nuevos filtros.

Para añadir nuevos filtros, será necesario seguir una serie de reglas, muy básicas, en las que en pocos minutos podremos tener un nuevo filtro a disposición de los usuarios de la aplicación. Los pasos para añadirlo se explicarán en las siguientes secciones.

9.2.1. Implementación de la interface ILifeCycleFilter

Como primer punto, se deberá implementar una clase que contenga la funcionalidad del filtro siguiendo la estructura de la interface **ILifeCycleFilter**. Todo filtro que no cumpla ésta clase no será controlado por la aplicación.

Los métodos de la interface **ILifeCycleFilter** serán los siguientes:

```
public interface ILifeCycleFilter {  
  
    /**  
     * Inicializa el filtro con la clase manager de la aplicacion.  
     */  
}
```

```

    * @param snsObject Objeto manager de la aplicación.
    * @param id Identificador del filtro.
    */
public void init(SNSAngelGuardFBManager snsObject, String id);

/**
 * Ejecuta la funcionalidad del filtro.
 *
 * @param Mapa de parámetros de entrada.
 * @return Cadena de texto con el resultado del filtro en formato HTML
 * @throws Exception
 */
public String executeFilter(Map<String, Object> args) throws Exception;

/**
 * Obtiene toda la información interna de Facebook necesaria para el
 * funcionamiento del filtro y la almacena en base de datos.
 *
 * @throws Exception
 */
public void getInformationFacebook() throws Exception;

/**
 * Actualiza la información interna en base de datos para analizarla.
 *
 * @throws Exception
 */
public void updateInformationFacebook() throws Exception;

/**
 * Obtiene el identificador del filtro.
 *
 * @return String
 */
public String getId();
}

```

9.2.2. Añadir una entrada al fichero de control de filtros

Como parte externa de la aplicación, tendremos un fichero de control de los filtros activos, denominado **filters.xml**. Este fichero deberá estar disponible en la ruta de configuración **/usr/local/snsangeguardfb/config** y deberá tener la siguiente estructura:

```
<filterList>
  <filter idFilter="FltWall" valueClass="es.uah.cc.ie.snsangeguardfb.sources.
filtersfuncionality.WallFilterFuncionality" />
```

```

<filter idFilter="FltFriends" valueClass="es.uah.cc.ie.snsangelguardfb.sources.
filtersfuncionality.FriendsFilterFuncionality" />
<filter idFilter="FltPriv" valueClass="es.uah.cc.ie.snsangelguardfb.sources.
filtersfuncionality.PrivacyFilterFuncionality" />
<filter idFilter="FltVist" valueClass="es.uah.cc.ie.snsangelguardfb.sources.
filtersfuncionality.VisitsFilterFuncionality" />
</filterList>

```

En ésta estructura se pueden distinguir las siguientes estructuras:

1. Entidad **filterList**: Será el elemento padre del fichero XML. En él se definirán todos los filtros.
2. Entidad **filter**: Será la entidad del fichero XML en el que se defina un filtro. Este elemento tendrá los siguientes campos:
 - a) **idFilter**: Será la clave-identificador único de cada uno de los filtros.
 - b) **valueClass**: Será la ruta a la clase Java que hemos implementado en el paso anterior bajo la interface **ILifeCycleFilter**.

Cada vez que queramos añadir un nuevo filtro a la aplicación, deberemos introducir una entrada a éste fichero con los datos especificados.

9.2.3. Añadir una imagen al nuevo filtro

Para ello, será necesario, dentro del proyecto de la aplicación, en la carpeta **resources/robots**, añadir una nueva imagen **png** cuyo nombre de fichero tenga la siguiente nomenclatura:

`robot + [nombreFiltro] + .png`

El campo **[nombreFiltro]** deberá ser la clave única identificadora del filtro que hayamos introducido en el fichero **filters.xml**. Ejemplo: `robotFltWall.png`.

9.2.4. Actualizar los recursos de idioma de la tabla `locale_settings`

Todo filtro debe tener un nombre y una descripción, que posteriormente se mostrarán en pantalla. Para ello, habrá que modificar dos campos de la tabla **locale_settings** de la base de datos:

1. Campo **titleVigSettVig**: Este campo contendrá un objeto JSON con los nombres de los filtros. Si queremos añadir un nombre, únicamente habrá que añadir la nueva clave del filtro con su nombre, de la siguiente manera:

```
{
    "FltWall": "Zeus",
    "FltFriends": "Hermes",
    "FltPriv": "Apollo",
    "FltVist": "Poseidon"
}
```

2. Campo **titleVigDescriptionSettVig**: En éste campo se almacenará la descripción para el vigilante de la siguiente forma:

```
{
    "FltWall": "Esta es la descripción para mi nuevo filtro...",
    ....
}
```

Nota: Cuando se añaden nuevos recursos de idioma, habrá que introducirlos en todos los idiomas que haya disponible en la base de datos, para evitar inconsistencias de traducción.

9.3. Traducción de la aplicación a un nuevo idioma

La aplicación actualmente está preparada para ser multiidioma, es decir, soporta cualquier idioma configurado previamente para ello. Actualmente, se da soporte para los idiomas castellano e inglés. Para añadir un nuevo idioma, se realizarán las siguientes modificaciones que se muestran a continuación.

9.3.1. Nuevo registro en la tabla locale_settings

Para introducir un nuevo idioma, el paso inicial será introducir en la tabla de recursos de idioma **locale_settings** la traducción de todos los literales contenidos en la misma tabla en el registro de idioma castellano e introducirlos para un nuevo registro ya traducidos al nuevo idioma para el que se va a dar soporte.

Notesé que la traducción debe hacerse en el mismo formato que está el resto de idiomas, es decir, en muchos de los campos de la tabla no hay sólo un literal, sino que puede haber varios literales formando un array separado por el carácter ;. Esta indicación se puede comprobar en el capítulo 4.2.2 del presente documento.

9.3.2. Generación de un nuevo fichero lexico de idioma

Para dar soporte al filtro de **Control de Idioma**, será necesario introducir un nuevo fichero de expresiones malintencionadas en el entorno de la aplicación del propio idioma. Su nombre deberá tener el siguiente formato:

`badWords_xx.txt`

Siendo los caracteres 'xx' los que indiquen, en letras minúsculas, el idioma al que pertenece el fichero. Éste se almacenará junto con el resto de ficheros léxicos de la aplicación para el resto de idiomas dentro de la carpeta **lexicalFiles**, contenida dentro del proyecto **SNSAngelGuardFB**.

9.3.3. Modificaciones en la clase UserSettingsDAO

Será necesario retocar el método **getIdLocale** para que se pueda obtener el identificador del nuevo idioma en base de datos en función del idioma del usuario de Facebook y los recursos de idioma sean obtenidos de forma correcta.

9.3.4. Modificaciones en la clase WallFilterFuncionality

En ésta clase será necesario retocar el método **getPathFileBadWords**, para que dé soporte al idioma indicado y poder seleccionar correctamente el fichero a comprobar en el filtro de **Control de Idioma**.

Capítulo 10

Trabajo a futuro y conclusiones

10.1. Trabajo a futuro

Tras analizar las posibles ampliaciones que se pueden hacer en ésta aplicación, toca realmente ver, además de éstas, qué posibles modificaciones hacen falta para que la aplicación sea más legible y modularizable para que, en un futuro, se pueda ampliar con facilidad y no sea un tormento a la hora de incluir un nuevo filtro de información.

10.1.1. Implementación del filtro de configuración

Esta es una de las tareas pendientes. Toda la infraestructura del filtro ya está disponible en la aplicación, la única tarea que queda pendiente es modelar la clase **Privacy-FilterFunctionality** para que tenga el negocio deseado para éste filtro. El resto de la infraestructura está preparada para la ejecución de éste filtro.

10.1.2. Mejorar el algoritmo de control de lenguaje

El algoritmo utilizado para controlar el lenguaje de un determinado perfil de usuario es demasiado básico. Consiste en contrastar las palabras de un diccionario contra las de un comentario en el muro de Facebook del usuario.

Una posible mejora consistiría en construir un algoritmo complejo utilizando ontologías¹ en las que se definan esquemas conceptuales de donde partan todas las expresiones malintencionadas que se pueden encontrar en el muro de un usuario.

¹El término ontología en informática hace referencia a la formulación de un exhaustivo y riguroso esquema conceptual dentro de uno o varios dominios dados; con la finalidad de facilitar la comunicación y el intercambio de información entre diferentes sistemas y entidades. Aunque toma su nombre por analogía, ésta es la diferencia con el punto de vista filosófico de la palabra ontología. Un uso común tecnológico actual del concepto de ontología, en este sentido semántico, lo encontramos en la inteligencia artificial y la representación del conocimiento. En algunas aplicaciones, se combinan varios esquemas en una estructura de facto completa de datos, que contiene todas las entidades relevantes y sus relaciones dentro del dominio. Los programas informáticos pueden utilizar así este punto de vista de la ontología para una variedad de propósitos, incluyendo el razonamiento inductivo, la clasificación, y una variedad de técnicas de resolución de problemas. Típicamente, las ontologías en las computadoras se relacionan estrechamente con vocabularios fijos –una ontología fundacional– con cuyos términos debe ser descrito todo lo demás.

En éste filtro también sería óptimo ampliar su funcionalidad para que también pudiera controlar conversaciones instantáneas tipo chat, las cuales son potencialmente más peligrosas que los comentarios públicos que terceras personas puedan hacer en el muro de un usuario.

10.1.3. Mejorar el mantenimiento de la aplicación

El API de Facebook cambia constantemente para mejorar funcionalmente dicha Red Social y, además, ganar seguridad en el resto de aplicaciones que utilizan su red y que pueden hacer un uso malintencionado de los datos que se captan.

Es por ello que la funcionalidad de la aplicación debe revisarse cada vez que se producen dichos cambios, porque puede cambiar y dejar de funcionar por estos motivos, por lo que, tras liberar esta versión inicial, será necesario mantener y actualizar las estructuras que la componen para mantener en uso esta aplicación.

10.1.4. Filtro de fotografías

Sería conveniente crear un nuevo filtro que controle las fotografías que otros usuarios suben a Facebook y etiquetan a sus víctimas para ser objeto de burla y acoso.

10.1.5. Filtro de chat

Como nueva mejora, sería bastante conveniente realizar un control del lenguaje en el chat instantáneo de Facebook, cuyo analizador sintáctico sea el utilizado para el filtro de control de lenguaje ofensivo. Para este caso, habría que investigar los diferentes plugins de mensajería instantánea que hay en el mercado y en cual de ellos se basa Facebook para implementar su chat.

10.2. Conclusiones

El mundo de las Redes Sociales está en auge actualmente. Todo el mundo es consciente de su importancia pero nadie lo es de lo peligrosas que son. Cada vez son más las personas que cuidan sus datos y los protegen de perfiles potencialmente peligrosos que pueden influir en terceras personas. Es por esta razón que aplicaciones de control cada vez son más demandadas, pero hay pocas en el mercado que puedan tener la filosofía que tiene **SNSAngelGuardFB**.

Con el desarrollo de ésta aplicación me he dado cuenta de lo fácil que es vulnerar las reglas para atacar perfiles potencialmente menos seguros que el resto, o lo fácil que es intimidar a ciertos perfiles de un rango de edad determinado que no denuncian estas agresiones por vergüenza o miedo.

Debido a que esto puede ocasionar representaciones pobres para ciertos dominios de problemas, se deben crear esquemas más especializados para convertir en útiles los datos a la hora de tomar decisiones en el mundo real. Fuente: Wikipedia, [https://es.wikipedia.org/wiki/Ontología_\(informática\)](https://es.wikipedia.org/wiki/Ontología_(informática))

Es por esta razón por lo que herramientas como **SNSAngelGuardFB** son completamente necesarias para un mundo como el de hoy en día, en el que cualquiera puede conectarse a una Red Social desde todo tipo de dispositivos electrónicos y puede ser atacado o perturbado. En ésta tesitura, entra en juego nuestra herramienta, facilitando a terceras personas, cómo los ángeles, ser conscientes de la situación de peligro que puede correr nuestros tutelados y tomar las decisiones que sean oportunas para que puedan dejar esa situación.

Nuestra herramienta es, por tanto, una nueva visión de futuro que, aunque nuestros algoritmos no sean lo suficientemente potentes como para detectar todas las posibles vulnerabilidades de un perfil determinado de Facebook, éstos sean una puerta de entrada a otros estudios mucho más amplios con los que se lleguen a objetivos como controlar a cualquier individuo que está amenazando a nuestros seres queridos o mantenernos en todo momento seguros cuando naveguemos por este tipo de sitios.

En definitiva, el desarrollo de ésta herramienta ha sido provechoso en el sentido del conocimiento adquirido respecto al alcance en Redes Sociales se refiere, ya que al ver todo lo que una aplicación puede hacer con unos simples permisos dados por cualquier usuario(ya sea intencionada o inintencionadamente) uno admite que puede estar constantemente en situaciones de riesgo, y debe ser consciente del peligro que supone 'regalar' nuestros datos de una forma descontrolada a fenómenos sociales como Facebook. Redes Sociales de éste tipo son uno de los mayores azañas de nuestro tiempo, con un sinfín de ventajas, pero no es condición indispensable para no tener ningún control en nuestra actividad cotidiana en ella, ya que Facebook tiene infinitas posibilidades, pero estamos más tranquilos si aprovechamos esas posibilidades manteniendo nuestra información un poco más segura.

Apéndice A

SNSAngelGuardFB: Manual de usuario y configuración

A.1. Instalación

En ésta sección se explicará el proceso de instalación de las herramientas necesarias para la ejecución de la aplicación **SNSAngelGuardFB**. Todo éste proceso deberá hacerse en un servidor que proporcione los servicios necesarios para su correcta ejecución.

A.1.1. Requisitos técnicos

Los requisitos técnicos para llevar a cabo la instalación serán los siguientes:

1. **Equipo servidor** con estructura **Unix** que dé acceso a un Servidor Web y un Servidor de Base de Datos.
2. Motor de base de datos **MySql**, versión **5.6.11** o superior, instalado en el Servidor.
3. **Certificado digital**, correctamente firmado mediante la herramienta Java **keytool**, instalado en el equipo Servidor, para dar soporte **https**.
4. Servidor Web de Aplicaciones **Apache Tomcat**, versión **7.0.34** o superior, preparado para dar soporte **https**.
5. Gestor de repositorios **Maven**, versión **3.0.4** o superior.

A.1.2. Instalando la base de datos SocialNetwork

En el motor de la base de datos será necesario importar el fichero:

socialNetwork _ YYYYMMDD.sql

adjuntado en el disco de instalación de la aplicación. En el momento de la importación, será necesario crear el usuario **root** con la clave **tragasables**.

A.1.3. Instalación del Servidor Web Apache Tomcat

A.1.3.1. Apache Tomcat

Será necesario instalar correctamente el servidor web de aplicaciones **Apache Tomcat**, con una versión igual o superior a la versión **3.0.4**.

A.1.3.2. Modificación para el certificado digital

Una vez instalado el servidor, será necesario modificar su fichero **server.xml** para dar soporte al certificado digital creado. Para ello, bastará con tener a mano la ruta donde está el fichero creado por el certificado digital y su contraseña del almacen de claves. Una vez obtenidos estos parámetros e instalado el certificado digital con la herramienta Java **keytool**, la modificación que habrá que hacer en el fichero **server.xml** será incluir el siguiente fragmento de código:

```
<Connector SSLEnabled="true" acceptCount="100" clientAuth="false"
    disableUploadTimeout="true" enableLookups="false" maxThreads="25"
    port="8443"
    keystoreFile="/Users/josejavierblecuadepedro1/.keystore"
    keystorePass="blecua"
    protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="https"
    secure="true" sslProtocol="TLS" />
```

Donde los parámetros **keystoreFile** y **keystorePass** harán referencia, respectivamente, a la ruta al fichero de claves del certificado digital y la contraseña de acceso a éste.

A.1.4. Creación del fichero de configuración config.properties

Para la correcta ejecución de la aplicación, será necesaria la creación de un fichero de configuración que tomará todos los parámetros propios del servidor que la aplicación tomará, tales como su contexto de ejecución, la dirección de la base de datos o las propiedades del certificado digital.

El nombre del fichero de configuración será **config.properties** y habrá que instalarlo en la ubicación **/usr/local/snsangeguardfb/config**, creando para ello las carpetas necesarias. El fichero **config.properties** tendrá la siguiente información:

1. Parámetro **configHostApplication**: Contendrá la dirección host del servidor.
2. Parámetro **configHostApplicationSSL**: Contendrá la dirección host **SSL** del servidor **Apache Tomcat** instalado.
3. Parámetro **configHostRESTFullWS**: Contendrá la dirección donde está instalado el proyecto que contiene todos los servicios RESTful de la aplicación.
4. Parámetro **apiKey**: Identificador propio de la aplicación proporcionado por Facebook a la hora de dar de alta ésta en el espacio de aplicaciones de Facebook. Este identificador debe ser proporcionado por el creador de la aplicación.

5. Parámetro **apiSecretKey**: Identificador privado de la aplicación, proporcionado por Facebook. Al igual que el parámetro anterior, será proporcionado por el creador de la aplicación.
6. Parámetro **pathKeyStoreSSL**: Ruta al certificado digital creado con la herramienta **keytool** instalado en el servidor.
7. Parámetro **passwordKeyStoreSSL**: Contraseña del almacen de claves del certificado digital.
8. Parámetro **pathApplicationFacebook**: Contendrá la ruta en Facebook, proporcionada por ésta, de la aplicación.
9. Parámetro **pathLexicalFiles**: Contendrá la ruta en configuración a los ficheros de recursos de idioma para el filtro de control de lenguaje ofensivo.
10. Parámetro **googleAppName**: Contendrá el nombre de la aplicación dada de alta en el Google API Console.
11. Parámetro **googleClientId**: Identificador de la aplicación generado automáticamente al dar de alta la aplicación en Google API Console.
12. Parámetro **googleClientSecret**: Identificador secreto de la aplicación generado automáticamente al dar de alta la aplicación en Google API Console.

Estos parámetros serán los que estén informados en el fichero **config.properties**. Para un ámbito de ejecución local, el fichero podrá tomar los siguientes valores:

```
# EXECUTION SERVER HOST
configHostApplication = http://localhost/

# EXECUTION SERVER HOST
configHostApplicationSSL = https://localhost/

# DATA BASE SERVER HOST
configHostRESTFullWS = http://localhost/

# API Key Application Facebook
apiKey = 578871802160869

# API Secret Key Application Facebook
apiSecretKey = 7188d26c40d74492a0b40e4ee2a9f646

# PATH al certificado de confianza SSL
pathKeyStoreSSL = /Users/josejavierblecuadepedro1/.keystore

# PASSWORD del certificado de confianza SSL
passwordKeyStoreSSL = blecua
```

```

# Dirección de la aplicación en Facebook
pathApplicationFacebook = https://apps.facebook.com/dev_snsangelguard/

# Dirección al path de los ficheros de recursos de idioma
pathLexicalFiles = /usr/local/snsangelguardfb/config/lexicalFiles/

# Nombre de la aplicación dada de alta en Google API Console
googleAppName = SNSAngelGuardFB

# Identificador de la aplicación generado automáticamente al dar de alta la aplicación en G
googleClientId = 541921444258.apps.googleusercontent.com

# Identificador secreto de la aplicación generado automáticamente al dar de alta la aplicación
googleClientSecret = iU3fBdbX_2PFSTwsGpVQe-Dm

```

Nótese que los parámetros propios de Facebook, tales como **apiSecretKey**, **apiSecret-Key** y **pathApplicationFacebook**, están informados con los valores reales de la aplicación, por lo que al crear el fichero deberán tener exactamente los mismos valores que en éste ejemplo.

A.1.5. Creación del fichero de configuración de base de datos **pool.properties**

Para el mantenimiento óptimo de la aplicación y para no tener que recomilar todo el proyecto **SNSdataBaseIntegratorServer** cada vez que se cambia de base de datos, se ha optado por almacenar los parámetros de la base de datos en un archivo **.properties** para que pueda ser modificado en cualquier momento y no alterar el comportamiento de la aplicación.

Este archivo estará ubicado, al igual que en el caso anterior, en la carpeta del servidor de la aplicación **/usr/local/snsangelguardfb/config** bajo el nombre **pool.properties** y contendrá la siguiente información:

1. Parámetro **url**: Contendrá la dirección url en la que está instalada la base de datos. Al utilizar un driver **jdbc** para su gestión, deberá ir en éste formato.
2. Parámetro **user**: Contendrá el usuario de la base de datos en la que está instalada.
3. Parámetro **password**: Contendrá la contraseña del usuario de la base de datos en la que está instalada.
4. Parámetro **driver**: Contendrá la clase del driver que gestionará la conexión. En éste caso, contendrá el driver genérico **MySql**.

Para un ámbito local, el fichero **pool.properties** podría almacenar la siguiente información:

```
# Path a la base de datos
url = jdbc:mysql://localhost:3306/socialNetwork

# Usuario de la base de datos
user = pepito

# Password del usuario de la base de datos
password = xxxxxx

# Driver de conexión JDBC
driver = com.mysql.jdbc.Driver
```

A.1.6. Creación del fichero de filtros activos filters.xml

Este fichero es el que contendrá los filtros activos que van a ser chequeados por la aplicación. Deberá guardarse en la ruta `/usr/local/snsangelguardfb/config` bajo el nombre `filters.xml` y contendrá la siguiente información:

```
<filterList>
    <filter idFilter="FltWall" valueClass="es.uah.cc.ie.snsangelguardfb.sources.
        filtersfuncionality.WallFilterFuncionality" />
    <filter idFilter="FltFriends" valueClass="es.uah.cc.ie.snsangelguardfb.sources.
        filtersfuncionality.FriendsFilterFuncionality" />
    <filter idFilter="FltPriv" valueClass="es.uah.cc.ie.snsangelguardfb.sources.
        filtersfuncionality.PrivacyFilterFuncionality" />
    <filter idFilter="FltVist" valueClass="es.uah.cc.ie.snsangelguardfb.sources.
        filtersfuncionality.VisitsFilterFuncionality" />
</filterList>
```

A.1.7. Instalando harvestedSNSAngelGuard

Esta aplicación será la que controle la ejecución de los procesos `offline` programados. Para instalarlo correctamente, serán necesarios los siguientes pasos.

A.1.7.1. Instalación del proyecto harvestedSNSAngelGuardFB

Este proyecto será necesario copiarlo a la dirección `/usr/local/snsangelguardfb/harvested`, creando para ello las carpetas necesarias. Este proyecto tendrá una clase principal `Main` que lanzará la conexión con un servlet que conectará con el módulo servidor de la aplicación para comenzar su ejecución, por lo que es de vital importancia.

A.1.7.2. Creación de la carpeta de logs

Todos los ficheros `.log` que se generen por la ejecución de los procesos `offline` será necesario almacenarlos. Para ello, será necesario crear la carpeta `logs` en la siguiente dirección:

```
/usr/local/snsangeguardfb/harvested/logs
```

Los ficheros que se generen tendrán el siguiente formato ya configurado dentro del script de ejecución:

```
execute_harvesting_SNSAngelGuardFB_YYYY_MM_DD.log
```

A.1.7.3. Modificación del fichero harvestedSNSAngelGuardFB.sh

Tras copiar el proyecto anterior a la dirección indicada, será necesario modificar el script que lanzará el proceso **offline** de actualización diaria de datos. El fichero **harvestedSNSAngelGuardFB.sh** se encontrará en la dirección:

```
/usr/local/snsangeguardfb/harvested/harvestedSNSAngelGuard/script
```

La modificación a realizar constará de, en la linea donde se llama a la clase **Main** del proyecto anterior, añadirle los siguientes parámetros:

1. Parámetro host donde se está ejecutando el servidor: Este será su primer parámetro. Notesé que deberá llamarse con la dirección host SSL para su funcionamiento óptimo.
2. Ruta al fichero almacén de claves del certificado digital: Este será su segundo parámetro
3. Contraseña del fichero almacén de claves del certificado digital.

Un ejemplo de llamada será el siguiente:

```
es.uah.cc.ie.harvestedsnsangelguard.Main
https://localhost:8443/
/Users/josejavierblecuadepedro1/.keystore
blecua
```

Y en su totalidad, el fichero **harvestedSNSAngelGuardFB.sh** tendrá la siguiente configuración para un ámbito local:

```
#!/bin/bash

DATE=$(date +%Y_%m_%d);
logFile=/usr/local/snsangeguardfb/harvested/logs/
execute_harvesting_SNSAngelGuardFB_$DATE\.log;
cd /usr/local/snsangeguardfb/harvested/harvestedSNSAngelGuard/
mvn clean dependency:copy-dependencies package >> $logFile
mvn exec:java -Dexec.mainClass="es.uah.cc.ie.harvestedsnsangelguard.Main"
-Dexec.args="http://localhost:8080/ /Users/user/.keystore passuser" >> $logFile
```

A.1.7.4. Modificación del cron del sistema

Será necesario modificar el fichero **cron** del sistema para que lance como proceso automático el fichero **harvestedSNSAngelGuardFB.sh**. Este proceso se lanzará todos los días a las 03 A.M., por lo que se tendrá que añadir al fichero cron una linea con las siguientes características:

```
00 3 * * * root /usr/local/snsangelguardfb/harvested/harvestedSNSAngelGuard/
script/harvestedSNSAngelGuardFB.sh
```

En el cual se está indicando al **cron** del sistema que deberá lanzar dicha tarea a las tres de la mañana cada día.

A.1.8. Instalando SNSdataBaseIntegratorServer

Este proyecto será el que controle los **Servicios RESTFul** que se pondrán a disposición de los programas aplicaciones para acceder a la base de datos **SocialNetwork**. Se comprobará que en su fichero de configuración **persistence.xml**, en el apartado de configuración de la base de datos, la configuración que apunta a ésta esté correcta, tanto su URL como el usuario y la contraseña, anteriormente indicada.

El fichero **SNSdataBaseIntegratorServer.war**, contenido en el disco de instalación tendrá esta configuración. Si por cualquier motivo, se cambiara de servidor, habría que volver a configurar el fichero **persistence.xml** y generar de nuevo el fichero war. Una vez generado, se subirá al Servidor Web de aplicaciones **Apache Tomcat** y se lanzará en ejecución.

A.1.9. Instalando SNSAngelGuardFB

Será el proyecto principal. Estará contenido en el archivo **SNSAngelGuardFB.war**, que se subirá al Servidor Web de aplicaciones **Apache Tomcat** y se lanzará en ejecución.

A.2. SNSAngelGuardFB: Manual de usuario

En ésta sección se describirán las principales pantallas de la aplicación, para su instalación y el alta de un nuevo usuario.

A.2.1. Alta de un nuevo usuario

Para acceder a la instalación mediante un navegador de internet, preferiblemente **Google Chrome**¹ o **Mozilla Firefox**, buscando en el App Center de Facebook o en Google la aplicación **SNSAngelGuard**.

¹La aplicación ha sido diseñada para éste tipo de navegador, garantizando el correcto funcionamiento en él de la totalidad de sus estructuras. Recomendamos el uso de éste navegador, ya que no se garantiza el correcto funcionamiento de la aplicación en el resto de navegadores

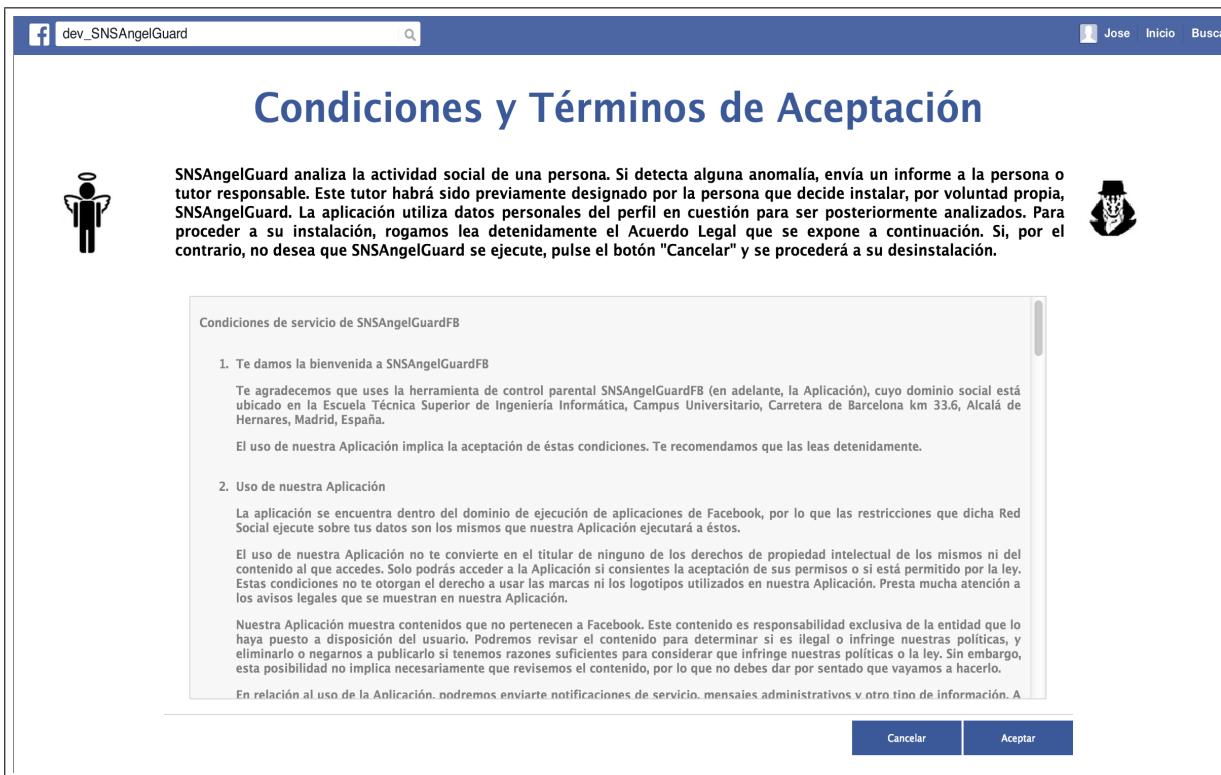


Figura A.1: Pantalla de Aceptación del Acuerdo Legal.

Una vez accedida a la aplicación, tocará aceptar el **Acuerdo Legal**, propio de la aplicación, mostrado en la imagen A.1.

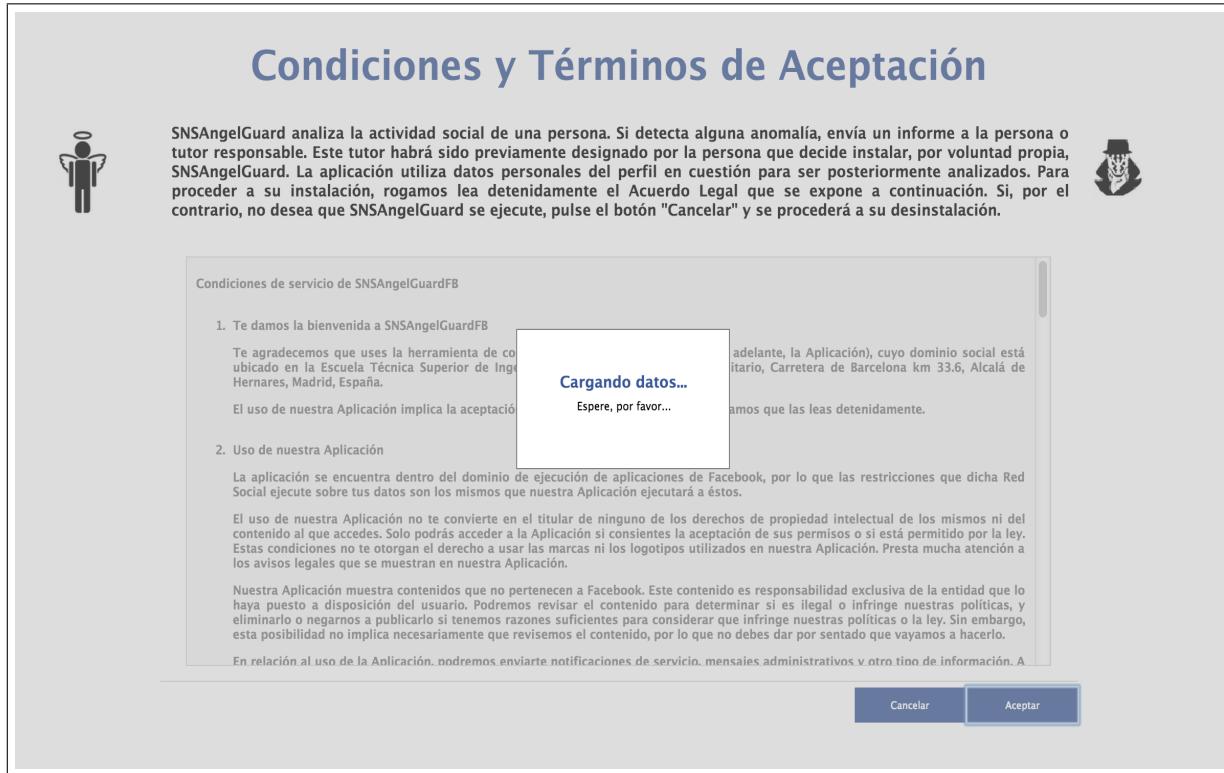


Figura A.2: Pantalla de Aceptación del Acuerdo Legal, con loader.

Éste se aceptará pulsando el botón **Aceptar**. En ese momento se empezará a descargar información del perfil del usuario de Facebook y aparecerá el **loader** de la aplicación mientras éste proceso esté activo, como muestra la imagen A.2. Si, por el contrario, el usuario no acepta el acuerdo legal, volveremos a la pantalla principal de Facebook y se terminará la ejecución.



Figura A.3: Pantalla de Selección de Ángeles inicial.

Una vez almacenados los datos del usuario en la base de datos, se cargará la pantalla principal de configuración de la aplicación. Ésta inicialmente mostará la pestaña de Selección de Ángeles, como se muestra en la imagen A.3. En ésta pantalla deberemos elegir los ángeles que formarán parte de nuestra configuración para que empiecen a recibir notificaciones producidas por anomalías detectadas en el perfil social del usuario en Facebook.

250APÉNDICE A. SNSANGELGUARDFB: MANUAL DE USUARIO Y CONFIGURACIÓN

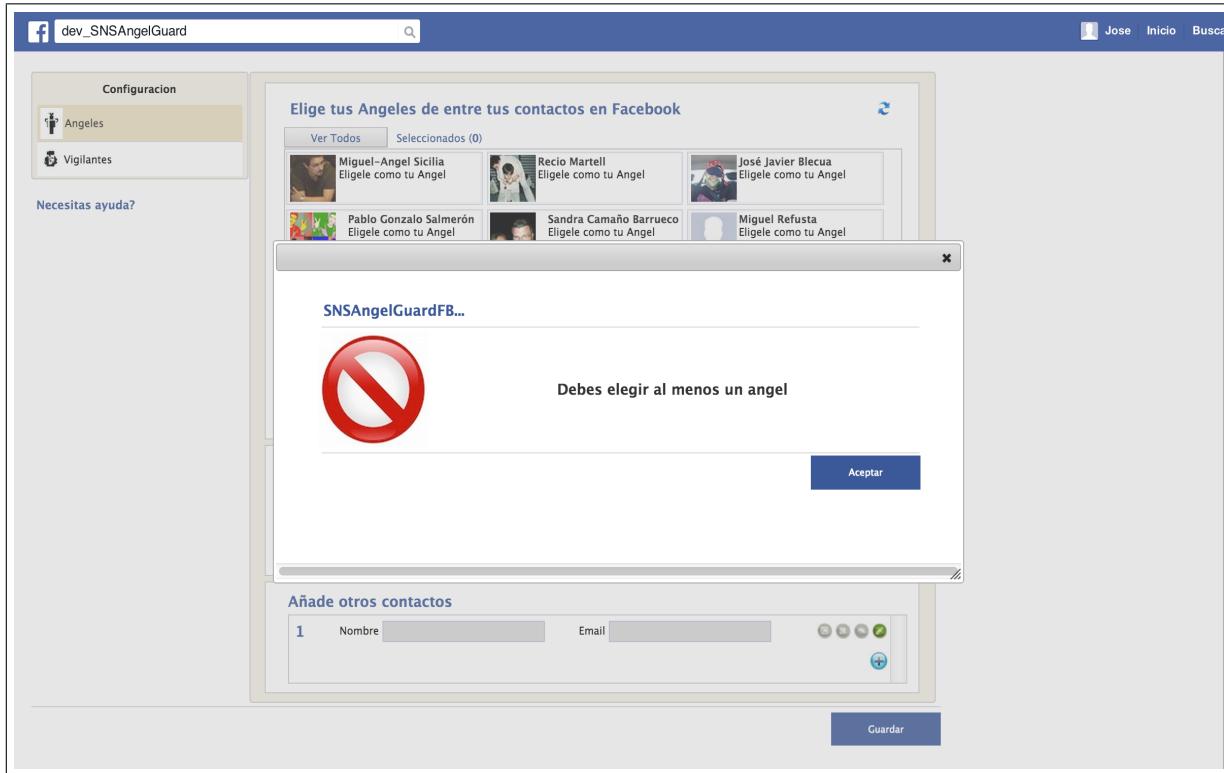


Figura A.4: Mensaje de aviso para seleccionar ángeles.

Notesé que no se podrá pasar a la pestaña de **Configuración de Vigilantes** si no se ha elegido ningún ángel. Si es así, al pulsar la pestaña **Vigilantes** se lanzará un mensaje al usuario que le indicará que debe elegir al menos un ángel para poder continuar, como muestra la imagen A.4.

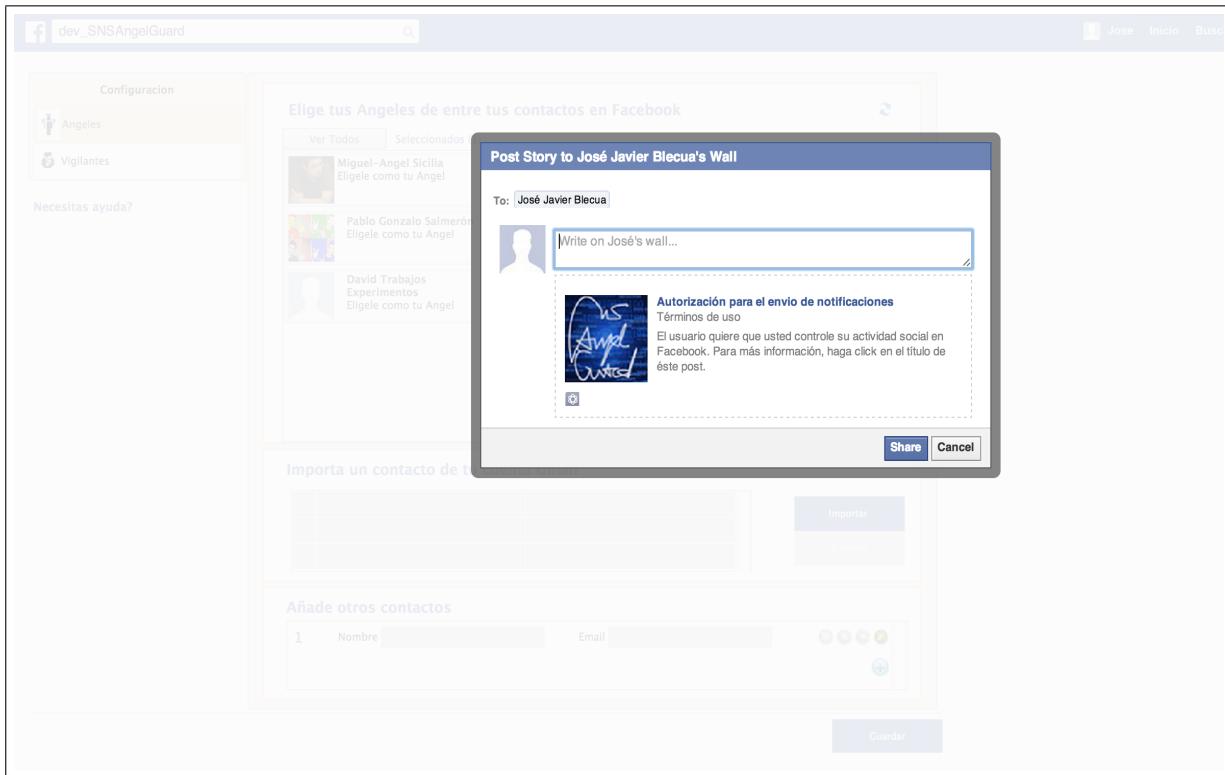


Figura A.5: Mensaje de post en Facebook.

Un ángel se podrá seleccionar de tres formas:

1. Seleccionando uno o varios contactos de Facebook en el contenedor superior de la página.
2. Importando uno o varios contactos de una cuenta válida de Gmail en el contenedor de mitad de página.
3. Introduciendo manualmente uno o varios contactos con direcciones válidas de correo electrónico en el contenedor inferior de la página.

Para seleccionar un contacto de Facebook, pulsamos sobre uno de nuestros contactos y se habilitará la estructura que Facebook pone a nuestra disposición para postear en su muro. Como desde la aplicación no podemos acceder al email de un amigo de Facebook, será necesario que éste pulse en el enlace adjunto al post para introducir su email y que se le empiecen a mandar notificaciones: A.5.

Para seleccionar un contacto de **Gmail**, pulsaremos el botón **Importar**, que abrirá una ventana modal con la información contenida en la imagen A.6.



Figura A.6: Ventana de selección de contactos de Gmail inicial.

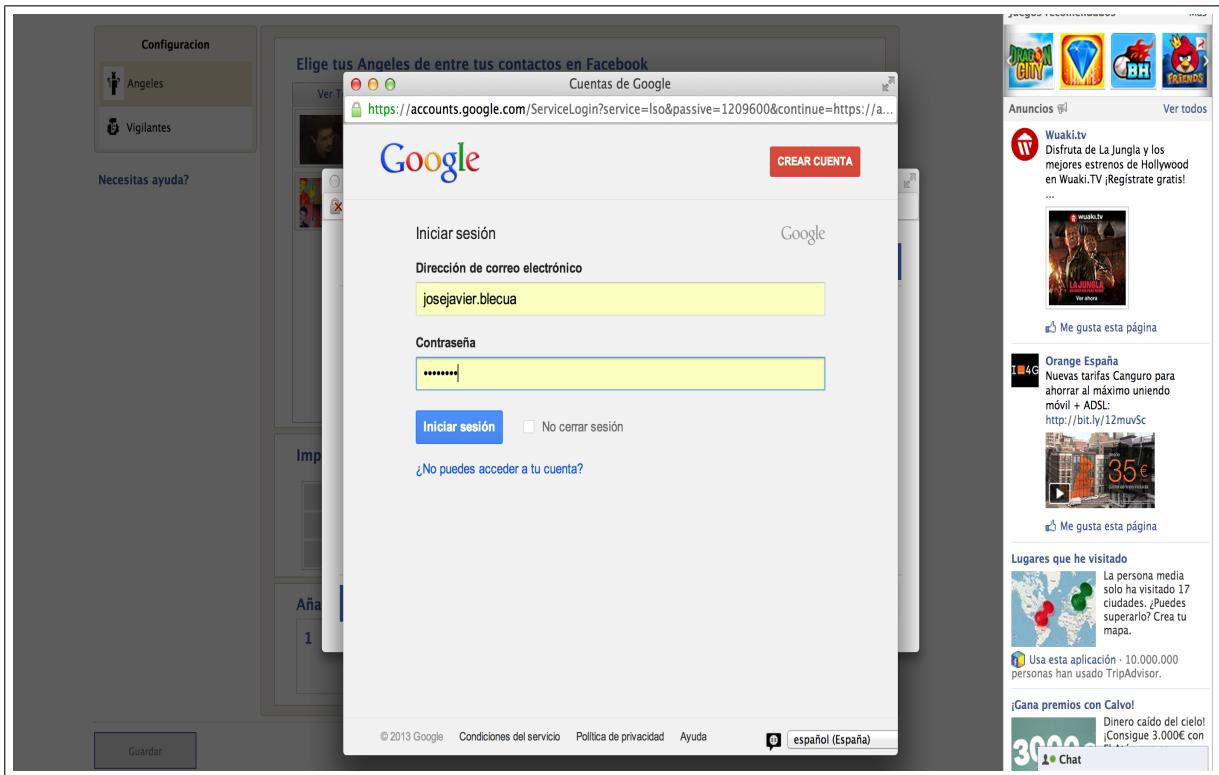


Figura A.7: Ventana de Login de Gmail.

Una vez abierta, nos aparecerá la ventana sin ningún contacto cargado. Para ello, será necesario iniciar sesión en **Gmail** pulsando en el botón **Iniciar Sesión Google** e introduciendo una cuenta válida, como se muestra en la imagen A.7. Esta pantalla pertenecerá a Google, es decir, nos logueamos directamente en Gmail, nunca se introducirá la información dentro de la aplicación, ya que éstos datos no nos pertenecen.



Figura A.8: Selección de un contacto de Gmail.

Una vez iniciada sesión, se cargarán todos los contactos de la cuenta en la ventana modal inicial, y se podrá seleccionar cualesquiera de ellos, como se muestra en la imagen A.8.



Figura A.9: Pantalla de Selección de Ángeles con un contacto de Gmail.

Al pulsar el botón **Aceptar**, se retornará a la ventana de Selección de Ángeles con el contacto cargado, como muestra la imagen A.9.

256APÉNDICE A. SNSANGELGUARDFB: MANUAL DE USUARIO Y CONFIGURACIÓN

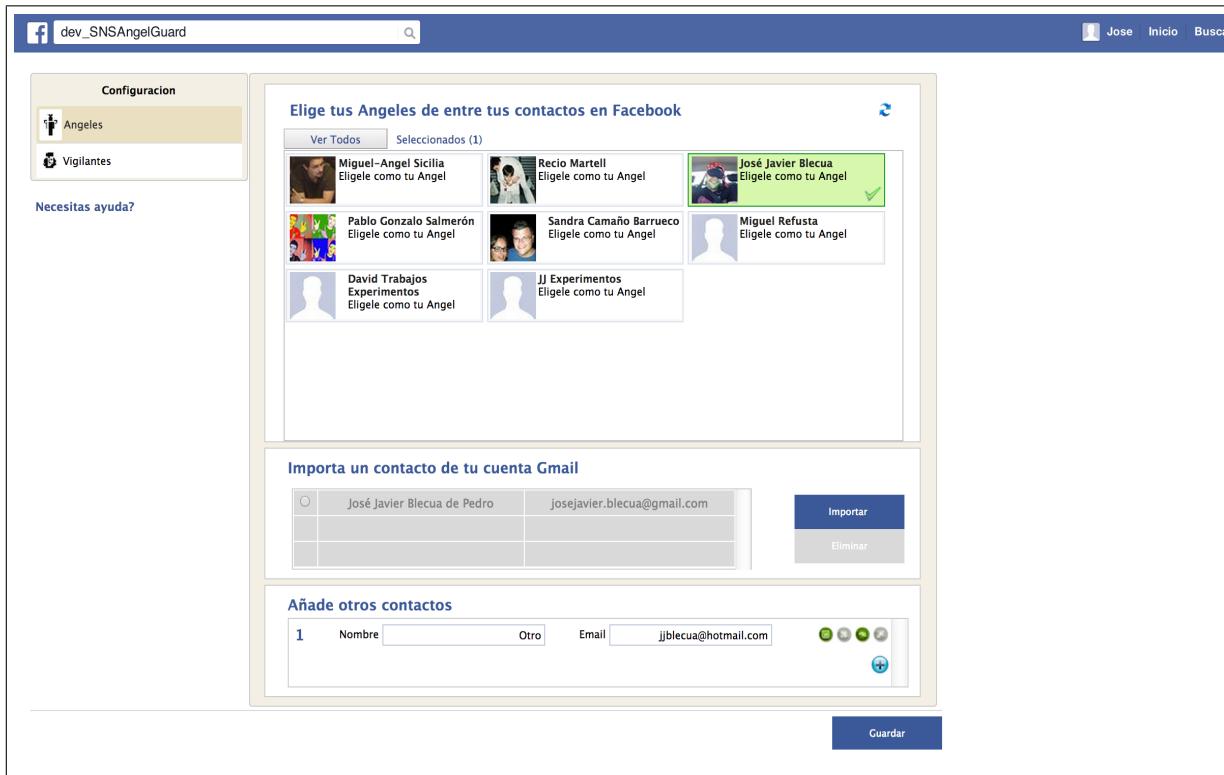


Figura A.10: Selección de un nuevo contacto manual habilitado.

La última forma de introducir un contacto, será introduciéndolo manualmente en el contenedor inferior de la página. Como hemos podido ver en las anteriores imágenes, los campos **Nombre** y **Email** se encuentran deshabilitados. Para habilitarlos, pulsaremos el botón **Editar**, como se muestra en la imagen A.10.

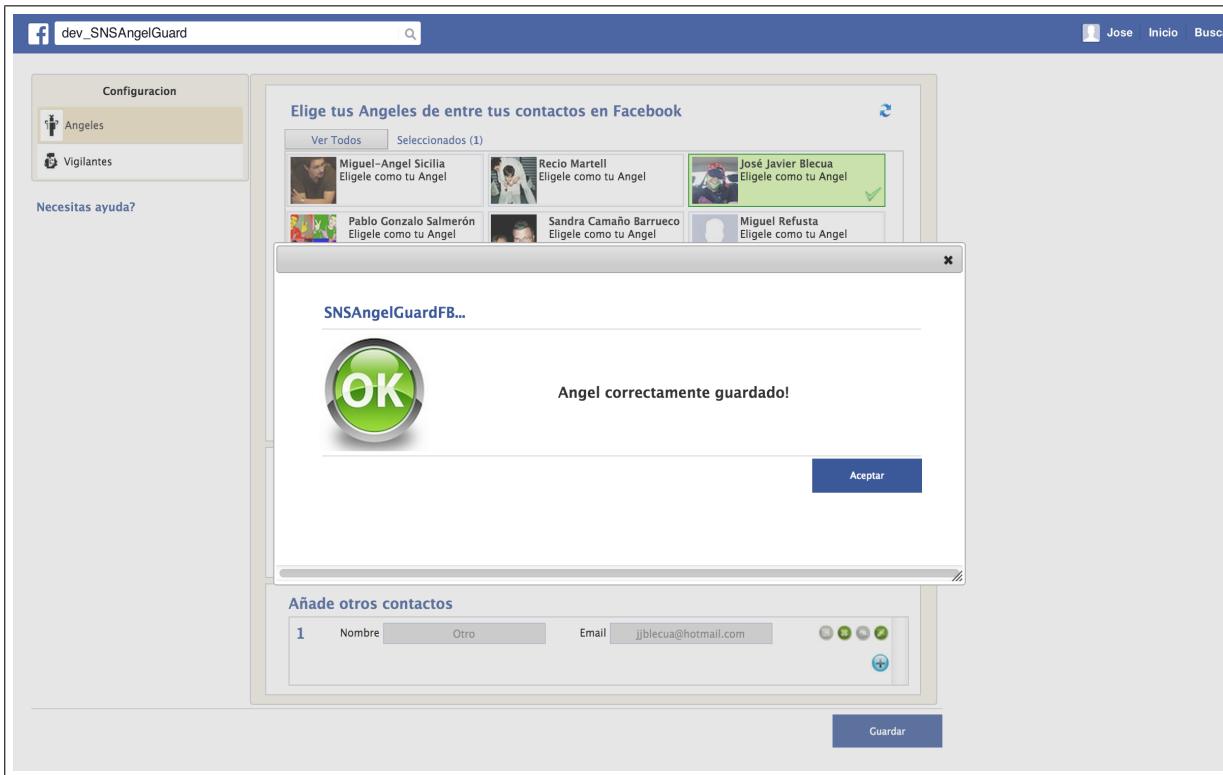


Figura A.11: Mensaje de confirmación para nuevo ángel introducido manualmente.

En éste caso, se habilitarán los botones **Cancelar**, que borrará el contenido de los campos de texto y deshabilitará los campos, y **Guardar**, que almacenará el ángel para poder asociarlo con algún vigilante. Si pulsamos éste último, se mostrará un mensaje informando al usuario de que se ha almacenado correctamente el nuevo ángel, mostrado en la imagen A.11.

258APÉNDICE A. SNSANGELGUARDFB: MANUAL DE USUARIO Y CONFIGURACIÓN



Figura A.12: Habilitada una nueva estructura para introducir un nuevo ángel manualmente.

Si se desea introducir más ángeles manualmente, bastará con pulsar el botón + y se habilitarán tantas estructuras como deseemos, de la manera indicada en la imagen A.12.



Figura A.13: Pantalla inicial de Configuración de Vigilantes.

Tras la elección de los ángeles, se pasará a la pestaña de **Configuración de Vigilantes**, como se muestra en la pantalla A.13.

260APÉNDICE A. SNSANGELGUARDFB: MANUAL DE USUARIO Y CONFIGURACIÓN

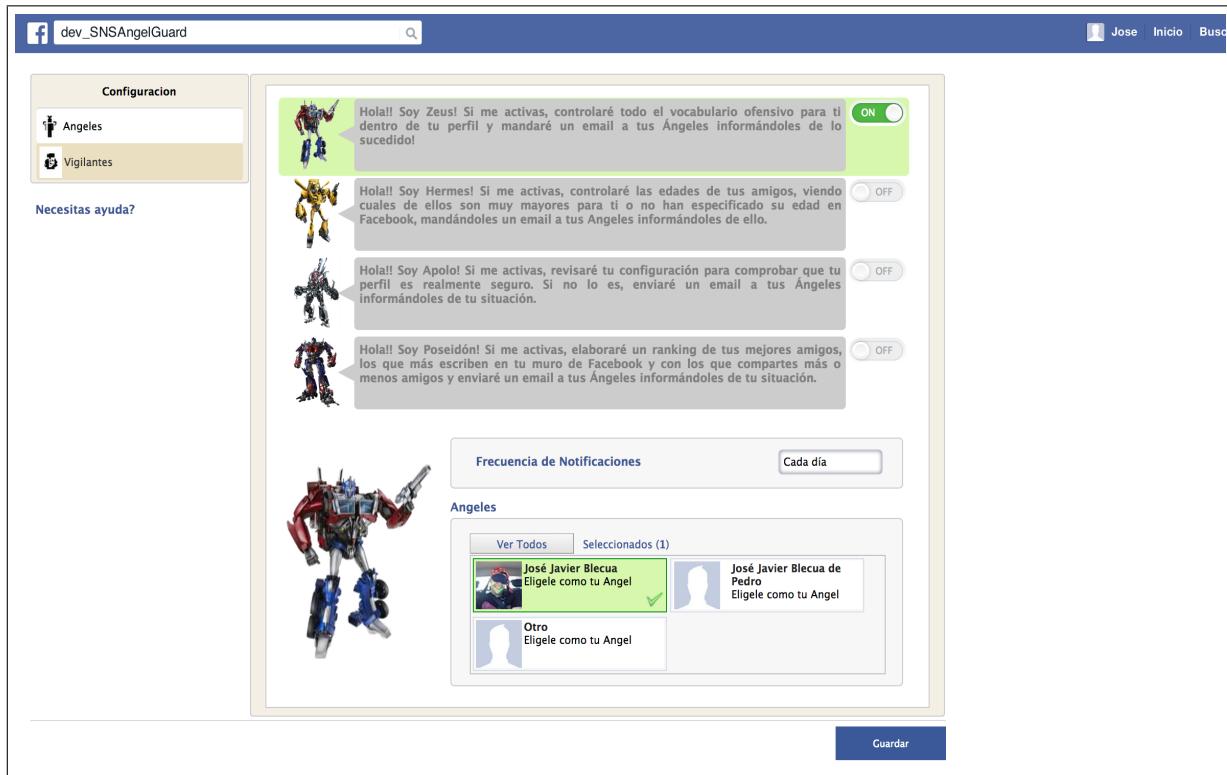


Figura A.14: Configuración del filtro de Control de Muro.

Para poder configurar un vigilante, será necesario activar su switch. A partir de ese momento, la parte inferior de la pantalla se habilitará y se podrá elegir la siguiente información:

1. La frecuencia determinada dentro de la lista desplegable de selección de frecuencia.
2. Elegir los ángeles que formarán parte de la configuración del filtro.

Este proceso esta indicado en la imagen A.14, donde se muestra la configuración seleccionada para el filtro de **Control de Muro**. Es análogo para el resto de filtros.

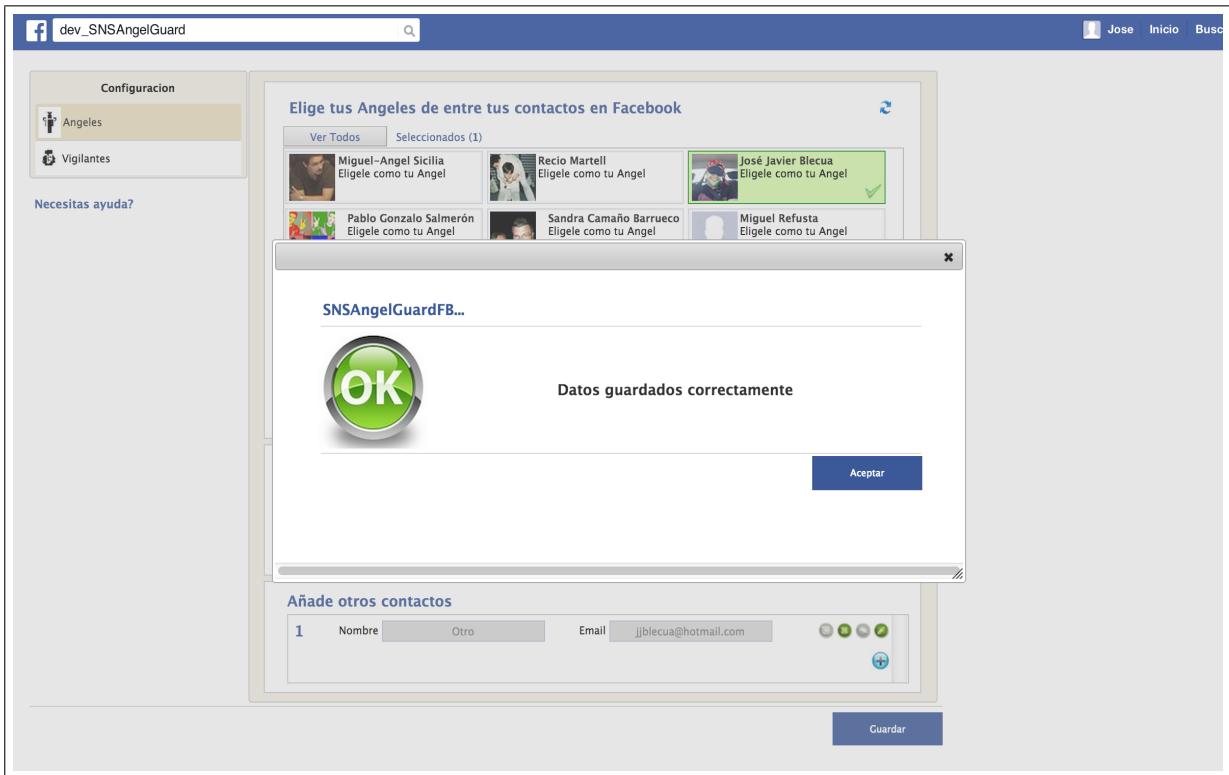


Figura A.15: Datos guardados correctamente.

Una vez seleccionados todos los filtros, se pulsará el botón **Guardar**, en el que se almacenarán todos los datos configurados. Una vez almacenados todos los datos, se volverá a la pantalla principal de la aplicación con la nueva configuración almacenada y mostrando un mensaje indicando que el proceso se ha realizado satisfactoriamente, como muestra la imagen A.15.

262APÉNDICE A. SNSANGELGUARDFB: MANUAL DE USUARIO Y CONFIGURACIÓN



Figura A.16: Pantalla inicial para usuarios de la aplicación.

A.2.2. Acceso de usuarios ya dados de alta

Para éstos usuarios, se cargará directamente la página principal de **Selección de Ángeles** con su configuración cargada, de la manera que se indica en la imagen A.16.



Figura A.17: Notificación de confirmación de ángel.

A.2.3. Confirmación de un ángel y envío de notificaciones

Una vez seleccionados los ángeles, a cada uno se le enviará una notificación de confirmación del usuario como muestra la imagen A.17

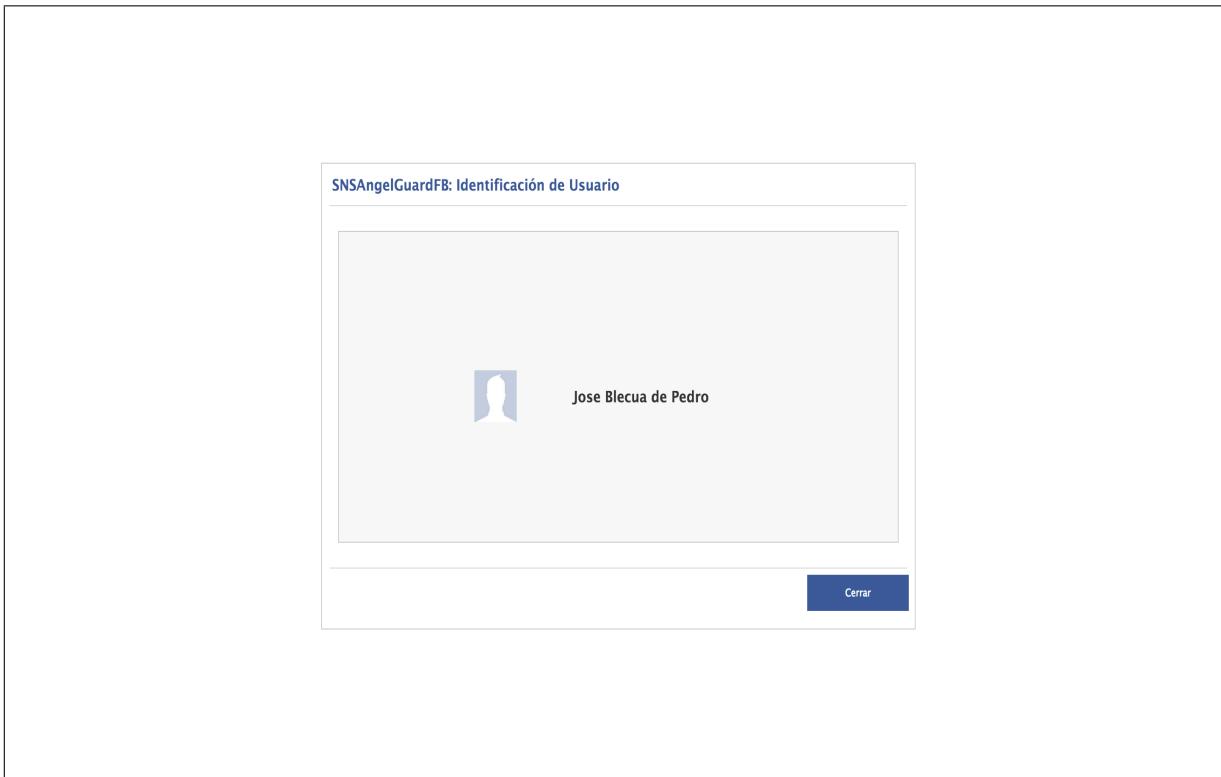


Figura A.18: Pantalla de Identificación del usuario

Si se quiere consultar la información del usuario que ha requerido de su tutelaje, basta con pulsar el enlace en el nombre del usuario de la notificación y se abrirá la siguiente pestaña en el navegador, mostrada en la imagen A.18.

Los ángeles seleccionados de Facebook deberán pulsar el enlace que la aplicación ha escrito en su muro y pasar por una pantalla intermedia para introducir su email, de la forma en la que indica la imagen A.19.



Figura A.19: Pantalla de Identificación del angel de Facebook

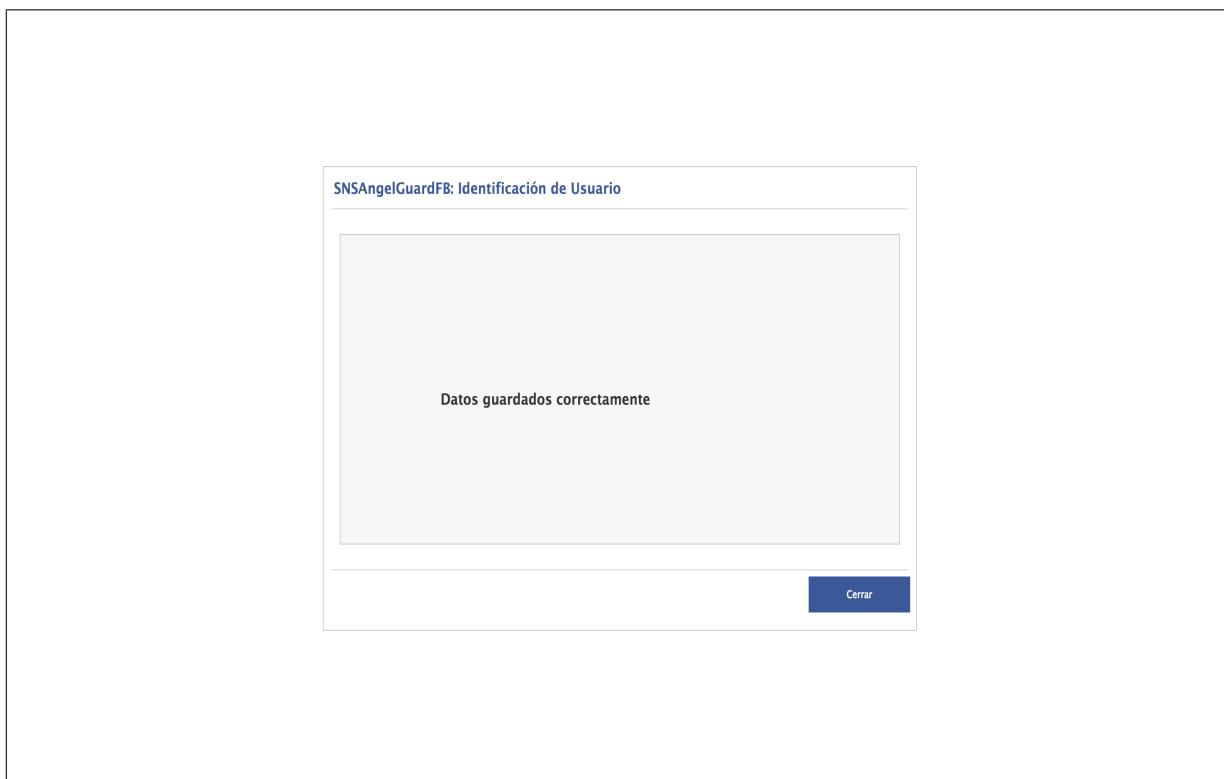


Figura A.20: Pantalla de Resultado a la confirmación de un ángel

Una vez confirmado a un ángel, al usuario se le presentará el resultado de la confirmación, en éste caso OK, mediante la pantalla A.20.

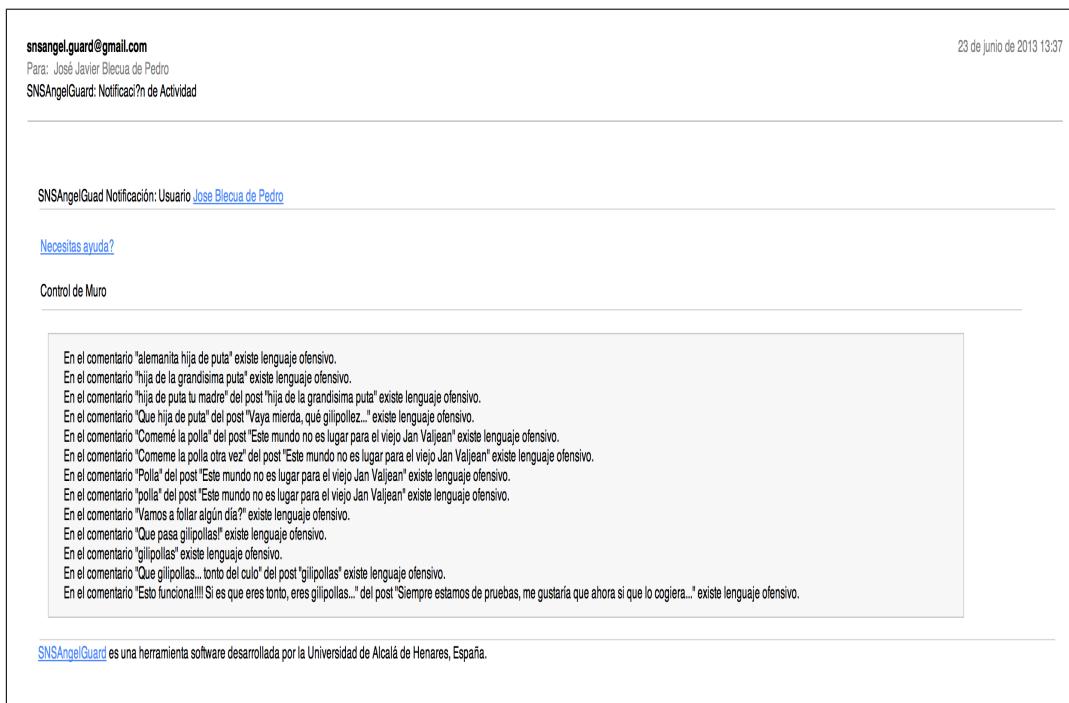


Figura A.21: Notificación de actividad.

Una vez procesada la confirmación, se le enviará al ángel un correo con el análisis de toda la información almacenada en la base de datos y procesada por los filtros. Esta notificación será del tipo mostrado en la imagen A.21. A partir de éste momento, el ángel quedará activado a la espera de notificaciones de información con la periodicidad configurada para él por parte del usuario en la pantalla de **Configuración de Vigilantes**.

Apéndice B

Base de Datos SocialNetwork: Tablas

En éste apéndice se enumerarán las tablas físicas del modelo de base de datos **SocialNetwork** divididas por módulos funcionales, tal y como están organizadas en el capítulo 4.

Tabla B.2: Tabla locale_settings

Campo	Tipo	Descripción
id_locale	VARCHAR(100)	Identificador único de la tabla.
acceptingTerms	VARCHAR(1000)	Cabecera del acuerdo legal.
legalAccepted	MEDIUMTEXT	Texto de aceptación de los términos de ejecución de la aplicación.
btnAgreeAT	VARCHAR(100)	Título del botón ‘Aceptar’ de la página de Aceptación de Términos.
btnCancelAT	VARCHAR(100)	Título del botón ‘Cancelar’ de la página de Aceptación de Términos.
titleAT	VARCHAR(100)	Título de la página de Aceptación de Términos.
titleSettings	VARCHAR(100)	Título de la página contenedora de la aplicación.
titleMenSettings	VARCHAR(100)	Título de las pestañas de la página contenedora.
btnSaveCheckSettings	VARCHAR(100)	Título del botón ‘Guardar’ de la página de contenedora de la aplicación.
titleSettAng	VARCHAR(100)	Título principal de la página de la pestaña ‘Ángeles’
titleFriendsContentSettAng	VARCHAR(100)	Título del frame de selección de ángeles de Facebook

Campo	Tipo	Descripción
titleFriendsImportSettAng	VARCHAR(100)	Títulos de los frames de selección de contactos de Google u otros contactos
txtNameTutorSettAng	VARCHAR(100)	Título para el nombre de un contacto del frame ‘Otros contactos’.
txtEmailTutorSettAng	VARCHAR(100)	Título para el email de un contacto del frame ‘Otros contactos’.
btnImportSettAng	VARCHAR(100)	Título del botón ‘Importar contactos de Google’.
titleFbListSettAng	VARCHAR(100)	Contiene los nombres de las pestañas que visualizan los contactos de Facebook.
subTitleAngelSettAng	VARCHAR(100)	Contiene los subtítulos que aparecen en cada contacto de Facebook.
titleSettVig	VARCHAR(100)	Título de la página de configuración de los vigilantes.
titleVigilantSettVig	VARCHAR(100)	Título del frame ‘Vigilantes’ de la pestaña ‘Vigilantes’.
titleVigSettVig	VARCHAR(100)	Título, en formato JSON, de los filtros vigilantes disponibles.
titleVigDescriptionSettVig	MEDIUMTEXT	Descripciones, en formato JSON, de cada vigilante.
titleVigFrecSettVig	VARCHAR(100)	Título de la lista desplegable ‘Frecuencia’.
titleVigFrecSelectSettVig	VARCHAR(100)	Valores disponibles de la lista desplegable ‘Frecuencia’.
titleVigAngSettVig	VARCHAR(100)	Título del frame ‘Ángeles’ de la pestaña ‘Vigilantes’.
titleAngConfirm	VARCHAR(100)	Título del email de confirmación enviado a un ángel.
desInfoAngConfirm	VARCHAR(1000)	Cuerpo del email de confirmación enviado a un ángel.
aceptConfAngConfirm	VARCHAR(100)	Título del link de confirmación enviado en el email a un ángel.
cancelConfAngConfirm	VARCHAR(100)	Título del link de cancelación enviado en el email a un ángel.
infoAngGuard	VARCHAR(100)	Información acerca de la aplicación enviada en cada email.
titleAngUser	VARCHAR(100)	Título de la página de información acerca del usuario que pide confirmación a un ángel.

Campo	Tipo	Descripción
nameUserAngUser	VARCHAR(100)	Título del nombre del usuario de Facebook.
btnCloseAngUser	VARCHAR(100)	Título del botón ‘Cerrar’ de la página de información de un usuario de Facebook.
titleGoogleCont	VARCHAR(100)	Título de la página de importación de contactos de Google.
titleContGoogleCont	VARCHAR(100)	Subtítulo de la página de importación de contactos de Google.
btnLogGoogleCont	VARCHAR(100)	Título del botón de login en Google.
titleNameContactGoogleCont	VARCHAR(100)	Nombre del contacto de Google.
titleEmailContactGoogleCont	VARCHAR(100)	Email del contacto de Google.
btnAcceptGoogleCont	VARCHAR(100)	Título del botón ‘Aceptar’ de la página de importación de contactos de Google.
btnCancelGoogleCont	VARCHAR(100)	Título del botón ‘Cancelar’ de la página de importación de contactos de Google.
helpMe	MEDIUMTEXT	Contenido de la página de ayuda de la aplicación.
warnings	MEDIUMTEXT	Contenido de los mensajes de aviso y errores que pueden producirse en la aplicación.
titleInformationMessage	VARCHAR(100)	Título para los mensajes de aviso y errores.
informationMessage	MEDIUMTEXT	Contenido de las páginas de confirmación offline para los ángeles.
mailDelete	MEDIUMTEXT	Contenido del email de eliminación de un ángel por parte de un usuario.
mailNotification	MEDIUMTEXT	Contenido del email de notificación de actividad de un usuario hacia sus ángeles.
altContactsAngelsEd	MEDIUMTEXT	Títulos para los botones de acción para otros contactos.
titleVisitsFilterOptions	MEDIUMTEXT	Títulos para las diferentes estadísticas utilizadas en el filtro de visitas.
post_friend_facebook	MEDIUMTEXT	Títulos para los post de Facebook.
titleActiveDesactiveVig	VARCHAR(45)	Títulos de activación para los vigilantes.

Campo	Tipo	Descripción
uid	VARCHAR(100)	Contiene el identificador del usuario en la aplicación. Correspondrá al usuario de Facebook.
user_name	LONGTEXT	Nombre del usuario
user_email	LONGTEXT	Email del usuario
legal_accepted	VARCHAR(1)	Indicador de aceptación del acuerdo legal por parte del usuario.
last_check	TIMESTAMP	Fecha de la última ejecución de la aplicación cliente.
uid_public	LONGTEXT	Identificador público del usuario.
app_activated	VARCHAR(1)	Indica si la aplicación está activada o no.
user_session	LONGTEXT	Token de sesión de Facebook para accesos offline.
locale_settings_id_locale	VARCHAR(100)	Índice de la tabla locale_settings que indica el idioma en el que se va a mostrar la aplicación.
bakup_check	TIMESTAMP	Indica la última actualización de información del usuario.

Tabla B.1: Tabla user_settings

Campo	Tipo	Descripción
uid_angel	INTEGER	Identificador único del ángel en la tabla.
id_angel	VARCHAR(100)	Id del ángel, es decir, su email.
name_angel	LONGTEXT	Nombre del ángel.
img_angel	LONGTEXT	Imagen del ángel.
type_angel	VARCHAR(10)	Tipo del ángel.
accept_angel	VARCHAR(1)	Indica si el ángel ha aceptado seguir a un usuario o no. Se representará con el valor 1 en caso afirmativo y 0 en caso negativo.
user_prop_angel	VARCHAR(100)	Usuario de la aplicación propietario del ángel.
confirm_angel	VARCHAR(1)	Indica si el ángel ha respondido a la solicitud. Se representará con el valor 1 en caso afirmativo y 0 en caso negativo.
id_facebook	VARCHAR(150)	Identificador del ángel en Facebook. Únicamente se llenará si el angel es un contacto de Facebook.

Tabla B.3: Tabla settings_angels

Tabla B.6: Tabla user_openSocial

Campo	Tipo	Descripción
id_user_openSocial	VARCHAR(100)	Identificador único de la tabla.
AGE	INTEGER	Edad del usuario.
BODYTYPE_BUILD	VARCHAR(100)	Complejión del usuario.
BODYTYPE_EYE_COLOR	VARCHAR(45)	Color de ojos.
BODYTYPE_HAIR_COLOR	VARCHAR(45)	Color de pelo.
BODYTYPE_HEIGHT	VARCHAR(45)	Altura.
BODYTYPE_WEIGHT	VARCHAR(45)	Peso.
CARS	VARCHAR(450)	Tendencias sobre coches.
CHILDREN	VARCHAR(450)	Hijos del usuario.
Address_openSocial_CURRENT_LOCATION	VARCHAR(10)	Dirección actual.
DATE_OF_BIRTH	DATE	Fecha de nacimiento.
Drinker_openSocial_id_Drinker_openSocial	INTEGER	Grado de alcoholismo.

Campo	Tipo	Descripción
ETHNICITY	VARCHAR(450)	Raza étnica.
FASHION	VARCHAR(450)	Tendencias sobre moda.
FOOD	VARCHAR(450)	Tendencias gastronómicas.
Gender_openSocial_id_Gender_openSocial	INTEGER	Sexo del usuario.
HAPPIEST_WHEN	VARCHAR(450)	Actividades favoritas.
HEROES	VARCHAR(450)	Heroes.
HUMOR	VARCHAR(450)	Tendencias humorísticas.
JOB_INTERESTS	VARCHAR(450)	Actividad laboral.
LANGUAGES_SPOKEN	VARCHAR(450)	Idiomas hablados por el usuario.
LIVING_ARRANGEMENT	VARCHAR(450)	Alojamiento.
LookingFor_openSocial_id_LookingFor_openSocial	INTEGER	Relaciones sociales sobre las que está interesado.
Presence_openSocial_id_Presence_openSocial	INTEGER	Presencia actual en la red.
NICKNAME	VARCHAR(100)	Apodo.
PETS	VARCHAR(450)	Mascotas.
PROFILE_URL	VARCHAR(100)	URL de la foto de perfil del usuario.
ROMANCE	VARCHAR(100)	Relación actual.
SCARED_OF	VARCHAR(450)	Miedos o temores.
Smoker_openSocial_id_Smoker_openSocial	INTEGER	Fumador.
SPORTS	VARCHAR(450)	Actividades deportivas del usuario.
TAGS	VARCHAR(450)	Etiquetas asociadas al usuario.
THUMBNAIL_URL	VARCHAR(100)	URL a la imagen thumbnail del usuario.
TIME_ZONE	DATE	Franja horaria en la que se encuentra el usuario.
TURNS_OFFS	VARCHAR(450)	Tendencias que desagradan al usuario.
TURNS_ONS	VARCHAR(450)	Tendencias que sorprenden al usuario.
URLS	VARCHAR(450)	URLs de interés del usuario.

Campo	Tipo	Descripción
id_filter	INTEGER	Identificador del filtro único en la aplicación.
frec_filter	VARCHAR(100)	Frecuencia a la que se ejecutará el filtro.
active_filter	VARCHAR(1)	Indicador de actividad del filtro. El valor 1 indicará que el filtro se encuentra activo y el valor 0 indicará que no se encuentra activado.
last_check	TIMESTAMP	Fecha de la última ejecución del filtro.
type_filter	VARCHAR(200)	Tipo de filtro del usuario.

Tabla B.4: Tabla settings_filter

Campo	Tipo	Descripción
user_settings_uid	VARCHAR(1000)	Identificador único del usuario de la aplicación.
SEX	VARCHAR(45)	Sexo del usuario de la aplicación.
RELIGION	LONGTEXT	Tendencias religiosas del usuario.
RELATIONSHIP_STATUS	LONGTEXT	Estado sentimental.
POLITICAL	LONGTEXT	Tendencias políticas.
ACTIVITIES	LONGTEXT	Actividades que realiza el usuario.
INTERESTS	LONGTEXT	Intereses del usuario.
IS_APP_USER	TINYINT	Indicador del perfil activo dentro de la Red Social.
MUSIC	LONGTEXT	Tendencias musicales del usuario.
TV	LONGTEXT	Programas televisivos que sigue o ha seguido actualmente.
MOVIES	LONGTEXT	Películas favoritas.
BOOKS	LONGTEXT	Libros favoritos.
ABOUT_ME	LONGTEXT	Descripción propia del usuario dentro de su perfil en la Red Social.
STATUS	LONGTEXT	Estatus actual del usuario.
QUOTES	LONGTEXT	Citas o frases famosas que definen a un usuario.
user_facebook_id_user_facebook	VARCHAR(100)	Identificador de su usuario en Facebook.
user_openSocial_id_user_openSocial	VARCHAR(100)	Identificador de su usuario en OpenSocial.

Tabla B.5: Tabla user

Campo	Tipo	Descripción
id_Phones_openSocial	INTEGER	Identificador único de la tabla.
NUMBER	VARCHAR(45)	Numero de teléfono asociado al usuario.
TYPE	VARCHAR(45)	Tipo de teléfono(fijo, móvil, etc.).
user_openSocial_id_user_openSocial	VARCHAR(100)	Identificador de su usuario en OpenSocial.

Tabla B.7: Tabla Phones_openSocial

Campo	Tipo	Descripción
id_Smoker_openSocial	INTEGER	Identificador único de la tabla.
DESCRIPTION	VARCHAR(45)	Descripción para cada uno de los valores anteriores.

Tabla B.8: Tabla Smoker_openSocial

Campo	Tipo	Descripción
id_LookingFor_openSocial	INTEGER	Identificador único de la tabla.
DESCRIPTION	VARCHAR(45)	Descripción para cada uno de los valores anteriores.

Tabla B.9: Tabla LookingFor_openSocial

Campo	Tipo	Descripción
id_Email_openSocial	INTEGER	Identificador único de la tabla.
ADDRESS	VARCHAR(100)	Dirección de correo electrónico.
TYPE	VARCHAR(45)	Descripción del tipo de cuenta de correo electrónico.
user_openSocial_id_user_openSocial	VARCHAR(100)	Identificador de su usuario en OpenSocial.

Tabla B.10: Tabla Email_openSocial

Campo	Tipo	Descripción
id_Gender_openSocial	INTEGER	Identificador único de la tabla.
DESCRIPTION	VARCHAR(45)	Descripción para cada uno de los valores anteriores.

Tabla B.11: Tabla Gender_openSocial

Campo	Tipo	Descripción
id_Presence_openSocial	INTEGER	Identificador único de la tabla.
DESCRIPTION	VARCHAR(45)	Descripción para cada uno de los valores anteriores.

Tabla B.12: Tabla Presence _ openSocial

Campo	Tipo	Descripción
id_Drinker_openSocial	INTEGER	Identificador único de la tabla.
DESCRIPTION	VARCHAR(45)	Descripción para cada uno de los valores anteriores.

Tabla B.13: Tabla Drinker _ openSocial

Campo	Tipo	Descripción
id_Address_openSocial	VARCHAR(100)	Identificador único de la tabla.
COUNTRY	VARCHAR(100)	País.
EXTENDED_ADDRESS	VARCHAR(150)	Dirección postal para direcciones largas.
LATITUDE	INTEGER	Línea de latitud del lugar.
LOCALITY	VARCHAR(100)	Localidad.
LONGITUDE	INTEGER	Línea de longitud del lugar.
PO_BOX	VARCHAR(10)	Apartado postal.
POSTAL_CODE	VARCHAR(10)	Código postal.
REGION	VARCHAR(100)	Región del lugar.
STREET_ADDRESS	VARCHAR(150)	Dirección postal.
TYPE	VARCHAR(45)	Tipo de la dirección(casa, trabajo, etc.).
UNSTRUCTURED	VARCHAR(200)	Si no se ha guardado la información estructuradamente, este campo almacenará toda la información de la dirección del usuario.

Tabla B.14: Tabla Address _ openSocial

Campo	Tipo	Descripción
id_Organization_openSocial	INTEGER	Identificador único de la tabla.
Address_openSocial_id_Address_openSocial	VARCHAR(10)	Referencia a la entrada en la tabla Address_openSocial en la que está almacenada su dirección.
DESCRIPTION	VARCHAR(100)	Descripción del trabajo.
END_DATE	DATE	Fecha fin del trabajo.
FIELD	VARCHAR(100)	Mercado de trabajo.
NAME	VARCHAR(100)	Nombre de la empresa.
SALARY	VARCHAR(45)	Salario.
START_DATE	DATE	Fecha de inicio del trabajo.
SUB_FIELD	VARCHAR(45)	Campo del mercado de trabajo.
TITLE	VARCHAR(45)	Puesto de trabajo que ostenta.
WEB_PAGE	VARCHAR(45)	Página web de la empresa.

Tabla B.15: Tabla Organization_openSocial

Campo	Tipo	Descripción
USER_UID	VARCHAR(45)	Identificador único del amigo en OpenSocial.
GROUP_ID	VARCHAR(45)	Grupo de amigos al que pertenece.
NETWORK_DISTANCE	INTEGER	Parámetro asignado por OpenSocial. Si el contacto pertenece a un grupo de amigos con los que tiene contacto frecuente, se le asignará una distancia menor, siendo mayor en caso contrario.

Tabla B.16: Tabla Friends_openSocial

Campo	Tipo	Descripción
id_url_openSocial	INTEGER	Identificador único de la tabla.
ADDRESS	VARCHAR(100)	Dirección url.
LINK_TEST	VARCHAR(100)	Dirección url test en caso de error.
TYPE	VARCHAR(100)	Tipo de url.

Tabla B.17: Tabla url_openSocial

Campo	Tipo	Descripción
id_Name_openSocial	INTEGER	Identificador único de la tabla.
ADDITIONAL_NAME	VARCHAR(45)	Nombre adicional del usuario.
FAMILY_NAME	VARCHAR(45)	Nombre con el que se conoce familiarmente al usuario.
GIVEN_NAME	VARCHAR(45)	Apodo del usuario.
HONORIFIC_PREFIX	VARCHAR(45)	Prefijo honorífico del usuario.
HONORIFIC_SUFFIX	VARCHAR(45)	Sufijo honorífico del usuario.
UNSTRUCTURED	VARCHAR(45)	Campo que almacena toda la información sin estructurar.

Tabla B.18: Tabla Name_openSocial

Campo	Tipo	Descripción
id_Message_Type_openSocial	INTEGER	Identificador único de la tabla.
DESCRIPTION	VARCHAR(45)	Descripción para cada uno de los valores anteriores.

Tabla B.19: Tabla Message_Type_openSocial

Campo	Tipo	Descripción
id_Message_openSocial	INTEGER	Identificador único de la tabla.
BODY	VARCHAR(1000)	Cuerpo del mensaje.
BODY_ID	VARCHAR(1000)	Cuerpo del mensaje codificado mediante templates.
TITLE	VARCHAR(200)	Título del mensaje.
TITLE_ID	VARCHAR(200)	Título del mensaje codificado mediante templates.
Message_Type_openSocial_id_Message_Type_openSocial	INTEGER	Tipo del mensaje.
user_openSocial_id_user_openSocial	VARCHAR(100)	Usuario de OpenSocial al que pertenece el mensaje.

Tabla B.20: Tabla Message_openSocial

Tabla B.21: Tabla user_facebook

Campo	Tipo	Descripción
id_user_facebook	VARCHAR(100)	Identificador único de la tabla.
FIRST_NAME	LONGTEXT	Nombre.
MIDDLE_NAME	LONGTEXT	Segundo nombre(para usuarios de Estados Unidos).
LAST_NAME	LONGTEXT	Apellidos.
NAME	LONGTEXT	Nombre completo.
PIC_SMALL	LONGTEXT	URL de la foto de perfil a tamaño pequeño(con un ancho máximo de 50 px y un alto máximo de 150px).
PIC_BIG	LONGTEXT	URL de la foto de perfil a tamaño grande(con un ancho máximo de 200 px y un alto máximo de 600px).
PIC_SQUARE	LONGTEXT	URL de la foto de perfil formateada y encuadrada(con un máximo, tanto de ancho como de alto, de 50 px).
PIC	LONGTEXT	URL de la foto de perfil(con un ancho máximo de 100 px y un alto máximo de 300px).
PROFILE_UPDATE_TIME	TIME	Fecha en la que se actualizó por última vez el perfil del usuario.
BIRTHDAY	LONGTEXT	Fecha de nacimiento.
BIRTHDAY_DATE	LONGTEXT	Fecha de nacimiento en formato americano.
SIGNIFICANT_OTHER_ID	LONGTEXT	Id de la persona con la que tiene una relación(novio, novia, marido, etc.)
HS1_NAME	LONGTEXT	Nombre de su primer instituto.
HS2_NAME	LONGTEXT	Nombre de su segundo instituto.
GRAD_YEAR	INTEGER	Año de graduación en el instituto.
HS1_ID	LONGTEXT	Id de su primer instituto.
HS2_ID	LONGTEXT	Id de su segundo instituto.
NOTES_COUNT	INTEGER	Numero de notas creadas por el usuario.
WALL_COUNT	INTEGER	Numero de mensajes de muro escritos por el usuario.
ONLINE_PRESENCE	LONGTEXT	Información sobre el estado de chat del usuario.
LOCALE	LONGTEXT	Código que indica el idioma configurado por el usuario.
PROXIED_EMAIL	LONGTEXT	Email interno cifrado del usuario que Facebook utiliza internamente.
PROFILE_URL	LONGTEXT	URL del perfil del usuario.

Campo	Tipo	Descripción
EMAIL_HASHES	LONGTEXT	Contiene los emails confirmados del usuario.
PIC_SMALL_WITH_LOGO	LONGTEXT	URL de la foto de perfil a tamaño pequeño(con un ancho máximo de 50 px y un alto máximo de 150px) con el logo de Facebook.
PIC_BIG_WITH_LOGO	LONGTEXT	URL de la foto de perfil a tamaño grande(con un ancho máximo de 200 px y un alto máximo de 600px) con el logo de Facebook.
PIC_SQUARE_WITH_LOGO	LONGTEXT	URL de la foto de perfil formateada y encuadrada(con un máximo, tanto de ancho como de alto, de 50 px) con el logo de Facebook.
PIC_WITH_LOGO	LONGTEXT	URL de la foto de perfil(con un ancho máximo de 100 px y un alto máximo de 300px) con el logo de Facebook.
ALLOWED_RESTRICTIONS	LONGTEXT	Lista, separada por ;, de restricciones demográficas. Actualmente, sólo devuelve el valor alcohol
VERIFIED	TINYINT(1)	Indica si el perfil de Facebook ha sido verificado por su usuario.
PROFILE_BLURB	LONGTEXT	Contiene el comentario del usuario acerca de su perfil en Facebook.
USERNAME	LONGTEXT	Nombre de usuario dentro de Facebook.
WEBSITE	LONGTEXT	URL al sitio web propio y externo del usuario.
IS_BLOCKED	TINYINT(1)	Indica si el perfil de Facebook está bloqueado.
CONTACT_EMAIL	LONGTEXT	Email de contacto.
EMAIL	VARCHAR(450)	Email del usuario.
MEETING_FOR	LONGTEXT	Lista de preferencias a la hora de conocer a alguien.
MEETING_SEX	LONGTEXT	Lista de preferencias a la hora de conocer a alguien para una relación.

Campo	Tipo	Descripción
id_work_history_facebook	INTEGER	Identificador único de la tabla.
LOCATION	VARCHAR(100)	Lugar de trabajo.
COMPANY_NAME	VARCHAR(100)	Nombre de la empresa de trabajo.
DESCRIPTION	VARCHAR(100)	Descripción del puesto de trabajo.
POSITION	VARCHAR(100)	Cargo en el puesto de trabajo.
START_DATE	VARCHAR(7)	Fecha de inicio del trabajo.
END_DATE	VARCHAR(7)	Fecha fin del trabajo.
user_facebook_id_user_facebook	VARCHAR(100)	Usuario de Facebook con el que está relacionado.

Tabla B.22: Tabla work_history_facebook

Campo	Tipo	Descripción
id_education_history_facebook	INTEGER	Identificador único de la tabla.
YEAR	VARCHAR(5)	Año de inicio.
NAME	VARCHAR(100)	Nombre del centro educativo.
DEGREE	VARCHAR(100)	Nombre del estudio universitario.
CONCENTRATIONS_0	VARCHAR(200)	Campo para comentarios y experiencias del primer año.
CONCENTRATIONS_1	VARCHAR(200)	Campo para comentarios y experiencias del segundo año.
CONCENTRATIONS_2	VARCHAR(200)	Campo para comentarios y experiencias del tercer año.
user_facebook_id_user_facebook	VARCHAR(100)	Usuario de Facebook con el que está relacionado.

Tabla B.23: Tabla education_history_facebook

Tabla B.25: Tabla stream_facebook

Campo	Tipo	Descripción
post_id	VARCHAR(100)	Identificador único del post.
viewer_id	VARCHAR(100)	Usuario al que va dirigido el comentario.
app_id	VARCHAR(100)	En caso de que el comentario sea escrito por una aplicación, este campo almacenará el ID de Facebook de dicha aplicación.
source_id	VARCHAR(100)	Usuario propietario del post.

Campo	Tipo	Descripción
updated_time	TIMESTAMP	Fecha de actualización del post. Se actualizará cada vez que a un usuario le guste el post o se añadan nuevos comentarios a él.
created_time	TIMESTAMP	Fecha de creación del post.
filter_key	VARCHAR(100)	Metadatos del post.
attribution	VARCHAR(100)	Nombre completo de la aplicación que genera el post.
actor_id	VARCHAR(100)	ID del usuario que genera el post.
target_id	VARCHAR(100)	Usuario al que referencia el post.
message	MEDIUMTEXT	Mensaje del post.
app_data	VARCHAR(10000)	Datos de la aplicación que genera el post.
attachment	VARCHAR(10000)	Información del archivo generado por el post en Facebook.
type	VARCHAR(100)	Tipo del post dependiendo de quien lo cree o lo actualice.
permalink	VARCHAR(300)	URL asociada al post.
xid	VARCHAR(45)	ID asociado a Live Stream Box.

Como se ha indicado anteriormente, la tabla **comments_facebook** se relacionará 1 a N con la tabla **comment_facebook**, la cual almacenará información de cada uno de los comentarios a los post y seguirá la estructura física de la tabla B.30.

Campo	Tipo	Descripción
USER_UID	VARCHAR(45)	Identificador único de Facebook del amigo del usuario.
USER_NAME	VARCHAR(45)	Nombre del amigo.
USER_BIRTHDAY	VARCHAR(45)	Fecha de nacimiento del amigo.
USER_PIC	VARCHAR(200)	URL a la foto de perfil del amigo en Facebook.

Tabla B.24: Tabla friends_facebook

Campo	Tipo	Descripción
id_likes_facebook	INTEGER	Identificador único de la tabla.
HREF	VARCHAR(100)	URL del post.
COUNT	INTEGER	Numero total de likes.
USER_LIKES	TINYINT(1)	Indica si a alguien le ha gustado el post.
CAN_LIKE	TINYINT(1)	Indica si a alguien puede gustarle el post.
stream_facebook_post_id	VARCHAR(100)	Post de la tabla stream_facebook con el que está relacionado.

Tabla B.26: Tabla likes_facebook

Campo	Tipo	Descripción
id_friends_likes_facebook	INTEGER	Identificador único de la tabla.
UID	VARCHAR(45)	UID del amigo de Facebook al que le ha gustado el post.
likes_facebook_id_likes_facebook	INTEGER	Entrada de la tabla likes_facebook con la que está relacionado.

Tabla B.27: Tabla friends_likes_facebook

Campo	Tipo	Descripción
id_privacy_facebook	INTEGER	Identificador único de la tabla.
UID	VARCHAR(45)	UID del amigo de Facebook que puede visualizar el post.
stream_facebook_post_id	VARCHAR(100)	Post de la tabla stream_facebook con el que está relacionado.

Tabla B.28: Tabla privacy_facebook

Campo	Tipo	Descripción
id_comments_facebook	INTEGER	Identificador único de la tabla.
CAN_REMOVE	TINYINT(1)	Indica si el post puede ser borrado.
CAN_POST	TINYINT(1)	Indica si el post puede ser comentado.
stream_facebook_post_id	VARCHAR(100)	Post de la tabla stream_facebook con el que está relacionado.

Tabla B.29: Tabla comments_facebook

Campo	Tipo	Descripción
ID	VARCHAR(100)	Identificador único de la tabla.
XID	VARCHAR(100)	ID asociado a Live Stream Box.
POST_ID	VARCHAR(100)	ID asociado al post de la tabla stream_facebook.
FROMID	VARCHAR(100)	Indica el usuario que ha generado el comentario.
TIME	TIMESTAMP	Fecha de creación.
TEXT	MEDIUMTEXT	Texto del comentario.
USERNAME	VARCHAR(100)	Nombre del usuario que ha generado el comentario.
REPLY_XID	VARCHAR(100)	Contestación generada desde Live Stream Box.
comments_facebook_id_comments_facebook	INTEGER	Post de la tabla comments_facebook con el que está relacionado.

Tabla B.30: Tabla comment_facebook

Campo	Tipo	Descripción
id_action_links_facebook	INTEGER	Identificador único de la tabla.
TEXT	VARCHAR(200)	Texto del post.
URL	VARCHAR(100)	URL al contenido del video o del clip de audio.
stream_facebook_post_id	VARCHAR(100)	Post de la tabla stream_facebook con el que está relacionado.

Tabla B.31: Tabla action_links_facebook

Campo	Tipo	Descripción
id_affiliations_facebook	INTEGER	Identificador único de la tabla.
YEAR	VARCHAR(10)	Año en el que comenzó la relación.
NAME	VARCHAR(200)	Nombre del usuario con el que está relacionado.
NID	INTEGER	ID del usuario de Facebook con el que está relacionado.
STATUS	VARCHAR(100)	Estado de la relación.
type_affiliations_facebook_id	INTEGER	Tipo de relación de la tabla type_affiliations_facebook.
user_facebook_id_user_facebook	VARCHAR(100)	Usuario de la tabla user_facebook con el que está relacionado.

Tabla B.32: Tabla affiliations_facebook

Campo	Tipo	Descripción
id_type_affiliations_facebook	INTEGER	Identificador único de la tabla.
DESCRIPTION	VARCHAR(100)	Descripción para cada uno de los tipos de relación en Facebook.

Tabla B.33: Tabla type_affiliations_facebook

Campo	Tipo	Descripción
id_family_facebook	INTEGER	Identificador único de la tabla.
relationship_facebook_id	INTEGER	Tipo de relación familiar de la tabla relationship_facebook.
UID	VARCHAR(45)	UID del usuario de Facebook con el que se mantiene una relación familiar.
NAME	VARCHAR(100)	Nombre del usuario con el que está relacionado.
BIRTHDAY	VARCHAR(45)	Fecha de nacimiento del usuario de Facebook con el que está relacionado.

Tabla B.34: Tabla family_facebook

Campo	Tipo	Descripción
id_relationship_facebook	INTEGER	Identificador único de la tabla.
DESCRIPTION	VARCHAR(100)	Descripción para cada uno de los tipos de relación familiar en Facebook.

Tabla B.35: Tabla relationship_facebook

Campo	Tipo	Descripción
id_location_facebook	INTEGER	Identificador único de la tabla.
CITY	VARCHAR(45)	Ciudad.
STATE	VARCHAR(45)	Estado o Comunidad Autónoma.
COUNTRY	VARCHAR(45)	País.

Tabla B.36: Tabla location_facebook

Apéndice C

Interface RESTFul: Definición de métodos

C.1. Método userSettings_getEntities

Este método obtiene todas los registros de la tabla **user_settings**. Será de utilidad en los procesos offline de la aplicación. Su información será la siguiente:

1. Entrada: Recibirá un parámetro con el tipo de dato que queremos a la salida **Class<T>responseType**.
2. Salida: Devolverá el tipo de dato que se ha indicado en el parámetro de entrada **responseType**.
3. Podrá devolver el tipo de excepción **UniformInterfaceException**.

Su definición será la siguiente:

```
public <T> T userSettings_getEntities(Class<T> responseType)
                                         throws UniformInterfaceException
```

C.2. Método userSettings_getUserByUidPublic

Este método obtiene el usuario de la tabla **user_settings** que coincida con el parámetro **uidPublic**, el cual será su **identificador público**, es decir, el identificador que puede observarse públicamente, obtenido de cifrar el identificador del usuario en Facebook con una clave privada proporcionada por el desarrollador de la aplicación. Será de utilidad cuando un ángel quiera acceder a la información del usuario que le ha enviado la petición de seguimiento. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.

- b) **String uidPublic:** Contendrá el identificador público del usuario que ejecuta la aplicación.
2. Salida: Devolverá toda la información del usuario al que referencia su uid pública en el tipo de dato que se ha indicado en el parámetro de entrada **responseType**.
 3. Podrá devolver el tipo de excepción **UniformInterfaceException**.

Su definición será la siguiente:

```
public <T> T userSettings_getUserByUidPublic(Class<T> responseType,
                                              String uidPublic)
                                              throws UniformInterfaceException
```

C.3. Método userSettings_setUpdateTime

Este método actualiza la fecha **lastCheck** o **backUpCheck** a la fecha en la que se está realizando la operación y, además, actualiza la sesión del usuario a la última sesión de conexión de Facebook. Que se actualice una u otra fecha dependerá del valor del parámetro **mode**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType:** Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid:** Contendrá el uid del usuario que ejecuta la aplicación.
 - c) **String mode:** Si mode es “1”, se actualizará la fecha **lastCheck**. Si por el contrario es “0”, se actualizará la fecha **backUpCheck**.
 - d) **String oauhToken:** Contendrá la nueva sesión para el usuario creada desde Facebook.
2. Salida: Devolverá la información del usuario con la fecha **lastCheck** o **backUpCheck** actualizada.

Su definición será la siguiente:

```
public <T> T userSettings_setUpdateTime(Class<T> responseType,
                                         String uid,
                                         String mode)
```

C.4. Método userSettings_setNewEntityUserSettings

Este método introduce un nuevo usuario, con toda su información, en la tabla **user_settings**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Objeto, en formato XML o JSON, que contendrá todos los datos del usuario que se da de alta en la tabla.
 2. Salida: Devolverá la información del usuario que se ha introducido.
 3. Podrá devolver el tipo de excepción **UniformInterfaceException**.

Su definición será la siguiente:

```
public <T> T userSettings_setNewEntityUserSettings(Class<T> responseType,  
                                                 Object requestEntity)  
throws UniformInterfaceException
```

C.5. Método userSettings_getUserSettingsByUid

Este método devuelve toda la información del usuario de la tabla `user_settings` al que pertenece el parámetro de entrada `uid`. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámetros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador del usuario que queremos obtener.
 2. Salida: Devolverá la información del usuario al que pertenezca el parámetro **uid** o, si no existe, devolverá una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userSettings_getUserSettingsByUid(Class<T> responseType,  
                                              String uid)  
        throws UniformInterfaceException
```

C.6. Método userSettings_setNewAngelsCollectionByUid

Este método establece un nuevo ángel en la colección de un usuario de la tabla **user_settings**, al que pertenece el parámetro de entrada **uidAngel**. Para que se lleve a cabo, dentro del objeto de entrada **Object requestEntity** deberá figurar la URI que relacione al Ángel con el usuario. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uidAngel**: Identificador del nuevo Ángel.
 - c) **Object requestEntity**: Información del nuevo Ángel.
2. Salida: Devolverá la información del nuevo Ángel o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userSettings_setNewAngelsCollectionByUid(Class<T> responseType,
String uidAngel,
Object requestEntity)
throws UniformInterfaceException
```

C.7. Método localeSettings_getLocaleSettingsByUid

Obtiene los recursos de idioma según el parámetro de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador de los recursos de idioma según la tabla **locale_settings**.
2. Salida: Devolverá los recursos de idioma asociados al parámetro **uid** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T localeSettings_getLocaleSettingsByUid(Class<T> responseType,
String uid)
throws UniformInterfaceException
```

C.8. Método settingsFilter_getFiltersByIdFilter

Obtiene la configuración de un filtro de la tabla **settings_filter** al que corresponde el identificador de entrada **idFilter**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String idFilter**: Identificador del filtro en la tabla **settings_filter**.
2. Salida: Devolverá la configuración del filtro asociado al parámetro **idFilter** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsFilter_getFiltersByIdFilter(Class<T> responseType,
String idFilter)
throws UniformInterfaceException
```

C.9. Método settingsFilter_getFiltersByUserSettingsUid

Obtiene todos los filtros de la tabla **settings_filter** asociados al usuario de la tabla **user_settings** al que corresponde el identificador de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String idFilter**: Identificador del filtro en la tabla **settings_filter**.
2. Salida: Devolverá la configuración del filtro asociado al parámetro **idFilter** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsFilter_getFiltersByUserSettingsUid(Class<T> responseType,
String uid)
throws UniformInterfaceException
```

C.10. Método settingsFilter_setNewFilter

Establece la configuración inicial de un filtro en la tabla **settings_filter**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Configuración inicial del filtro.
2. Salida: Devolverá la configuración inicial del filtro o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsFilter_setNewFilter(Class<T> responseType,
Object requestEntity)
throws UniformInterfaceException
```

C.11. Método settingsFilter_updateFilterByIdFilter

Actualiza la configuración de un filtro de la tabla **settings_filter** identificado con el parámetro de entrada **idFilter**. Tiene dos modos de funcionamiento establecido por el parámetro de entrada **mode**:

1. **0**: Actualizará la información del filtro sin actualizar la fecha de ejecución del mismo.
2. **1**: Actualizará la información del filtro actualizando a su vez, a la fecha y día actual, la fecha de ejecución del filtro.

Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String idFilter**: Identificador del filtro en la tabla **settings_filter**.
 - c) **Object requestEntity**: Nueva configuración del filtro.
 - d) **String mode**: Modo de funcionamiento.
2. Salida: Devolverá la nueva configuración del filtro o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsFilter_updateFilterByIdFilter(Class<T> responseType,
String idFilter,
Object requestEntity,
String mode)
throws UniformInterfaceException
```

C.12. Método settingsFilter_getAngelsCollectionByIdFilter

Obtiene la colección de ángeles definidos para el filtro identificado por el parámetro de entrada **idFilter**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String idFilter**: Identificador del filtro en la tabla **user_settings**.
2. Salida: Devolverá la colección de ángeles definidos para el filtro **idFiltro** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsFilter_getAngelsCollectionByIdFilter(Class<T> responseType,
String idFilter)
throws UniformInterfaceException
```

C.13. Método settingsAngels_setAngelByUid

Actualiza la información del ángel en la tabla **settings_angels** que está identificado por el parámetro de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador del ángel de la tabla **settings_angels**.
 - c) **Object requestEntity**: Nueva información del ángel actualizada.
2. Salida: Devolverá la nueva información del ángel actualizada o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsAngels_setAngelByUid(Class<T> responseType,
String uid,
Object requestEntity)
throws UniformInterfaceException
```

C.14. Método settingsAngels_delAngelByUid

Borra la información del ángel en la tabla **settings_angels** que está identificado por el parámetro de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **String uid**: Identificador del ángel de la tabla **settings_angels**.
2. Salida: No devolverá nada si el borrado se ha realizado correctamente o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public void settingsAngels_delAngelByUid(String uid)
throws UniformInterfaceException
```

C.15. Método settingsAngels_setNewAngel

Introduce un nuevo ángel en la tabla **settings_angels**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Información del nuevo ángel.
2. Salida: Devolverá la información del nuevo ángel o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsAngels_setNewAngel(Class<T> responseType,
Object requestEntity)
throws UniformInterfaceException
```

C.16. Método settingsAngels_getAngelsByPropUid

Obtiene la colección de ángeles definidos para un usuario de la tabla **user_settings** que está identificado por el parámetro de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.

- b) **String uid:** Identificador del usuario de la tabla **user_settings**.
- 2. Salida: Devolverá la colección de ángeles definida para el usuario o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsAngels_getAngelsByPropUid(Class<T> responseType,
String uid)
throws UniformInterfaceException
```

C.17. Método settingsAngels_getAngelsByUid

Obtiene un angel de la tabla **settings_angels** que está identificado por el parámetro de entrada **uid**. Su información será la siguiente:

- 1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType:** Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid:** Identificador del angel en la tabla **settings_angels**.
- 2. Salida: Devolverá el angel en cuestión o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsAngels_getAngelsByUid(Class<T> responseType,
String uid)
throws UniformInterfaceException
```

C.18. Método settingsAngels_getAngelsByUidFacebook

Obtiene la colección de ángeles pertenecientes a Facebook definidos para un usuario de la tabla **user_settings** que está identificado por el parámetro de entrada **uidFacebook**. Su información será la siguiente:

- 1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType:** Contendrá el tipo de dato que queremos a la salida.
 - b) **String uidFacebook:** Identificador del usuario de Facebook de la tabla **user_settings**.
- 2. Salida: Devolverá la colección de ángeles definida para el usuario de Facebook o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T settingsAngels_getAngelsByUidFacebook(Class<T> responseType,
String uidFacebook)
throws UniformInterfaceException
```

C.19. Método user_getEntities

Obtendrá todos los usuarios de la tabla **user**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
2. Salida: Devolverá todos los usuarios de la tabla **user** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T user_getEntities(Class<T> responseType)
throws UniformInterfaceException
```

C.20. Método user_getUserByUid

Obtiene el usuario de la tabla **user** que indica el identificador de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador del usuario de la tabla **user**.
2. Salida: Devolverá la información del usuario en la tabla **user** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T user_getUserByUid(Class<T> responseType,
String uid)
throws UniformInterfaceException
```

C.21. Método user_setNewUser

Almacena a un nuevo usuario en la tabla **user**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Información del nuevo usuario.
2. Salida: Devolverá la información del nuevo usuario en la tabla **user** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T user_setNewUser(Class<T> responseType,
Object requestEntity)
throws UniformInterfaceException
```

C.22. Método user_setUserByUid

Actualiza la información del usuario de la tabla **user** que indica el identificador de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Información del usuario.
 - c) **String uid**: Identificador del usuario de la tabla **user**.
2. Salida: Devolverá la información actualizada del usuario en la tabla **user** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T user_setUserByUid(Class<T> responseType,
Object requestEntity,
String uid)
throws UniformInterfaceException
```

C.23. Método userFacebook_getEntities

Obtendrá todos los usuarios de la tabla **user_Facebook**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
2. Salida: Devolverá todos los usuarios de la tabla **user_Facebook** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getEntities(Class<T> responseType)
throws UniformInterfaceException
```

C.24. Método userFacebook_setNewUserFacebook

Almacena a un nuevo usuario en la tabla **user_Facebook**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Información del nuevo usuario.
2. Salida: Devolverá la información del nuevo usuario en la tabla **user_Facebook** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_setNewUserFacebook(Class<T> responseType,
Object requestEntity)
throws UniformInterfaceException
```

C.25. Método userFacebook_getUserFacebookByUid

Obtiene el usuario de la tabla **user_Facebook** que indica el identificador de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:

- a) **Class<T>responseType:** Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid:** Identificador del usuario de la tabla **user_Facebook**.
2. Salida: Devolverá la información del usuario en la tabla **user_Facebook** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getUserFacebookByUid(Class<T> responseType,
String uid)
throws UniformInterfaceException
```

C.26. Método userFacebook_setUserFacebookByUid

Actualiza la información del usuario de la tabla **user_Facebook** que indica el identificador de entrada **uid**. Su información será la siguiente:

- 1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType:** Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity:** Información del usuario.
 - c) **String uid:** Identificador del usuario de la tabla **user_Facebook**.
- 2. Salida: Devolverá la información actualizada del usuario en la tabla **user_Facebook** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_setUserFacebookByUid(Class<T> responseType,
Object requestEntity,
String uid)
throws UniformInterfaceException
```

C.27. Método userFacebook_setNewStreamCommentsByUid

Inserta en la tabla **stream_Facebook** los nuevos comentarios en el muro de Facebook del usuario que indica el identificador de entrada **uid**. Su información será la siguiente:

- 1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType:** Contendrá el tipo de dato que queremos a la salida.

- b) **Object requestEntity:** Información del nuevo comentario.
 - c) **String uid:** Identificador del usuario de la tabla **user_Facebook**.
 - d) **String updatedTime:** Fecha de actualización del comentario.
 - e) **String createdTime:** Fecha de creación del comentario
2. Salida: Devolverá la información insertada del comentario en la tabla **stream_Facebook** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_setNewStreamComentsByUid(Class<T> responseType,
Object requestEntity,
String uid,
String updatedTime,
String createdTime)
throws UniformInterfaceException
```

C.28. Método userFacebook_getStreamComentByUid

Devuelve de la tabla **stream_Facebook** el comentario referenciado por el campo “postId”. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType:** Contendrá el tipo de dato que queremos a la salida.
 - b) **String postId:** Información del usuario de Facebook que creó el comentario.
2. Salida: Devolverá la información del comentario de la tabla **stream_Facebook** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getStreamComentByUid(Class<T> responseType,
String postId)
throws UniformInterfaceException
```

C.29. Método userFacebook_setStreamComentsByUid

Actualiza un comentario de la tabla **stream_Facebook** con su nueva fecha de creación y de actualización. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Información del comentario.
 - c) **String postId**: Información del usuario de Facebook que creó el comentario.
 - d) **String updatedTime**: Fecha de actualización del comentario.
 - e) **String createdTime**: Fecha de creación del comentario
2. Salida: Devolverá la información actualizada del comentario en la tabla **stream_Facebook** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_setStreamComentsByUid(Class<T> responseType,
Object requestEntity,
String postId,
String updatedTime,
String createdTime)
throws UniformInterfaceException
```

C.30. Método userFacebook_setComentsPost

Establece un comentario a un post de muro de la tabla **stream_Facebook** con su nueva fecha de creación en la tabla **comment_facebook**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Información del comentario.
 - c) **String time**: Fecha de creación del comentario
2. Salida: Devolverá la información actualizada del comentario en la tabla **stream_Facebook** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_setComentsPost(Class<T> responseType,
Object requestEntity,
String time)
throws UniformInterfaceException
```

C.31. Método userFacebook_getComentsPostById

Obtiene todos los comentarios a un post de muro de la tabla **stream_facebook**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String postId**: Identificador del post de la tabla **stream_facebook**.
2. Salida: Devolverá todos los comentarios a un post del muro del usuario en Facebook o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getComentsPostById(Class<T> responseType,
String postId)
throws UniformInterfaceException
```

C.32. Método userFacebook_getComentsPostByTime

Obtiene todos los comentarios a un post de muro de la tabla **stream_facebook** que se han producido posteriormente a una fecha indicada por el parámetro **time**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String postId**: Identificador del post de la tabla **stream_facebook**.
 - c) **String time**: Fecha a partir de la cual se obtienen los comentarios producidos.
2. Salida: Devolverá todos los comentarios a un post del muro del usuario en Facebook producidos a partir de una fecha determinada o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getCommentsPostByTime(Class<T> responseType,
String postId,
String time)
throws UniformInterfaceException
```

C.33. Método userFacebook_getStreamFacebookByUid

Obtiene todos los post del muro de un usuario cuyo identificador está contenido en el parámetro de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador del usuario propietario del muro de Facebook.
2. Salida: Devolverá todos los post del muro del usuario en Facebook o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getStreamFacebookByUid(Class<T> responseType,
String uid)
throws UniformInterfaceException
```

C.34. Método userFacebook_getStreamFacebookByUpdatedTime

Obtiene todos los post del muro de un usuario cuyo identificador está contenido en el parámetro de entrada **uid** y que se han producido a partir de la fecha contenida en el parámetro de entrada **updatedTime**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador del usuario propietario del muro de Facebook.
 - c) **String updatedTime**: Fecha a partir de la cual se obtienen los post producidos.
2. Salida: Devolverá todos los post del muro del usuario en Facebook producidos posteriormente a la fecha **updatedTime** o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getStreamFacebookByUpdatedTime(Class<T> responseType,
String uid,
String updatedTime)
throws UniformInterfaceException
```

C.35. Método userFacebook_getFriendsFacebookByUidCollection

Obtiene la colección de amigos en Facebook del usuario identificado por el parámetro de entrada **uid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador del usuario de Facebook.
2. Salida: Devolverá la colección de amigos de Facebook de un usuario o una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getFriendsFacebookByUidCollection(Class<T> responseType,
String uid)
throws UniformInterfaceException
```

C.36. Método userFacebook_isNewFriendsFacebookByUid

Indica si un amigo de Facebook, identificado por el parámetro de entrada **friendUid**, está o no registrado en la base de datos SocialNetwork. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String friendUid**: Identificador del amigo de Facebook.
2. Salida: Devolverá la información del amigo de Facebook si existe. Si no existe devolverá una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_isNewFriendsFacebookByUid(Class<T> responseType,
String friendUid)
throws UniformInterfaceException
```

C.37. Método userFacebook_getFriendsFacebookByUid

Devuelve toda la información de un amigo de Facebook, identificado por el parámetro de entrada **friendUid**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador del usuario de Facebook.
 - c) **String friendUid**: Identificador del amigo de Facebook.
2. Salida: Devolverá la información del amigo de Facebook si existe y es amigo del usuario. Si no existe devolverá una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_getFriendsFacebookByUid(Class<T> responseType,
String uid,
String friendUid)
throws UniformInterfaceException
```

C.38. Método userFacebook_setNewFriendFacebook

Almacena en la base de datos SocialNetwork la información de un nuevo amigo de Facebook del usuario. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Información del amigo de Facebook.
2. Salida: Devolverá la información del amigo de Facebook si se ha creado correctamente. Si no devolverá una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_setNewFriendFacebook(Class<T> responseType,
Object requestEntity)
throws UniformInterfaceException
```

C.39. Método userFacebook_setFriendsFacebookByUid

Actualiza en la base de datos SocialNetwork la información de un amigo de Facebook del usuario. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **Object requestEntity**: Información del amigo de Facebook.
 - c) **String uid**: Identificador del amigo de Facebook.
2. Salida: Devolverá la información del amigo actualizada de Facebook si se ha procesado correctamente. Si no devolverá una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userFacebook_setFriendsFacebookByUid(Class<T> responseType,
Object requestEntity,
String uid)
throws UniformInterfaceException
```

C.40. Método userOpenSocial_getEntities

Obtiene todos los usuarios de la tabla **user_opensocial**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
2. Salida: Devolverá rodas las entidades de la tabla **user_openSocial** si no se produce un error. Si no devolverá una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userOpenSocial_getEntities(Class<T> responseType)
throws UniformInterfaceException
```

C.41. Método userOpenSocial_setUserOpenSocialByUid

Actualiza la información de un usuario de la tabla **user_opensocial**. Su información será la siguiente:

1. Entrada: Recibirá los siguientes parámentros:
 - a) **Class<T>responseType**: Contendrá el tipo de dato que queremos a la salida.
 - b) **String uid**: Identificador del usuario de OpenSocial.
 - c) **Object requestEntity**: Información del usuario de OpenSocial.
2. Salida: Actualizará la información del usuario de la tabla **user_opensocial** si no se produce un error. Si no devolverá una excepción **UniformInterfaceException** con el código de error HTTP asociado.

Su definición será la siguiente:

```
public <T> T userOpenSocial_setUserOpenSocialByUid(Class<T> responseType,  
String uid,  
Object requestEntity)  
throws UniformInterfaceException
```


Bibliografía

An HTTP Extension Framework (n.d.), <http://tools.ietf.org/html/rfc2774>.

Aulet, J. (n.d.), *Introducción a REST-1.* <http://www.sindikos.com/2011/05/breve-introduccion-a-rest-1/>.

BibTex (n.d.), *Organización de Librerías*, <http://es.wikipedia.org/wiki/BibTeX>.

B.O.E. (1999), ‘Ley orgánica de protección de datos’, *B.O.E.* .

Cabal, R. D. (2014), ‘Cyberbullying: el acoso a través de las redes sociales’, *Revista Digital Cabal* .

Crespo, R. (2011), ‘La teoría de los seis grados de separación’.

Developers, F. (2002), *Facebook FQL Api*, <https://developers.facebook.com/docs/reference/fql>.

Domínguez, D. C. (2010), ‘Las redes sociales. tipología, uso y consumo de las redes 2.0 en la sociedad digital actual’, Universidad Complutense, Madrid.

Fielding, R. (2000), ‘Rest’.

Hardcastle, M. (2014), ‘What is cyberbullying?’, www.teenadvice.com .

HTTP Extensions for Distributed Authoring – WEBDAV (n.d.), <http://tools.ietf.org/html/rfc2518>.

HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV) (n.d.), <http://tools.ietf.org/html/rfc4918>.

Hypertext Transfer Protocol – HTTP/1.1 (n.d.), <http://tools.ietf.org/html/rfc2616>.

Karinthy, F. (1930), ‘Chains’.

Marsel, R. N. (n.d.), *REST vs Web Services.* users.dsic.upv.es/~rnarrowo/NewWeb/docs/RestVsWebServices.pdf.

of Gothenburg, U. (2010), ‘Cyberbullying: A growing problem’, *ScienceDaily* .

OpenSocial Resources API (n.d.), GoogleCode, <http://opensocial-resources.googlecode.com/>.

RestFB (2009), ‘Restfb’. <http://restfb.com/>.

Rosenblum, D. (2007), ‘What anyone can know: The privacy risks of social networking sites’, *Security & Privacy, IEEE* 5(3), 49.

Transparent Content Negotiation in HTTP (n.d.), <http://tools.ietf.org/html/rfc2295>.

Upgrading to TLS Within HTTP/1.1 (n.d.), <http://tools.ietf.org/html/rfc2817>.

Wikipedia (2007), *OpenSocial*, <http://es.wikipedia.org/wiki/OpenSocial>.

Wikipedia (n.d.a), ‘Ajax’. <http://es.wikipedia.org/wiki/AJAX>.

Wikipedia (n.d.b), *Códigos de estado HTTP*, http://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

Wikipedia (n.d.c), ‘Html’. <http://es.wikipedia.org/wiki/HTML>.

Wikipedia (n.d.d), ‘Javascript’. <http://es.wikipedia.org/wiki/JavaScript>.

Wikipedia (n.d.e), ‘jquery’. <http://es.wikipedia.org/wiki/Jquery>.

Wikipedia (n.d.f), ‘Json’. <http://es.wikipedia.org/wiki/JSON>.

Wikipedia (n.d.g), ‘Jsp’. <http://es.wikipedia.org/wiki/JSP>.

Zuckerberg, M. (2002), ‘Facebook’. <http://es.wikipedia.org/wiki/Facebook>.