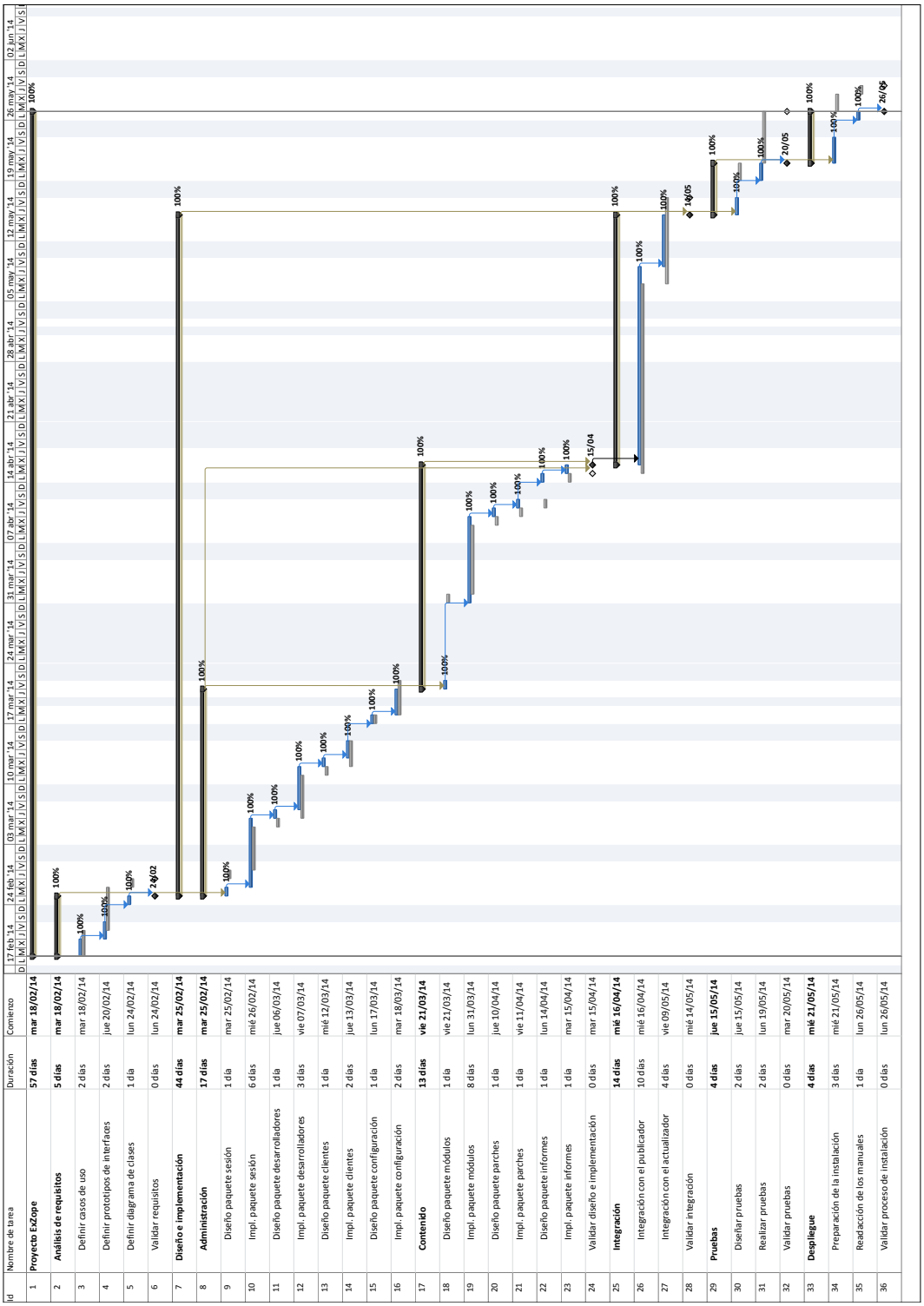

Id	Nombre de tarea	Duración	Comienzo
1	**Proyecto ExZope**	**57 días**	**mar 18/02/14**
2	**Análisis de requisitos**	**5 días**	**mar 18/02/14**
3	Definir casos de uso	2 días	mar 18/02/14
4	Definir prototipos de interfaces	2 días	jue 20/02/14
5	Definir diagrama de clases	1 día	lun 24/02/14
6	Validar requisitos	0 días	lun 24/02/14
7	**Diseño e implementación**	**44 días**	**mar 25/02/14**
8	**Administración**	**17 días**	**mar 25/02/14**
9	Diseño paquete sesión	1 día	mar 25/02/14
10	Impl. paquete sesión	6 días	mié 26/02/14
11	Diseño paquete desarrolladores	1 día	jue 06/03/14
12	Impl. paquete desarrolladores	3 días	vie 07/03/14
13	Diseño paquete clientes	1 día	mié 12/03/14
14	Impl. paquete clientes	2 días	jue 13/03/14
15	Diseño paquete configuración	1 día	lun 17/03/14
16	Impl. paquete configuración	2 días	mar 18/03/14
17	**Contenido**	**13 días**	**vie 21/03/14**
18	Diseño paquete módulos	1 día	vie 21/03/14
19	Impl. paquete módulos	8 días	lun 31/03/14
20	Diseño paquete parches	1 día	jue 10/04/14
21	Impl. paquete parches	1 día	vie 11/04/14
22	Diseño paquete informes	1 día	lun 14/04/14
23	Impl. paquete informes	1 día	mar 15/04/14
24	Validar diseño e implementación	0 días	mar 15/04/14
25	**Integración**	**14 días**	**mié 16/04/14**
26	Integración con el publicador	10 días	mié 16/04/14
27	Integración con el actualizador	4 días	vie 09/05/14
28	Validar integración	0 días	mié 14/05/14
29	**Pruebas**	**4 días**	**jue 15/05/14**
30	Diseñar pruebas	2 días	jue 15/05/14
31	Realizar pruebas	2 días	lun 19/05/14
32	Validar pruebas	0 días	mar 20/05/14
33	**Despliegue**	**4 días**	**mié 21/05/14**
34	Preparación de la instalación	3 días	mié 21/05/14
35	Readacción de los manuales	1 día	lun 26/05/14
36	Validar proceso de instalación	0 días	lun 26/05/14

Figura 3.6: Diagrama de GANTT de seguimiento. Proyecto completo.

A pesar de estos pequeños desvíos, en todo el desarrollo no ha ocurrido ningún contratiempo que haya causado un retraso considerable. En general, la planificación ha sido correcta y el proyecto ha transcurrido según lo previsto.

Debido a que el proyecto ha finalizado dos días antes de lo esperado, se ha ampliado ligeramente su alcance. En las tareas de integración con los scripts publicador y actualizador tan solo se debía incorporar el código para que estos se comunicaran con el nuevo sistema, sin implementar funcionalidad adicional. Sin embargo, en última instancia, se ha completado el proyecto con nueva funcionalidad para los scripts que les permite publicar y desplegar informes de manera automática.

Capítulo 4

Análisis de requisitos

Índice

En este capítulo se presenta la primera fase del desarrollo del proyecto. Como ya se ha comentado en capítulos anteriores, a lo largo de esta etapa se han analizado las necesidades del cliente para delimitar el alcance de la aplicación.

Expresado con otras palabras, durante esta fase se ha determinado qué funcionalidades ha de ofrecer el nuevo sistema y de qué manera debe comportarse. Para lograr este propósito, se han obtenido los tres documentos que se exponen a continuación.

4.1. Documento de casos de uso

En el documento de casos de uso se recopilan una serie de diagramas de casos de uso que describen la funcionalidad de los diferentes paquetes que va a tener la aplicación y se presentan explicaciones detalladas para cada uno de ellos. Además, cuando un caso de uso tiene un comportamiento complejo se describe el flujo de ejecución del mismo.

El objetivo principal de este documento es obtener un contrato entre el cliente y el desarrollador que determine qué debe hacer y qué no es necesario

que haga el sistema que se va a desarrollar. Gracias a este documento, se logra un acuerdo entre ambas partes que delimita el alcance del proyecto.

Para obtener el documento, en primer lugar se han determinado los paquetes de trabajo a partir de las conclusiones obtenidas en la reunión inicial. Como ya se ha explicado con anterioridad, los paquetes de trabajo son conjuntos de funcionalidades relativas a un mismo concepto o con propósitos similares. Por ejemplo, para este proyecto existen ocho paquetes de trabajo:

⇒ *Sesión.* Este paquete incluye funcionalidad para gestionar las reglas de acceso al sistema: a qué páginas pueden acceder usuarios anónimos y en cuáles es necesario autenticarse, qué acciones puede realizar cada usuario y cuáles no, así como sistemas para iniciar y cerrar sesión.

⇒ *Desarrolladores.* Este paquete permite al administrador realizar la gestión de desarrolladores en el sistema: consultarlos, crearlos, editarlos y darlos de baja, así como bloquearlos para que no puedan acceder al sistema ni realizar publicaciones.

⇒ *Clientes.* Similar al anterior, este paquete permite al administrador gestionar clientes: consultarlos, crearlos, editarlos y darlos de baja. Igualmente, permite bloquearlos para que no puedan acceder al sistema ni descargar nuevas versiones. Además, incluye la creación y asignación de personas de contacto.

⇒ *Módulos.* En este paquete se recoge toda la funcionalidad relativa a la administración de módulos y sus versiones. Incluye la creación, modificación y borrado de informes, así como la consulta del histórico de versiones de cada uno de ellos. También engloba el borrado y la descarga de versiones.

⇒ *Parches.* Este paquete permite visualizar y descargar las versiones de parches publicadas en el sistema.

⇒ *Informes.* Homólogo al anterior, permite visualizar y descargar las versiones de informes.

⇒ *Configuración.* En este paquete se comprende toda la funcionalidad relativa a la configuración del sistema: parámetros que ajustan el funcionamiento sistema a las necesidades variables del cliente.

⇒ *Integración.* Este paquete incluye la integración del sistema central con los dos módulos externos que deben interactuar con él. Se divide en

dos subpaquetes: publicación de nuevas versiones, que incluye versiones de módulo, de parches y de informes y actualización, que incluye la descarga y el despliegue en los servidores del cliente de los elementos publicados.

Para cada uno de dichos paquetes, se obtienen los casos de uso que lo componen y se desarrolla un diagrama de casos de uso como el que se puede observar en la Figura 4.1. Una vez elaborado el diagrama, se explica detalladamente cada uno de sus casos de uso con el fin de obtener una descripción precisa y sin ambigüedad.

En el Anexo A se adjunta el documento de casos de uso completo elaborado para este proyecto.

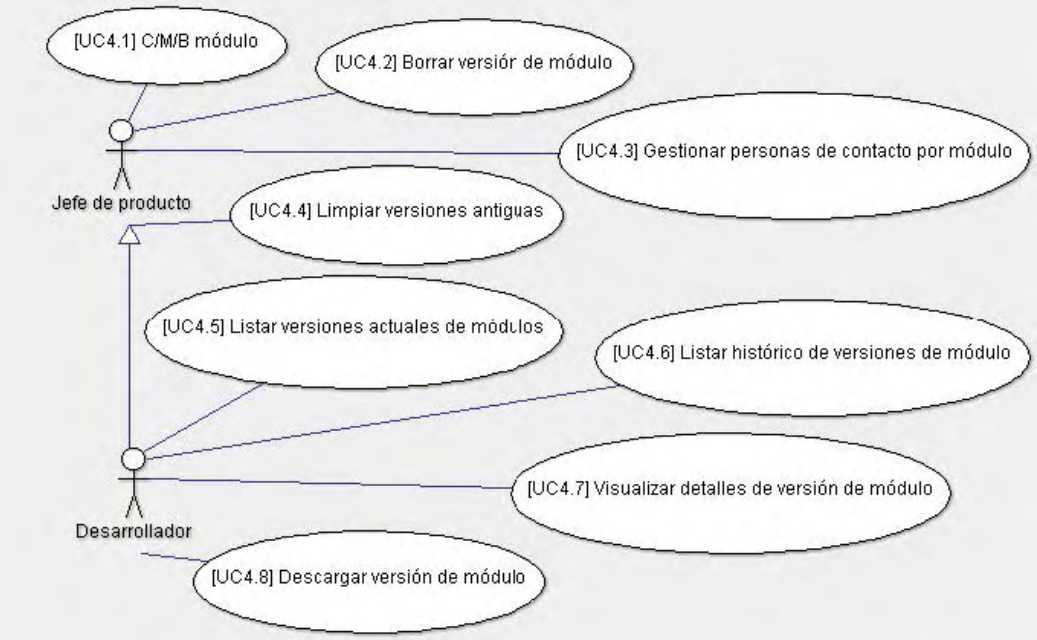

Figura 4.1: Ejemplo de diagrama de casos de uso.

4.2. Diagrama de clases

El segundo documento que se ha elaborado en esta fase ha sido un diagrama de clases. En él se determina qué clases existirán en el sistema que se va a desarrollar y de qué manera se relacionarán entre ellas.

En base una vez más a los resultados obtenidos en la reunión inicial con el cliente, se han identificado los elementos del modelo. Una vez definidos, se han creado las relaciones entre ellos.

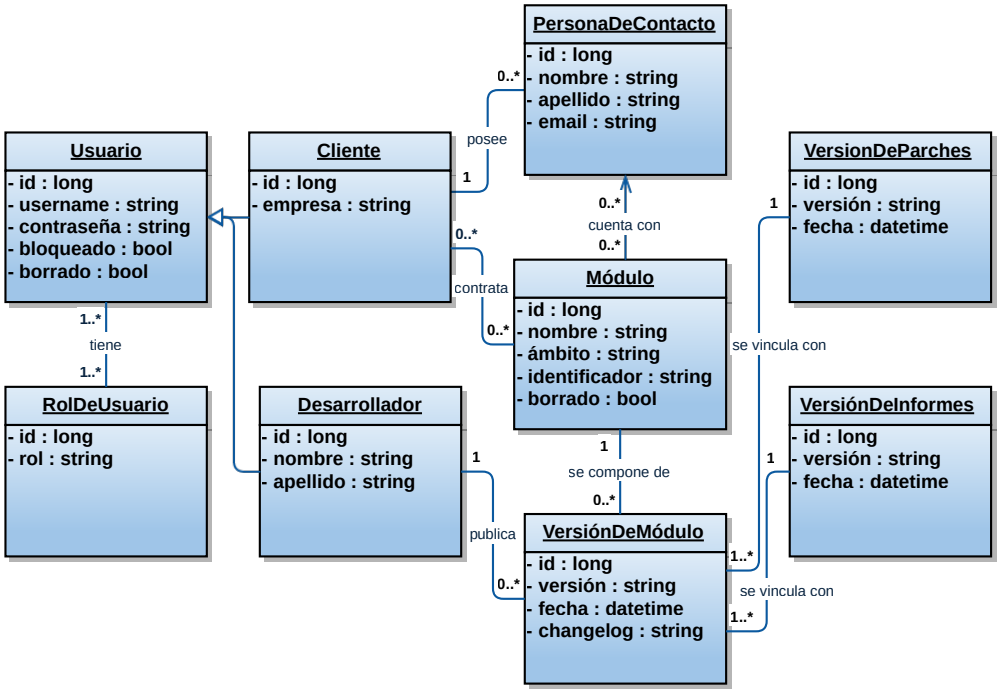

Figura 4.2: Diagrama de clases.

En la Figura 4.2 se puede observar el diagrama de clases obtenido en esta fase, cuyas clases y relaciones se detallan a continuación.

⇒ *Usuario*. Esta clase representa un usuario que puede acceder al sistema e interactuar con los datos que en él se almacenan. Cuenta con un nombre de usuario y una contraseña que le permiten identificarse e iniciar sesión en la aplicación. Cabe destacar que los usuarios pueden encontrarse en estado bloqueado, hecho que no les permite acceder al sistema ni realizar ninguna acción sobre el mismo.

⇒ *Rol de usuario*. Esta clase representa los tipos de usuarios del sistema. Existen tres roles diferentes: administrador, desarrollador y cliente. Cada uno de ellos tiene unos permisos distintos y puede realizar unas acciones u otras en el sistema. Por ejemplo, un administrador puede dar de alta clientes, módulos y desarrolladores; un desarrollador puede

publicar nuevas versiones de módulos, parches e informes; y un cliente puede descargar las versiones de los módulos que tiene contratados.

Esta clase está relacionada con la clase *Usuario*, pues cada uno de ellos debe tener asignado, al menos, un rol para poder acceder al sistema.

⇒ *Cliente* y *Desarrollador*. Estas dos clases están fuertemente relacionadas, ya que ambas son tipos de usuarios de los cuales el sistema almacena cierta información adicional.

⇒ *Persona de contacto*. Además de información sobre los módulos contratados, la clase *Cliente* también mantiene información sobre qué personas de contacto existen para un usuario de este tipo. Dado que los clientes son, por norma general, empresas, esta clase se utiliza para administrar a qué personas de dicha empresa se debe notificar cuando una nueva versión de algún módulo es publicada.

Se trata de una clase sencilla que cuenta únicamente con un nombre y un correo electrónico de contacto.

⇒ *Módulo*. Esta clase representa los módulos que componen el ERP. Como ya se ha comentado en el Capítulo 2, cada cliente puede contratar unos módulos u otros, en función de sus necesidades y presupuesto. El propósito de esta clase, a parte de mantener la información relativa a cada módulo, es determinar qué clientes pueden descargar las versiones de cada uno de ellos.

Obviamente, no es muy inteligente dejar que todos los clientes puedan descargar todos los módulos. Con esta clase se pretende limitar dicha libertad: cada cliente solamente podrá descargar las versiones de los módulos que tenga contratados.

Cabe destacar la relación que existe entre esta clase y *Persona de contacto*. Su propósito es organizar los contactos de un cliente por módulos, de manera que cuando se genere una notificación relativa a un módulo en concreto, esta pueda ser enviada al responsable de dicho módulo en la empresa cliente.

⇒ *Versión de módulo*. Esta clase almacena información sobre las versiones de cada módulo. La relación entre ambas clases es muy estrecha. Una versión de módulo debe pertenecer, necesariamente, a un único módulo.

Además de la información sobre la versión, cada instancia de esta clase cuenta con una referencia a los ficheros binarios que la componen. Este detalle facilita la tarea de ofrecer dichos ficheros por descarga directa a los clientes.

Como se puede observar en el diagrama de la Figura 4.2, existe una relación entre *Versión de módulo* y *Desarrollador*. Esta relación permite llevar un seguimiento de las versiones que publica cada desarrollador.

⇒ *Versión de parches* y *Versión de informes*. Estas dos son clases similares a la *Versión de módulo*. La principal diferencia reside en que se corresponden con las versiones de parches y de informes, respectivamente. Del mismo modo, también cuentan con referencias a los ficheros que componen cada una de ellas para permitir su descarga a los clientes.

Ambas están relacionadas únicamente con la clase *Versión de módulos*. Esto se debe a que cada nueva versión que se publica para un módulo cuenta, necesariamente, con una versión de parches y una de informes mínimas para poder funcionar de manera correcta.

Posiblemente el lector se esté preguntando por qué existe una clase llamada *Versión de informes* si en el Capítulo 2 se ha especificado que los informes no cuentan con número de versión. Aunque realmente las de informes no sean versiones propiamente dichas, sí que se trata de instancias que evolucionan en el tiempo. Sencillamente, se ha utilizado la palabra versión para mantener una nomenclatura similar en las tres clases.

4.3. Prototipos de la interfaz

Por último, se han creado una serie de prototipos o *mockups* de las interfaces más relevantes de la aplicación. A pesar de ser esquemas de la apariencia de la interfaz, no pertenecen a la fase de diseño. No se trata de diseños finales para el desarrollador, sino de una pequeña aproximación que sirve para que el cliente pueda verificar de una forma muy sencilla y visual que su aplicación va a contar con mecanismos que le permitirán realizar todas las acciones que ha solicitado.

Un ejemplo de los prototipos desarrollados en esta fase se encuentra en la Figura 4.3. En ella se puede observar que el prototipo contiene prácticamente todos los requisitos que el cliente había solicitado para el paquete de módulos (el resto se incluyen en un segundo prototipo) y que se recogen en el diagrama de casos de uso de la Figura 4.1.

Figura 4.3: Ejemplo de prototipo de la interfaz. Detalles de módulo.

En el Anexo B se pueden ver las diferencias entre los prototipos creados en esta etapa y los resultados obtenidos en la fase de diseño e implementación (Capítulo 5).

Capítulo 5

Diseño e implementación

Índice

En este capítulo se detalla el trabajo realizado durante la segunda fase del proyecto. Una vez finalizado el análisis de requisitos y con los tres documentos resultantes validados por el cliente, se ha procedido a realizar el desarrollo de la aplicación.

Conceptualmente, esta fase se divide en tres subfases.

⇒ La primera de ellas ha consistido en realizar el diseño interno de la aplicación. En ella se ha determinado la arquitectura que se ha utilizado, qué clases se han implementado y cómo se han relacionado entre sí. Para ello se ha utilizado como base el diagrama de clases elaborado en la Sección 4.2.

⇒ La segunda subfase ha consistido en diseñar el aspecto visual de las distintas interfaces de usuario que componen la aplicación. Este diseño se ha realizado de manera informal y no estandarizada. Partiendo de los prototipos obtenidos en la Sección 4.3 y utilizando HTML puro se han ido definiendo los modelos básicos de cada pantalla. Más tarde, durante la fase de implementación, se han ido puliendo dichos modelos y se han obtenido los resultados finales.

⇒ Por último, se ha procedido a la implementación de la aplicación en base a los diseños realizados en las dos fases anteriores. Durante esta fase, se ha realizado un seguimiento de todas las características implementadas y se han contrastado con los casos de uso incluidos en el documento elaborado durante la Sección 4.1. De esta manera, se ha verificado que todos los requisitos que el cliente ha solicitado están disponibles en la aplicación.

Tal y como se ha mencionado anteriormente, las dos últimas subfases se han realizado de forma iterativa. Una vez por cada paquete de trabajo se han diseñado las interfaces necesarias y se ha implementado la lógica de la aplicación.

5.1. El nuevo repositorio de versiones

Antes de comenzar con el desarrollo de la aplicación, se han instalado todas las herramientas necesarias en el equipo. Una versión de PHP, el entorno de desarrollo Aptana Studio 3, la última versión estable del framework Symfony2, Bootstrap 3 y un servidor FTP.

Una vez todo instalado y configurado, y dado el desconocimiento de las tecnologías por parte del alumno, se ha procedido a la lectura de la documentación de PHP y jQuery. Una vez entendidos los conceptos teóricos de estas tecnologías, se han realizado unas pruebas previas al desarrollo para obtener cierta soltura. Tras esto, se ha iniciado la implementación de la aplicación.

5.1.1. Paquete de sesión

Durante la iteración del paquete de sesión, se han diseñado cinco interfaces de usuario diferentes: el formulario de inicio de sesión (Figura 5.1), un formulario de recordatorio de contraseña, la página de perfil de usuario, un

Figura 5.1: Diseño final de la pantalla de login.

formulario de edición de usuario y otro para realizar los cambios de contraseña.

Con los modelos diseñados, se ha procedido a realizar la implementación. En primer lugar se ha configurado el *bundle* de seguridad de Symfony. Pero, ¿qué es un *bundle* en Symfony? El concepto de *bundle* se refiere a un paquete que ofrece funcionalidad para una característica concreta. Existen gran cantidad de estos paquetes en Symfony por defecto, como el de seguridad o el de persistencia, y existen muchos otros *bundles third party* fácilmente incorporables a cualquier proyecto.

El *bundle* de seguridad de Symfony ofrece funcionalidad para gestionar la autenticación y la autorización en el sistema. Gracias a él, se pueden implementar de manera sencilla el inicio de sesión y el control de usuarios de la aplicación.

Para definir esta configuración, se han especificado una serie de *firewalls* que regulan si es necesaria la autenticación o no en un determinado conjunto de páginas, y unas normas de control de acceso que regulan qué tipos de usuarios pueden acceder a cada uno de estos conjuntos de páginas.

Para esta aplicación se han necesitado varios *firewalls* con diferentes tipos de autenticación. Por ejemplo, la zona de la aplicación a la que acceden los desarrolladores vía web ha requerido una autenticación mediante usuario y contraseña con un formulario de inicio de sesión, mientras que la zona a la que se conectan los scripts actualizador y publicador ha requerido una autenticación básica HTTP, mediante la cual se envían el usuario y la contraseña en la misma URL de la petición.

Una porción ilustrativa de la configuración del control de acceso de la aplicación se encuentra en el siguiente fragmento de código.

```
security:
    firewalls:
        login:
            pattern:  ^/login$
            security: false

        xmlrpc:
            pattern:  ^/xmlrpc
            security: true
            http_basic: true

        secured_area:
            pattern:    ^/
            form_login:
                login_path: login
                check_path: login_check
            logout:
                path:   /logout
                target: /

    access_control:
        - { path: ^/login, roles : IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin, roles : ROLE_ADMIN }
        - { path: ^/xmlrpc/cli, roles : ROLE_CLIENT }
        - { path: ^/xmlrpc/dev, roles : ROLE_DEVELOPER }
        - { path: ^/client/download, roles : ROLE_CLIENT }
        - { path: ^/, roles : ROLE_DEVELOPER }
```

Con esto, se han definido las reglas de acceso a la aplicación. A continuación se han implementado las entidades *Usuario* y *Rol de usuario* que aparecen en el diagrama de clases (Figura 4.2) para poder comprobar las credenciales a partir de los datos almacenados en la base de datos.

De nuevo, Symfony cuenta con un *bundle* que facilita esta tarea. Se trata de Doctrine, un ORM que se encarga de crear vínculos entre las clases PHP de la aplicación y las tablas de la base de datos para poder cargar las filas a memoria de forma automática cuando se le solicita.

Para configurar los vínculos entre la base de datos y las clases PHP, se ha creado un fichero YAML en el que se ha especificado con qué atributo de la clase se corresponde cada columna de la base de datos. Además, también se han especificado las relaciones entre entidades. En este caso existe una relación de muchos a muchos entre *Usuario* y *Rol de usuario*. Finalmente, también se han definido otras restricciones como la longitud del texto de una columna o si dicha columna es única, entre otras.

A continuación se muestra un fragmento ilustrativo del código utilizado para configurar el mapeo de la entidad *Usuario*.

```
Planatec\AroUpdateBundle\Entity\User:
    type: entity
    table: p_users

    id:
        id:
            type: integer
            generator: { strategy: AUTO }

    fields:
        username:
            type: string
            length: 50
            unique: true

        password:
            type: string
            length: 255

        blocked:
            type: boolean

    manyToMany:
        roles:
            targetEntity: Role
            inversedBy: users
            joinTable:
                name: p_users_roles
                joinColumns:
                    user_id:
```

```
            referencedColumnName: id
    inverseJoinColumns:
        role_id:
            referencedColumnName: id
```

Con todo configurado, se ha procedido a desarrollar la lógica. Como Symfony es un *framework* basado en el patrón MVC, y tanto el modelo como las vistas ya han sido creadas, se han implementado los controladores que hacen accesibles las páginas al cliente.

Un controlador en Symfony se compone de tres elementos: una vista HTML, la ruta a la que accede el cliente desde su navegador y el método que se ejecuta cuando se accede a dicha ruta y que devuelve una respuesta HTTP con la vista HTML.

En el caso del inicio de sesión, se ha implementado un controlador que obtiene las credenciales introducidas por el cliente y comprueba si son correctas, contrastándolas con las entradas de la base de datos. Si las credenciales coinciden, se permite el acceso a la aplicación; mientras que si la autenticación no es satisfactoria, se devuelve un código de error.

Se han creado nueve controladores para el paquete de sesión: uno para mostrar el formulario de login; otro para gestionar los envíos de dicho formulario; uno más para mostrar el formulario de olvido de contraseña; el cuarto, para gestionar los envíos del mismo; el quinto, para mostrar la página de perfil; dos más para mostrar los formularios de edición y cambio de contraseña y los dos últimos, para tramitar los cambios introducidos por el usuario en ellos.

Por último, se han desarrollado una serie de scripts que se ejecutan en el lado del cliente para facilitar la experiencia del usuario. La función de estos scripts es ofrecer *feedback* informativo en ciertos casos, como por ejemplo, al introducir datos erróneos en cualquier formulario o para informar de que los cambios se han realizado satisfactoriamente.

Cabe destacar que, aunque no se mencione en los apartados siguientes, este tipo de scripts han sido desarrollados para todas las interfaces de la aplicación. Es importante ofrecer *feedback* informativo al usuario en todo momento acerca de los cambios que ha realizado en el sistema. Del mismo modo, también es importante realizar una validación de los campos de los formularios en el mismo navegador. Gracias a ello, se evita el envío peticiones al servidor cuando los datos introducidos no son correctos, hecho que ahorra tiempo y mejora la experiencia del usuario.

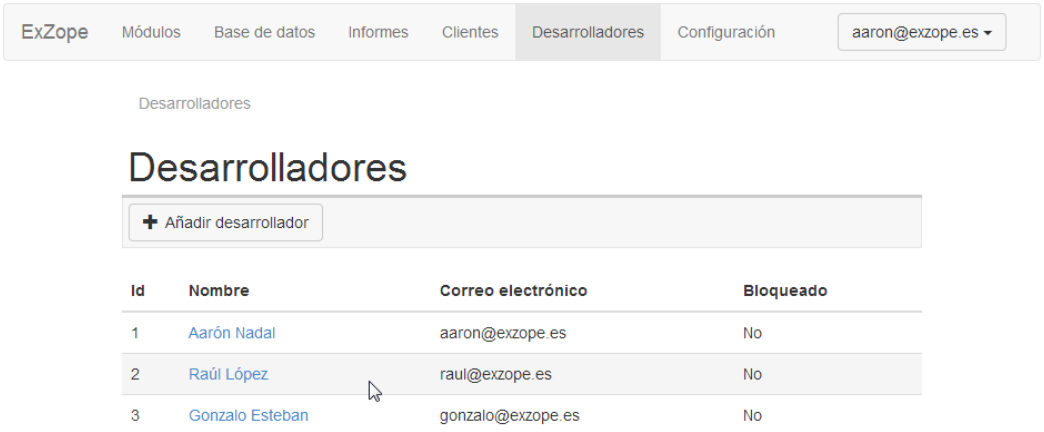

Figura 5.2: Diseño final de la pantalla de listado de desarrolladores.

5.1.2. Paquete de desarrolladores

El primer paso para implementar este paquete de funcionalidad ha sido construir una lista de desarrolladores como la de la Figura 5.2. Para ello, se ha credo una nueva entidad *Desarrollador*. Al igual que se ha hecho en el paquete anterior con *Usuario* y *Rol de usuario*, se ha implementado la clase PHP que representa un desarrollador y se ha creado el fichero YAML correspondiente para que Doctrine sepa cómo realizar el mapeo.

Esta nueva entidad se ha relacionado con la entidad *Usuario*, ya que un desarrollador, para poder acceder al sistema, requiere de la existencia de un usuario propio. Dado que cada desarrollador tan solo puede tener asignada una cuenta de usuario, y cada cuenta de usuario solo puede pertenecer a un desarrollador, la relación que se ha implementado tiene cardinalidad uno a uno.

Debido a esta gran dependencia, ha sido necesario implementar un método que crea de forma automática un usuario cuando el administrador da de alta un nuevo desarrollador en el sistema. Para conseguir este propósito, en el formulario de añadir nuevo desarrollador (Figura 5.3), además de los campos pertenecientes a la entidad *Desarrollador*, se han añadido tres campos adicionales vinculados a la cuenta de usuario: el correo electrónico, que se utiliza a modo de nombre de usuario; la contraseña, que puede ser introducida manualmente o generada de forma automática y el *checkbox* de 'insertar bloqueado'.

Como se puede observar en la Figura 5.3, no se ha implementado ninguna opción para insertar un desarrollador como administrador. Esto se ha

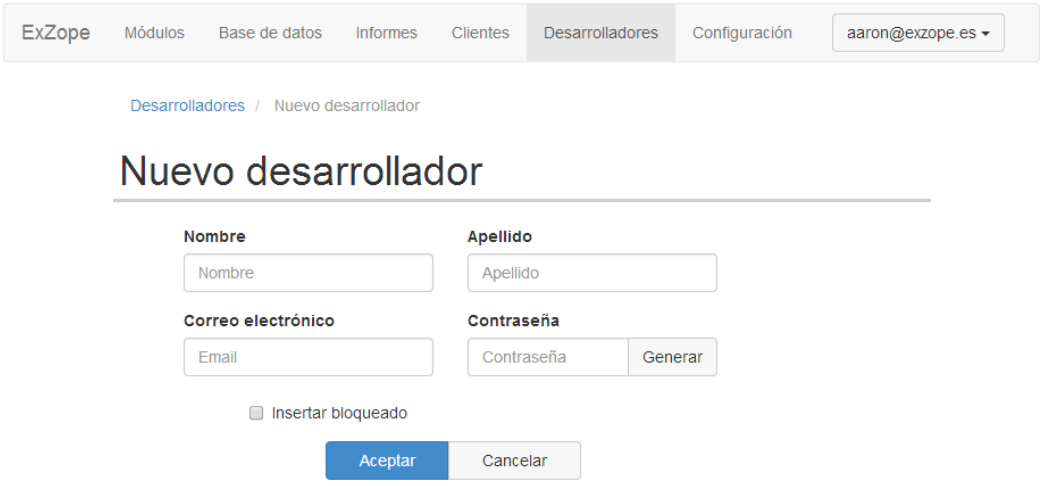

Figura 5.3: Diseño final de la pantalla para añadir desarrolladores.

debido a que en el documento de casos de uso no consta dicha funcionalidad. En la reunión inicial, el cliente especificó que no debía existir más de un administrador en el sistema.

Teniendo en cuenta este requisito, y dado que debe existir necesariamente un administrador en el sistema, este rol se asigna al primer usuario registrado, tal y como se explica en la Sección 7.1.

Se ha creado un vínculo que, al hacer click sobre alguno de los desarrolladores del listado de la Figura 5.2, permite acceder a la vista de detalle de los datos del desarrollador. En ella, se ha diseñado un listado con la información relativa a este de una manera más completa, así como acciones adicionales que se pueden realizar sobre él (editar sus datos, cambiar su contraseña de acceso, bloquearlo o darlo de baja).

En esta vista, además de lo nombrado anteriormente, se ha implementado una tabla que muestra el historial de versiones publicadas por el desarrollador. Para mejorar la experiencia del usuario, se ha implementado un script que ofrece la posibilidad de alterar el orden de los ítems de la tabla pulsando sobre el encabezado de cada columna. Dependiendo de la columna pulsada, las filas se ordenan alfabéticamente por módulo o cronológicamente por identificador de versión o fecha de publicación.

Por último, cabe destacar que, por exigencias del cliente, al dar de baja un desarrollador, este no se elimina de la base de datos, sino que su información permanece en ella. A pesar de que el desarrollador sigue presente en la base

de datos, se ha utilizado un *flag* que indica al sistema que debe ignorarlo cuando se realizan consultas sobre su tabla.

5.1.3. Paquete de clientes

La implementación de este paquete ha sido muy similar a la implementación del paquete de desarrolladores. Para empezar, se ha construido una lista de clientes homóloga a la lista de desarrolladores de la Figura 5.2. Para ello, también se ha creado una nueva entidad, en este caso, *Cliente*.

Al igual que la entidad *Desarrollador*, esta nueva entidad también se ha relacionado con *Usuario*: cada cliente necesita de la existencia de un usuario propio para el correcto funcionamiento del sistema. En caso contrario, el script actualizador del cliente no puede acceder al sistema para descargar las nuevas versiones de los módulos que tiene contratados.

De igual manera, el sistema ha sido preparado para crear automáticamente un usuario cuando el administrador da de alta un nuevo cliente. Para lograr dicho propósito, tal y como ocurre en el formulario de añadir nuevo desarrollador, en el de crear clientes también se han añadido los tres campos vinculados a la cuenta de usuario.

Por otra parte, cuando el administrador se dispone a dar de baja un cliente, ocurre lo mismo que ocurría al borrar desarrolladores. El controlador que gestiona esta acción ha sido implementado para que no borre la información de la base de datos. Mediante un *flag* booleano se indica al sistema que dicho cliente ha sido borrado. En la reunión inicial, el cliente especificó de forma explícita que la información debía permanecer en la base de datos para poder ser recuperada en caso de necesidad.

Las particularidades del paquete de clientes se hacen visibles en la pantalla de detalles. En ella, se muestran, además de la información básica del cliente, dos tablas adicionales: la primera de ellas con los módulos que tiene contratados y la otra con información acerca de sus personas de contacto.

Dado que cada cliente puede tener contratados unos u otros módulos, se ha dotado a la aplicación de un mecanismo para mostrar, tanto a los desarrolladores como al administrador, con cuáles de ellos cuenta cada cliente. Además, se ha facilitado la posibilidad de añadir módulos contratados y eliminar contratos existentes, aunque, tal y como se puede observar en el documento de casos de uso del Anexo A, esta acción solo puede realizarla el administrador.

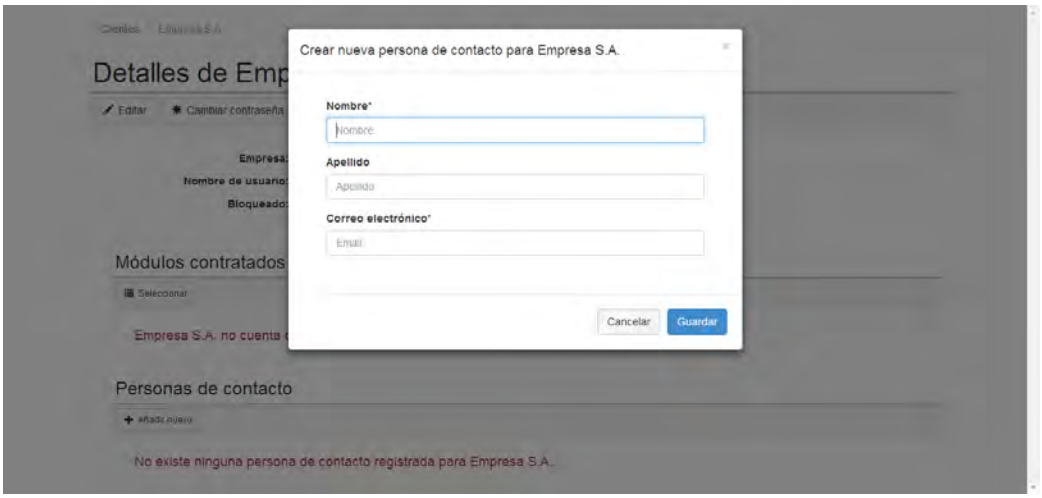

Figura 5.4: Diseño final de la pantalla para añadir personas de contacto.

El desarrollo del paquete de módulos y su relación con la entidad *Cliente* se explican con más detalle en la Subsección 5.1.4.

Por otro lado, se ha gestionado la información sobre las personas de contacto. Forman parte del cliente, por lo que se han incluido en el mismo paquete de trabajo. Asimismo, dado que las personas de contacto no tienen sentido si no pertenecen a un cliente concreto, también se han introducido en la vista de detalles del cliente.

Sin embargo, a pesar de estar tan fuertemente relacionadas con el cliente, se trata de instancias de una entidad independiente: *Persona de contacto*. Esta entidad mantiene una relación de uno a muchos con la entidad *Cliente*, debido a que un cliente puede tener varias personas de contacto, pero una persona de contacto tan solo puede pertenecer a un único cliente.

Las acciones que se han implementado sobre la tabla de personas de contacto son las acciones básicas de persistencia: añadir una nueva persona de contacto a un cliente, tal y como se ilustra en la Figura 5.4; editar una persona de contacto existente y eliminarla.

Para esta entidad, y a diferencia de lo que se ha implementado para *Desarrollador* y *Cliente*, al eliminar una persona de contacto del sistema, esta se elimina definitivamente. Esto es debido a que, en este caso, mantener la información de un contacto después de borrarlo carece de interés para el cliente.

5.1.4. Paquete de módulos

El paquete de módulos es el más extenso y, al mismo tiempo, el más importante del sistema. Por esta razón, su desarrollo se ha llevado consigo gran parte del esfuerzo total del proyecto.

En primer lugar, el diseño de este paquete de trabajo ha sido el más costoso y el que más difiere de los prototipos iniciales, tal y como se muestra en el Anexo B. Si se realiza la comparación, se puede observar que en la vista definitiva únicamente se lee la información relativa al módulo y se omite la referente a la versión seleccionada. Como se explica más adelante en este mismo apartado, la información de las versiones se muestra en un diálogo modal independiente.

Para el desarrollo de este paquete son necesarias dos entidades nuevas: *Módulo* y *Versión de módulo*. La primera de ellas cuenta con los campos nombre del módulo, ámbito en el que se centra, una cadena única y sin caracteres especiales que identifica al módulo y un *flag* booleano que determina si dicho módulo ha sido dado de baja o no.

Además, se relaciona con otras tres entidades: *Versión de módulo*, *Cliente* y *Persona de contacto*.

⇒ Debido a que cada nueva *Versión de módulo* publicada pertenece a un *Módulo*, ambas entidades cuentan con una relación de uno a muchos: un módulo puede contar con varias versiones, pero cada versión pertenece a un único módulo.

⇒ Un *Módulo* se relaciona con un *Cliente* cuando el segundo solicita la contratación del primero. Se trata de una relación de muchos a muchos porque un cliente puede tener contratados muchos módulos y, a su vez, un mismo módulo puede estar contratado por varios clientes.

⇒ La relación entre *Versión de módulo* y *Persona de contacto* surge a partir de la necesidad de organizar los contactos de un cliente por módulo. Gracias a esta relación, al publicar nuevas versiones de un módulo, en lugar de notificar a todos y cada uno de los contactos del cliente, el sistema notifica únicamente a aquellos asignados al módulo en cuestión.

La segunda entidad, *Versión de módulo*, cuenta con un identificador de la versión de la forma *xxx.xxx.xxx.xxx*, la fecha en la que se ha realizado la publicación y una lista con los cambios introducidos en la misma.

Las versiones de módulo se relacionan con cuatro entidades: *Desarrollador*, *Módulo*, *Versión de parches* y *Versión de informes*.

⇒ La relación entre *Versión de módulo* y *Desarrollador* aparece en el momento en el que un desarrollador publica una versión. Esta relación se utiliza para llevar un seguimiento de qué versiones publica cada desarrollador: cuando se publica una nueva versión, el sistema obtiene al desarrollador a partir del usuario autentificado y la relación se crea automáticamente.

⇒ Como se ha explicado anteriormente, una *Versión de módulo* se relaciona con el *Módulo* al que pertenece.

⇒ Las relaciones de *Versión de módulo* con *Versión de parches* y *Versión de informes* se explican en la Subsección 5.1.5.

Con las dos entidades creadas y para ofrecer una visión general de los módulos del sistema, se ha implementado una lista que muestra todos los módulos registrados, así como información relevante de los mismos: su nombre, el ámbito en el que se centran, el identificador de la última versión, la fecha de publicación de esta, y el desarrollador que la ha publicado.

Por otro lado, el controlador que se encarga de crear nuevos módulos ha sido dotado de funcionalidad que le permite organizar el sistema de ficheros en el que se almacenarán los binarios de las versiones. Al insertar un nuevo módulo en el sistema, este controlador también crea un directorio con el identificador del módulo por nombre. Es en este directorio en el que se almacenan los ficheros de las versiones pertenecientes a dicho módulo.

Del mismo modo, el controlador que gestiona las modificaciones de los módulos también ha sido preparado para actualizar el sistema de ficheros cuando se cambie el identificador del módulo. Este ha sido un aspecto vital, debido a que si el sistema de ficheros no se mantiene, las versiones publicadas antes de la modificación podrían quedar huérfanas: al intentar acceder a la ruta de los binarios, el sistema no la encontraría, pues no existiría ningún directorio con el identificador actualizado del módulo por nombre.

Detalles de módulo

Al igual que se ha realizado con los desarrolladores y los clientes, para observar con más detalle cada uno de los módulos, se ha implementado la vista de detalles que se muestra en la Figura 5.5. Para acceder a ella, se ha creado un hipervínculo que conecta cada elemento de la lista de módulos con su vista de detalles particular.

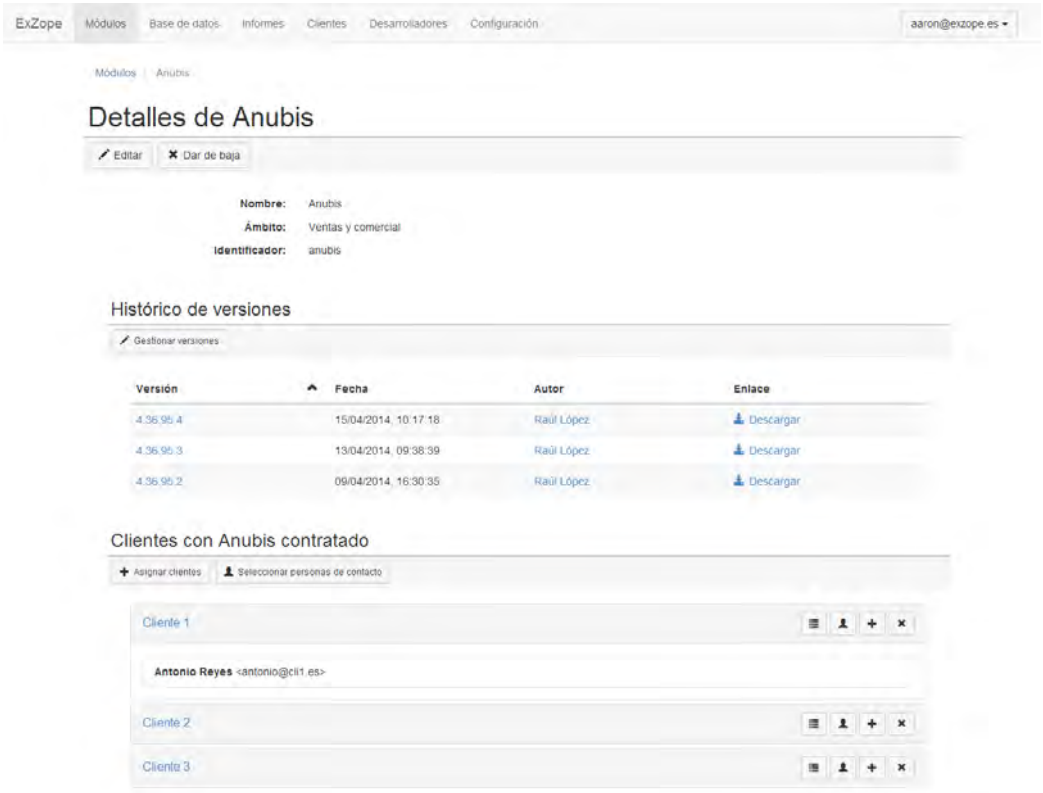

Figura 5.5: Diseño final de la pantalla de detalles de módulo.

La vista de detalles de módulo cuenta con la información básica del módulo, una tabla con un historial de las versiones publicadas y una lista con las personas de contacto organizadas por clientes. Además, desde esta vista se permite realizar una gran cantidad de acciones: es posible modificar o dar de baja un módulo, gestionar las versiones publicadas, asignar nuevos clientes y desvincular los existentes, seleccionar las personas de contacto de cada cliente y, en caso de que no existan, crear nuevas personas de contacto para un cliente y relacionarlas con el módulo de manera automática.

La mayoría de estas acciones se llevan a cabo mediante scripts jQuery y peticiones AJAX. Esto ofrece una mayor fluidez al sistema, debido a que la mayor parte del cómputo se realiza en el navegador del usuario y a que no es necesario refrescar la página tras cada envío al servidor. Gracias a ello, se mejora considerablemente la experiencia del usuario.

Para realizar un mantenimiento de las versiones publicadas, se ha utilizado un sistema de tres pasos más confirmación para evitar errores del usuario:

1. Se hace click en el botón de *Gestionar versiones* que aparece en la Figura 5.5. Tras ello, se despliega un checkbox por cada versión y aparecen los botones de *Borrar seleccionadas* y *Cancelar* (ver Figura 5.6).
2. Se seleccionan las versiones que se desean eliminar.
3. Se pulsa el botón de *Borrar seleccionadas*.
4. Se confirma la acción en el diálogo que aparece en última instancia.

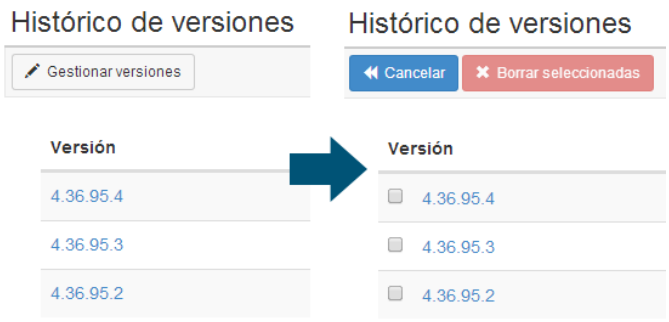

Figura 5.6: Funcionamiento del botón de gestión de versiones.

Se ha tomado la decisión de implementarlo de esta manera en lugar de ofrecer un mecanismo más sencillo debido a que se trata de una acción crítica y, al mismo tiempo, muy poco frecuente.

Por otro lado, para seleccionar las personas de contacto de cada cliente, se han implementado dos posibilidades. Ambas utilizan un diálogo modal como el que aparece en la Figura 5.7.

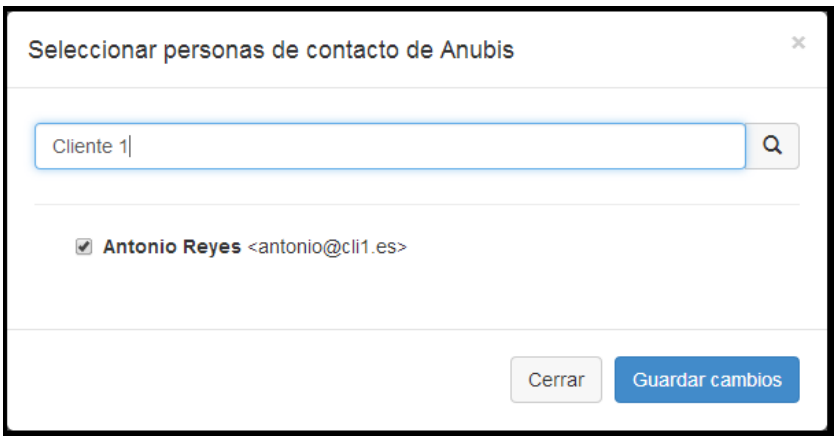

Figura 5.7: Diálogo modal de selección de contactos de módulo.

Al hacer click sobre el botón de *Seleccionar personas de contacto* en la barra de botones de la lista de clientes, se muestra el diálogo modal donde el usuario realiza la búsqueda del cliente al que pertenece la persona de contacto que se desea asignar. Como ayuda, el campo de búsqueda cuenta con un mecanismo de autocompletado, de manera que en la mayoría de ocasiones no es necesario teclear el nombre completo.

Por otro lado, al hacer click sobre el icono de *Seleccionar personas de contacto* en la barra de botones de un cliente concreto, se muestra el diálogo modal con el campo de búsqueda ya rellenado con el nombre de dicho cliente.

Al realizar la búsqueda, se envía una petición AJAX al servidor, solicitando los contactos del cliente buscado. Al recibir la respuesta, la lista de clientes se muestra en el diálogo modal, permitiendo al usuario seleccionar y deseleccionar los contactos correspondientes.

Detalles de versión

Para mostrar los detalles de cada versión se ha recurrido a un diálogo modal que aparece al seleccionar cualquier versión de la vista de detalles de módulo (ver Figura 5.8). En él, se muestra una ficha detallada de la versión seleccionada, que contiene: el módulo, el identificador de la versión, el desarrollador, la fecha de publicación, el histórico de cambios y las versiones de parches (bases de datos) e informes mínimas necesarias.

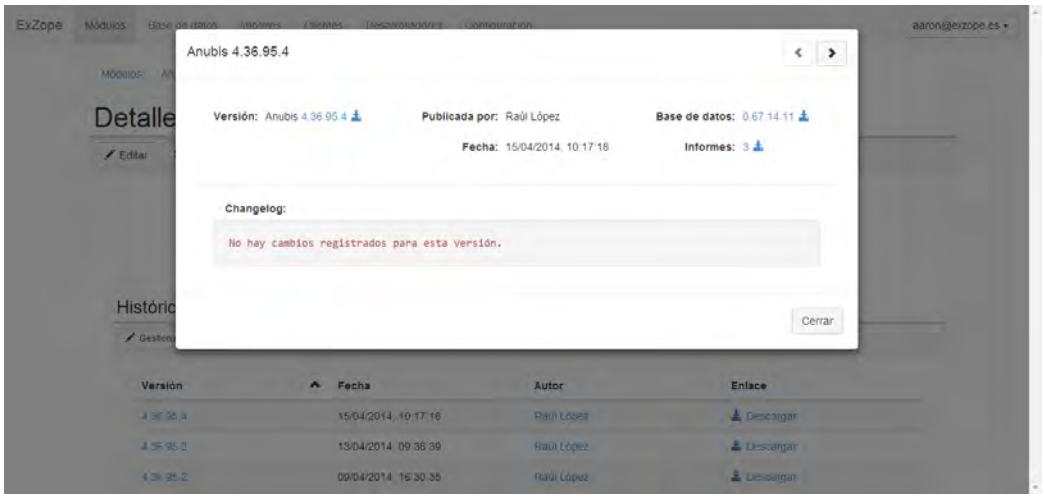

Figura 5.8: Diseño final de la pantalla de detalles de versión de módulo.

Para facilitar la comparación de versiones, se ha añadido un panel de navegación en la esquina superior derecha del modal. Gracias a él, el usuario tiene la posibilidad de navegar adelante y atrás entre las diferentes versiones del módulo sin necesidad de cerrar el diálogo.

Además, desde esta vista, se permite al cliente descargar los ficheros de la versión detallada, así como los de las respectivas versiones de parches e informes. Como se explica en la Subsección 5.2.2, al publicar una nueva versión, el sistema crea automáticamente una ruta para almacenar sus binarios. Cuando el usuario procede a descargar estos ficheros, el sistema los obtiene de esta ruta y los devuelve como descarga directa.

Cabe destacar, que antes de devolver el fichero, el sistema comprueba las credenciales del usuario registrado, de manera que si no se trata de un desarrollador o de un cliente con el módulo al que pertenece la versión contratada, no se permite la descarga.

A continuación, se muestra el código que implementa esta funcionalidad:

```
class DownloadController extends Controller {

    ...

    /**
     *  Comprueba las credenciales del usuario para determinar
     *  si tiene permisos para realizar la descarga.
     */
    private function isDownloadAllowed($version) {
        // Si es administrador, puede descargar.
        if ($this -> get('security.context') ->
             isGranted('ROLE_ADMIN')) {
            return true;
        }

        // Si es desarrollador, puede descargar.
        if ($this -> get('security.context') ->
             isGranted('ROLE_DEVELOPER')) {
            return true;
        }

        // Si es cliente, puede descargar
        // solo si tiene el módulo contratado.
```

```
        if ($this -> get('security.context') ->
             isGranted('ROLE_CLIENT')) {

            $client = $this -> get('query') ->
                findClientByUser($this -> get('security.context') ->
                getToken() -> getUser());

            return $client -> getModules() ->
                contains($version -> getModule());
        }
        return false;
    }

    public function downloadModuleVersionAction(Request $request) {
        $modname = $request -> query -> get('mod');
        $modversion = $this ->
            desnormalizeVersion($request -> query -> get('ver'));

        // Código de validación de los parámetros.
        ...

        // Si la descarga no está permitida,
        // se lanza una excepción
        if(!$this -> isDownloadAllowed($version)) {
            throw new AccessDeniedException("Acceso denegado.
                No tienes permisos para descargar la versión
                $modversion de $modname");
        }

        // Se obtiene la ruta con los ficheros.
        $filename = $this ->
            get('files') -> getModuleVersionFilePath($version);

        // Se devuelven los ficheros por descarga directa.
        return $this -> createFileDownloadResponse($filename);
    }

    // Controladores para la descarga de parches e informes.
    ...

}
```

5.1.5. Paquetes de parches e informes

Estos dos paquetes se analizan en una misma sección debido a que ambos son muy similares y sencillos. Si se observan los diagramas de casos de uso del Anexo A relativos a estos dos paquetes, se puede comprobar que cada uno de ellos cuenta únicamente con dos casos de uso: listar un histórico de versiones y descargarlas.

Para implementar las listas de versiones se han creado las entidades *Versión de parches* y *Versión de informes*, y se han implementado los controladores que obtienen las versiones de la base de datos y las muestran al cliente.

Como se observa en la Figura 5.9, en el caso de que una misma versión de parches o informes sea compartida por varias versiones de módulo diferentes, se agrupan en la misma fila. Al hacer click sobre dicha fila, se despliegan las versiones de módulo que dependen de ella.

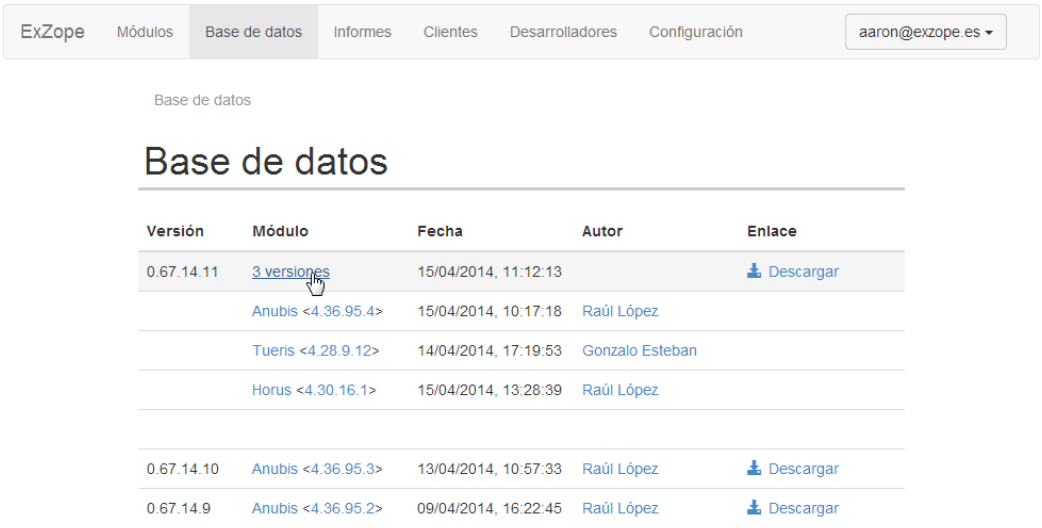

Figura 5.9: Diseño final del listado de versiones de parches.

Para iniciar la descarga de una versión, se ha colocado un hipervínculo en la parte derecha de cada fila de la lista. Se han creado también dos controladores que se encargan de leer los binarios de las versiones de parches e informes, respectivamente, y de devolverlas al usuario mediante descarga directa. El funcionamiento de estos controladores es homólogo al descrito en la Subsección 5.1.4.

5.1.6. Paquete de configuración

En un principio, el paquete de configuración no formaba parte de los requisitos del proyecto, sin embargo, al comenzar el desarrollo, ha aparecido una serie de variables cuyo valor no estaba definido, como la dirección de envío de correos, o que podían variar con el tiempo, como el tamaño del historial de versiones. Como solución, ha surgido la idea de agrupar este conjunto de variables en una pantalla que permite al usuario realizar cambios sobre ellas. El diseño final de la vista se puede observar en la Figura 5.10.

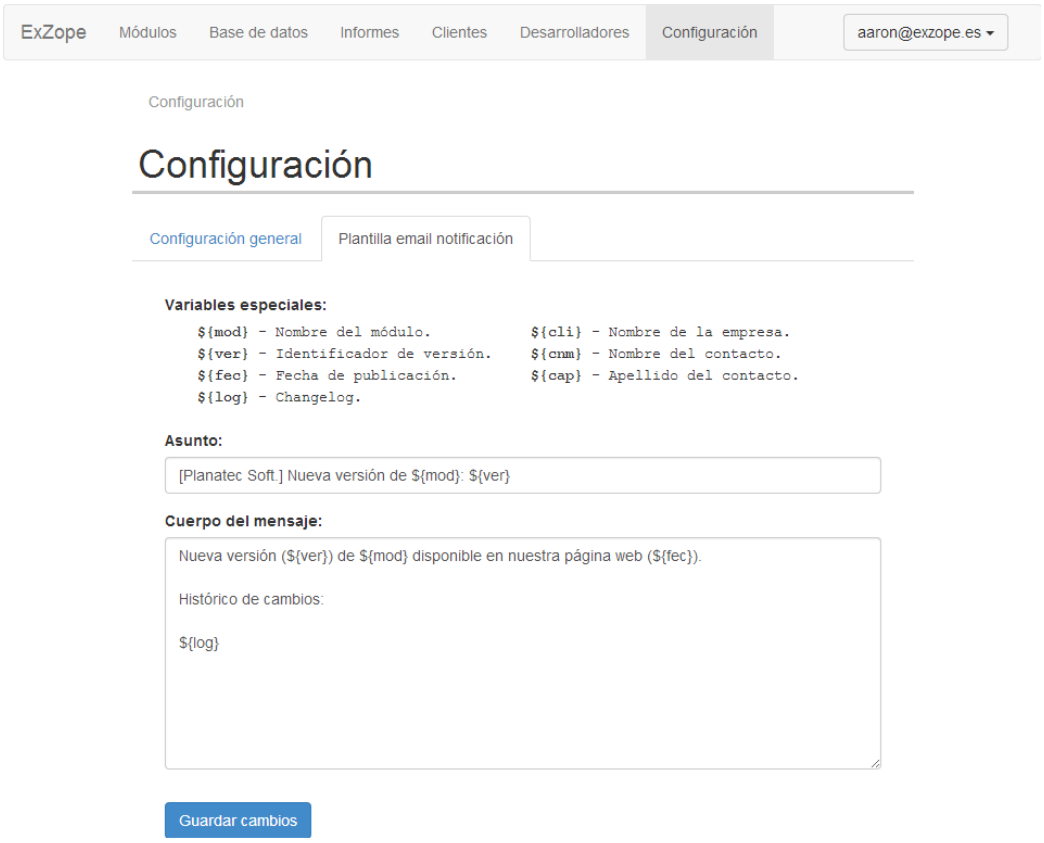

Figura 5.10: Diseño final de la pantalla de configuración.

Organizados en dos pestañas (*configuración general* y *plantilla email configuracion*), los campos que el usuario puede modificar son los siguientes:

⇒ *Nombre de la aplicación.* Nombre que se mostrará cuando se haga referencia a la aplicación en correos o el portal web.

⇒ *Dirección de correo remitente*. Dirección de correo electrónico con la que se enviarán los mensajes desde la aplicación.

⇒ *Número máximo de versiones en el historial*. Cantidad máxima de versiones que se mantendrán en el historial de versiones de cada módulo. Una vez se sobrepase este límite, se eliminarán las versiones más antiguas hasta alcanzarlo de nuevo.

⇒ *Asunto del mensaje de notificación*. Texto que se mostrará en el asunto del correo de notificación de nueva versión. Se pueden utilizar algunas variables especiales como ${mod}, que inyecta el nombre del módulo actualizado; ${ver}, que inyecta el identificador de versión o ${fec}, que inyecta la fecha, entre otros.

⇒ *Cuerpo del mensaje de notificación*. Texto que se mostrará en el cuerpo del correo de notificación de nueva versión. Al igual que en el ítem anterior, también se pueden utilizar las variables especiales.

5.2. Integración con los módulos externos

Una vez completado y validado el desarrollo de la aplicación web, se ha procedido a realizar la integración con los módulos publicador y actualizador. Para ello, en primer lugar, se ha implementado un servidor XML-RPC en el módulo central que posibilita la comunicación de este con los scripts externos.

Una vez habilitada la comunicación, se ha integrado el nuevo sistema con el script publicador. Este hecho permite al script publicar las nuevas versiones en el repositorio central desarrollado.

Finalmente, se ha integrado la aplicación con el script actualizador. De este modo, el actualizador es capaz de descargar las versiones más actuales del nuevo repositorio central y desplegarlas en el ordenador del cliente.

5.2.1. Implementación del protocolo XML-RPC

Por exigencias del cliente y dado que los módulos externos utilizan este protocolo, se ha tenido que implementar un servidor XML-RPC en el nuevo sistema. Del mismo modo, se ha tenido que modificar ligeramente el cliente que utilizan los scripts para que realice las peticiones al nuevo sistema en lugar de enviarlas al antiguo.

Servidor en PHP con Symfony2

Para la creación del servidor XML-RPC en Symfony2, se ha utilizado un *Bundle third party* que implementa dicho protocolo. Para utilizar este *Bundle*, se han implementado dos controladores, uno para las acciones del publicador y otro para las del actualizador. Del mismo modo, se han creado dos clases auxiliares que implementan los métodos que se podrán ejecutar remotamente.

A continuación se muestra el código que implementa los dos controladores. El primero, `xmlrpcCliAction()`, se corresponde con el que controla las acciones del actualizador y el segundo, `xmlrpcDevAction()`, se corresponde con el que controla las del publicador.

```
class XmlRpcController extends Controller {

    public function xmlrpcCliAction() {
        $server = new Server();
        $xmlrpc = $this -> get('xmlrpc.cli');

        $server -> addHandler('xmlrpc', $xmlrpc);

        return $server -> handle($this -> getRequest());
    }

    public function xmlrpcDevAction() {
        $server = new Server();
        $xmlrpc = $this -> get('xmlrpc.dev');

        $server -> addHandler('xmlrpc', $xmlrpc);

        return $server -> handle($this -> getRequest());
    }
}
```

La segunda instrucción de cada controlador obtiene una de las clases auxiliares mencionadas anteriormente: la del primer controlador obtiene la relativa al actualizador y la del segundo la referente al publicador.

Cliente en Python

Para el cliente, se ha utilizado la implementación de XML-RPC nativa de Python. Se ha instanciado un cliente XML-RPC y se ha asignado la ruta del controlador de desarrolladores al publicador y la del de clientes al actualizador. En el fragmento siguiente se muestra el código utilizado para la conexión del publicador:

```
auth = 'aaron@exzope.es:aaron@localhost:8085'
con = xmlrpclib.ServerProxy(
     'http:' + auth + '/Symfony/web/xmlrpc/dev')
```

Dependiendo del usuario que ejecuta el script, la variable 'auth' se inicializa con unos valores u otros. Esto es debido a que en dicha variable constan las credenciales del usuario que intenta acceder al sistema.

Una vez realizada la conexión, se han implementado las llamadas a los métodos remotamente de la siguiente manera: `res = con.xmlrpc.ping()`, siendo *ping()* el método que se desea ejecutar.

5.2.2. Integración con el publicador

Para realizar la integración con el publicador, se ha tenido que replicar la funcionalidad que ofrecía el antiguo sistema al script. Un total de siete métodos han sido implementados:

⇒ `ping()`. Devuelve una cadena estática. Sirve para comprobar que la conexión se realiza correctamente.

⇒ `listModules()`. Lista los identificadores de todos los módulos registrados en el sistema. Sirve para determinar para qué módulos se pueden publicar versiones.

⇒ `listVersions($module)`. Lista los identificadores de versión de todas las versiones publicadas para el módulo `$module`. Sirve para saber si una versión ya ha sido publicada.

⇒ `listPatches()`. Lista todas las versiones de parches publicadas en el sistema. Sirve para determinar si se deben publicar los parches relativos a la versión que se va a publicar.

⇒ `listReports()`. Homólogo a `listPatches()`, pero con las versiones de informes.

⇒ `generateTempDir()`. Genera un directorio temporal en el servidor. Sirve para colocar en él, vía FTP, los binarios de la versión que se va a publicar.

⇒ `createVersion()`. Recibe diez parámetros con la información de la versión y los *checksums* para comprobar la validez de los ficheros. Sirve para registrar nuevas versiones en el sistema.

Este método realiza todas las acciones necesarias para la correcta publicación de las versiones: obtiene el histórico de cambios del repositorio SVN, copia los ficheros subidos por el publicador del directorio temporal al de despliegue, publica las versiones de parches e informes si todavía no existen en el sistema, establece las relaciones entre la versión del módulo y las de parches e informes, así como con el desarrollador que realiza la publicación y notifica a las personas de contacto del módulo en cuestión.

5.2.3. Integración con el actualizador

Para realizar la integración con el actualizador, se han implementado los diez métodos que el script utilizaba del sistema anterior:

⇒ `ping()`. Devuelve una cadena estática. Sirve para comprobar que la conexión se realiza correctamente.

⇒ `getLastVersion($module)`. Devuelve la última versión publicada para el módulo `$module`. Sirve para comprobar si el cliente tiene dicho módulo actualizado o, por contra, se ha publicado alguna nueva versión.

⇒ `getLastDatabase()`. Devuelve la última versión de parches publicada en el sistema. Sirve para comprobar si se debe actualizar o no la base de datos.

⇒ `getLastReports()`. Homólogo al método anterior, pero con versiones de informes.

⇒ `getVersionChecksum($module, $version)`, `getDatabaseChecksum($version)` y `getReportsChecksum($version)`. Tres métodos muy similares que devuelven el *checksum* de los ficheros comprimidos en zip de la versión de

módulo, de parches o de informes, respectivamente. Sirven para comprobar la integridad de los ficheros descargados.

⇒ `getVersionUrl($module, $version)`, `getDatabaseUrl($version)` y `getReportsUrl($version)`. Tres métodos que devuelven la URL para descargar la versión de módulo, de parches o de informes, respectivamente. Sirven para que el actualizador sepa a qué URL debe acceder para descargar la últimas últimas versiones.

Capítulo 6

Pruebas

Índice

Una vez finalizada la implementación de la aplicación y con la integración terminada, ha sido necesario realizar una serie de pruebas sobre el resultado obtenido para verificar su correcto funcionamiento.

Según la metodología de trabajo de la empresa, las pruebas han sido planificadas para una fase final del proyecto. A pesar de ello, se han ido realizando varias pruebas básicas durante el desarrollo del mismo. Estas pruebas intermedias han verificado el buen funcionamiento de las relaciones entre entidades, la correcta visualización de las distintas páginas, la validación de los campos de formulario y el funcionamiento apropiado de los diferentes hipervínculos de la aplicación, entre otras cosas.

Con los aspectos básicos ya validados, en esta fase se han realizado las pruebas que han certificado el correcto funcionamiento del sistema completo. Estas pruebas han sido de dos tipos diferentes: pruebas del sistema y tests de aceptación.

6.1. Pruebas del sistema

Las pruebas del sistema sirven para probar porciones individuales del sistema, ya sean módulos, entidades, clases o paquetes de desarrollo, y verificar que su comportamiento se corresponde con el precisado en la fase de análisis.

En primera instancia, se han estudiado los aspectos más importantes del sistema y se ha diseñado una batería de pruebas. Este análisis y diseño de las pruebas se ha realizado de manera informal y no estandarizada en una reunión oral con el jefe del proyecto.

En la batería de pruebas resultante, se han incluido, entre otros aspectos, la validación del control de usuarios en los controladores de descarga de versiones, la publicación de versiones en diferentes circunstancias (versiones de parches o informes existentes, versión ya publicada con anterioridad, etc.) y la integridad del sistema de ficheros tras la creación de nuevos módulos, la modificación de los existentes y la publicación y borrado de las versiones.

Dentro de las pruebas del sistema también se han incluido aspectos de seguridad, como la protección del sistema ante ataques de SQL-injection o el correcto funcionamiento de los roles de usuario.

6.2. Tests de aceptación

Para realizar los tests de aceptación, se ha definido una serie de escenarios de utilización del sistema y el comportamiento esperado del mismo en dichos escenarios. Tras ello, se ha reproducido cada uno de los escenarios y se ha contrastado el comportamiento obtenido con el comportamiento esperado. A continuación se enumeran algunos ejemplos de escenario:

⇒ Dados tres clientes con un mismo módulo contratado, teniendo el primero de ellos todos sus contactos asignados a dicho módulo, el segundo solamente uno y el tercero ninguno, al publicar una nueva versión para ese módulo, el sistema debe notificar únicamente a los contactos asignados.

⇒ Dado un módulo con, al menos, una versión publicada, al borrar dicha versión, los ficheros que la componen deben dejar de estar disponibles para su descarga.

⇒ Dado un usuario bloqueado y otro dado de baja, el sistema no debe permitir a ninguno de ellos iniciar sesión ni realizar ninguna acción.

De nuevo, no se ha seguido ningún proceso formal o estandarizado para la realización de los tests de aceptación. La definición de los escenarios ha sido realizada junto con el cliente de manera oral en una reunión informal y

la ejecución de la batería de pruebas se ha llevado a cabo manualmente por el alumno.

Tanto las pruebas del sistema como los tests de aceptación han sido realizados sobre el entorno de desarrollo. Este entorno, íntegramente creado por el alumno, podría no representar con total fidelidad un entorno real.

Para realizar una validación más exhaustiva del nuevo sistema, el supervisor ha sugerido probar el nuevo sistema en el entorno de explotación. Para ello, se ha creado una nueva versión del script publicador en la que las publicaciones de nuevas versiones se realizan en ambos sistemas al mismo tiempo.

Como se puede observar en la Figura 6.1, se ha añadido funcionalidad al publicador para que publique en ambos sistemas paralelamente. Sin embargo, para evitar que los posibles problemas que pueda generar el nuevo sistema afecten a los clientes, estos han seguido obteniendo las actualizaciones del sistema antiguo.

La prueba en explotación del actualizador desarrollado para la nueva aplicación se ha realizado sobre una réplica del entorno de los clientes. Se ha simulado la contratación de diversos módulos en un servidor de la empresa y se ha desplegado ARO tal y como se hubiera hecho en un nuevo cliente. Luego, se ha sustituido el actualizador existente por el que descarga las versiones del nuevo sistema.

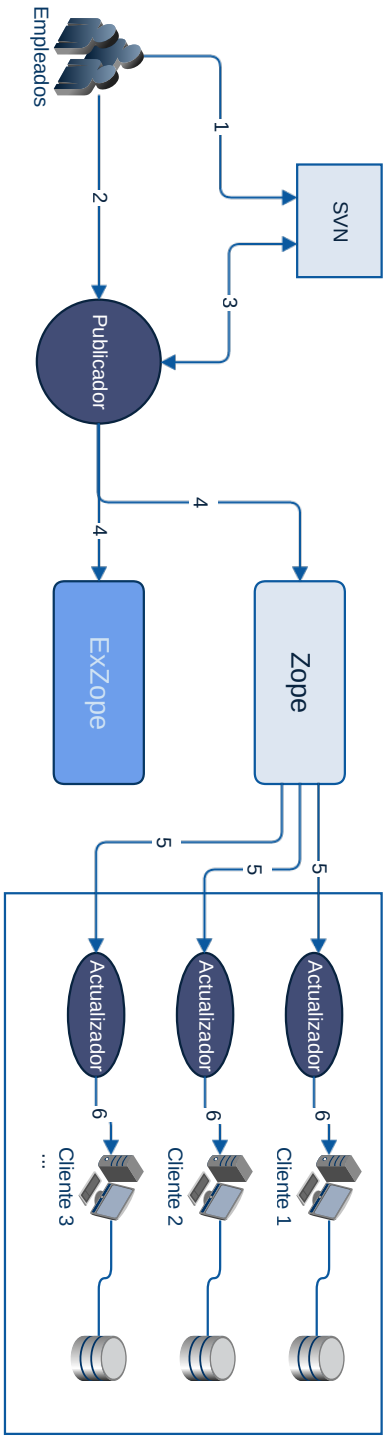

Figura 6.1: El sistema de actualizaciones durante los tests de aceptación.

Capítulo 7

Despliegue

Índice

Aunque en un principio estaba planificado realizar un despliegue completo de la aplicación, durante la estancia han surgido una serie de problemas en Planatec Software S.L. que han causado que no sea beneficioso para la empresa realizar dicho despliegue todavía. En su lugar, se ha preparado el sistema para poder ser instalado, configurado y mantenido de una forma sencilla y se han redactado dos manuales que explican cómo realizar dichas acciones.

7.1. Preparando la instalación

El jefe del proyecto ha solicitado que el sistema cuente con un proceso de instalación sencillo. Para ello, se ha realizado un conjunto de cambios en el sistema que lo preparan para ofrecer una instalación fácil y prácticamente automatizada:

⇒ Se han creado tres scripts independientes que inicializan y configuran el sistema automáticamente, facilitando la tarea al usuario que realiza la instalación.

⇒ Se ha implementado un nuevo controlador que, a partir de un formulario de iniciación, registra en el sistema el usuario administrador. Por razones de seguridad, este formulario es accesible únicamente si no existe ningún usuario en el sistema. Una vez registrado el primer usuario, al intentar acceder, se devolverá una respuesta con código de estado *403 - Forbidden* que informará al usuario de que el acceso al recurso ha sido denegado.

⇒ Se ha realzado una nueva fase de pruebas para verificar que los cambios introducidos para facilitar la instalación funcionan correctamente y no afectan al resto del sistema.

7.2. Manual de instalación

Para que cualquier usuario sea capaz de instalar y configurar el sistema, el jefe de proyecto ha solicitado la redacción de un manual de instalación en el que se detalle paso a paso el proceso de instalación del sistema. Además de este proceso, se ha añadido una guía con los parámetros de configuración del sistema, así como una sección de problemas frecuentes y cómo solucionarlos.

En el Anexo C se adjunta el documento resultante.

7.3. Manual de administración

Del mismo modo, para que el usuario sepa cómo realizar una correcta administración del sistema, el jefe de proyecto ha requerido la redacción de un manual de administración. En él se detallan las tareas adicionales que deben realizarse tras efectuar algunas acciones en el sistema. Se trata de tareas externas que no pueden ser automatizadas y que deben ser efectuadas de forma manual.

En el Anexo D se adjunta el documento resultante.