

1. Resumen

La ingeniería es el conjunto de conocimientos y técnicas científicas aplicadas al desarrollo, implementación, mantenimiento y perfeccionamiento de estructuras (tanto físicas como teóricas) para la resolución de problemas que afectan la actividad cotidiana de la sociedad.

Tras observar el método de trabajo de medianas empresas de revisión de sistemas de extinción, he entendido lo tedioso y manual que puede llegar a ser su tarea, por lo que empezo a fraguarse la idea de facilitar al máximo las labores que tienen que hacer estas empresas desde la oficina hasta las instalaciones del cliente.

Poco después, tras abstraerme lo máximo posible del negocio del cliente, me percaté de que existen muchos elementos comunes en lo que respecta a la revisión in situ de elementos de cualquier índole, por lo que surgió la idea, ya centralizada, de desarrollar un programa que fuese capaz de adaptarse a cualquier negocio que se dedique a la revisión y el mantenimiento de cualquier entidad, desde sistemas de extinción hasta grifería, iluminación, catenaria, etc.

SIRME (Sistema Integral de Revisión y Mantenimiento de Equipos) nace consecuencia del vacío tecnológico que existe en las empresas de revisión y mantenimiento de cualquier sistema y con el propósito de agilizar el trabajo a las personas que estén en las oficinas y a los técnicos que tengan que realizar las revisiones de las instalaciones.

2. Agradecimientos

Me lo contaron y lo olvidé; lo vi y lo entendí; lo hice y lo aprendí.

A **Francisco Gortázar, Patxi**, por dirigir este último paso para conseguir mi meta, por su infinita paciencia contestando mis correos electrónicos y su guía para no perderme en banalidades.

A la **URJC** y a aquellos profesores que se han volcado con nosotros, que han sabido ofrecer una enseñanza de calidad independientemente de la modalidad en la que se imparta y que han conseguido formarme y abrirme los ojos.

Especialmente quiero agradecérselo a **Ana E. García Sipols**, por animarme a no dejar de lado su asignatura, a **Javier Garzás**, por su blog y su visión real del mundo del Software, a **Luis Rincón Córcoles**, por recibirmelos en tutorías incluso sin haberlas concertado, a **Sofía Bayona Beriso**, por sus vídeos, sin los que no hubiera podido aprobar su asignatura, a **Patxi** (nuevamente), por abrirme la mente a nuevos paradigmas y tecnologías...

A mis inseparables **compañeros** de Universidad y al grupo que hemos ido formando estos últimos años, ayudándonos en las materias que cada uno más dominamos y repasando a horas intempestivas los días antes de los exámenes. Nunca olvidaré esas explicaciones de última hora en la biblioteca. Roberto, Javier, este trabajo lleva un poquito de vosotros.

A mi **familia**, por aguantarme y comprenderme, por el cariño, el apoyo y la confianza que han depositado en mí. Gracias por quererme y aceptarme tal y como soy y por haber hecho de mi la persona que soy.

A todos mis **amigos y compañeros** de trabajo, por animarme con la aplicación y darme ideas según su experiencia.

Y, en definitiva, a todos aquellos que han hecho posible que llegue a este punto.

3. Introducción

Do cool things that matters.

3.1. Propósito

El propósito de esta sección es recoger los requerimientos que se han de implementar en el proyecto **SIRME**, de forma que sirva como referencia para el sistema que ofrecerá la solución buscada.

3.2. Alcance

3.2.1 Situación actual

Actualmente la mayoría de las empresas de revisión de sistemas de extinción realiza las verificaciones de los elementos de extinción asignando a los técnicos una orden de trabajo con los elementos que debe revisar para diferentes clientes y las pruebas a realizar por cada elemento. El técnico realizará las verificaciones que se le indique e irá anotando los resultados a mano.

Una vez terminada la labor de revisión por parte del técnico, debe entregar la ficha de resultados al administrativo para que éste prepare una ficha de revisión que certifica el estado de los elementos y devolverla al cliente.

3.2.2 Objetivos y Beneficios

Con este desarrollo se persigue automatizar el proceso de verificación de elementos de extinción.

- Gestión de las fichas de revisión de los elementos verificados. Esto incluye almacenaje, edición, búsqueda y centralización.
- Generación de partes de trabajo para los técnicos.
- Mejora en la realización de las revisiones, siendo anotadas desde una aplicación.
- Consulta automática de las órdenes de trabajo por parte de un técnico.

3.3. Usuarios del sistema

El sistema utilizará cuatro grupos o roles de usuarios bien diferenciados: administradores, administrativos, técnicos y comerciales.

- Los Administradores serán aquellos usuarios que tendrán pleno acceso al sistema, pudiendo realizar todas las labores de los administrativos y gestionar los usuarios. De igual manera tendrán acceso a las zonas de gestión de los parámetros de la aplicación.
- Los Administrativos podrán gestionar y modificar todos los partes de trabajo y las revisiones que los técnicos generen. De igual manera, podrán gestionar cualquier elemento del sistema que necesiten, excepto usuarios.
- Los Técnicos que, al comenzar su jornada laboral tendrán asignados partes de trabajo, serán los que utilicen las interfaces REST.
- Los Comerciales, si bien aún no están claras las partes de la aplicación a las que deben acceder (el programa permite que tengan solo acceso al sistema), tienen que estar dados de alta en el sistema para poder ser utilizados como personas de contacto para las empresas.

4. Índice de Contenidos

1.	Resumen	I
2.	Agradecimientos	II
3.	Introducción.....	III
3.1.	Propósito	III
3.2.	Alcance	III
3.2.1	Situación actual.....	III
3.2.2	Objetivos y Beneficios.....	III
3.3.	Usuarios del sistema	III
4.	Índice de Contenidos	IV
5.	Índice de Ilustraciones	VI
6.	Descripción Funcional	1
6.1.	Clientes.....	2
6.2.	Gestores	3
6.3.	Contratos.....	3
6.4.	Partes de trabajo.....	3
6.5.	Fichas de revisión.....	4
6.6.	Equipos.....	6
6.7.	Usuarios	6
6.8.	Configuración.....	6
6.9.	Mensajes	7
7.	Descripción informática	8
7.1.	Ciclo de desarrollo incremental en espiral	8
7.1.1	Especificación y Análisis de Requisitos	8
7.1.2	Diseño	8
7.1.3	Implementación	8
7.1.4	Pruebas	9
7.2.	Tecnologías empleadas.....	9
7.2.1	Java.....	9
7.2.2	XML	10
7.2.3	JSON	10
7.2.4	HTML.....	11
7.2.5	JavaServer Faces	11
7.2.6	Apache Maven	12
7.2.7	Spring	13

7.2.8	JPA e Hibernate	13
7.2.9	SLFJ	14
7.2.10	Estilos CSS	15
7.2.11	Servicios web y REST	15
7.2.12	JUnit	17
7.2.13	MySQL	18
7.2.14	Git	18
7.2.15	Jasper Report	19
7.2.16	HSQLDB	19
7.2.17	TravisCI	20
8.	Arquitectura	21
8.1.	MODEL	23
8.1.1	util	25
8.1.2	data	25
8.1.3	dao	25
8.2.	VIEW	26
8.2.1	webapp	27
8.2.2	java	30
8.3.	CONTROLLER	36
8.3.1	data	36
8.3.2	dto	38
8.3.3	transform	38
8.3.4	util	38
8.3.5	validator	39
8.3.6	rest	39
8.3.7	services	39
9.	Despliegue y Pruebas	40
9.1.	Despliegue	40
9.1.1	Servidor de Aplicaciones	40
9.1.2	Base de Datos	40
9.1.3	Clientes	40
9.1.4	Instalación en Linux	40
9.2.	Pruebas	44
10.	Conclusiones y trabajo futuro	48
11.	Bibliografía	49

5. Índice de Ilustraciones

ILUSTRACIÓN 6-1. DIAGRAMA DE NAVEGACIÓN DE SIRME	1
ILUSTRACIÓN 6-2. MENÚ DE NAVEGACIÓN DE SIRME	1
ILUSTRACIÓN 6-3. PANTALLA DE LOGIN DE SIRME	2
ILUSTRACIÓN 6-4. PANTALLA DE BÚSQUEDA DE CLIENTES.....	2
ILUSTRACIÓN 6-5. PANTALLA DE BÚSQUEDA DE GESTORES	3
ILUSTRACIÓN 6-6. PANTALLA DE BÚSQUEDA DE CONTRATOS.....	3
ILUSTRACIÓN 6-7. PANTALLA DE BÚSQUEDA DE PARTES DE TRABAJO.....	4
ILUSTRACIÓN 6-8. PANTALLA DE VISTA AVANZADA DE PARTES DE TRABAJO	4
ILUSTRACIÓN 6-9. EJEMPLO DE ACCIONES SOBRE FICHA DE REVISIÓN	4
ILUSTRACIÓN 6-10. EJEMPLO DE FICHA DE REVISIÓN EN FORMATO PDF.....	5
ILUSTRACIÓN 6-11. DESCARGA DE FICHEROS COMPRIMIDOS	5
ILUSTRACIÓN 6-12. PANTALLA DE VISTA AVANZADA DE FICHA DE REVISIÓN	5
ILUSTRACIÓN 6-13. PANTALLA DE BÚSQUEDA DE EQUIPOS	6
ILUSTRACIÓN 6-14. PANTALLA DE BÚSQUEDA DE USUARIOS	6
ILUSTRACIÓN 6-15. PANTALLA DE ADMINISTRACIÓN	6
ILUSTRACIÓN 6-16. EJEMPLOS DE DIFERENTES MENSAJES.....	7
ILUSTRACIÓN 7-1. CICLO DE VIDA EN ESPIRAL	9
ILUSTRACIÓN 7-2. LOGO DE JAVA	9
ILUSTRACIÓN 7-3. LOGO DE XML	10
ILUSTRACIÓN 7-4. LOGO DE JSON	10
ILUSTRACIÓN 7-5. LOGO DE HTML5	11
ILUSTRACIÓN 7-6. LOGO DE JSF.....	11
ILUSTRACIÓN 7-7. LOGO DE PRIMEFACES	12
ILUSTRACIÓN 7-8. LOGO DE MAVEN	12
ILUSTRACIÓN 7-9. LOGO DE SPRING	13
ILUSTRACIÓN 7-10. LOGO DE JPA	13
ILUSTRACIÓN 7-11. LOGO DE HIBERNATE	14
ILUSTRACIÓN 7-12. LOGO DE SL4J.....	14
ILUSTRACIÓN 7-13. LOGO DE CSS.....	15
ILUSTRACIÓN 7-14. LOGO DE REST.....	15
ILUSTRACIÓN 7-15. TABLA DE VERBOS HTTP	16
ILUSTRACIÓN 7-16. LOGO DE JERSEY.....	17
ILUSTRACIÓN 7-17. LOGO DE JUNIT	17
ILUSTRACIÓN 7-18. LOGO DE MYSQL	18
ILUSTRACIÓN 7-19. LOGO DE GITHUB	18
ILUSTRACIÓN 7-20. LOGO DE JASPER SOFT	19
ILUSTRACIÓN 7-21. LOGO DE HSQLDB	19
ILUSTRACIÓN 7-22. LOGO DE TRAVISCI	20
ILUSTRACIÓN 8-1. MVC	21
ILUSTRACIÓN 8-2. SUBPROYECTOS SIRME	21
ILUSTRACIÓN 8-3. DIAGRAMA DE SECUENCIA DE EJEMPLO DE BÚSQUEDA.....	22
ILUSTRACIÓN 8-4. DETALLE DEL DIAGRAMA DE CLASES DE USUARIOS	23
ILUSTRACIÓN 8-5. DETALLE DEL DIAGRAMA DE CLASES DE TRABAJOS	23
ILUSTRACIÓN 8-6. DETALLE DEL DIAGRAMA DE CLASES DE PREGUNTAS Y RESPUESTAS	24
ILUSTRACIÓN 8-7. DETALLE DEL DIAGRAMA DE CLASES COMPLETO	24
ILUSTRACIÓN 8-8. DEPENDENCIAS MAVEN DE SIRME-CORE	25
ILUSTRACIÓN 8-9. PAQUETES SIRME-CORE.....	25
ILUSTRACIÓN 8-10. DEPENDENCIAS MAVEN PARA SIRME-WEB	26
ILUSTRACIÓN 8-11. DIRECTORIOS DE SIRME-WEB	26

ILUSTRACIÓN 8-12. DIRECTORIOS WEBAPP	27
ILUSTRACIÓN 8-13. DIFERENCIA PLANTILLA DE BÚSQUEDA Y EDICIÓN.....	28
ILUSTRACIÓN 8-14. CONTENIDO WEB-INF	28
ILUSTRACIÓN 8-15. EJEMPLO DE SECURIZACIÓN DE BOTONES	30
ILUSTRACIÓN 8-16. DIRECTORIOS DE CÓDIGO EN SIRME-WEB	30
ILUSTRACIÓN 8-17. EXTRACTO DE CONFIGURACIÓN DE LA APLICACIÓN	31
ILUSTRACIÓN 8-18. EXTRACTO DE HIBERNATE.CFG.XML	31
ILUSTRACIÓN 8-19. PAQUETES DE SIRME-WEB.....	32
ILUSTRACIÓN 8-20. MÉTODO DOINIT.....	33
ILUSTRACIÓN 8-21. VISTA AVANZADA DE CLIENTES.....	33
ILUSTRACIÓN 8-22. ACTUALIZACIÓN DE CLIENTES.....	33
ILUSTRACIÓN 8-23. PAQUETE UTIL.....	34
ILUSTRACIÓN 8-24. EJEMPLO DE EXPORTACIÓN DE LISTADO A FORMATO PDF.....	34
ILUSTRACIÓN 8-25. DEPENDENCIAS MAVEN DE SIRME-BUSINESS	36
ILUSTRACIÓN 8-26. PAQUETES DE SIRME-BUSINESS.....	36
ILUSTRACIÓN 8-27. DIAGRAMA DE CLASES DE USUARIOS	37
ILUSTRACIÓN 8-28. DIAGRAMA DE CLASES DE OBJETOS DE NEGOCIO CON MARCAS.....	37
ILUSTRACIÓN 8-29. DIAGRAMA DE CLASES DE OBJETOS DE NEGOCIO	38
ILUSTRACIÓN 8-30. CLASES DEL PAQUETE UTIL	38
ILUSTRACIÓN 9-1. CONSOLA LINUX - JAVA.....	41
ILUSTRACIÓN 9-2. CONSOLA LINUX - MAVEN	41
ILUSTRACIÓN 9-3. CONSOLA LINUX - MYSQL	41
ILUSTRACIÓN 9-4. SERVIDOR ARRANCADO	42
ILUSTRACIÓN 9-5. CONSOLA LINUX – INSTALACIÓN DE FUENTE ARIAL.....	42
ILUSTRACIÓN 9-6. CONSOLA LINUX - GIT	43
ILUSTRACIÓN 9-7. CONSOLA LINUX – CONSTRUCCIÓN DE SIRME	43
ILUSTRACIÓN 9-8. CONSOLA LINUX – CARGA DE ESQUEMA DDL	43
ILUSTRACIÓN 9-9. PANTALLA DE LOGIN DESPLEGADA.....	44
ILUSTRACIÓN 9-10. PANTALLA DE INICIO DESPLEGADA.....	44
ILUSTRACIÓN 9-11. TRAVISCI.....	45
ILUSTRACIÓN 9-12. EJEMPLOS DE TESTS	45
ILUSTRACIÓN 9-13. CONFIGURACIÓN Y ESTRUCTURA DE LOS TESTS	46
ILUSTRACIÓN 9-14. COBERTURA	46
ILUSTRACIÓN 9-15. EJEMPLO DE COBERTURA EN CLASE	46
ILUSTRACIÓN 9-16. COBERTURA EN CAPA CONTROL.....	47
ILUSTRACIÓN 9-17. CICLO DE VIDA DEL SOFTWARE.....	47

6. Descripción Funcional

Los programas deben ser escritos para que los lean las personas, y sólo incidentalmente, para que lo ejecuten las máquinas.

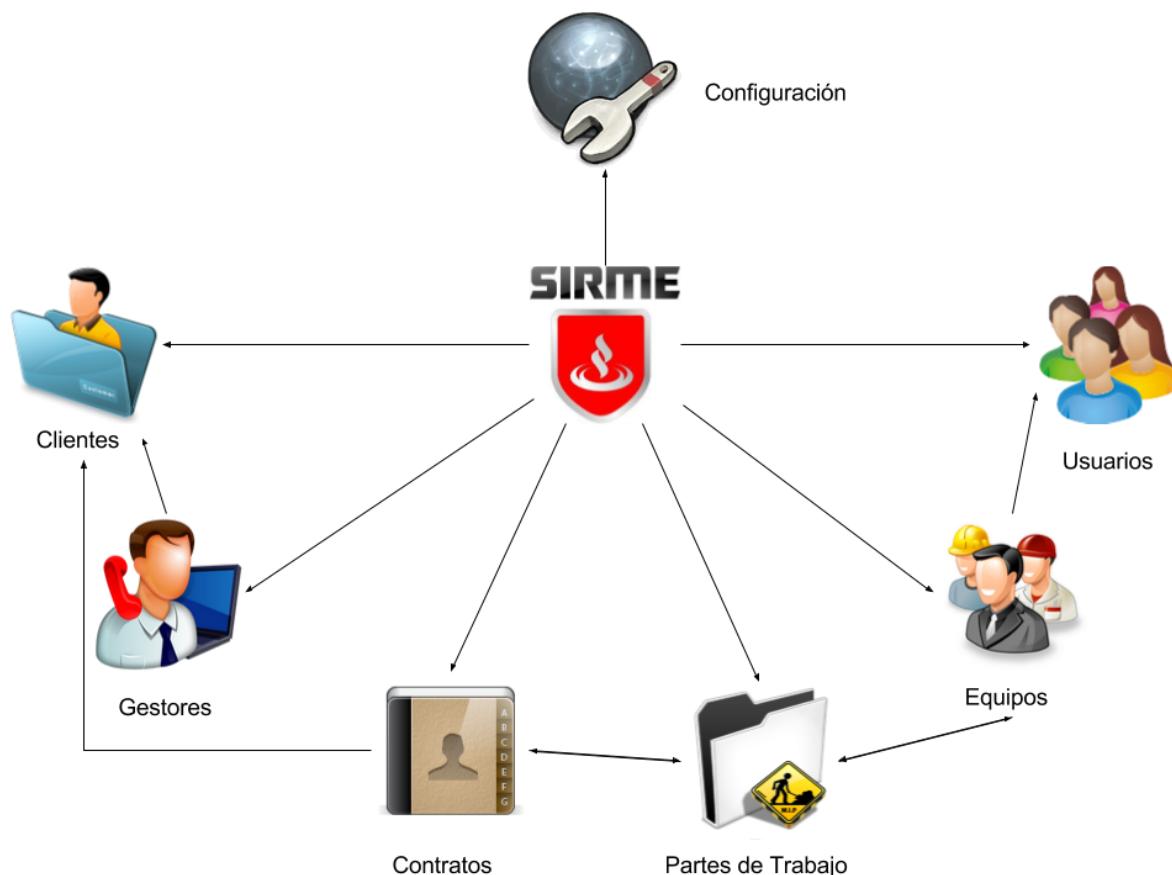


Ilustración 6-1. Diagrama de Navegación de SIRME

La aplicación tiene 7 módulos bien diferenciados: clientes, gestores, contratos, partes de trabajo, equipos, usuarios y configuración. Se accede a los módulos a través del menú situado en la parte superior de la aplicación o bien mediante los accesos que hay en las diferentes pantallas acordes al diagrama de Navegación anteriormente descrito.



Ilustración 6-2. Menú de Navegación de SIRME

Así, por ejemplo, para manipular los partes de trabajo deberemos seleccionar el contrato al cual pertenece ese parte de trabajo y el equipo al cual está asociado. De manera recíproca podremos ver todos los partes de trabajo que tiene un contrato o un equipo.

SIRME tendrá una pantalla de login por la que los usuarios podrán entrar a todas las características de la aplicación, que serán recogidas según los roles y permisos establecidos. Se ha elegido MD5 como algoritmo hash de cifrado de la contraseña.



Ilustración 6-3. Pantalla de Login de SIRME

6.1. Clientes

Los clientes son las comunidades de vecinos, centros comerciales, locales, oficinas, naves, colegios, hospitales, hoteles y en definitiva todas aquellas instalaciones susceptibles de tener elementos para ser supervisados por los técnicos.

Cada cliente, a parte de los datos propios del mismo, tendrá asociado (si así lo requiriese) un Gestor, un Comercial (usuarios del sistema con este rol) y tantos contratos como necesite.

Clientes del Sistema										
Tipo	Activo	Código Cliente	Nombre	Dirección Servicio	CIF	Teléfono Principal	Periodo	Importe	Fecha	Acciones
Cliente	No	243	Asociación Profesionales de la Construcción, S.L.U. (APC)	C/ Generalí, 11	9495000007		May	0,0	2014-05-24 02:36:25,0	
Cliente	Si	1000	Avea Perla Europa	Ave. de Torrelodones, 46	9157000004	9155000000	Mar	82,0	2014-05-24 02:36:27,0	
Cliente	Si	101	José Antonio Miguel Gómez	C/ Alfonso, nº 1 - 1A	9155000000	9155000010	Sep	36,0	2014-05-24 02:36:27,0	
Cliente	Si	1010	José Antonio Miguel Gómez	C/ Alfonso, 1 Nave 1/F/UF	1000000000	9155000000	Jul	42,35	2014-05-24 02:36:28,0	
Cliente	No	1013	José T. Sánchez de Carrizosa	C/ Andrés Mellado, 36	30000001-2	9147000004	Abr	263,68	2014-05-24 02:36:29,0	
Cliente	Si	1017	La Pineda de Colmenar, S.L.	C/ Pineda, 27	0000000000	9155000000	May	49,95	2014-05-24 02:36:33,0	
Cliente	Si	102	MAPAQUET, S.L.	C/ Roma, s/nif. Torre d'	9155000000	9155000000	Nov	64,0	2014-05-24 02:36:34,0	
Cliente	No	1032	Asociación de Pequeños Empresarios	C/ Francisco, 110	10017001-00	9155000000	May	28,68	2014-05-24 02:36:35,0	
Cliente	No	1034	SPN-Pascual Pérez Pérez	Ave. Monte Igualada, 76	71000000-00	9147000005	Jun	128,7	2014-05-24 02:36:36,0	
Cliente	Si	1036	José-Rafael Vicente	C/ General Ricardos, 200	9155000000	9155000007	Oct	30,0	2014-05-24 02:36:37,0	

Ilustración 6-4. Pantalla de búsqueda de Clientes

6.2. Gestores

Los gestores son las personas que nos sirven de enlace para los Clientes. Podemos asemejarlos a los responsables con los que tenemos que contactar en cualquier caso.

The screenshot shows a web-based application interface titled 'Gestores'. At the top, there are three logos: 'Universidad Rey Juan Carlos', a set of colorful icons, and 'SIRME'. Below the title is a toolbar with a 'Nuevo' button. The main area is a table titled 'Gestores del Sistema' with columns for Nombre, Teléfono, Correo, and Acciones. The table lists various users with their names, phone numbers, emails, and action buttons. At the bottom of the table, it says 'Registros del 101 al 110 de 191 (Página 11 de 20)' with a navigation bar.

Nombre	Teléfono	Correo	Acciones
Gestor Gestión	913019602	gcf_mmlm@hotmail.com	
ADMINISTRADORES DE FINCAS INMUEBLADA ABASCAL	918000001	a.m@inmueblada-abascal@gmail.com	
JOAQUIN SERGIO RUIZ	910420221	l.sergio@atmadrid.es	
IMPOR SERVICIOS, S.L.	913000979	e.s@imporservicios.com	
INTERGRA ADMINISTRACIÓN DE FINCAS (Centro: Centro)	911020001	g.prieto@intergrainf.com.es; g.prieto@intergrainf.telefonica.net	
IFUTURAS ASOCIADOS, S.L.	919070075	l.santos@ifuturas.es	
J.Fernández Gómez	913767198	info@josefernandezgomez.com; jfernandezgomez@ortiz.es	
Administración de Procesos		l.lopez@administraciondeprocesos.es	
JAVIER GUTIÉRREZ	919157700	j.gutierrez@icronsoft.es	
JORGE ANGEL VALLEJO LARRO	910000001	jgut@jgut.es	

Ilustración 6-5. Pantalla de búsqueda de Gestores

6.3. Contratos

Cada cliente podrá tener asignados 1 o varios contratos, que serán básicamente las direcciones en las que se tienen que prestar los servicios. Un aspecto importante, a parte de los datos básicos de la dirección del servicio, es que los contratos conllevan una periodicidad mensual por la que los servicios de revisión deben realizarse en unos meses determinados. La aplicación debe facilitar la búsqueda de clientes por este criterio.

The screenshot shows a web-based application interface titled 'Contratos'. At the top, there are three logos: 'Universidad Rey Juan Carlos', a set of colorful icons, and 'SIRME'. Below the title is a toolbar with a 'Nuevo' button. The main area is a table titled 'Direcciones de Servicio de los Clientes' with columns for Tipo, Activo, Código Contrato, Código Cliente, Nombre, Dirección Servicio, CIF, Período, Importe, and Acciones. The table lists service addresses for different clients, including their type (Comunidad), status, contract and client codes, names, addresses, CIF numbers, periods, and amounts. At the bottom, it says 'Registros del 1 al 4 de 4 (Página 1 de 1)' with a navigation bar.

Tipo	Activo	Código Contrato	Código Cliente	Nombre	Dirección Servicio	CIF	Período	Importe	Acciones
Comunidad	Si	861	3658	Comunidad de Bienes Alquiler, S.L.	CL. Mariano Iriarte, 4	E-3000000000	Oct	45.66	
Comunidad	Si	454	3029	Comunidad de Propietarios	CL. Alcalde Angel Arroyo, 4-6	3000000000	Oct	979.81	
Comunidad	Si	1157	3317	Comunidad de Propietarios	CL. Alcalde Roque de Beronda, 58	CIF 3313	Dic	691.22	
Comunidad	Si	1321	3797	Comunidad de Propietarios	CL. Alcalde Roque de Beronda, 74	M-000000073	Abr	0.0	

Ilustración 6-6. Pantalla de búsqueda de Contratos

6.4. Partes de trabajo

Un Parte de Trabajo es la asignación, en una fecha concreta, de un grupo de Técnicos a un Contrato para que vaya a revisar sus instalaciones.

El parte de trabajo contendrá el grupo de técnicos asociado, el Contrato, la fecha y un campo Observaciones que el administrativo puede utilizar a su libre disposición. Cuando un Parte de Trabajo

sea descargado, los datos que tenga relacionados serán bloqueados (no se podrán modificar) hasta que el Técnico termine su trabajo.

Partes de Trabajo											
Estado	Tipo	Albarán	Cliente	Nombre Cliente	Dirección Servicio	Equipo	Fecha	Estado	Recibido	Clasificación	Acciones
Aviso	2015.359	2817	BUSIT INF STORNUC, S.L.	Gobernación, 5	sergoyalvaro	24/07/2015	24/07/2015				<input type="checkbox"/> Acciones
Aviso	2015.358	3566	MANCOMUNIDAD DE PRIOL	LOS GALLINARES	sergoyalvaro	24/07/2015	24/07/2015				<input type="checkbox"/> Acciones
Aviso	2015.353	507	TELEMML, S.L.	CL. 100 MOLLET, 10 P.I. Alborondas	sergoyalvaro	16/07/2015	16/07/2015				<input type="checkbox"/> Acciones
Aviso	2015.352	3156	Comunidad de Propietarios	CI. TELLO, 96	sergoyalvaro	16/07/2015	16/07/2015				<input type="checkbox"/> Acciones
Parte de Trabajo	2015.327	3031	Comunidad de Propietarios	Pav. 101 la italiana, nº 26	sergoyalvaro	14/07/2015	14/07/2015			Revisión Anual	<input type="checkbox"/> Acciones
Parte de Trabajo	2015.328	3030	Comunidad de Propietarios	Pav. 101 la italiana, 24	sergoyalvaro	14/07/2015	14/07/2015			Revisión Anual	<input type="checkbox"/> Acciones
Aviso	2015.323	2654	Cooperativa de Apeis. Purcella 17	CL. Riverin c/nr CL. Mayerardo	sergoyalvaro	07/07/2015	07/07/2015				<input type="checkbox"/> Acciones
Aviso	2015.322	507	TELEMML, S.L.	CL. San Martín, 10 P.I. Alborondas	sergoyalvaro	08/07/2015	08/07/2015				<input type="checkbox"/> Acciones
Aviso	2015.319	3742	Comunidad de Propietarios	CI. Llanuras la Foret, 8	sergoyalvaro	03/07/2015	03/07/2015				<input type="checkbox"/> Acciones
Aviso	2015.318	3758	Comunidad de Propietarios	CL. 100 MOLLET, 1	sergoyalvaro	03/07/2015	03/07/2015				<input type="checkbox"/> Acciones

Ilustración 6-7. Pantalla de búsqueda de Partes de Trabajo

Estos partes de trabajo serán modificados y creados por cualquier rol con los permisos necesarios. La vista de los Partes contendrá los datos del parte, así como del cliente y el grupo de trabajo asociado.

Ilustración 6-8. Pantalla de Vista Avanzada de Partes de Trabajo

Cada una de las fichas, a su vez, podrá ser visualizada, exportada a PDF o modificada, si el usuario tiene permisos y se encuentra en una acción de edición.

Ilustración 6-9. Ejemplo de Acciones sobre Ficha de revisión

6.5. Fichas de revisión

Los resultados de las revisiones se anotan en las fichas de revisión que los técnicos han ido rellenando a lo largo del tiempo.

Estas fichas varían en función del tipo de elemento que se haya revisado (extintor, detector de humos, manguera...).

El sistema debe de ser capaz de generar estas fichas en PDF.

Nº Ref.	Nº Placa	Ubicación	Fabricació n	Último p	Caducidad	Tipo/Eficacia	Peso CO2 Correcto	Acces	Selaz	Estado (Exterior)	Necesario Revisar	Deseado	Codicio	Fuera Norma	Retirado	Operación
1	Plata 1º	Plata 1º	2007	2012		Peso 6 Kg 27A 1800	SI	Bien	Bien	No	No	No	No	No	No	No
2	Plata 2º	Plata 2º	2012	2014		Peso 6 Kg 27A 1800	SI	Bien	Bien	No	No	No	No	No	No	No
3	Plata 4º	Plata 4º	2012	2014		Peso 6 Kg 27A 1800	SI	Bien	Bien	No	No	No	No	No	No	No
4	Plata 5º	Plata 5º	2012	2014		Peso 6 Kg 27A 1800	SI	Bien	Bien	No	No	No	No	No	No	No

Ilustración 6-10. Ejemplo de Ficha de Revisión en formato PDF

También se dará la posibilidad de descargar todas las fichas de revisión en un único archivo comprimido en formato ZIP.

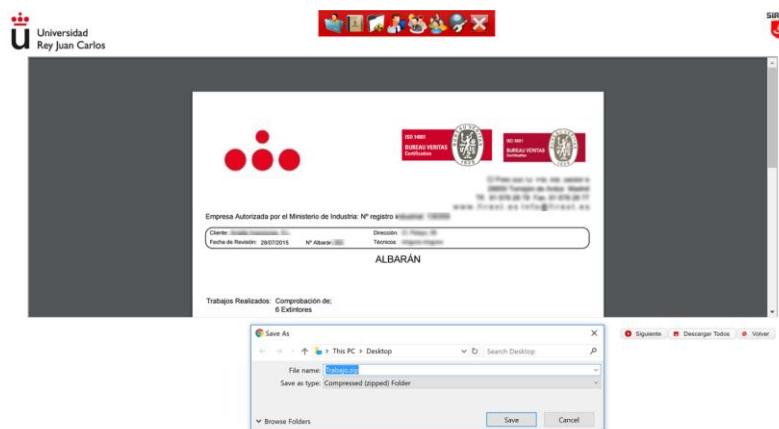


Ilustración 6-11. Descarga de ficheros comprimidos

Los partes de Trabajo podrán ser visualizados o modificados por cualquier usuario con los permisos necesarios.

1386 (Extintor)		240448 (Extintor)		2180337 (Extintor)	
Número de referencia	1	Número de referencia	2	Número de referencia	4
Número de Placa	1386	Número de Placa	240448	Número de Placa	2180337
Ubicación	Plata 1º	Ubicación	Plata	Ubicación	Plata 2º
Fabricación	2007	Fabricación	2014	Fabricación	2003
Último P.	Último P.	Último P.	2014	Último P.	2014
Caducidad		Caducidad		Caducidad	
Tipo / Eficacia	Peso 6 Kg 27A 1800	Tipo / Eficacia	Peso 6 Kg 27A 1800	Tipo / Eficacia	Peso 6 Kg 27A 1800
Peso CO2 Correcto		Peso CO2 Correcto		Peso CO2 Correcto	
Acceso	Bien	Acceso	Bien	Acceso	Bien
Selazación	Bien	Selazación	Bien	Selazación	Bien
Estado Exterior	Bien	Estado Exterior	Bien	Estado Exterior	Bien
Necesario Recoger	No	Necesario Recoger	No	Necesario Recoger	No
Desprendido	No	Desprendido	No	Desprendido	No
Caducado	No	Caducado	No	Caducado	No
Fuera Norma	No	Fuera Norma	No	Fuera Norma	No
Retirado	No	Retirado	No	Retirado	No
Operación		Operación		Operación	

Ilustración 6-12. Pantalla de Vista Avanzada de Ficha de Revisión

6.6. Equipos

Los equipos son una agrupación de 0 a N usuarios, y serán los encargados de realizar los trabajos. Sólo podrán ser integrantes de los equipos de trabajo aquellos usuarios con un rol de Técnico.

The screenshot shows a search interface for 'Equipos'. At the top, there are filters for 'Nombre' and 'Apellido'. Below the filters is a table titled 'Equipos de Técnicos' with columns for 'Código', 'Nombre', 'Apellido', 'Técnicos', 'Teléfono', and 'Acciones'. The table contains 22 rows of data, each representing a technician. The 'Técnicos' column lists names like Daniel Llorente, Sergio Mato, Alvaro Alonso, etc. The 'Teléfono' column lists phone numbers like 608 476 100, 649 256 650, etc. The 'Acciones' column contains small red icons for each row.

Ilustración 6-13. Pantalla de búsqueda de Equipos

6.7. Usuarios

Los usuarios del sistema que tienen acceso a la aplicación. Cualquiera que esté dado de alta en el sistema, podrá acceder a la aplicación, dependiendo del rol que desempeñe.

The screenshot shows a search interface for 'Usuarios'. At the top, there are filters for 'Nombre', 'Apellido 1', and 'Apellido 2'. Below the filters is a table titled 'Usuarios del Sistema' with columns for 'Código', 'Nombre', 'Apellido 1', 'Apellido 2', 'Acceso', and 'Acciones'. The table contains 15 rows of user data. The 'Acceso' column shows dates like 31/05/2017, 18/06/2015, 28/07/2015, etc. The 'Acciones' column contains small red icons for each row. At the bottom of the table, there is a message 'Registros del 1 al 10 de 15 (Página 1 de 2)' and a set of navigation icons.

Ilustración 6-14. Pantalla de búsqueda de Usuarios

6.8. Configuración

Los usuarios del sistema que tienen acceso a la aplicación. Cualquiera que esté dado de alta en el sistema, podrá acceder a la aplicación, dependiendo del rol que desempeñe.

The screenshot shows the 'Configuración' administration interface. It features a header with the 'Universidad Rey Juan Carlos' logo and the 'SIRME' logo. The main area has tabs for 'Configuración' and 'SIRME'. On the left, there is a sidebar with 'Opciones de Usuario' and a 'Acerca De' link. The central part of the screen displays several configuration panels: 'Datos Básicos' (with fields for Código de Usuario, Nombre, Apellido, etc.), 'Super User' (with checkboxes for 'Forzar DynDNS', 'Conexiones Web', etc.), and 'Super User' (with checkboxes for 'Procesar Directorio de Contratos', 'Procesar Directorio de Gestores', etc.). A small 'Acerca De' dialog box is also visible in the center.

Ilustración 6-15. Pantalla de Administración

6.9. Mensajes

La aplicación también contará con un sistema de aviso de mensajes al usuario, en los casos de que encuentre algún campo incorrecto o para indicar el resultado de las distintas operaciones realizadas.



Ilustración 6-16. Ejemplos de diferentes mensajes

7. Descripción informática

Las pruebas son el rigor ingenieril del Desarrollo del Software.

La Metodología de Desarrollo elegida para este proyecto ha seguido un modelo evolutivo para el desarrollo rápido de versiones incrementales. El nombre que lleva este tipo de metodologías de desarrollo software es ciclo de vida en espiral.

El desarrollo en espiral es un modelo de ciclo de vida del software definido por primera vez por Barry Boehm en 1986. Las actividades de este modelo se conforman en una espiral, en la que cada bucle o iteración representa un conjunto de actividades. Las actividades no están fijadas a ninguna prioridad, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior.

Partiendo de un concepto básico, basado en un análisis de requisitos inicial, se desarrolla rápidamente un sencillo prototipo que permite demostrar la validez del concepto. Una vez realizado este paso, se va incrementando la funcionalidad del prototipo, validando en cada paso la exactitud del estado actual en el que se encuentra el desarrollo. De este modo se obtiene una evolución consistente del sistema.

7.1. Ciclo de desarrollo incremental en espiral

Las etapas que caracterizan la metodología elegida son las siguientes:

7.1.1 Especificación y Análisis de Requisitos

Es la etapa donde se recogen los requisitos de la siguiente etapa o las funcionalidades que se desean añadir al prototipo obtenido en la anterior iteración.

En caso de que se trata de la iteración inicial, los requisitos definirán los pasos a seguir hacia el primer prototipo funcional.

Los requisitos pueden ser obtenidos a partir de nuevos casos de uso del sistema o a partir de mejoras observadas sobre los casos de uso ya existentes.

7.1.2 Diseño

Se elabora la arquitectura del producto objeto de la iteración y las clases o paquetes necesarios para cubrir los requisitos de la etapa anterior.

Se analizan las herramientas necesarias para llevar a cabo la implementación, estudiando las alternativas disponibles para obtener la solución descrita por la arquitectura.

7.1.3 Implementación

Es la parte de codificación de las funcionalidades descritas por los casos de uso, usándose para ello la estructura de clases y paquetes definida anteriormente.

Esta etapa incluye también la creación o modificación de documentación. Parte de los artículos servirá también a los futuros desarrolladores de proyecto.

7.1.4 Pruebas

Donde probamos si el código que acabamos de implementar es correcto del punto de vista del funcionamiento.

En teoría, cada clase y cada método deberían tener su batería de test, es decir, una cobertura del 100% es la meta que cada desarrollo software debe perseguir.

En este proyecto se han implementado pruebas para la mayoría de los casos de uso más habituales.



Ilustración 7-1. Ciclo de vida en Espiral

7.2. Tecnologías empleadas

Una de las características más interesantes, con diferencia, de este proyecto ha sido la variedad de las tecnologías y herramientas software que se han ido investigando y manipulando a lo largo de su desarrollo.

Se pueden destacar los marcos de trabajo (frameworks) que en algún punto del desarrollo se han tenido que acoplar al proyecto para conseguir los objetivos establecidos por los requisitos de usuario. De este modo, el proyecto finalizado presenta una estructura heterogénea del punto de vista de los módulos que lo componen.

7.2.1 Java



Ilustración 7-2. Logo de Java

La definición estándar del lenguaje Java expresa que es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90 (Wikipedia). En sus orígenes fue una evolución y a la vez, una simplificación de los lenguajes C y C++. Sus principales características son la orientación a objetos (el modelo de objetos se ha visto simplificado con respecto a C++), la independencia de plataforma (write once, run everywhere) y el recolector de basura.

La independencia de plataforma se obtiene mediante la ejecución del código fuente compilado a bytecode (código en instrucciones máquina de la plataforma Java) utilizando un programa llamado Java Virtual Machine (o JVM), específico de la plataforma destino. JVM es un programa escrito en código nativo para la plataforma donde se quiere ejecutar la aplicación que se ha desarrollado en Java, y se puede obtener desde la página oficial de Java, <http://java.com/en/download/index.jsp>.

Todas las tecnologías Java se ofrecen actualmente por la empresa Oracle que adquirió recientemente (en 2009) la empresa Sun, desarrolladora original de Java y sus derivados.

Para el desarrollo de este proyecto se han utilizado los paquetes y clases Java incluidos en la API JDK (Java Development Kit) de la edición estándar (Java Standard Edition). La versión del JDK ha sido la 1.6.

7.2.2 XML



XML es un metalenguaje extensible de etiquetas (*eXtensible Markup Language*) desarrollado por el World Wide Consortium (<http://www.w3.org/>) que se usa principalmente para el intercambio de datos (en Internet y otras partes). Además representa un formato de texto que permite la compatibilidad entre sistemas para compartir información estructurada de manera segura y fiable.

Se ha empleado para la configuración usada por maven así como todos los ficheros de configuración de la aplicación, como la gestión de LOG, datos de conexión a la base de datos o configuraciones internas para usuarios y administradores.

7.2.3 JSON



Ilustración 7-4. Logo de JSON

JavaScript Object Notation es un formato ligero para la representación e intercambio de datos en formato texto. Está basado en el lenguaje JavaScript y se usa mayormente para representar estructuras de datos simples y vectores asociativos. Aunque proviene de JavaScript, es independiente de lenguaje y existen generadores o lectores JSON para casi todos los lenguajes de programación.

En este proyecto se ha utilizado para representar el intercambio de información entre los servidores y los dispositivos móviles. La librería que da soporte se llama Jackson y permite convertir o recuperar objetos Java complejos a y desde objetos String en formato JSON.

7.2.4 HTML



Ilustración 7-5. Logo de HTML5

HTML, sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del World Wide Web Consortium (W3C) o Consorcio WWW, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la World Wide Web (WWW). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

Dado que se requiere que la aplicación esté disponible para cualquier dispositivo y desde cualquier lugar, se ha optado por desarrollar una aplicación basada en entorno web siguiendo los estándares de HTML5 y CSS3 para su compatibilidad absoluta.

HTML es un sistema normalizado para marcar los archivos de texto con el fin de lograr los efectos de fuente, color, gráficos e hipervínculos en las páginas del World Wide Web.

7.2.5 JavaServer Faces



Ilustración 7-6. Logo de JSF

JavaServer Faces (JSF) es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. JSF usa JavaServer Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías como XUL (acrónimo de XML-based User-interface Language, lenguaje basado en XML para la interfaz de usuario). JSF incluye:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
- Un conjunto por defecto de componentes para la interfaz de usuario.
- Dos bibliotecas de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaServer Faces dentro de una página JSP.
- Un modelo de eventos en el lado del servidor.
- Administración de estados.
- Beans administrados.

La especificación de JSF fue desarrollada por la Java Community Process como JSR 127, que definía JSF 1.0 y 1.1, JSR 252 que define JSF 1.2 y JSR 314 para JSF 2.0

7.2.5.1 PrimeFaces



Ilustración 7-7. Logo de PrimeFaces

PrimeFaces es una biblioteca de componentes para JavaServer Faces (JSF) de código abierto que cuenta con un conjunto de componentes enriquecidos que facilitan la creación de las aplicaciones web. Primefaces está bajo la licencia de Apache License V2. Una de las ventajas de utilizar Primefaces, es que permite la integración con otros componentes como por ejemplo RichFaces.

Las principales propiedades son:

- Conjunto de componentes ricos (Editor de HTML, autocompletar, cartas, gráficas o paneles, entre otros).
- Soporte de ajax con despliegue parcial, lo que permite controlar qué componentes de la página actual se actualizarán y cuáles no.
- 25 temas prediseñados
- Componente para desarrollar aplicaciones web para teléfonos móviles, especiales para iPhones, Palm, Android y teléfonos móviles Nokia.

7.2.6 Apache Maven



Ilustración 7-8. Logo de maven

Maven, actualmente un proyecto de nivel superior de la Apache Software Foundation, es una herramienta para la gestión y construcción de proyectos, especialmente en el lenguaje Java siguiendo unos estándares bien definidos.

Maven ha sido pensado para aliviarle al programador las tareas de configuración del proyecto software siempre y cuando se adhiere al principio de Convención sobre Configuración.

Los proyectos que siguen la arquitectura Maven deben incluir en su raíz un fichero XML de configuración llamado pom.xml (las siglas de Project Object Model, o modelo de objetos del proyecto). Este fichero incluye detalles sobre el proyecto y cómo va a ser construido por Maven. Una de sus grandes ventajas con respecto a otras herramientas de este tipo (como, por ejemplo, Ant) es que permite coordinar las dependencias de librerías, internas como externas, a través del fichero POM de una manera sencilla (basta con especificar el nombre y el número de versión de la librería que se quiere incorporar al proyecto).

Maven es un programa que se puede ejecutar desde línea de comandos (en Linux, el comando mvn), o bien, a través de un entorno de desarrollo integrado como Eclipse. En el caso de Eclipse, su uso ha sido posible debido al complemento (o plugin) M2E: <http://www.eclipse.org/m2e/>.

7.2.7 Spring



Ilustración 7-9. Logo de Spring

Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java; permite la configuración de los componentes de aplicación y la administración del ciclo de vida de los objetos Java, se lleva a cabo principalmente a través de la inyección de dependencias.

La primera versión fue escrita por Rod Johnson, quien lo lanzó junto a la publicación de su libro Expert One-on-One J2EE Design and Development (Wrox Press, octubre 2002). El framework fue lanzado inicialmente bajo la licencia Apache 2.0 en junio de 2003. El primer gran lanzamiento fue la versión 1.0, que apareció en marzo de 2004 y fue seguida por otros hitos en septiembre de 2004 y marzo de 2005. La versión 1.2.6 de Spring Framework obtuvo reconocimientos Jolt Awards y Jax Innovation Awards en 2006. Spring Framework 2.0 fue lanzada en 2006, la versión 2.5 en noviembre de 2007, Spring 3.0 en diciembre de 2009, y Spring 3.1 dos años más tarde. El inicio del desarrollo de la versión 4.0 fue anunciado en enero de 2013. La versión actual es 4.3.7.

Si bien las características fundamentales de Spring Framework pueden ser usadas en cualquier aplicación desarrollada en Java, existen variadas extensiones para la construcción de aplicaciones web sobre la plataforma Java EE. A pesar de que no impone ningún modelo de programación en particular, este framework se ha vuelto popular en la comunidad al ser considerado una alternativa, sustituto, e incluso un complemento al modelo EJB (Enterprise JavaBean).

7.2.8 JPA e Hibernate



Ilustración 7-10. Logo de JPA

Java Persistence API, más conocida por sus siglas JPA, es la API de persistencia desarrollada para la plataforma Java EE. La cual es un framework del lenguaje de programación Java, que maneja datos relacionales en aplicaciones usando la plataforma Java en sus ediciones Standard (Java SE) y Enterprise (Java EE).

Por su parte, Hibernate es una herramienta de Mapeo objeto-relacional (ORM), para la plataforma Java –también disponible para .Net, bajo el nombre de NHibernate –que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación; mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones. Este es software libre, distribuido bajo los términos de la licencia GNU LGPL.

Es importante destacar que JPA es una parte de la especificación de EJB 3, es decir que no es un framework, sino que es simplemente un documento en el cual se especifica los principios básicos de gestión de la capa de persistencia en el mundo de Java EE. En cambio, Hibernate, si se trata de un framework que gestiona la capa de persistencia a través de ficheros xml o anotaciones.



Ilustración 7-11. Logo de Hibernate

La relación que existe entre JPA e Hibernate, radica en que este último implementa como parte de su código la especificación de JPA; es decir que se puede usar Hibernate para construir una capa de persistencia apoyándose en las definiciones y reglas que la especificación de JPA, aunque esto no sea obligatorio.

Sin embargo, esto no quiere decir que Hibernate simplemente implemente el standard de JPA; Hibernate es mucho más grande que la especificación de JPA y añade más funcionalidad.

7.2.9 SLFJ



Ilustración 7-12. Logo de SL4J

Es una API de registro Java a través de un simple patrón de fachada (en inglés, Simple Logging Facade for Java). La ventaja que proporciona es el desacoplamiento entre la aplicación y el marco de registro (logging framework), que permite cambiar del proveedor de servicio de registro en cualquier momento.

SLF4J establece cinco niveles de registro, desde trace (grano muy fino) hasta error (grano muy grueso). Todas las llamadas a los métodos de registro especifican (a través de su nombre) el nivel de prioridad. En tiempo de ejecución se verifica de manera automática el nivel de prioridad de la aplicación y las llamadas de registro tienen efecto solamente si su nivel de prioridad es mayor o igual que el de la aplicación.

Se han empleado las clases y los métodos de SLF4J de manera constante durante el desarrollo del proyecto para informarle al desarrollador de los problemas en tiempo de ejecución. Dado que el nivel de granularidad (en desarrollo) de los mensajes es muy fino, de bajo nivel, se ha tenido en cuenta a la hora de publicar el proyecto para cambiar el nivel de prioridad de la aplicación y dejar que se le avisara al usuario solamente en caso de problema grave.

7.2.10 Estilos CSS



Ilustración 7-13. Logo de CSS

Los selectores CSS y XPath se han utilizado para especificar elementos HTML de interés en la estructura de las páginas web visitadas.

XPath es un lenguaje para seleccionar nodos en un fichero XML o HTML; aunque no es muy rápido comparado con el otro mecanismo (CSS), tiene más versatilidad y es más preciso, a veces siendo la única opción para conseguir la identificación de un elemento HTML desde JavaScript o algunos avances que hice son Selenium.

CSS es un lenguaje para definir las semánticas de la presentación de los documentos escritos en formato de marcado, como XML o HTML. Permite separar el contenido de la presentación (estilo y formato) del documento. CSS utiliza selectores (conjuntos de reglas) para dar estilo a ciertos elementos o nodos de un documento HTML.

7.2.11 Servicios web y REST



Ilustración 7-14. Logo de REST

Los servicios web son un enfoque moderno y ligero para las aplicaciones distribuidas. Se construyen encima del protocolo de aplicación HTTP pero su finalidad no es proporcionarle al cliente una página en formato HTML que sea mostrada en su navegador web, sino que hacerle llegar información estructurada que suele ser procesada en su lado.

SOAP (originalmente las siglas de Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de un protocolo creado por Dave Winer en 1998, llamado XML-RPC. SOAP fue creado por Microsoft, IBM y otros. Está actualmente bajo el auspicio de la W3C. Es uno de los protocolos utilizados en los servicios Web.

La Transferencia de Estado Representacional (en inglés Representational State Transfer) o REST es un estilo de arquitectura software para sistemas hipermédia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.

REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.

Los principios generales de un sistema REST son los siguientes:

- Ser un sistema cliente – servidor
- No tener estado (stateless), es decir que no tenga que soportar sesiones para los usuarios y cada petición es independiente de las demás
- Incluir un sistema de caché por niveles
- Los recursos tienen que ser accesible a través de una única dirección de acceso, es decir el sistema será accesible de forma uniforme
- Ser escalable

Los sistemas que cumplan los principios REST se llaman RESTful.

7.2.11.1 Idempotencia

En matemática y lógica, la idempotencia es la propiedad para realizar una acción determinada varias veces y aun así conseguir el mismo resultado que se obtendría si se realizase una sola vez. Un elemento que cumple esta propiedad es un elemento idempotente, o un idempotente. De esta manera, si un elemento al multiplicarse por sí mismo sucesivas veces da él mismo, este elemento es idempotente. Por ejemplo, los dos únicos números reales que son idempotentes, para la operación producto (\cdot), son 0 y 1. ($0 \cdot 0 = 0, 1 \cdot 1 = 1$).

El principio de idempotencia aplicado a REST (realmente es a los verbos HTTP) nos dice que, la ejecución repetida de una petición con los mismos parámetros sobre un mismo recurso tendrá el mismo efecto en el estado de nuestro recurso en el sistema si se ejecuta 1 o N veces.

La respuesta del servidor puede cambiar según el número de petición. Por ejemplo, si es una petición de creación/actualización de un recurso, si dicho recurso no existiese, la primera vez que realizamos la petición el servidor nos podría devolver un código 201 (CREATED) y las siguientes veces podría devolver 200 o 204.

Aquí podemos ver un listado de los verbos HTTP más comúnmente utilizados donde se indica si son o no idempotentes.

VERBO	IDEMPOTENTE
GET	SI
POST	NO
PUT	SI
DELETE	SI

Ilustración 7-15. Tabla de Verbos HTTP

La gran ventaja de la idempotencia es que podemos reintentar una petición contra el servidor tantas veces como necesitemos hasta que tengamos la certeza de que el estado de nuestro recurso en el sistema está justo como nosotros queríamos. Esto adquiere especial importancia en operaciones de creación de recursos donde necesitamos garantizar que ese recurso se creará una única vez en el servidor y donde queremos contar con una política de reintentos en caso de que no obtengamos respuesta del servidor.

7.2.11.2 Semántica

Se ha hecho un uso apropiado de la semántica a la hora de elegir los nombres de los servicios REST.

7.2.11.3 Versionado

Quedaría para futuras implementaciones el uso de un correcto versionado para los servicios REST, mediante códigos metadata en la estructura del objeto o mediante las cabeceras “Content-Type” y “Accept”, o incluso añadiendo la versión del API en la dirección URI, lo cual no es el método más apropiado pero puede resultar pragmático.

También, si el sistema creciese, deberíamos añadir paginación a los listados de resultados mediante los verbos GET, dado que ahora mismo está devolviendo todos los registros.

7.2.11.4 Richardson Maturity Model

Existen tres niveles de calidad a la hora de aplicar REST en el desarrollo de una aplicación web y más concretamente una API que se recogen en un modelo llamado Richardson Maturity Model en honor al tipo que lo estableció, Leonard Richardson padre de la arquitectura orientada a recursos. Estos niveles son:

- Uso correcto de URIs
- Uso correcto de HTTP.
- Implementar Hypermedia.

Además de estas tres reglas, nunca se debe guardar estado en el servidor, toda la información que se requiere para mostrar la información que se solicita debe estar en la consulta por parte del cliente.

Al no guardar estado, REST nos da mucho juego, ya que podemos escalar mejor sin tener que preocuparnos de temas como el almacenamiento de variables de sesión e incluso, podemos jugar con distintas tecnologías para servir determinadas partes o recursos de una misma API.

7.2.11.5 Jersey



Ilustración 7-16. Logo de Jersey

Es la implementación por referencia de SUN; la librería open source que implementa la especificación JAX-RS se llama Jersey y es ofrecida por Oracle (es la implementación de referencia). Su objetivo declarado es aliviar el desarrollo de los servicios web RESTful con Java y la máquina virtual Java.

7.2.12 JUnit



Ilustración 7-17. Logo de JUnit

Es un marco de trabajo diseñado para construir y ejecutar pruebas en los proyectos desarrollados en el lenguaje Java.

Utiliza las anotaciones para identificar los métodos que deberían ser considerados métodos de test.

Durante la ejecución del método de test, se prueba si la parte del proyecto (una clase, método, servicio, etc.) que se está inspeccionando, está haciendo lo correcto o no.

Para asegurar la fiabilidad de los test, se crea un entorno de test para aislar lo que se quiere probar del resto del sistema, y sin necesidad de ejecutar toda la aplicación. Por esta razón, se considera que los test creados con JUnit son unitarios, ya que se ponen a prueba las unidades que forman el sistema completo.

7.2.13 MySQL



Ilustración 7-18. Logo de MySQL

MySQL es uno de los sistemas de gestión de bases de datos relacionales más populares a nivel mundial (actualmente en la primera posición (MySQL)), ofreciendo un altísimo nivel de consistencia y disponibilidad, además de ser gratuito.

El acceso a las bases de datos gestionadas por MySQL se ofrece por medio de un servidor multihilo y multiusuario. Es muy habitual el uso de la edición Community, gratuita y open source (o de código abierto), en los proyectos open source que precisan de un sistema completo de gestión de bases de datos. También está teniendo mucho éxito en los proyectos relacionados con el desarrollo de las páginas o aplicaciones web (el famoso conjunto de tecnologías para el desarrollo de servidores web, LAMP).

Además, la instalación y el uso de MySQL son unos procesos sencillos y a la vez estandarizados, con muchos artículos y tutoriales disponibles en Internet.

7.2.14 Git



Ilustración 7-19. Logo de GitHub

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

7.2.14.1 GitHub

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por GitHub, Inc. (anteriormente conocida como Logical Awesome). Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc.

El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

Todo el código fuente de SIRME está disponible en <https://github.com/ganzux/SIRME>

7.2.15 Jasper Report



Ilustración 7-20. Logo de Jasper Soft

JasperReports es una biblioteca de creación de informes que tiene la habilidad de entregar contenido enriquecido al monitor, a la impresora o a ficheros PDF, HTML, XLS, CSV y XML.

Está escrito completamente en Java y puede ser usado en gran variedad de aplicaciones de Java, incluyendo J2EE o aplicaciones web, para generar contenido dinámico.

JasperReports se usa comúnmente con iReport, un front-end gráfico de código abierto para la edición de informes, si bien a partir de la versión 5.5.0 iReport ha sido sustituido por Jaspersoft Studio, un front-end gráfico de código abierto basado en Eclipse.

Se encuentra bajo licencia libre GNU, por lo que es Software libre. Forma parte de la iniciativa apilada open source Lisog.

7.2.16 HSQLDB



Ilustración 7-21. Logo de HSQLDB

HSQLDB (Hyperthreaded Structured Query Language Database) es un sistema gestor de bases de datos libre escrito en Java. Lo interesante es que permite un modo “en memoria” que resulta ideal para ejecutarlo junto con las pruebas unitarias, controlando perfectamente que información tenemos en la base de datos y podremos saber cual es el resultado que queremos obtener de nuestra prueba unitaria.

7.2.17 TravisCI



Ilustración 7-22. Logo de TravisCI

Se trata de un servidor de integración continua que incluye la posibilidad de ser integrado con GitHub de manera automática, por lo que cada vez que realizamos un cambio en nuestro código podemos especificar qué acciones queremos hacer con él, como la ejecución de tests, construcción del proyecto, etc.

8. Arquitectura

La simplicidad llevada al extremo se convierte en elegancia.

Modelo–vista–controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

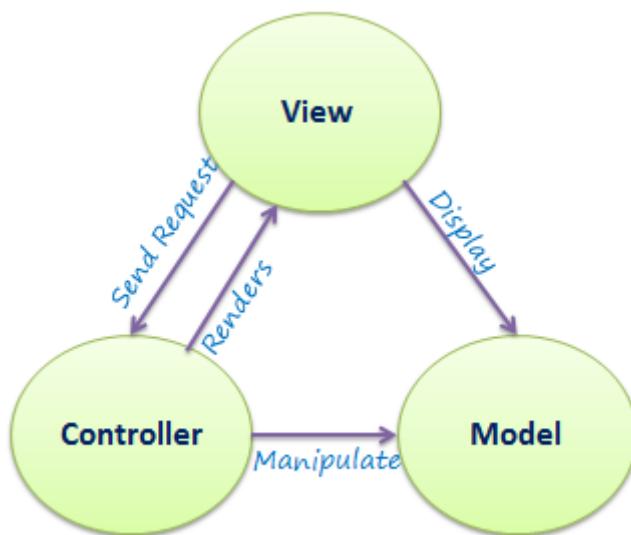


Ilustración 8-1. MVC

SIRME hace uso de esta arquitectura, separando el código a 3 niveles: **web**, **business** y **core**.

	ganzux committed on GitHub correct spaces and tabs
	SIRME-business new date in data base and sort rest services
	SIRME-core fixed jta dependency
	SIRME-web new date in data base and sort rest services

Ilustración 8-2. Subproyectos SIRME

Grosso modo la capa de persistencia utiliza los estándares de **JPA** bajo la pseudoimplementación que hace Hibernate siempre por medio de unas clases **DAO** bajo el proyecto **SIRME-CORE**.

Estas clases serán las encargadas de acceder a los recursos en base de datos utilizando la potencia que ofrece **JPA**, por el que podemos cambiar de motor de base de datos sin tener que tocar apenas nuestro código.

Es esta misma capa de persistencia, y fuera de las bases de datos, se pueden manejar ficheros del sistema.

La capa de control o servicio utiliza **Spring Transaction** para controlar las transacciones en base de datos, y será la encargada de mantener las sesiones de aplicación, así como la administradora de las estructuras de datos, cachés, configuraciones y demás elementos que necesite la vista.

Dentro de esta capa se engloban también los accesos por **REST** que pueden hacerse desde fuera, y que accederán a los Servicios. Estas clases se encuentran en el paquete.

Existe una capa de generación de informes mediante la librería Jasper Reports.

La capa de vista está basada en el estándar **JSF2** y hace uso de la librería **PrimeFaces** bajo su versión 4; básicamente están formados por las clases “**ManagedBean**” y aquellas que puedan serle de utilidad, así como de todas las páginas **XHTML** que tiene la aplicación.

Los diferentes módulos del proyecto tienen paquetes de pruebas unitarias gestionadas por el framework **JUnit**, revisando un porcentaje de código superior al 60% que, si bien no asegura que el software no tenga fallos, nos transmite cierta tranquilidad al respecto.

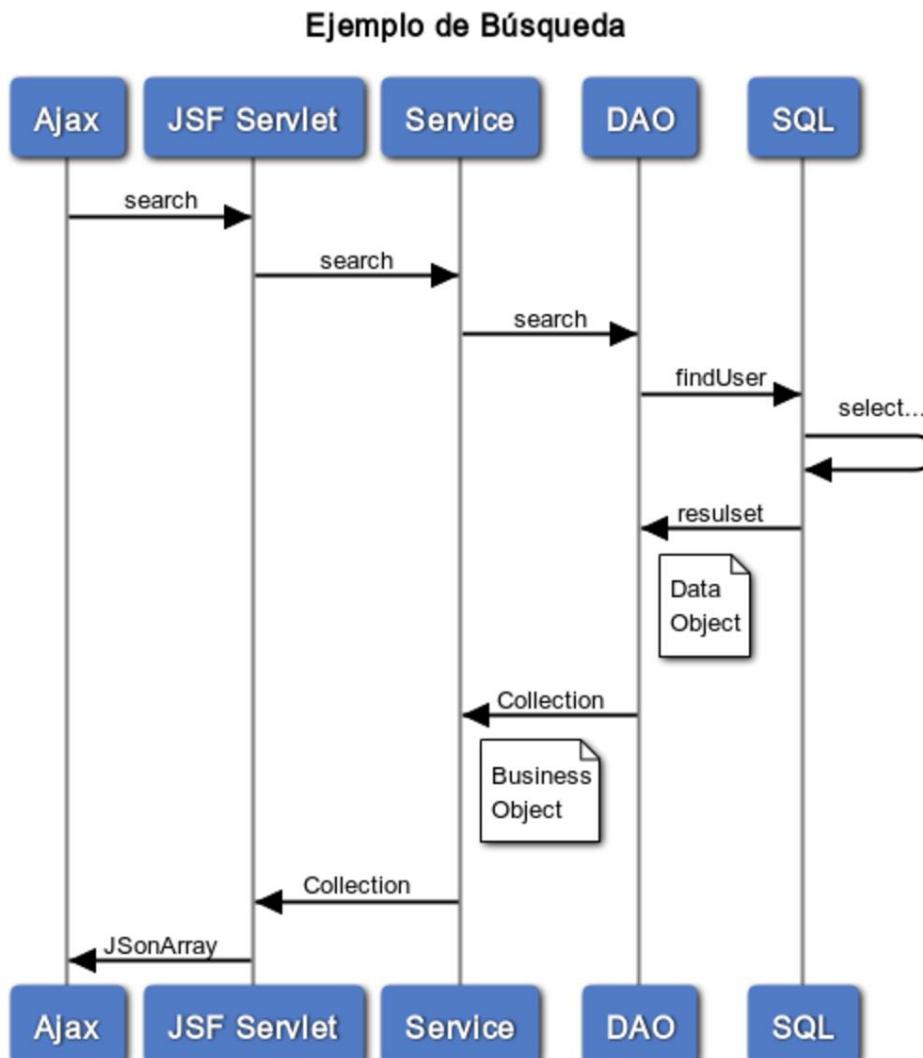


Ilustración 8-3. Diagrama de secuencia de ejemplo de búsqueda

8.1. MODEL

SIRME-CORE es el núcleo de la aplicación, que contiene la capa de persistencia. El modelo de datos se compone de 3 partes bien diferenciadas. Por un lado, tenemos las entidades referentes a los usuarios, con sus roles, perfiles y permisos. Estos serán usados para definir un usuario del sistema.

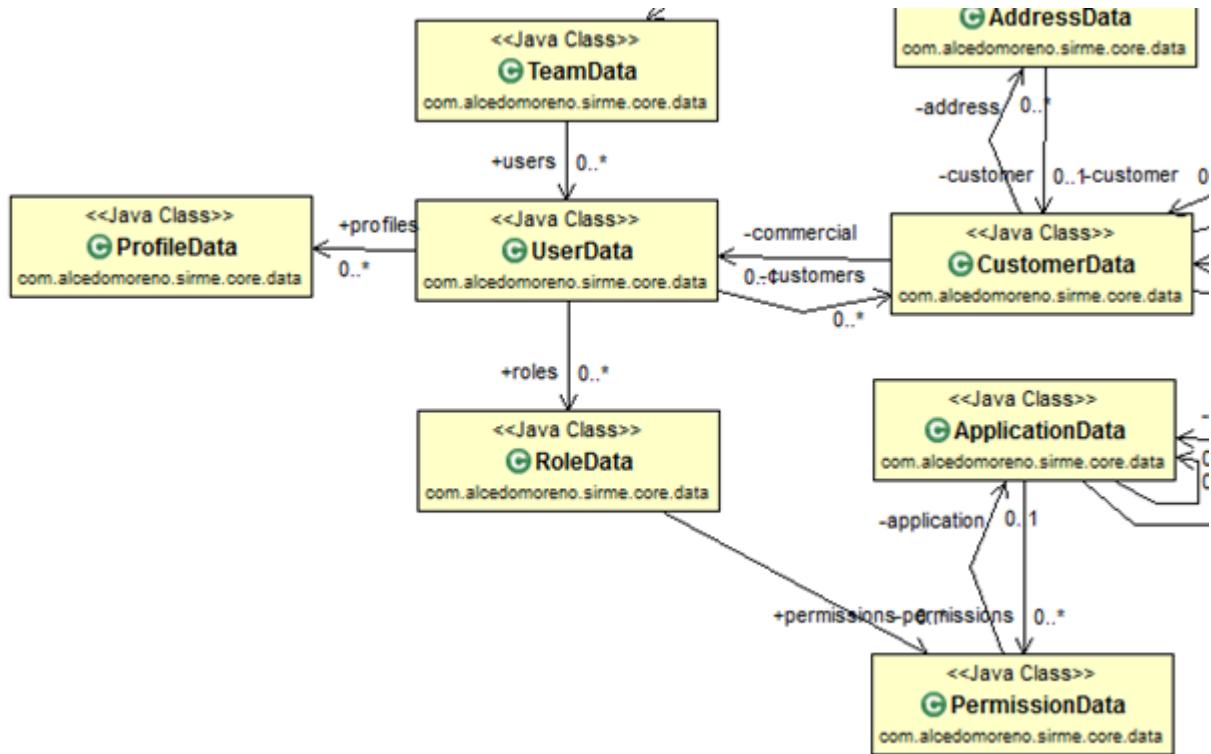


Ilustración 8-4. Detalle del Diagrama de Clases de Usuarios

Por otro lado, tenemos los trabajos realizados por los equipos de trabajo en una dirección de un cliente. Los equipos de trabajo estarán formados por cero o varios usuarios del sistema (punto de unión con el anterior extracto del esquema). El trabajo será el encargado de mantener la información del mismo, con notas adicionales, fecha, estado, etc.

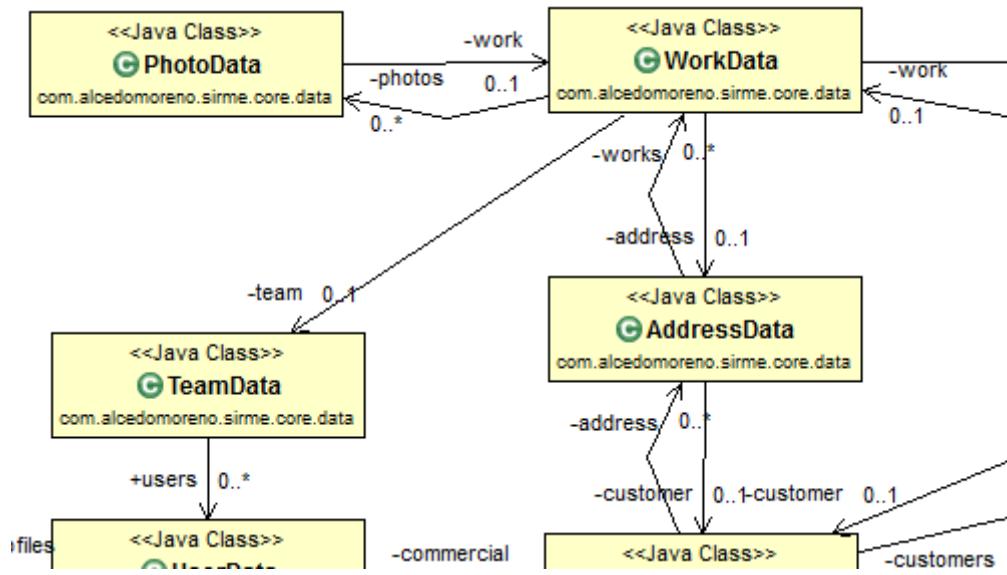


Ilustración 8-5. Detalle del Diagrama de Clases de Trabajos

Y por último los datos propios de este trabajo, con su “configuración previa” y las preguntas/resuestas almacenadas; estas son un grupo de respuestas, que pertenecen a un informe concreto. Básicamente aquí podemos observar la configuración de las preguntas y la tipología de respuesta esperada y en ReplyData es donde se van almacenando todas las respuestas.

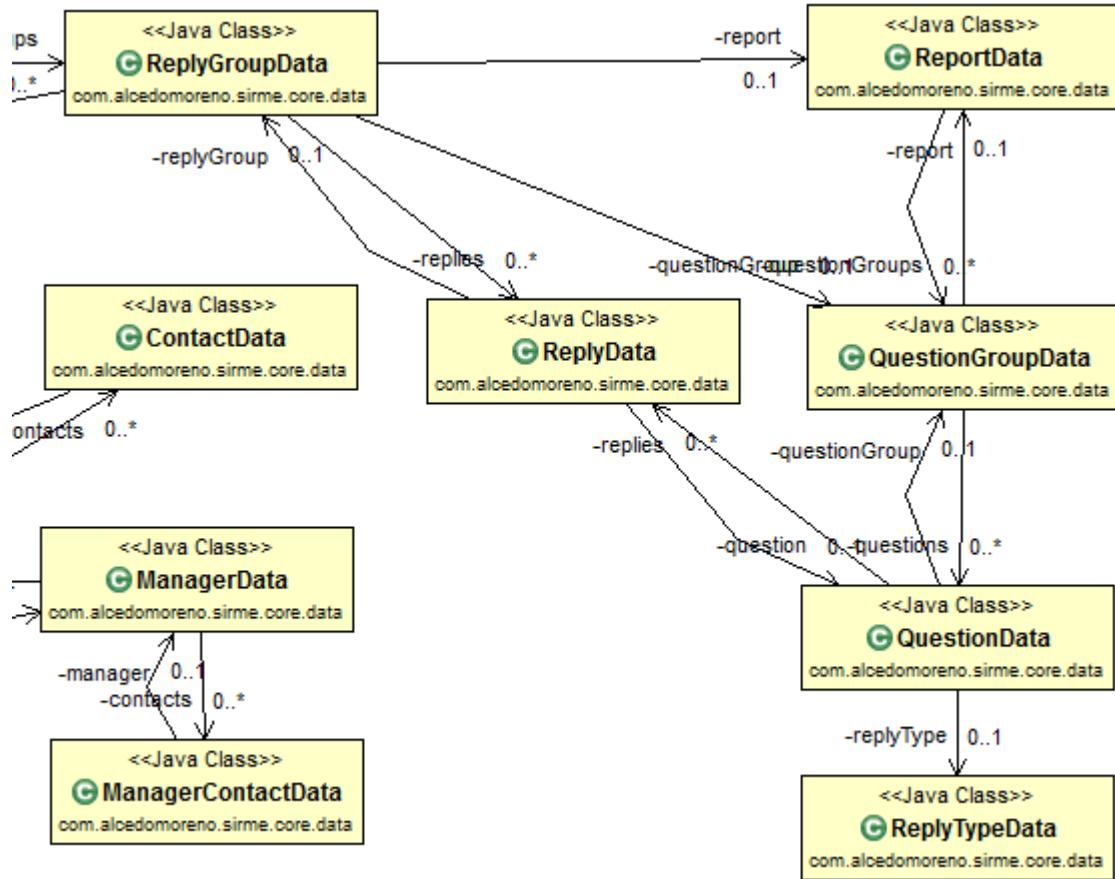


Ilustración 8-6. Detalle del Diagrama de Clases de Preguntas y Respuestas

Aquí podemos ver el diagrama completo con las relaciones entre las 3 partes anteriormente explicadas.

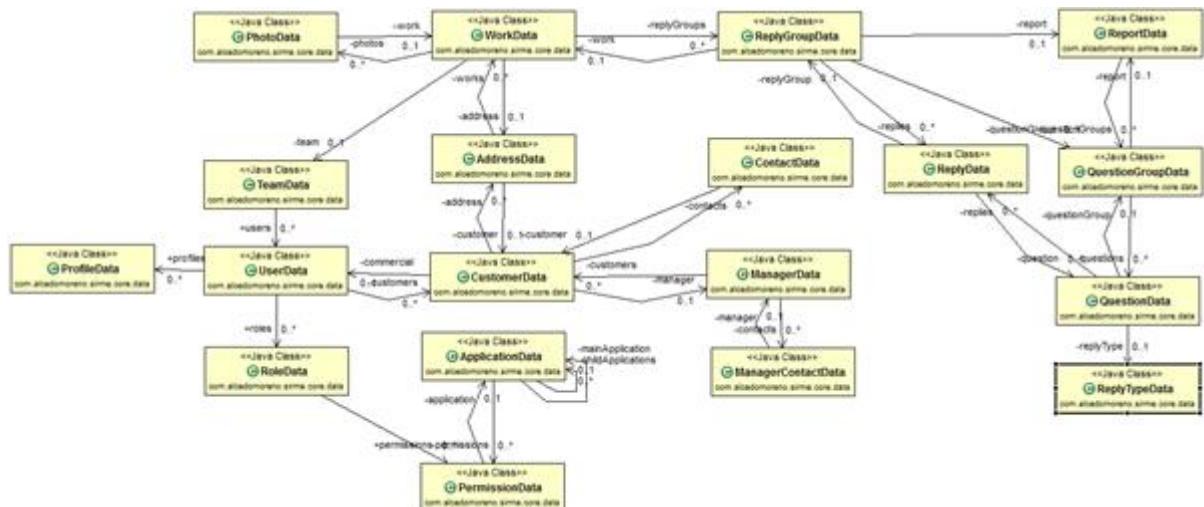


Ilustración 8-7. Detalle del Diagrama de Clases completo

A nivel de código, en su fichero POM, observamos que incluimos las dependencias a hibernate o mysql entre otras. Estas referencias, por supuesto, no serán encontradas en otros proyectos dado que son librerías de la capa de persistencia.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>3.3.2.GA</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-annotations</artifactId>
    <version>3.4.0.GA</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.15</version>
</dependency>
```

Ilustración 8-8. Dependencias maven de SIRME-CORE

El proyecto tiene 3 grandes paquetes, dao, data y util.

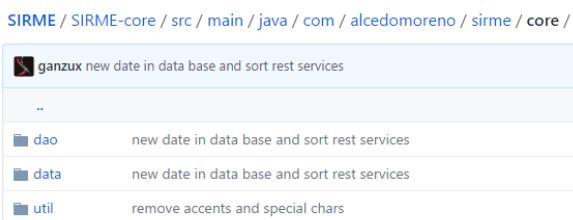


Ilustración 8-9. Paquetes SIRME-CORE

8.1.1 util

El paquete contendrá algunas clases con utilidades varias y constantes.

8.1.2 data

De manera análoga a la capa de negocio, este paquete contendrá los Bean asociados a la base de datos, siendo un fiel reflejo de la misma y estando anotados con las anotaciones de Hibernate.

Todos estos objetos heredarán de DataObject con el único objetivo de poder definirlos como objetos de la capa de persistencia, dado que la interfaz estará vacía.

Al igual que en la capa de datos, la capa de persistencia tendrá 3 partes bien diferenciadas. Por un lado tenemos los usuarios con sus roles y sus perfiles, que pertenecen (o no) a un equipo de trabajo.

8.1.3 dao

Es el paquete donde tenemos la capa de persistencia, el último paso para conectar con la librería de Hibernate, ya sea mediante anotaciones, HQL o accediendo a las Entities directamente. De manera análoga a los servicios, tendremos una interfaz por cada uno de los DAO y una implementación.

8.2. VIEW

SIRME-web se encarga de la capa de la presentación (**VIEW**); en su fichero de configuración de **maven** se incluyen las librerías pertenecientes a **jstl**, **faces** y **primefaces**, **poi** y **jasperreport**.

```
<dependency>
    <groupId>jstl</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>

<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.1.7</version>
</dependency>
<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-impl</artifactId>
    <version>2.1.7</version>
</dependency>
```

```
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>4.0</version>
</dependency>

<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-scratchpad</artifactId>
    <version>3.10.1</version>
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-contrib</artifactId>
    <version>3.6</version>
</dependency>
<dependency>
    <groupId>net.sf.jasperreports</groupId>
    <artifactId>jasperreports</artifactId>
    <version>5.5.2</version>
</dependency>
```

Ilustración 8-10. Dependencias maven para SIRME-WEB

La estructura de directorios está dividida en 2: **webapp** (donde se encuentran todos los recursos estáticos de presentación) y **java** (donde tenemos el código fuente que controla las vistas y algunos ficheros de configuración necesarios para el correcto funcionamiento).

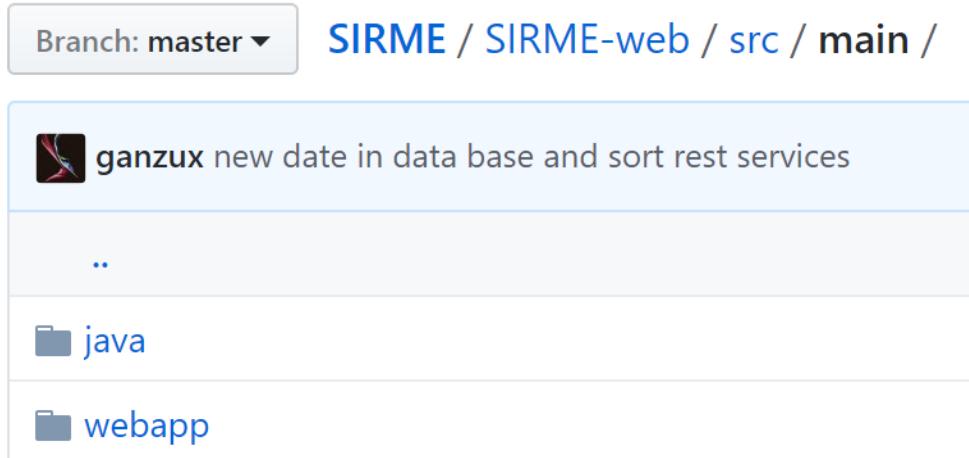


Ilustración 8-11. Directorios de SIRME-WEB

8.2.1 webapp

A continuación, se detallan los ficheros más importantes o representativos de cada directorio.

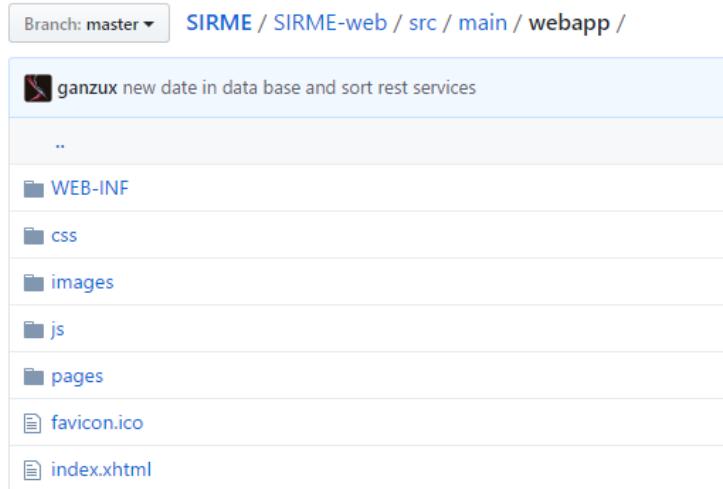


Ilustración 8-12. Directorios webapp

8.2.1.1 index.xhtml

Es la página de entrada a la aplicación, habiendo quedado configurada en el fichero webapp/WEB-INF/web.xml

Contiene la redirección a la página de inicio de la aplicación, perteneciente al directorio “pages”. Esta decisión se ha llevado a cabo por el sistema de plantillas empleado, dado que quería que todas las páginas siguieran la misma estructura.

8.2.1.2 pages

Para el desarrollo de las vistas me he basado en una estructura de plantillas utilizando, excepto en contadas ocasiones, la plantilla ubicada en “SIRME-web/src/main/webapp/WEB-INF/templates/tpl_str_non_load.xhtml”. Esta plantilla establece las cabeceras de las páginas HTML así como incorpora los estilos de la aplicación y el código javascript.

También define las diferentes zonas de la pantalla mediante elementos CSS DIV, como puede ser el menú de la aplicación o el contenido de la misma.

Las pantallas se han decidido en aquellas que realizan una búsqueda y aquellas que permiten la manipulación o visualización de registros concretos. Así, por ejemplo, estamos definiendo la misma cabecera, menú y pie de página para la búsqueda y manipulación de clientes, cambiando sólo el elemento “content” de la plantilla:

```

<cui:define name="header">
    <ui:include src="/WEB-INF/header/hdr_general.xhtml" />
</ui:define>

<cui:define name="menu">
    <ui:include src="/WEB-INF/menu/menu_general.xhtml" />
</ui:define>

<cui:define name="content">
    <ui:include src="/WEB-INF/contents/cnt_customers.xhtml"/>
</ui:define>

<cui:define name="footer">
    <ui:include src="/WEB-INF/footer/ftr_general.xhtml" />
</ui:define>

<cui:define name="modals">
    <ui:include src="/WEB-INF/modals/mod_general.xhtml" />
</ui:define>

```



```

<cui:define name="header">
    <ui:include src="/WEB-INF/header/hdr_general.xhtml" />
</ui:define>

<cui:define name="menu">
    <ui:include src="/WEB-INF/menu/menu_general.xhtml" />
</ui:define>

<cui:define name="content">
    <ui:include src="/WEB-INF/contents/cnt_customers_e.xhtml"/>
</ui:define>

<cui:define name="footer">
    <ui:include src="/WEB-INF/footer/ftr_general.xhtml" />
</ui:define>

<cui:define name="modals">
    <ui:include src="/WEB-INF/modals/mod_general.xhtml" />
</ui:define>

```

Ilustración 8-13. Diferencia plantilla de búsqueda y edición

De esta manera se consigue homogeneidad en las vistas de la aplicación y un rápido desarrollo de nuevas funcionalidades.

8.2.1.3 js

Contiene los ficheros con el código Javascript de la aplicación. Tiene el fichero con jQuery v1.9.1 y otro “util.js” donde se coloca todo el código javascript que podamos utilizar.

8.2.1.4 images

Contiene las imágenes, logos e iconos de la aplicación.

8.2.1.5 css

Contiene los estilos de la aplicación en CSS3.

8.2.1.6 WEB-INF

El directorio WEB-INF contiene el fichero de configuración “web.xml”, de suma importancia para todos los aspectos de despliegue y ejecución en un servidor web así como diferentes carpetas con los recursos estáticos y dinámicos que se utilizan para la renderización de la página en pantalla.

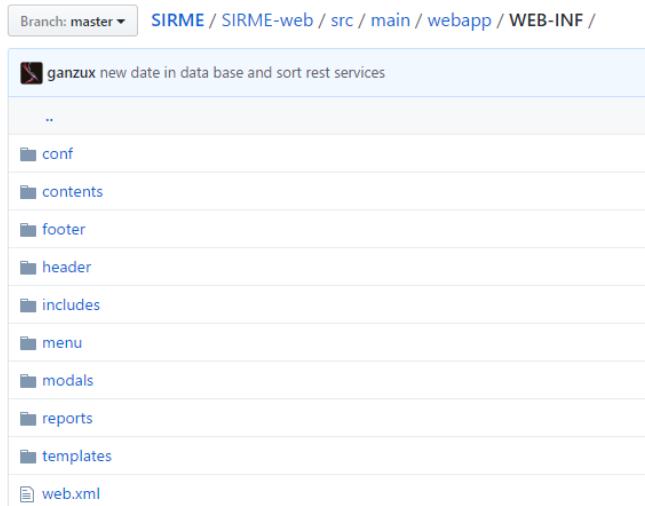


Ilustración 8-14. Contenido WEB-INF

8.2.1.6.1 web.xml

Quedan definidas las configuraciones para JSF, mvc-dispatcher de Spring, Facelets, Servlets, encoding, filtros de seguridad, Primefaces, página de inicio, etc.

8.2.1.6.2 templates

Plantillas de la aplicación que se utilizan desde las “pages”.

8.2.1.6.3 reports

Aquí se ubican todas las plantillas de “Jasper Reports”, así como sus compilaciones y las imágenes que utiliza.

8.2.1.6.4 modals

Ventanas modales que se utilizan para información emergente.

8.2.1.6.5 menu

Contiene un único fichero “menu_general.xhtml” con los enlaces a las distintas pantallas de la aplicación. A la hora de dibujar los iconos con los enlaces, hará una primera comprobación de seguridad por si el usuario tiene o no acceso al módulo correspondiente.

El acceso a los diferentes módulos de la aplicación seguirá el mismo patrón una y otra vez, esto es, ejecutará el método “dolnit” del “Bean” correspondiente al módulo.

8.2.1.6.6 includes

Algunas páginas de aspectos avanzados que se incluyen a nivel de plantilla, como la comprobación de expiración de la sesión del usuario (configurada, de igual manera, en web.xml) y la animación “Cargando” que se visualiza cada vez que una petición es enviada al servidor.

8.2.1.6.7 header

Contiene la cabecera de las páginas de la aplicación.

8.2.1.6.8 footer

Contiene el pie de las páginas de la aplicación.

8.2.1.6.9 contents

Contiene los ficheros con el contenido de la búsqueda de entidades y la manipulación de las mismas.

8.2.1.6.9.1 Búsqueda de Elementos

La búsqueda se basa en una tabla con filtros en las cabeceras. Dependiendo del elemento a buscar se pueden definir filtros numéricos, de texto, fechas, etc. Al lado de cada registro se muestra un botón con las diferentes opciones que se pueden realizar con éste, como Ver, Modificar, Borrar, etc.

Las tablas de registros quedan configuradas para que sean todas iguales, con el mismo número de filas, plantilla para la paginación, ordenación de elementos, etc.

De manera análoga, cada columna de resultados queda configurada para mostrar el dato correspondiente del elemento sobre el que itera la tabla. Esto se consigue mediante reflexión gracias a la nomenclatura “getter” de los bean.

Merecen especial atención los botones de acción sobre los elementos. Estos redirigirán al método “prepareAction” del “Bean” correspondiente y establecerán 2 propiedades para poder realizar el seguimiento de las acciones. El primero será la acción que estamos realizando, que será almacenada en una pila (Stack) que comprobaremos a la hora de habilitar campos o dejarlos en modo solo de lectura, así como para dibujar el botón de acción correspondiente. La otra acción que llevará a cabo será la de enviar al servidor el registro concreto que queremos manipular.

Clientes del Sistema										
Tipo	Activo	Código Cliente	Nombre	Dirección Servicio	CIF	Teléfono Principal	Periodo	Importe	Fecha	Acciones
Cliente	No	243	ANETOS Promociones Inmobiliarias, S.L.	CL. Camarillas, 11	B-6100007		May	0.0	2014-05-24 02:38:25.0	
Cliente	SI	1000	JPC ALQUILERES, S.L.P.	Avd. de Trespuentes, 46	B-6100008	91811138	Mar	82.0	2014-05-24 02:38:27.0	
Cliente	SI	101	Viajes Punto Europa	CL. Errázuriz, nº 1 - 1 A	B-6100009	600673519	Sep	36.0	2014-05-24 02:38:27.0	

Ilustración 8-15. Ejemplo de securización de Botones

8.2.1.6.9.2 Manipulación de elementos

Las páginas de manipulación de elementos siguen una mecánica similar a las búsquedas, haciendo uso del modificador “disabled” de las etiquetas para deshabilitar el elemento en caso de que no queramos manipularlo (normalmente cuando no estamos editando o creando un elemento nuevo).

Los botones de acción se dibujarán dependiendo del estado en el que nos encontremos, y cada uno redirigirá a las distintas acciones, como save, update, delete, etc.

8.2.1.6.9.3 conf

Contiene los ficheros de configuración básicos, como log4j o spring. Cabe destacar:

8.2.1.6.9.4 securityContext.xml

Define filtros de seguridad, interceptores y beans de aplicación que utiliza Spring de fondo.

8.2.1.6.9.5 mvc-dispatcher-servlet.xml

Define los Servicios REST que quedan abiertos en la aplicación, así como el conversor que utilizará para JSON (Jackson).

8.2.1.6.9.6 faces-config.xml

Define las reglas de navegación de la aplicación, para poder tener “accesos directos” o “shortcuts” y no escribir rutas continuamente sino palabras clave.

8.2.1.6.9.7 context.xml

Define la principal configuración de Spring, como pueden ser los paquetes a escanear, la ubicación del fichero de configuración de Hibernate, y datos injectados a diferentes Beans de la aplicación, como el gestor de eventos “cron” o el actualizador de IP.

8.2.2 java

Contiene los ficheros multi-idioma, la configuración de usuario y las clases Java que controlan la vista.

Branch: master ▾ [SIRME](#) / [SIRME-web](#) / [src](#) / [main](#) / [java](#) /

- ganzux new date in data base and sort rest services
- ..
- com/alcedomoreno/sirme/web
- conf
- resources

Ilustración 8-16. Directories de código en SIRME-WEB

8.2.2.1 resources

Contiene todos los pares “clave-valor” para las cadenas de texto de la aplicación, así como la configuración de las vistas (número de filas, tipo de paginación, plantilla de paginadores, etc.).

```
1 # Número de filas en las vistas principales
2 generalRows      = 10
3 # Número de filas en las pantallas de edición
4 generalEditRows = 8
5
6 # Plantilla del Paginador para las pantallas principales
7 generalPaginator          = {CurrentPageReport} {FirstPageLink} {PreviousPageLink} {NextPageLink} {LastPageLink}
8 # Parte de la izquierda de las tablas de contenidos para las pantallas principales
9 generalPageReportTemplate = Registros del {startRecord} al {endRecord} de {totalRecords}
10 editPageReportTemplate   = {startRecord} al {endRecord}/{totalRecords} (Pg {currentPage} de {totalPages})
11 # Paginador siempre visible, aunque sólo tengamos una página para las pantallas principales
12 generalPaginatorAlwaysVisible = false;
13 # Tamaño por defecto de la columna de acciones sobre un elemento
14 widthOptionsTable = 300
15
16 # Orden en las tablas
17 generalSortOrder      = descending
18
19 # Tiempo de vida del Mensaje, tanto de error como de información
20 lifeGrowl           = 7500
21
22 # Formato de las fechas para mostrar en la Aplicación
23 dateFormat           = dd/MM/yyyy
24 dateFormatseconds    = dd/MM/yyyy HH:mm:ss
```

Ilustración 8-17. Extracto de configuración de la aplicación

8.2.2.2 conf

Contiene la configuración de acceso a base de datos y del sistema.

8.2.2.2.1 hibernate.cfg.xml

```
<!--      <property name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property> -->
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<!--      <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<!--      <property name="connection.driver_class">net.bull.javamelody.JdbcDriver</property> -->
<property name="hibernate.connection.driver">com.mysql.jdbc.Driver</property>

<!--
<property name="hibernate.connection.url">jdbc:mysql://</property>
<property name="hibernate.connection.username"></property>
<property name="hibernate.connection.password"></property>
-->

<!--
<property name="hibernate.connection.pool_size">5</property>
<property name="hibernate.connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider</property>
<property name="hibernate.connection.zeroDateTimeBehavior">convertToNull</property>
<property name="hibernate.order_updates">true</property>
<nonempty name="hibernate.cron.acquire_increment">1</nonempty>
```

Ilustración 8-18. Extracto de hibernate.cfg.xml

Los 3 aspectos fundamentales a tener en cuenta son “hibernate.connection.url”, “hibernate.connection.username” y “hibernate.connection.password”.

8.2.2.2.2 config.properties

Contiene la configuración para la aplicación “cron” ejecutada en segundo plano, los datos de acceso a DynDNS, el directorio donde se almacenarán las imágenes, etc.

web

Contiene las clases Java de la capa de presentación

ganzux new date in data base and sort rest services	
..	
aut	Update BSUserAuthenticationSuccessHandler.java
dyn	1st mavenized version
excel	new date in data base and sort rest services
jasper	- Updated iText version, now compatible with Jasper actual version
schedulers	Update IPCronService.java
util	Work queries more fast
AddressBean.java	Update AddressBean.java
ApplicationBean.java	Update ApplicationBean.java
CustomersBean.java	Update CustomersBean.java
IManagedBean.java	1st mavenized version
ManagedBean.java	1st mavenized version
ManagerBean.java	Update ManagerBean.java
MyBean.java	1st mavenized version
SettingsBean.java	new date in data base and sort rest services
StackNavigationBean.java	1st mavenized version
TeamsBean.java	Update TeamsBean.java
UsersBean.java	Update UsersBean.java
WorksBean.java	Search for Works By year (default, server year)

Ilustración 8-19. Paquetes de SIRME-WEB

8.2.2.2.3 ManagedBeans

Todos los Bean de la aplicación, que son los encargados del manejo de las vistas y de la conexión con la capa “Control” extienden de ManagedBean y son anotados mediante Spring para definir un ámbito de sesión (con esto conseguimos que cada usuario tenga el suyo propio) y el nombre con el que lo referenciaremos en las páginas xhtml.

ManagedBean implementará la interfaz IManagedBean, que obligará a las clases a definir el comportamiento de los métodos doInit, prepareAction, save, update, delete, back y cancel.

- doInit: encargado de preparar la búsqueda inicial y cargar todos los objetos que se utilizarán en la pantalla.
- prepareAction: preparará el elemento seleccionado previamente cargando sus objetos asociados y manejando la acción que se haya elegido.
- Save: guardará el elemento mediante la capa de persistencia.
- Delete: eliminará el elemento mediante la capa de persistencia.
- Update: actualizará el elemento mediante la capa de persistencia según su identificador.
- Back: éste método eliminará de la pila de acciones el último elemento y cargará la página anterior.
- Cancel: cancelará la acción que estábamos realizando.

A continuación se detalla un ejemplo del manejo de la capa de presentación de direcciones mediante la clase AddressBean.

En la zona de atributos tenemos declarados los pertenecientes a los log de la aplicación (marcados con el número 1), las inyecciones de Servicios mediante la anotación de Spring “@Service” para conectar con la capa de “Control” (número 2) y los atributos básicos usados en las vistas, con una colección de las direcciones, las direcciones filtradas y la dirección que se ha seleccionado, para poder ser utilizada desde los diferentes métodos de la clase.

8.2.2.2.3.1 doInit

El método doInit es el que se ejecuta en primer lugar antes de cargar la página. Éste método, mediante los objetos “Service” cargará los elementos que se utilizan en esta pantalla e inicializará la pila de llamadas, dejándola vacía. Al final del mismo, redirigirá a la pantalla de búsqueda correspondiente.

```
@Override  
public String doInit(){  
    MyLogger.debug(log, CLASS_NAME, "doInit", "IN");  
  
    selectedCustomer = null;  
    stackBean.initStack();  
  
    Collection<Customer> customers = customerService.getAllCustomers();  
    addresses = new ArrayList<Address>();  
    for ( Customer c:customers )  
        addresses.addAll( c.getAddress() );  
    filteredAddresses = addresses;  
  
    MyLogger.debug(log, CLASS_NAME, "doInit", "OUT");  
    return BeanNameUtil.PAGE_PRINCIPAL_ADDRESS;  
}
```

Ilustración 8-20. Método doInit

8.2.2.2.3.2 prepareAction

prepareAction preparará el objeto a utilizar, creando uno nuevo en caso de un alta o recuperando los datos necesarios para mostrarlo por pantalla en el resto de casos.

Aquí podemos observar la diferencia entre las pantallas de Vista avanzada y de edición, aun siendo exactamente la misma página.

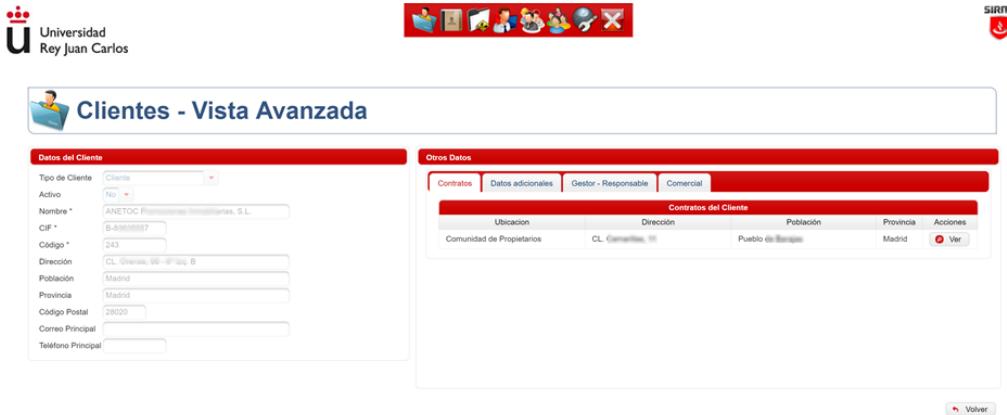


Ilustración 8-21. Vista Avanzada de Clientes



Ilustración 8-22. Actualización de Clientes

8.2.2.2.3.3 save

Save será el encargado de almacenar la dirección que estamos creando. También capturará las excepciones que sean lanzadas desde el resto de capas (estas habrá pasado por el LOG, luego no es necesario capturar el error aquí). Mediante un Bean de aplicación con scope de sesión “ApplicationBean” se generará un mensaje de error o de aceptación al usuario.

8.2.2.2.3.4 Update y delete

Los métodos de actualización y de borrado serán gestionados de igual manera que el de guardado.

8.2.2.2.3.5 Back y cancel

El método para volver sacará el último elemento de la pila de páginas para poder volver a la misma y reiniciará el filtro de direcciones con todas las que se encuentren en el sistema.

Por supuesto, al tratarse de un Bean, todos los atributos a los que queramos acceder estarán declarados como privados pero tendrán sus getter y sus setter.

8.2.2.2.4 util

Este paquete contiene algunas clases con utilidades varias para constantes, el manejo de imágenes, compresión en formato ZIP, fechas y horas, etc.

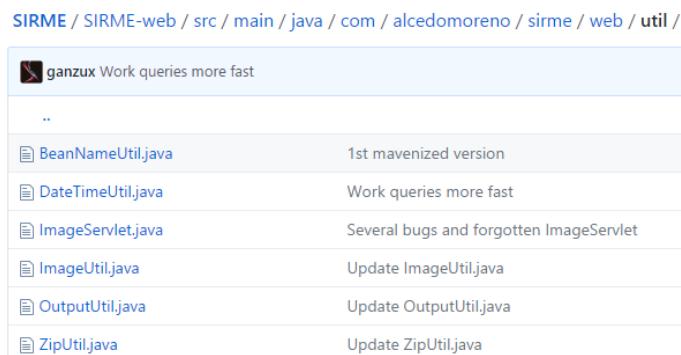


Ilustración 8-23. Paquete util

8.2.2.2.5 schedulers

Contiene la clase que es ejecutada en segundo plano con un hilo independiente mediante un comando “cron”. En este caso recogerá los parámetros de configuración de DynDNS para tratar de actualizar el servicio con la IP del usuario.

8.2.2.2.6 jasper

Contiene la clase encargada de generar el reporte según la configuración seleccionada por el usuario, formando los datos acorde a las agrupaciones de los informes.

8.2.2.2.7 excel

Este paquete contiene las clases encargadas de exportar los listados en las búsquedas a formato Excel.

Tipo	Activo	Código	Nombre	Dirección	CIF	Teléfono Principal	Período
Cliente	No	243	Juan Pérez	Calle Alcalá, 123	K-1234567890	9112345678	May
Cliente	S	1000	Ángela Martínez	Calle de la Constitución, 100	K-1234567890	9112345678	31 Mar
Cliente	S	101	David Rodríguez	Calle de la Constitución, 101	K-1234567890	9112345678	31 Sep
Cliente	S	1010	Isabel Gómez	Calle de la Constitución, 1010	K-1234567890	9112345678	31 Jul
Cliente	No	1013	Pedro Martínez	Calle de la Constitución, 1013	K-1234567890	9112345678	31 Abo
Cliente	S	1017	María Pérez	Calle de la Constitución, 1017	K-1234567890	9112345678	04 May
Cliente	S	102	Antonio Gómez	Calle de la Constitución, 102	K-1234567890	9112345678	08 Nov

Ilustración 8-24. Ejemplo de exportación de listado a formato PDF

8.2.2.2.8 dyn

Este paquete contiene la clase que realiza la conexión con DynDNS. Cabe destacar que está construída siguiendo un patrón Singleton, dado que no se necesita una instancia de la misma por cada hilo de ejecución sino que pueden compartir siempre la misma instancia de la clase.

8.2.2.2.9 aut

Éste paquete contiene las clases encargadas de poner en la sesión del usuario los Roles a los que pertenece junto con sus permisos en caso de que la autenticación haya sido exitosa y de gestionar el fallo de autenticación.

Para ello, la clase BSAuthenticationProvider extenderá de la clase de Spring DaoAuthenticationProvider, declarando los métodos public Authentication authenticate(Authentication auth) y public boolean supports(Class<? extends Object> authentication), donde podremos implementar nuestro propio negocio.

Si la autenticación de usuario ha sido incorrecta, se ejecutará el método “onAuthenticationFailure” de la clase BSUserAuthenticationErrorHandler, que implementa AuthenticationFailureHandler.

En caso contrario, cuando las credenciales son correctas, nos iremos al método “onAuthenticationSuccess” de la clase BSUserAuthenticationSuccessHandler que implementa AuthenticationSuccessHandler.

8.3. CONTROLLER

```
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-client</artifactId>
    <version>1.8</version>
</dependency>                                <dependency>
                                                <groupId>org.codehaus.jackson</groupId>
                                                <artifactId>jackson-mapper-asl</artifactId>
                                                <version>1.9.7</version>
</dependency>
```

Ilustración 8-25. Dependencias maven de SIRME-BUSINESS

Dentro del fichero pom de maven encontramos dependencias a Jackson y a Jersey, por ejemplo.

Este Proyecto se trata de puro JAVA, teniendo una estructura de 7 paquetes bien diferenciados, con data, dto, rest, services, transform, util y validator.

SIRME / SIRME-business / src / main / java / com / alcedomoreno / sirme / business /	
	ganzux new date in data base and sort rest services
..	
	data new date in data base and sort rest services
	dto new date in data base and sort rest services
	rest new date in data base and sort rest services
	services new date in data base and sort rest services
	transform new date in data base and sort rest services
	util Several bugs and forgotten ImageServlet
	validator Validator codification

Ilustración 8-26. Paquetes de SIRME-BUSINESS

8.3.1 data

Se trata de un paquete con todos los Bean que representan objetos planos de la aplicación usados en la capa de presentación.

Todos ellos heredan de BusinessObject, que obliga a implementar el método getTransformator por el que el objeto de negocio será transformado a un objeto Data, encargado de manejar la base de datos.

Un claro ejemplo lo tenemos en Customer, donde quedan almacenados los datos referentes al cliente, con su nombre, cif, dirección, etc.

Por supuesto, todos estos atributos pueden hacer referencia a otros Bean, como por ejemplo un Customer tendrá una colección de direcciones, de contactos y de trabajos, y estará asociado a una tipología. No hace falta decir que todos los atributos deben tener sus getter y sus setter.

Y el método getTransformator, que devuelve un objeto CustomerTransform.

A continuación se detalla el diagrama de clases para los objetos Data de la aplicación. Por una parte encontramos el diagrama de clases para los usuarios. Estos se basarán en Spring Security con usuarios, roles, privilegios/permisos. Básicamente por cada usuario tendremos un Rol asociado al cual le corresponderán diferentes permisos para una aplicación.

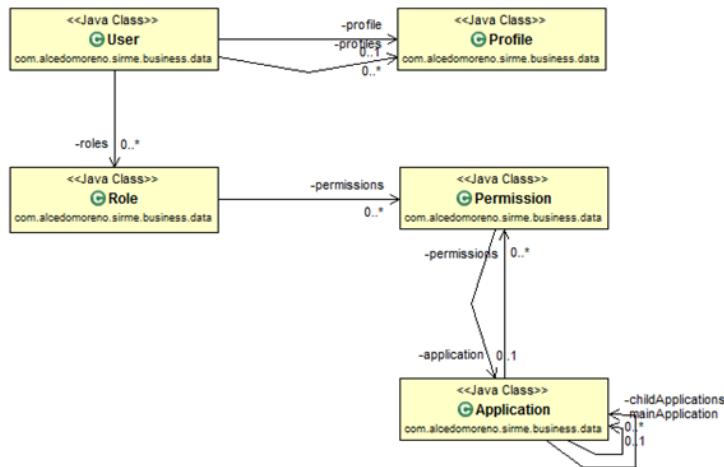


Ilustración 8-27. Diagrama de Clases de Usuarios

El grueso del negocio se detalla a continuación.

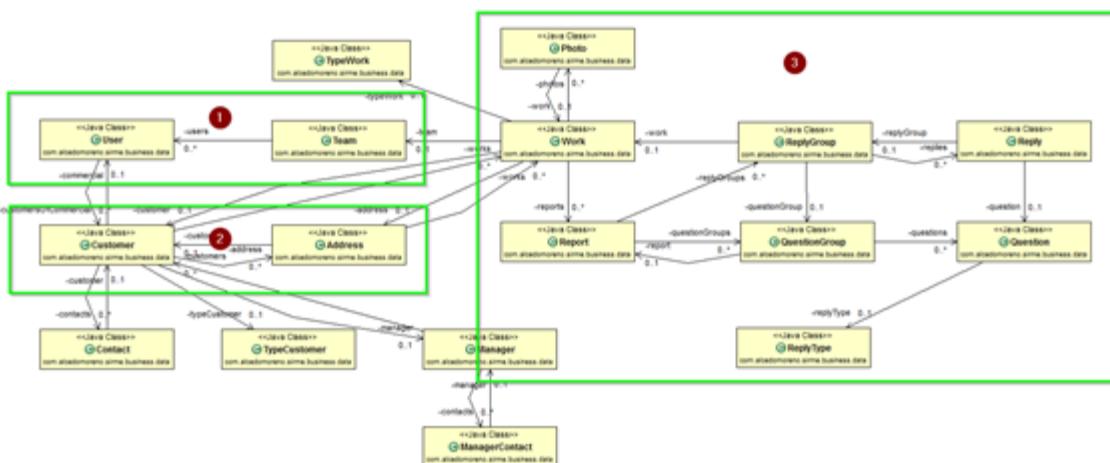


Ilustración 8-28. Diagrama de Clases de objetos de negocio con marcas

En la zona marcada con el número 1 vemos como un grupo de usuarios formará un equipo, que será el encargado de realizar un trabajo (punto de unión entre los bloques 1 y 3).

En la zona 2 tenemos al cliente (Customer), que tendrá diferentes direcciones (será en una de ellas donde el trabajo será realizado), una tipología y formas de contacto.

Este cliente también tendrá un Manager, que es el responsable del mismo.

La zona 3 contiene el grueso del denominado trabajo.

- Un trabajo es dado para un cliente en una dirección.
- El trabajo podrá contener fotos.
- El trabajo generará un informe (Report) con grupos de preguntas y respuestas, para ir verificando o almacenando cada uno de los elementos del sistema y su estado actual. No es la intención del sistema el inventariar los diferentes elementos sino el responder al estado de los mismos según sea necesario.

A continuación el diagrama completo:

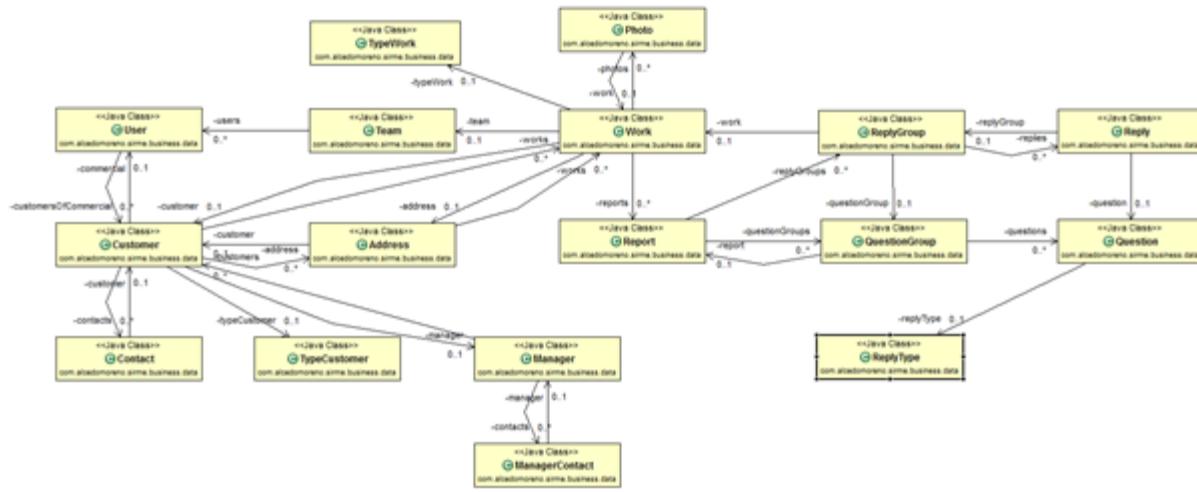


Ilustración 8-29. Diagrama de Clases de Objetos de negocio

8.3.2 dto

El paquete contiene, al igual que el paquete data, los Bean que utilizaremos para ser transformados en un archivo JSON. Su principal característica es que están anotados con @JSon, que permite definir qué atributos serán los que queramos incluir o donde parar (hay que tener especial cuidado con los bucles infinitos, como un cliente con direcciones y una dirección perteneciente a un cliente).

8.3.3 transform

Se trata de un paquete con los transformadores de objetos de negocio a objetos de la capa de persistencia (se verán en el core). Todos ellos implementan la interfaz Transformator y extienden de DefaultTransformator.

Mediante la interfaz les obligamos a realizar las transformaciones de objetos data a objetos business tanto para objetos específicos como para colecciones de los mismos.

Al extender de DefaultTransformer estamos añadiendo la funcionalidad de replicar el comportamiento para el objeto “single” a las colecciones, por lo que no es necesario repetirlo en cada una de las clases que transforman.

Un ejemplo de transformador, del que sólo tenemos que definir aquellas transformaciones para objetos simples será la de direcciones, donde podemos anidar las llamadas a distintos transformadores.

8.3.4 util

Contiene algunas clases que pueden sernos útiles para el manejo de cadenas, constantes o excepciones.

SIRME / SIRME-business / src / main / java / com / alcedomoreno / sirme / business / util	
 ganzux	Several bugs and forgotten ImageServlet
..	
 FileUtil.java	Several bugs and forgotten ImageServlet
 ServiceConstants.java	1st mavenized version
 StringUtil.java	1st mavenized version
 SuperUser.java	1st mavenized version
 TransactionException.java	1st mavenized version
 ValidationException.java	1st mavenized version

Ilustración 8-30. Clases del paquete util

8.3.5 validator

Contiene las clases encargadas de validar los tipos de datos; dado que todas las respuestas son almacenadas en formato de texto, al configurar las preguntas se define qué tipo de respuesta queremos. La clase Validator es la encargada de comprobar si la respuesta concuerda o no con lo configurado.

Por ejemplo, para validar que tenemos un entero mayor que cero o una fecha, las siguientes validaciones son las más adecuadas:

8.3.6 rest

Paquete con los controladores REST de la aplicación. Harán uso de los servicios, siendo estas llamadas REST un mero usuario más, por lo que no deberán nunca tener su propio negocio más que el mapeo de datos, log e información al usuario.

Las clases serán anotadas con @Controller

Inyectarán los servicios que vayamos a utilizar en los diferentes métodos

Y por último declararán aquellos métodos necesarios, diciendo el verbo HTTP a utilizar

En este ejemplo, dado que un trabajo está empezando a ser almacenado y dado que necesitamos transaccionalidad con los servicios REST, todo será almacenado hasta que la última llamada REST sea recibida, que será entonces cuando sea persistido todo y se devolverá un código en el último servicio REST. Como se puede comprobar aquí se hace uso de los servicios de la aplicación, que son los que contienen el negocio, para formar los diferentes objetos, pero ninguna clase de negocio es ejecutada en esta capa.

8.3.7 services

Son los que manejan el negocio y la transaccionalidad de las operaciones. Todas las clases tendrán su interfaz, que será el contrato que deberán cumplir a la hora de implementar los métodos y nuestra implementación, que por nomenclatura llevará el mismo nombre acabado en Impl.

Un ejemplo de un método para recuperar datos desde la capa de persistencia puede ser el método getAll del servicio de Preguntas, que devuelve todas las preguntas almacenadas en base de datos.

Simplemente hará uso del objeto injectado DAO y llamará al transformador, dado que los DAO devuelven objetos de la capa de persistencia pero los servicios sólo entienden de objetos de negocio, por lo que haremos uso de los transformadores.

A la hora de realizar una transacción, haremos uso de la excepción TransactionException, haciendo un rollBack en caso de que la misma sea lanzada. Como antes, estamos conectando directamente a la capa de persistencia, manteniendo el patrón MVC en mente.

9. Despliegue y Pruebas

Programar sin una arquitectura o diseño en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado ni hacia dónde vas.

9.1. Despliegue

9.1.1 Servidor de Aplicaciones

En caso de que la empresa no tenga contratada una IP estática por parte de su proveedor de servicios de Internet, se ofrecen alternativas (que pueden ser de pago, como la actualización dinámica mediante Dynamic DNS) al cliente. No obstante, para poder acceder desde el exterior de la empresa a la aplicación o a los servicios REST, se necesitará conocer su ubicación DNS.

Se tiene la necesidad técnica de un servidor de aplicaciones J2EE, con una versión de Java a partir de la 1.7. Dada la retro compatibilidad de Java, se podrá hacer uso a partir de esta versión para adelante.

El proyecto cumple los estándares J2EE según la arquitectura maven, por lo que bastará con ejecutar el fichero “pom” del directorio raíz para descargar las dependencias y proceder a la construcción del mismo en un fichero WAR, desplegado en un servidor de aplicaciones (Tomcat 8 sirve perfectamente).

Para su descarga e instalación el código se encuentra en <https://github.com/ganzux/SIRME>.

9.1.2 Base de Datos

La base de datos escogida para realizar las pruebas es MySQL en su versión 5.7.

El script DDL de creación de las tablas así como el SQL para las tablas de configuración se encuentran en la raíz del proyecto bajo el directorio “scripts”.

9.1.3 Clientes

Los clientes que accedan a la aplicación necesitarán disponer de un navegador web que soporte HTML5 (cualquier navegador web del año 2017 soporta este lenguaje de marcas), conocer la URL de la aplicación y un usuario y contraseña que será facilitada por el administrador de la aplicación.

9.1.4 Instalación en Linux

Para la instalación de ejemplo me he basado en una distribución de Linux Ubuntu 16.04. Los pasos para la instalación en un entorno Linux serán los siguientes:

9.1.4.1 Conexión al equipo

Si no tenemos acceso directo al terminal, bastara con abrirlo; si no, tendremos que conectarnos a la máquina de manera remota mediante ssh. Lo primero que haremos será actualizar la lista de paquetes y sus versiones de apt-get (mediante update) y los actualizaremos (mediante upgrade).

```
ssh root@178.62.106.94
sudo apt-get update
sudo apt-get upgrade
```

9.1.4.2 Instalacion de JDK

Debido a que compilaremos el Proyecto, necesitamos instalar la JDK y no solo el JRE. En este caso utilizaremos “Open JDK”. La instalación del mismo se realiza mediante el siguiente comando:

```
sudo apt-get install openjdk-8-jdk
```

Una vez finalizada la instalación comprobaremos que está correctamente instalado mediante el comando

```
Java -version
```

Que deberá indicarnos la versión instalada de la siguiente manera:

```
[root@alvaro:~# java -version
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-0ubuntu1.16.04.2-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
```

Ilustración 9-1. Consola Linux - Java

9.1.4.3 Instalacion de maven

Maven no esta instalado por defecto, y nos servirá para construir el proyecto una vez haya sido descargado con git (que si está instalado por defecto en la distribución de Linux utilizada).

```
sudo apt-get install maven
```

Y podemos comprobar la version que hemos instalado

```
mvn --version
```

```
[root@alvaro:~# mvn --version
Apache Maven 3.3.9
Maven home: /usr/share/maven
Java version: 1.8.0_131, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.4.0-79-generic", arch: "amd64", family: "unix"
root@alvaro:~# ]
```

Ilustración 9-2. Consola Linux - Maven

9.1.4.4 Instalacion de MySQL

Procedemos a instalar la versión 5.7 del servidor.

```
sudo apt-get install mysql-server-5.7
```

Al final de la instalacion nos pedira la contraseña del usuario “root”, que he dejado como “root” para este ejemplo, dado que por defecto la distribución de SIRME utilizara ese usuario con esa contraseña.

Comprobamos que lo hemos instalado correctamente y que el servicio esta arrancado:

```
mysql --version
sudo service mysql status
```

```
[root@alvaro:~# mysql --version
mysql Ver 14.14 Distrib 5.7.18, for Linux (x86_64) using EditLine wrapper
[root@alvaro:~# sudo service mysql status
● mysql.service - MySQL Community Server
   Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2017-06-20 11:10:24 UTC; 8min ago
     Main PID: 21620 (mysqld)
        CGroup: /system.slice/mysql.service
                  └─21620 /usr/sbin/mysqld

Jun 20 11:10:23 alvaro systemd[1]: Starting MySQL Community Server...
Jun 20 11:10:24 alvaro systemd[1]: Started MySQL Community Server.
```

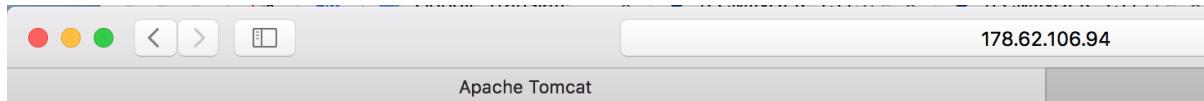
Ilustración 9-3. Consola Linux - MySQL

9.1.4.5 Instalacion de Tomcat

Instalaremos lo básico del servidor de aplicaciones, sin consola ni ejemplos.

```
sudo apt-get install tomcat8
```

Comprobaremos que el servicio funciona e iremos a un navegador para el "Hello World".



It works !

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

This is the default Tomcat home page. It can be found on the local filesystem at: /var/lib/tomcat8/webapps/ROOT/index.html

Tomcat8 veterans might be pleased to learn that this system instance of Tomcat is installed with CATALINA_HOME instead of CATALINA_HOME. This is done by setting the CATALINA_HOME environment variable in the /etc/default/tomcat8 file according to the rules from /usr/share/doc/tomcat8-common/RUNNING.txt.gz.

You might consider installing the following packages, if you haven't already done so:

tomcat8-docs: This package installs a web application that allows to browse the Tomcat 8 documentation locally.

tomcat8-examples: This package installs a web application that allows to access the Tomcat 8 Servlet and JSP examples.

tomcat8-admin: This package installs two web applications that can help managing this Tomcat instance. Once installed, they can be accessed via the manager-gui application.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager-gui". The host-manager application is controlled by the manager-gui application. It is located at /etc/tomcat8/tomcat-users.xml.

Ilustración 9-4. Servidor arrancado

9.1.4.6 Instalacion de fuente Arial

Debido al uso de Jasper Reports, necesitaremos instalar la fuente Arial en el sistema; esto se realiza mediante los siguientes comandos:

```
sudo apt-get install ttf-mscorefonts-installer  
sudo fc-cache
```

Y lo comprobamos:

```
[root@alvaro:~# fc-match Arial  
Arial.ttf: "Arial" "Regular"
```

Ilustración 9-5. Consola Linux – Instalación de fuente Arial

9.1.4.7 Construcción del proyecto

Descargaremos el proyecto mediante el siguiente comando git:

```
git clone https://github.com/ganzux/SIRME.git
```

Una vez descargado, lo construiremos. Este proceso se encargará de descargar todas las librerías correspondientes, pasar los tests y empaquetar el fichero WAR que desplegaremos en TomCat.

```

root@alvaro:/# git clone https://github.com/ganzux/SIRME.git
Cloning into 'SIRME'...
remote: Counting objects: 2545, done.
remote: Compressing objects: 100% (58/58), done.
remote: Total 2545 (delta 14), reused 48 (delta 6), pack-reused 2479
Receiving objects: 100% (2545/2545), 43.80 MiB | 21.75 MiB/s, done.
Resolving deltas: 100% (967/967), done.
Checking connectivity... done.
root@alvaro:/# cd /SIRME/
root@alvaro:/SIRME# mvn package

```

Ilustración 9-6. Consola Linux - GIT

Al finalizar el proceso tendremos un fichero en /SIRME/SIRME-web/target/SIRME.war

```

[INFO] Packaging webapp
[INFO] Assembling webapp [SIRME-web] in [/SIRME/SIRME-web/target/SIRME]
[INFO] Processing war project
[INFO] Copying webapp resources [/SIRME/SIRME-web/src/main/webapp]
[INFO] Webapp assembled in [924 msecs]
[INFO] Building war: /SIRME/SIRME-web/target/SIRME.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] SIRME_ ..... SUCCESS [ 0.007 s]
[INFO] SIRME-core ..... SUCCESS [03:44 min]
[INFO] SIRME-business ..... SUCCESS [ 4.359 s]
[INFO] SIRME ..... SUCCESS [ 18.664 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 04:08 min
[INFO] Finished at: 2017-06-20T11:36:50+00:00
[INFO] Final Memory: 36M/197M
[INFO] -----

```

Ilustración 9-7. Consola Linux – Construcción de SIRME

9.1.4.8 Creacion de la base de datos

Nos logaremos en la base de datos, crearemos el esquema “sirme” y cargaremos el script DDL con los datos básicos de configuración.

```

[root@alvaro:/# mysql -u root -p
[Enter password]:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.18-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

[mysql]> create database sirme;
Query OK, 1 row affected (0.01 sec)

[mysql]> exit
Bye
root@alvaro:/#

```

Ilustración 9-8. Consola Linux – Carga de esquema DDL

```
mysql -u root -p sirme</SIRME/scripts/sql.sql
```

9.1.4.9 Arranque del servidor

Solo nos falta copiar el fichero WAR al servidor para que este se despliegue automáticamente.

```
cp /SIRME/SIRME-web/target/SIRME.war /var/lib/tomcat8/webapps/
```

A partir de este momento podremos acceder a los ficheros de LOG y, por supuesto, a nuestra aplicación.



Ilustración 9-9. Pantalla de Login desplegada



Ilustración 9-10. Pantalla de Inicio desplegada

9.2. Pruebas

Las pruebas de la aplicación están automatizadas mediante la construcción del proyecto. Para ello se ha hecho uso de JUnit y del servidor de integración continua TravisCI. A la hora de construir el software todos los tests son ejecutados, cosa que no asegura que el software no vaya a fallar, pero nos indica qué pruebas hemos tenido en cuenta, por lo que sabemos dónde no vamos a fallar.

Las pruebas que se han realizado han sido sobre todo de la capa de persistencia y de la capa de negocio. Existen soluciones para poder automatizar las pruebas en la capa de presentación (como por ejemplo Selenium, que provee una herramienta “record-play” muy útil y sencilla para poder automatizar la navegación por las interfaces web sin necesidad de conocer el código que se ejecuta por debajo), pero han quedado fuera del alcance de este proyecto debido a la complejidad que estaba

alcanzando. He querido centrarme en estas capas porque creo firmemente que la persistencia y el negocio han de funcionar a la perfección dado que no puedo permitir ningún tipo de inconsistencia en la información del cliente, que será de vital importancia para ellos.

The screenshot shows the Travis CI interface for the repository 'ganzux / SIRME'. On the left, there's a sidebar with 'My Repositories' and two listed: 'ganzux/SIRME' (3 builds) and 'ganzux/TestJUnit' (4 builds). The main area is titled 'Current' and shows the 'master' branch with a green checkmark and the text 'some tests with travisCI'. Below it, a detailed view of build #3 is shown, which passed. It includes information about the commit (e6f986b), compare (b4a39ad..e6f986b), branch (master), and authors (Álvaro authored, GitHub committed). The total time for this build was 2 min 24 sec. At the bottom, there are sections for 'Build Jobs' with two more successful builds: #3.1 and #3.2.

Ilustración 9-11. TravisCI

```
-----  
TESTS  
-----  
Running com.alcedomoreno.sirme.core.data.AccessDataTest  
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.046 sec  
Running com.alcedomoreno.sirme.core.data.ConfigDataTest  
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 sec  
Running com.alcedomoreno.sirme.core.data.BussinessDataTest  
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.006 sec  
Running com.alcedomoreno.sirme.core.data.UsersDataTest  
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.007 sec  
Running com.alcedomoreno.sirme.core.dao.WorkBaoTest  
log4j:WARN No appenders could be found for logger (org.springframework.test.context.junit4.SpringJUnit4ClassRunner).  
log4j:WARN Please initialize the log4j system properly.  
Hibernate: select this_.idaddress as idaddress4_0_, this_.address as address4_0_, this_.amount as amount4_0_, this_.idCustomer as idCustomer4_0_, this_.location as location4_0_, this_.monthMask as monthMask4_0_, this_.other as other4_0_, this_.pobl as pobl4_0_, this_.postalCode as postalCode4_0_, this_.prov as prov4_0_ from address this_ where this_.idaddress=?  
Hibernate: insert into work (idwork, idAddress, albaran, data, "date", dateCreated, dateReceived, memo, signName, signpath, status, idTeam, typeWork, year) values (null, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)  
Hibernate: call identity()  
Hibernate: insert into replygroup (idReplyGroup, nameReplyGroup, idQuestionGroup, idReport, idWork) values (null, ?, ?, ?, ?)
```

Ilustración 9-12. Ejemplos de tests

Me he ayudado de los plugin de maven “Surefire” y “Cobertura” para generar “reportes” de errores y cobertura del código, con lo que con cada construcción puedo saber qué tests se han ejecutado, si ha fallado algo, el porcentaje de cobertura, etc.

Por ejemplo, en el subproyecto “core” tenemos el paquete “test”, donde están almacenadas todas las pruebas unitarias del mismo. Este está dividido en “resources” (que contiene los recursos, estos son los ficheros “persistence.properties” con la configuración de la base de datos en memoria y “test.sql”, con los datos que utilizaremos para hacer las pruebas) y “java” (con el código fuente de los tests). Dentro del directorio java se encuentran las clases de tests, siguiendo la misma estructura que se ha utilizado a la hora de la codificación, por lo que encontraremos una clase de Test por cada clase de la aplicación.

Para la capa de persistencia se ha hecho uso de HSQLDB (Hyper SQL Database), que es una base de datos en memoria que nos ayuda a probar la capa de persistencia en vez de tener que conectar con una base de datos de test o “mockear” los resultados esperados.

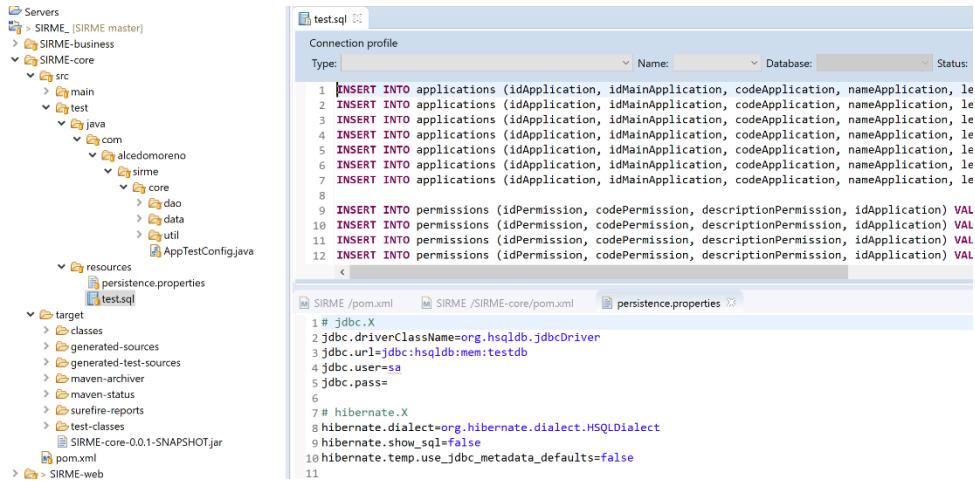


Ilustración 9-13. Configuración y estructura de los tests

Si bien soy de la opinión de que el porcentaje de cobertura de código no es indicativo de que el código no vaya a fallar, también soy consciente de que las pruebas resultan ser el rigor ingenieril en el desarrollo de software, por lo que todo lo que no haya sido debidamente probado puede decirse que no es formalmente valido. Cobertura es un plug-in que me ha ayudado a mostrar gráficamente el porcentaje de código que ha sido recorrido en la fase de test. Al ejecutar el comando mvn –site obtenemos un informe detallado (en diferentes versiones) a modo de “Java Doc” con los paquetes del proyecto y las clases probadas.

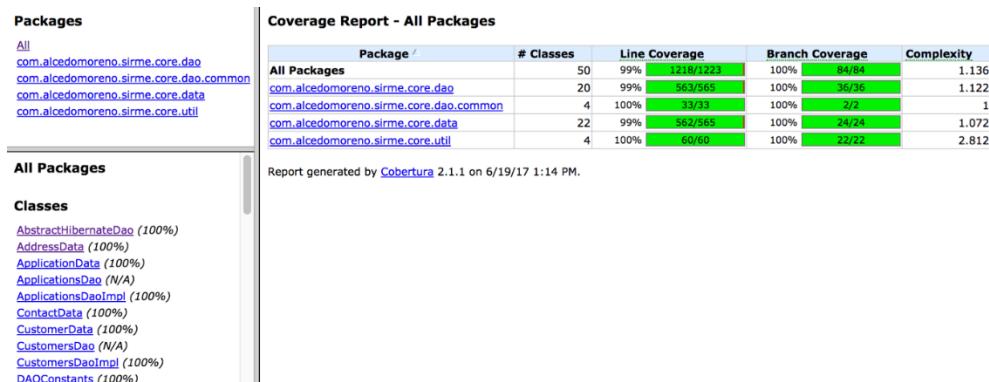


Ilustración 9-14. Cobertura

Podemos, de igual manera, navegar por las clases para ver cómo se han ido probando estas, cuantas veces se ha pasado por cada línea, etc.

```

118     @Override
119     public Collection<WorkData> getOpenAdvicesOrWorksFromTeam(int idTeam, Date date, boolean work){
120         MyLogger.info(log , CLASS_NAME, "getAdvicesFromTeam", "INIT", idTeam);
121
122         Calendar dateCalendar = Calendar.getInstance();
123
124         if (date != null){
125             dateCalendar.setTime(date);
126         } else {
127             dateCalendar.setTime(new Date());
128         }
129
130         SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
131         Date startDate = date;
132
133         try {
134             startDate = format.parse(dateCalendar.get(Calendar.YEAR) + " - "
135                                     + dateCalendar.get(Calendar.MONTH) + " - "
136                                     + dateCalendar.get(Calendar.DAY_OF_MONTH));
137
138         } catch (ParseException e) {
139             e.printStackTrace();
140         }
141
142         return getAdvicesOrWorksFromTeam(idTeam, startDate, work);
143     }

```

Ilustración 9-15. Ejemplo de Cobertura en Clase

Aunque, como ya he comentado, el porcentaje de código que ha sido recorrido por las pruebas unitarias no es indicativo de nada, he obtenido un 100% de cobertura en la capa de persistencia y alrededor de un 20% en la capa de negocio. Sin duda alguna este es uno de los aspectos a mejorar en futuras actualizaciones.

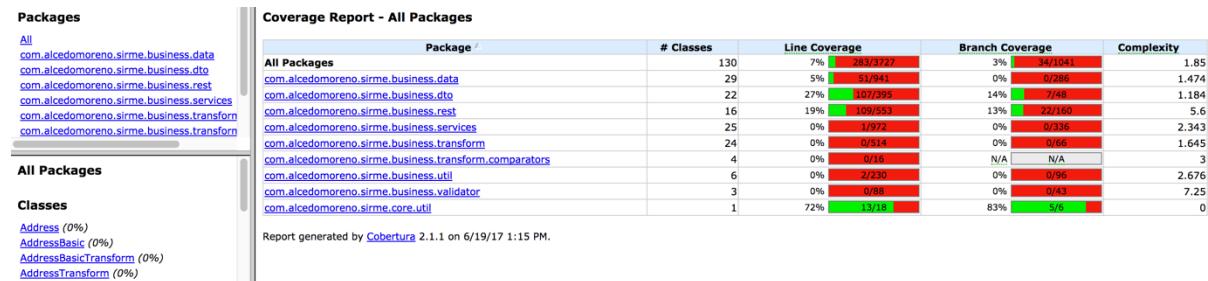


Ilustración 9-16. Cobertura en capa Control

Básicamente, el ciclo de vida para el proyecto es que cada vez que alguien realiza un commit en el repositorio de código, el servidor de integración continua trata de construir el proyecto, ejecutando todos los tests que se hayan realizado. De esta manera, si alguno de los test llegase a fallar, seria un claro indicativo de que hemos cambiado alguna funcionalidad que no debíamos, teniendo que revisar los test o la funcionalidad modificada.



Ilustración 9-17. Ciclo de vida del Software

10. Conclusiones y trabajo futuro

Un programa nunca está completo por debajo del 90% ni por encima del 95%.

El programa funciona y es útil. Si bien requiere algo de esfuerzo configurar todo el motor de preguntas y respuestas para que los técnicos puedan supervisar y realizar sus trabajos, una vez hecho todo es transparente.

Debido a la naturaleza del programa, he podido aprender cómo abstraerme de la idea original y hacer algo que pueda evolucionar hacia diferentes vertientes o negocios, teniendo siempre claras las limitaciones del modelo relacional y la idea de “inspeccionar elementos a nivel de campo”.

La arquitectura de la aplicación está realizada para soportar futuros evolutivos del sistema. A continuación se proponen actualizaciones que pueden ser llevadas a cabo por el sistema.

- Actualizar a la última versión de Spring e incluir Spring Boot.
- Actualizar a gradle.
- Mejorar la cobertura del código en la capa de negocio e introducir pruebas de la capa de presentación.
- Aplicación móvil que utilice los servicios REST que ofrece la aplicación y pueda establecer el flujo de trabajo.
- Gestión de los equipos mediante códigos inteligentes: los técnicos serán capaces de localizar los equipos propiedad de la empresa mediante etiquetas generadas por la aplicación de escritorio. Esto agiliza en gran medida el trabajo de campo que realizan los técnicos.
- Impresión de Partes de Trabajo a pie de campo: con un sistema de impresión compatible con Android sería posible efectuar una impresión de los trabajos realizados o un resumen de la misma para facilitársela al cliente.
- Estadísticas: se puede crear un módulo de estadísticas que permita a un usuario administrador tener gráficos sobre eficiencia, tiempos de respuesta, carga de trabajo, etc.
- Gestión avanzada de Elementos: pudiendo ser extrapolable a más elementos, se podría llevar un control sobre las fechas de retimbrado o de llenado, por ejemplo, de los extintores (saber al momento en qué estado se encuentran).
- Avisos automáticos del sistema: se podría preparar un “motor de reglas de negocio” por el cual el sistema pueda generar avisos de manera automática según las reglas que sean definidas. Por ejemplo, podríamos definir una regla por la que el sistema nos avise de los clientes que tengan un sistema de extinción automática mediante Ansul y la última prueba hidrostática haya sido hace 9 años y 6 meses. Se les podría avisar recordándoles que esa prueba debe realizarse cada 10 años.

11. Bibliografía

- <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
- <http://rogerdudler.github.io/git-guide/>
- https://www.tutorialspoint.com/hibernate/hibernate_examples.htm
- <https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/tutorial.html>
- https://www.udacity.com/course/javascript-basics--ud804?gclid=Cj0KCQjw7J3KBRCxARIsAPMXNVJfQC6G9p1teAluBvnkrunVTRMEmcjxfubjIFP1gg4IN75S2wXv48kaAstYEALw_wcB
- <https://www.tutorialspoint.com/spring/>
- <https://www.tutorialspoint.com/jsf/>
- <http://www.mkyong.com/tutorials/jsf-2-0-tutorials/>
- <https://www.primefaces.org/gettingstarted/>
- https://www.tutorialspoint.com/continuous_integration/index.htm
- <https://www.tutorialspoint.com/listtutorials/java/j2ee/1>
- <http://j2eetutorials.50webs.com/>
- <https://www.codecademy.com/en/tracks/coding-with-javascript-for-dummies>
- <https://www.tutorialspoint.com/restful/>
- <https://www.mkyong.com/spring-boot/spring-boot-hello-world-example-mustache/>
- <https://www.mkyong.com/>
- <https://www.digitalocean.com/community/questions/fresh-tomcat-takes-loong-time-to-start-up>
- <https://www.websequencediagrams.com/>
- <https://askubuntu.com/questions/651441/how-to-install-arial-font-in-ubuntu>
- <http://www.stackoverflow.com>
- <http://www.google.com/>