Informe de Análisis

Integrantes del grupo C2.026:

* Ignacio Blanquero Blanco ([ignblabla@alum.us.es](mailto:ignblabla@alum.us.es))
* María de la Salud Carrera Talaverón ([marcartal1@alum.us.es](mailto:marcartal1@alum.us.es))
* Joaquín González Ganfornina ([joagongan@alum.us.es](mailto:joagongan@alum.us.es))

Fecha: Sevilla, 8 de Julio 2024

Tabla de contenido

[Resumen del Informe 3](#_Toc171358911)

[Historial de Versiones 4](#_Toc171358912)

[Introducción 5](#_Toc171358913)

[Registro de análisis 6](#_Toc171358914)

[Requisito 201 6](#_Toc171358915)

[Descripción 6](#_Toc171358916)

[Análisis y decisiones 6](#_Toc171358917)

[Validación por el profesor 7](#_Toc171358918)

[Requisito 202 7](#_Toc171358919)

[Descripción 7](#_Toc171358920)

[Análisis y decisiones 7](#_Toc171358921)

[Validación por el profesor 8](#_Toc171358922)

[Requisito 204 9](#_Toc171358923)

[Descripción 9](#_Toc171358924)

[Análisis y decisiones 9](#_Toc171358925)

[Validación por el profesor 12](#_Toc171358926)

[Requisito 205 12](#_Toc171358927)

[Descripción 12](#_Toc171358928)

[Análisis y decisiones 12](#_Toc171358929)

[Validación por el profesor 16](#_Toc171358930)

[Requisito 206 16](#_Toc171358931)

[Descripción 16](#_Toc171358932)

[Análisis y decisiones 16](#_Toc171358933)

[Validación por el profesor 18](#_Toc171358934)

[Requisito 208 19](#_Toc171358935)

[Descripción 19](#_Toc171358936)

[Análisis y decisiones 19](#_Toc171358937)

[Validación por el profesor 22](#_Toc171358938)

[Conclusiones 23](#_Toc171358939)

[Bibliografía 24](#_Toc171358940)

# Resumen del Informe

El objetivo de este informe es proporcionar detalles sobre el análisis de todos los tipos de requisitos especificados en esta entrega del proyecto, así como las decisiones tomadas para llevarlos a cabo y enlazarlo, si procediera, a las validaciones y observaciones planteadas por el profesor encargado de la asignatura.

# Historial de Versiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Contenidos** | **Fecha** | **Contribuyente** |
| V1.0 | Versión inicial. Estructura y contenido del documento. | 16/02/2024 | Mª Salud Carrera Talaverón |
| V2.0 | Modificación de la plantilla para que coincida con el entregable actual. Introducción. Requisitos implementados. Bibliografía. | 01/07/2024 | Mª Salud Carrera Talaverón |
| V2.1 | Vistas de los requisitos 206 y 208 corregidas. | 02/07/2024 | Mª Salud Carrera Talaverón |
| V2.2 | Requisitos 204 y 205 descritos. | 06/07/2024 | Joaquín González Ganfornina |
| V2.3 | Requisito 203 descrito | 07/07/2024 | Ignacio Blanquero Blanco |

# Introducción

En esta entrega de la segunda convocatoria cada estudiante debe implementar aquellos requisitos que no fueron validados en la entrega final de la primera convocatoria. Estos requisitos incluyen arreglar la implementación de la entidad notice y de diversas funcionalidades, como las operaciones de cualquier usuario sobre claims, además de implementar funcionalidades que no se realizaran en la entrega de la primera convocatoria. Además, habrá que repetir aquellas pruebas que dejen de funcionar como resultado de estos cambios (o de otros introducidos por los estudiantes en los archivos de población de la base de datos) y volver a realizar el análisis de rendimiento del proyecto. También se incluyen diversos informes que no se entregasen en junio.

En este informe sólo se describirán aquellas tareas que requieran modificaciones en el código de las entidades grupales y no aquellas referentes a informes. La descripción de las pruebas realizadas se encuentra en el informe de testing grupal.

# Registro de análisis

Los requisitos se describirán en su totalidad y no únicamente los cambios añadidos en esta entrega. Sin embargo, se especificará de qué tratan estas modificaciones y por qué se han realizado de acuerdo a la corrección indicada por el profesor tras la entrega final de la primera convocatoria.

## Requisito 201

### Descripción

Una notificación es un mensaje publicado por un director autenticado. El sistema debe almacenar los siguientes datos sobre ellas: el momento de instanciación (en el pasado), un título (no en blanco, de menos de 76 caracteres), un autor (no en blanco, de menos de 76 caracteres), un mensaje (no en blanco, de menos de 101 caracteres), una dirección de correo electrónico opcional y un enlace opcional. El autor debe computarse de la siguiente manera: “〈nombre de usuario〉 - 〈apellido, nombre〉”, donde “〈nombre de usuario〉” denota el nombre de usuario del autor que ha publicado la notificación y “〈apellido, nombre〉” denota su nombre completo.

### Análisis y decisiones

Para implementar este requisito, se ha creado una clase Java que extiende a la clase AbstractEntity de Acme Framework. La clase, Notice.java, tiene las anotaciones “@Entity” de Javax y “@Getter” y “@Setter” de Lombok. Se ha definido en el paquete “acme/entities/notices”. No tiene relación con ninguna otra entidad.

Imagen que contiene Tabla

Descripción generada automáticamenteLos atributos definidos son: instantiationMoment, de tipo Date, que no puede ser nulo y debe estar en el pasado; title, de tipo String, que no puede estar en blanco y tiene como máximo 75 caracteres; message, de tipo String, que no puede estar en blanco y tiene como máximo 100 caracteres; email, de tipo String, que debe estar formado como una dirección de correo electrónico; link, de tipo String, que debe estar formado como un enlace; username, de tipo String, que no puede estar en blanco; fullName, de tipo String, que no puede estar en blanco y debe seguir un patrón “cadena, cadena”. Tanto username como fullName se utilizan para definir al autor de la notificación, que se trata de un atributo derivado. Esto se indica utilizando la anotación “@Transient” de Javax.

Texto

Descripción generada automáticamenteEl autor se crea cogiendo el atributo username, añadiéndole el separador “ – “ y concatenándole al final el atributo fullName. La forma en la que se definen tanto el nombre de usuario como el nombre completo se verán en el requisito que implementa la funcionalidad de creación de una notice, al igual que la restricción de que el atributo author no puede estar en blanco y debe tener como máximo 75 caracteres.

### Validación por el profesor

Tras la entrega final de la primera convocatoria, este requisito no fue validado por el profesor por el siguiente motivo: “Corrección: El modelo considera que el campo ‘author’ es un campo persistente, cuando claramente se indica que se trata de un valor calculado. Se comprueba en la implementación que tampoco se hace el cálculo del atributo tal y como se indica en los requisitos.”

Para corregir el requisito, se añadieron los atributos username y fullName, se eliminó el atributo persistente author y se definió este atributo de nuevo como uno derivado. Hubo que modificar el archivo CSV de población para que se ajustara a la nueva implementación.

## Requisito 202

### Descripción

Operaciones de cualquier usuario sobre reclamaciones:

* Enumerar las reclamaciones y mostrar sus detalles.
* Publicar una reclamación. Esta operación requiere confirmación porque las reclamaciones no se pueden actualizar ni eliminar.

### Análisis y decisiones

Para implementar todos los servicios de dicho requisito, se ha seguido la metodología discutida en clase. En consecuencia, se ha decidido crear un servicio para cada funcionalidad solicitada: listar, mostrar, crear y publicar una reclamación.

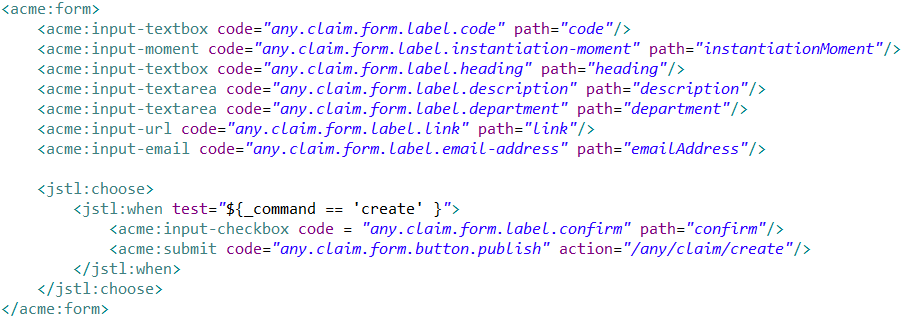
Al listar las reclamaciones, se ha considerado que no es necesario mostrar todos los atributos por razones de diseño. Por lo tanto, se ha decidido mostrar únicamente los atributos “code” y “heading”.

Al mostrar las reclamaciones, se exhibe la totalidad de la información contenida en dicha entidad, incluyendo así los atributos “code”, “instantiationMoment”, “heading”, “description”, “department”, “emailAddress” y “link”.

En relación al servicio de creación, en esta clase, dentro del método validate(), se ha verificado que el atributo “code” no estuviera publicado, y que, para la creación, se debe haber marcado el botón de confirmar.

Con el fin de crear una interfaz de usuario más sencilla e intuitiva, se ha decidido implementar un menú desplegable titulado ”Any”, situado en el AppNavBar, donde se encuentra la opción de Listar todas las reclamaciones.

A continuación, se muestra como ha sido implementado la casilla para confirmar la creación de la reclamación:



### Validación por el profesor

Este requisito no fue validado correctamente por el profesor en la primera convocatoria, por lo que ha sido implementado de nuevo para la segunda.

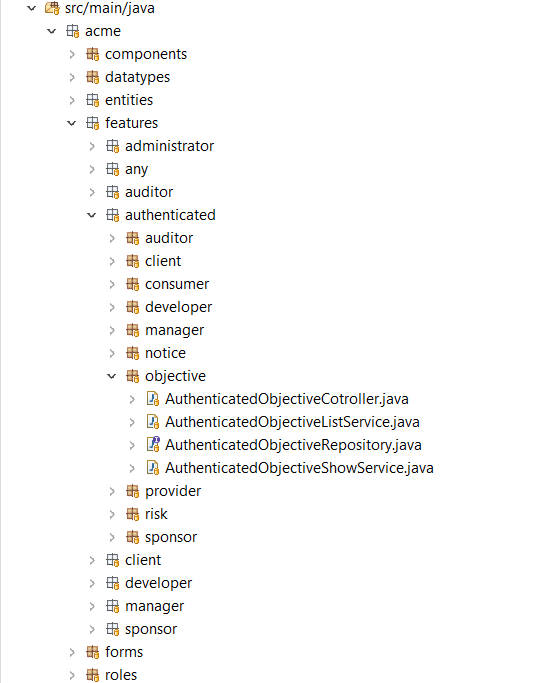
## Requisito 204

### Descripción

Un objetivo permite a un usuario autentificado definir una meta hacia la cual las acciones u operaciones de un proyecto son dirigidas. Los usuarios autentificados deben poder listar los objetivos y mostrar sus detalles.

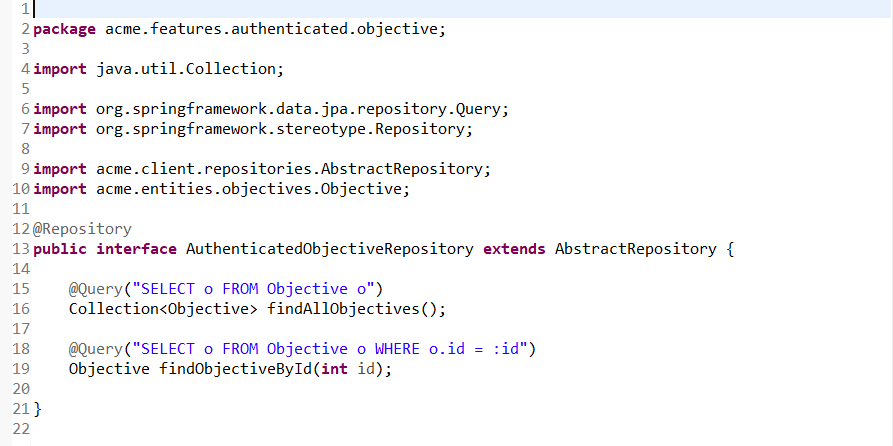
### Análisis y decisiones

Para completar la parte del backend de este requisito, se creó una nueva carpeta llamada “objective” en “src/main/java/acme/features/authenticated”. En esta carpeta se crearon el repositorio, el controlador y los servicios necesarios para satisfacer este requisito. En la siguiente imagen se puede apreciar con más detalle:



#### AuthenticatedObjectiveRepository.java

Para el repositorio, solo se necesitaron dos funciones, una para encontrar todos los objetivos existentes y otra para encontrar un objetivo dada un id.



#### AuthenticatedObjectiveListService.java

Para la autorización del servicio que permite listar los objetivos, solo es necesario que el usuario esté autentificado. Además, en este servicio se usa la función que permite encontrar todos los objetivos.



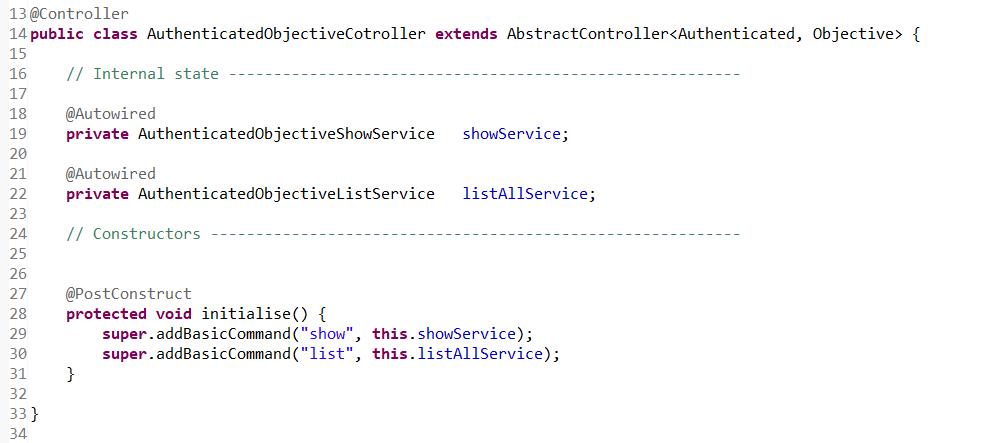
#### AuthenticatedObjectiveShowService.java

Para la autorización del servicio que permite mostrar los detalles de un objetivo, es necesario comprobar que dicho objetivo no sea nulo. Además, como uno de sus atributos es un tipo “enum”, se utiliza un SelectChoices en el método del unbind.

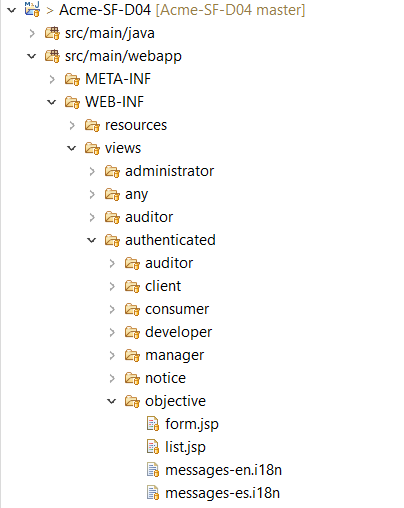


#### AuthenticatedObjectiveController.java

En el controlador, se añadieron los servicios de “show” y “list”.

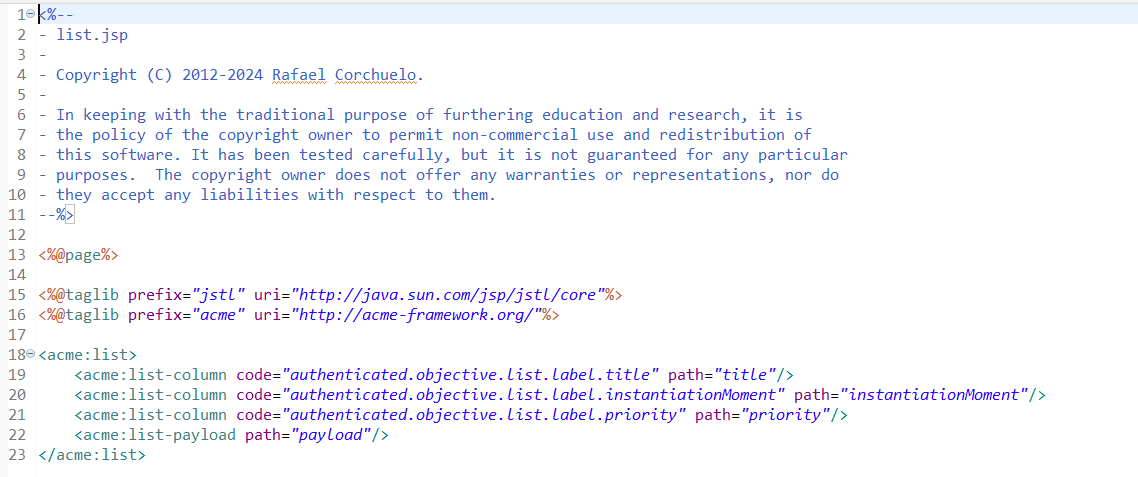


Para el frontend, se creó una carpeta llamada “objective” en “src/main/webapp/WEB-INF/views/authenticated”. En esta carpeta se crearon dos archivos “jsp”, uno para el listado de objetivos y otro para el formulario que se muestra al querer ver los detalles de un único objetivo. Además, se crearon dos archivos más, uno para mostrar los mensajes en español y otro para mostrar los mensajes en inglés.



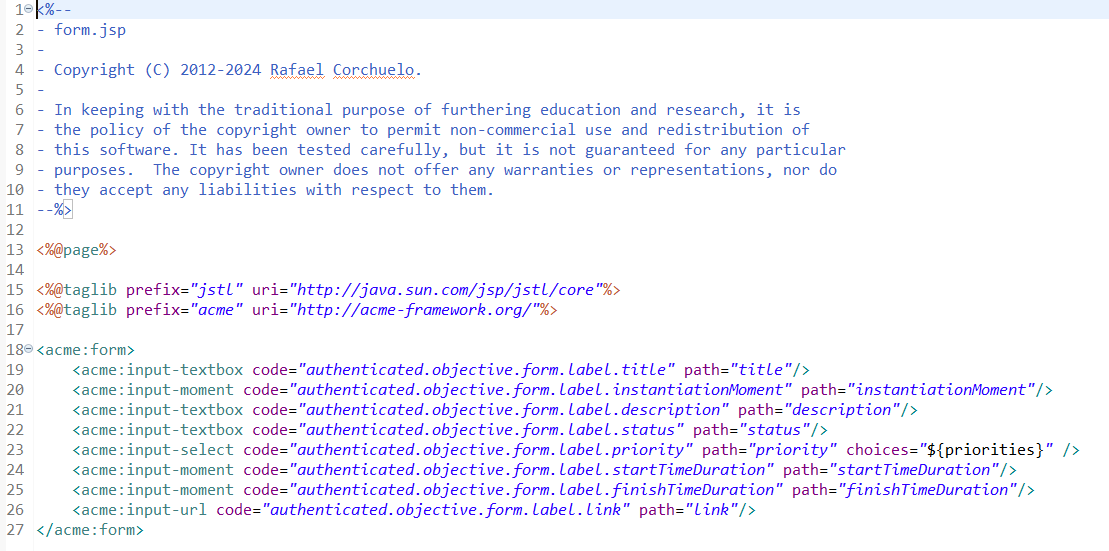
#### list.jsp

En el archivo del listado de objetivos, solo se muestran los siguientes atributos:



En el archivo que muestra los detalles de un único objetivo, se muestran todos los atributos de dicho objetivo, como se puede apreciar en la siguiente imagen:

#### form.jsp

Además, se modificaron los archivos ”src/main/webapp/WEB-INF/views/fragments/menu-en.i18n”, ” src/main/webapp/WEB-INF/views/fragments/menu-es.i18n” y ”src/main/webapp/WEB-INF/views/fragments/menu.jsp” para que se muestre en el menú un desplegable que permita a todos los usuarios autentificados listar los objetivos. Una vez se listan los objetivos, al hacer click en cualquiera de ellos se muestran sus detalles.

### Validación por el profesor

Este requisito no se completó para la entrega de la primera convocatoria, por lo que no fue validado por el profesor.

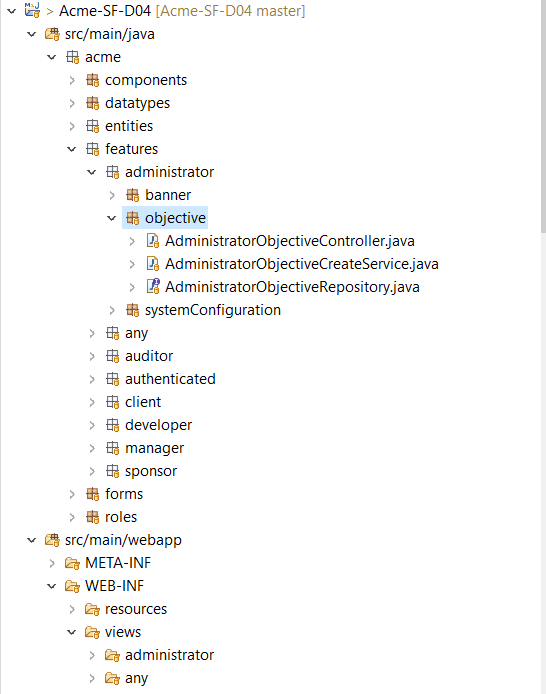
## Requisito 205

### Descripción

Un objetivo permite a un usuario autentificado definir una meta hacia la cual las acciones u operaciones de un proyecto son dirigidas. Los administradores pueden publicar un objetivo. Esta acción necesita una confirmación.

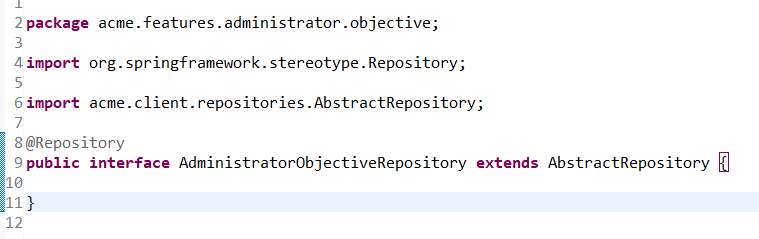
### Análisis y decisiones

Para completar la parte del backend de este requisito, se creó una nueva carpeta llamada “objective” en “src/main/java/acme/features/administrator”. En esta carpeta se crearon el repositorio, el controlador y el servicio necesario para satisfacer este requisito. En la siguiente imagen se puede apreciar con más detalle:



#### AdministratorObjectiveRepository.java

Para el repositorio, no hay ninguna función ya que las funciones que se necesitan para el servicio ya vienen implementadas en AbstractRepository.



#### AdministratorObjectiveCreateService.java

Para la autorización del servicio para crear los objetivos, se comprueba que el usuario que está intentando realizar la acción es un usuario del tipo administrador. En la siguiente imagen se puede ver esto, además de los métodos de load() y bind().

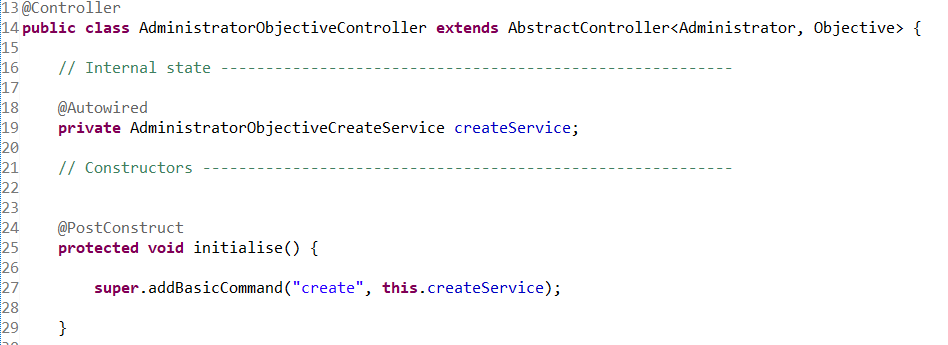


Hay que destacar la función validate(), la cual comprueba que todos los datos sean correctos, por ejemplo, la fecha de inicio de un periodo no puede ser posterior a la fecha final de dicho periodo. Además, en esta función se valida si se ha confirmado la acción de crear el objetivo. Para realizar dicha comprobación, se añade un atributo booleano en el dataset del unbind(), y en la función validate() salta una excepción siempre que dicho atributo sea falso.

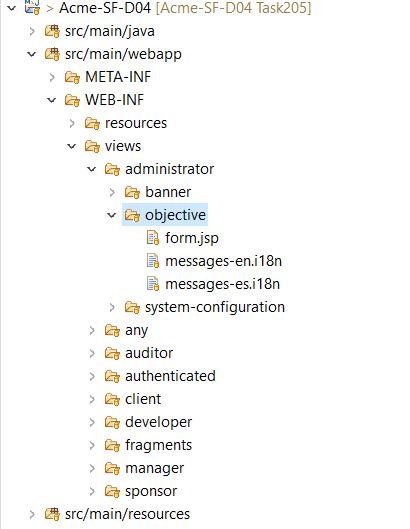


#### AdministratorObjectiveController.java

En el controlador solo se añade la función de crear un objetivo.

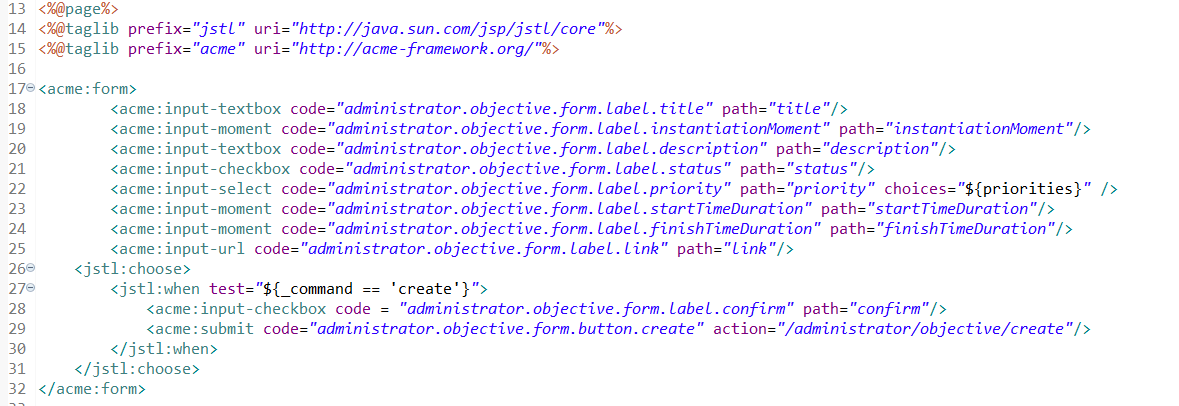


Para el frontend, se creó una carpeta llamade objective en “src/main/webapp/WEB-INF/views/administrator”. En esta carpeta se crearon un archivo “jsp”, para otro el formulario que se muestra crear el objetivo. Además, se crearon dos archivos más, uno para mostrar los mensajes en español y otro para mostrar los mensajes en inglés.



#### Form.jsp

En el archivo del formulario, se muestran todos los atributos de dicho objetivo. Además, se muestra un botón que es necesario pulsar para confirmar la acción y otro que al pulsarlo se crea el objetivo.

Además, se modificaron los archivos ”src/main/webapp/WEB-INF/views/fragments/menu-en.i18n”, ” src/main/webapp/WEB-INF/views/fragments/menu-es.i18n” y ”src/main/webapp/WEB-INF/views/fragments/menu.jsp” para que se muestre en el menú un desplegable que permita a todos los administradores crear un objetivo.

### Validación por el profesor

Este requisito no se completó para la entrega de la primera convocatoria, por lo que no fue validado por el profesor.

## Requisito 206

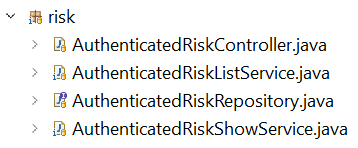
### Descripción

Implementar las operaciones de los usuarios autenticados sobre los riesgos, que incluyen:

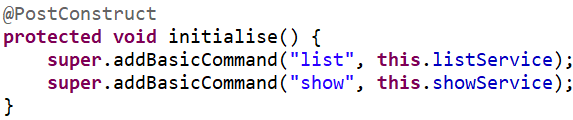
* Listar los riesgos que hay en el sistema.
* Mostrar los detalles de dichos riesgos.

### Análisis y decisiones

Para implementar este requisito, se ha creado un repositorio que define las consultas necesarias a la base de datos, un servicio por cada una de las acciones requeridas y un controlador que recoge todos los servicios implementados. Además, se han definido las vistas correspondientes y se han modificado los fragmentos necesarios para facilitar la navegación por estas funcionalidades. Se ha utilizado, y modificado ligeramente, la entidad Risk.java creada por Adrián Cabello Martín para la entrega de junio, cuando el grupo de trabajo era el C1.026. Los cambios añadidos han sido: modificar el tipo del atributo impact, que era un número entero mayor o igual que 1, para que sea un número real positivo (anotación “@Positive” de Javax). También, para aquellos atributos que estuvieran definidos con tipos envolventes y no pudieran ser nulos, se ha eliminado la anotación “@NotNull” y se ha cambiado su tipo al tipo primitivo correspondiente.

En esta imagen podemos ver los archivos creados en el paquete “acme/features/authenticated/risk”. El repositorio incluye la anotación “@Repository” de Spring Framework antes de la definición de la interfaz y extiende a la interfaz AbstractRepository de Acme Framework. Las consultas que define son: encontrar un riesgo por su ID y encontrar todos los riesgos existentes.

El controlador tiene la anotación “@Controller” de Spring Framework y extiende a la clase AbstractController de Acme Framework, especificando que va a tratar operaciones de usuarios tipo Authenticated sobre entidades tipo Risk. Recoge los servicios usando la anotación “@Autowired” de Spring Framework sobre la definición de cada uno de ellos, e implementa un constructor en el que especifica qué operación realiza cada servicio de la siguiente manera:

Cada servicio corresponde a un tipo de operación de lectura o escritura. Estas operaciones son “list” para leer todos los riesgos y “show” para leer un solo riesgo.

Todos los servicios utilizan la anotación “@Service” de Spring Framework y extienden a la clase AbstractService de Acme Framework, especificando que va a tratar operaciones de usuarios tipo Authenticated sobre objetos tipo Risk. Definen el repositorio utilizando la anotación “@Autowired” ya mencionada.

#### AuthenticatedRiskListService.java

El método “authorise()” controla si el usuario que intenta acceder a la URL de la funcionalidad está autorizado o no. En este caso, para listar los riesgos un usuario debe haber iniciado sesión.

El método “load()” carga los datos que se necesitan para realizar la operación. En este caso, se obtiene la colección de riesgos creados existentes en el sistema.

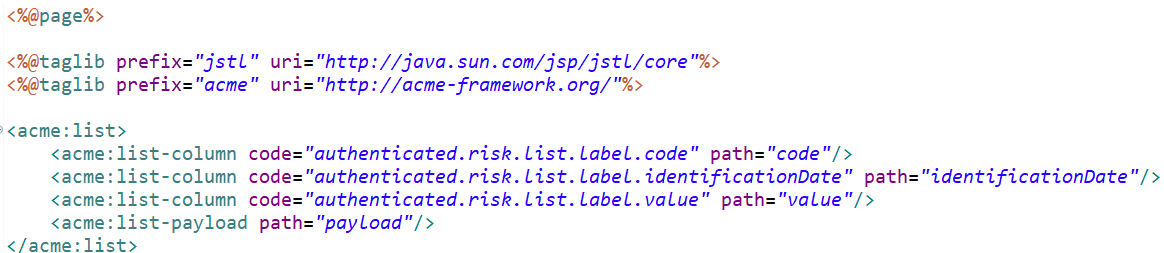
Por último, el método “unbind(final Risk object)” crea una tupla con los parámetros de la entidad que van a ser necesarios para mostrar la información en pantalla. En este caso, el listado de riesgos sólo va a mostrar su código, fecha de identificación y valor (que debe calcularse primero). Sin embargo, se han añadido los atributos restantes al payload para poder utilizarlos en la barra de búsqueda o filtrado de la página.

#### AuthenticatedRiskShowService.java

Este servicio es muy similar al anterior: sólo se puede mostrar la información de un riesgo si este existe y si se ha iniciado sesión; se cargan los datos mediante el ID del riesgo, que se extrae de la petición; se incluyen todos los parámetros de la entidad, excepto el impacto y la probabilidad, que se sustituyen por el valor derivado de ambos.

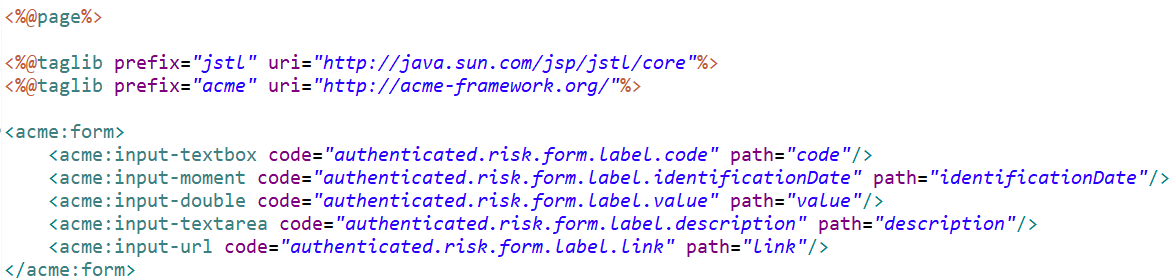
En cuanto a las vistas de la aplicación, los fragmentos creados han sido los siguientes:

#### risk/list.jsp



Este fragmento define una tabla con una columna por cada uno de los atributos especificados en el método unbind del servicio de listado. Cada fila de la tabla corresponderá con un risk distinto y, si se pulsa en ella, conducirá al formulario que muestra los detalles del riesgo. Se añade también el payload a la tabla.

#### risk/form.jsp



Este fragmento servirá para la funcionalidad de show. Hay un campo por cada atributo que se muestra del risk, especificando el tipo del mismo. Estos campos no se podrán modificar en ningún caso.

Los mensajes en español e inglés de estas dos vistas están definidos en sus correspondientes archivos.

### Validación por el profesor

Tras la entrega final de la primera convocatoria, este requisito no fue implementado, por lo que no existe validación por parte del profesor. Se ha desarrollado tomando como referencia las transparencias de la asignatura, así como implementaciones de otros requisitos parecidos que sí fueron validados por el profesor.

## Requisito 208

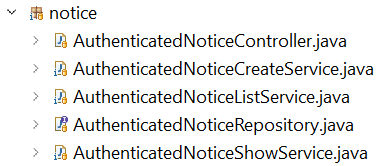
### Descripción

Implementar las operaciones de usuarios autenticados sobre las notificaciones, que incluyen:

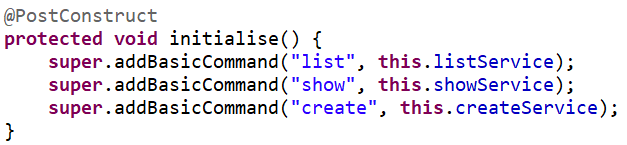
* Listar las notificaciones más recientes (del último mes) y mostrar sus detalles.
* Publicar una notificación. Esta operación requiere confirmación porque las notificaciones no se pueden borrar ni editar.

### Análisis y decisiones

Para implementar este requisito, se ha creado un repositorio que define las consultas necesarias a la base de datos, un servicio por cada una de las acciones requeridas y un controlador que recoge todos los servicios implementados. Además, se han definido las vistas correspondientes y se han modificado los fragmentos necesarios para facilitar la navegación por estas funcionalidades. Se ha utilizado la entidad Notice.java modificada en el requisito 201 y se han realizado los cambios necesarios en las funcionalidades implementadas por Natalia Olmo Villegas (cuando el grupo de trabajo era el C1.026) tras estas modificaciones en la entidad.

En esta imagen podemos ver los archivos creados en el paquete “acme/features/authenticated/notice”. El repositorio incluye la anotación “@Repository” de Spring Framework antes de la definición de la interfaz y extiende a la interfaz AbstractRepository de Acme Framework. Las consultas que define son: encontrar una noticia por su ID, encontrar todas las noticias posteriores a una determinada fecha y encontrar una cuenta de usuario por su username.

El controlador tiene la anotación “@Controller” de Spring Framework y extiende a la clase AbstractController de Acme Framework, especificando que va a tratar operaciones de usuarios tipo Authenticated sobre objetos tipo Notice. Recoge los servicios usando la anotación “@Autowired” de Spring Framework sobre la definición de cada uno de ellos, e implementa un constructor en el que especifica qué operación realiza cada servicio de la siguiente manera:

Cada servicio corresponde a un tipo de operación de lectura o escritura. Estas operaciones son “list” para leer el listado de notificaciones, “show” para leer una sola notificación y “create” para escribir una notificación nueva.

Todos los servicios utilizan la anotación “@Service” de Spring Framework y extienden a la clase AbstractService de Acme Framework, especificando que va a tratar operaciones de usuarios tipo Authenticated sobre entidades tipo Notice. Definen el repositorio utilizando la anotación “@Autowired” ya mencionada.

#### AuthenticatedNoticeListService.java

El método “authorise()” controla si el usuario que intenta acceder a la URL de la funcionalidad está autorizado o no. En este caso, para listar las notificaciones un usuario debe haber iniciado sesión.

El método “load()” carga los datos que se necesitan para realizar la operación. En este caso, se obtiene la colección de notificaciones que se han publicado en el último mes, obteniendo para ello la fecha correspondiente al momento presente y calculando la fecha un mes anterior.

Por último, el método “unbind(final Notice object)” crea una tupla con los parámetros de la entidad que van a ser necesarios para mostrar la información en pantalla. En este caso, el listado de riesgos sólo va a mostrar su fecha de instanciación, título y autor (que debe calcularse primero). Sin embargo, se han añadido los atributos restantes al payload (excepto username y fullName, porque ya van dentro de author) para poder utilizarlos en la barra de búsqueda o filtrado de la página.

#### AuthenticatedNoticeShowService.java

Este servicio es muy similar al anterior: sólo se puede mostrar la información de una notificación si esta existe y si se ha iniciado sesión; se cargan los datos mediante el ID de la notificación, que se extrae de la petición; se incluyen todos los parámetros de la entidad, excepto el nombre de usuario y el nombre completo, que se sustituyen por el autor derivado de ambos.

#### AuthenticatedNoticeCreateService.java

En este caso, sólo se puede acceder al formulario de creación de una notificación si el usuario se ha autenticado previamente. El método de carga creará una nueva notificación vacía, a la que asignará como instantiationDate el momento presente, como username el nombre de usuario del usuario autenticado y como fullName el nombre completo del usuario autenticado. El nombre completo se obtiene buscando en la base de datos la cuenta de usuario que tiene el username ya mencionado. La clase UserAccount dispone de un atributo de tipo DefaultUserIdentity, que a su vez tiene un atributo derivado fullName. El método “unbind” utilizará todos los parámetros del proyecto excepto los definidos en “load”. Además, incluirá un nuevo parámetro “accept” que será el que requiera la confirmación para poder crear la notificación.

Este servicio cuenta con otros métodos: “bind(final Notice object)”, “validate(final Notice object)” y “perform(final Notice object)”.

El método bind se encarga de procesar los datos que se envíen en la petición. En este caso, sólo aceptará valores para los atributos título, mensaje, email y enlace.

El método validate se encarga de validar los datos recibidos por el método bind. Aparte de comprobar las restricciones simples definidas en la clase Notice.java (como que el título debe tener menos de 75 caracteres y no puede estar vacío), se definen las restricciones más complejas, como que el campo computado author no puede estar vacío (en este caso debe tener más de 3 caracteres pues la cadena separadora tiene esa longitud) ni tener más de 75 caracteres o que el parámetro “accept” debe ser verdadero.

Por último, el método perform es el que realiza la acción del servicio si el método validate no ha fallado. En este caso, guarda en la base de datos el objeto de notificación recién creado.

En cuanto a las vistas de la aplicación, los fragmentos creados han sido los siguientes:

#### notice/list.jsp



Este fragmento define una tabla con una columna por cada uno de los atributos especificados en el método unbind del servicio de listado. Cada fila de la tabla corresponderá con una notificación distinta y, si se pulsa en ella, conducirá al formulario que muestra los detalles de la notificación. Se añade también el payload a la tabla. Al final de la página a la izquierda, aparecerá un botón que conducirá al formulario de creación de una notificación.

#### notice/form.jsp



Este fragmento servirá para las funcionalidades de show y create. Hay un campo por cada atributo que se muestra de la notificación. En el caso de la operación “show”, los campos mostrarán los valores correspondientes y no se podrán modificar; para la operación “create”, los campos del formulario aparecerán vacíos y bajo éste únicamente aparecerá un botón para enviar el formulario y crear la notificación.

Los mensajes en español e inglés de estas dos vistas están definidos en sus correspondientes archivos.

### Validación por el profesor

Tras la entrega final de la primera convocatoria, este requisito fue validado por el profesor (o, al menos, no existe indicación de lo contrario). Sin embargo, al haber modificado el modelo de la entidad Notice, ha habido que arreglar sus funcionalidades. Principalmente, los cambios implementados han sido hacer que las operaciones sean de usuarios autenticados sobre notificaciones, en lugar de realizarlas cualquier usuario, y la forma de almacenar el autor de la misma.

# Conclusiones

Tras este análisis, concluimos que la implementación de todos los requisitos no ha supuesto mayor complicación, dado que la corrección del profesor indicaba con claridad cuál había sido el error en la primera entrega. Analizando el resto de requisitos del estudiante 1 implementados, no se han encontrado otros fallos que corregir, por lo que los dos requisitos de código sin validar de la primera convocatoria han sido los únicos a abarcar en esta segunda convocatoria.

Consideramos exitoso el desarrollo de las tareas individuales de este entregable.

# Bibliografía

Transparencias de DP2 – L02, S01 – A foundation.

Transparencias de DP2 – L03, S01 – Display features.

Transparencias de DP2 – L03, S02 – Edition features.