# Decisiones de diseño

En esta sección describiremos las decisiones de diseño que se han tomado a lo largo del desarrollo de la aplicación que vayan más allá de la mera aplicación de patrones de diseño o arquitectónicos.

## Decisión 5: Recursividad en citas de comentarios

### Descripción del problema:

En nuestra aplicación, los comentarios de un hilo tienen la característica de citar o responder a otro. La idea era que el comentario que responde o cita al original apareciese justo debajo, con una marca que representase que es una cita o respuesta. La cuestión es que esa cita o respuesta es también un comentario, y puede ser citado o respondido de igual forma. Esto quiere decir que al mostrar la lista de comentarios, se debían mostrar los comentarios y sus respuestas de forma recursiva.

### Alternativas de solución evaluadas:

Alternativa 5.a: Importar librerías de jsp que implementasen la función recursiva.

**Ventajas**:

* Es una solución eficaz al problema, en teoría sencilla.

**Inconvenientes**:

* Requeriría que nos familiarizáramos con una librería que no habíamos usado antes.
* Había escasa información de la librería en línea.

Alternativa 5.b: Hacer uso de javascript para hacer llamadas recursivas.

**Ventajas**:

* Solucionaría el problema si se implementase bien.
* En nuestra experiencia, Javascript tiende a dar muy buen manejo de muchas situaciones.

**Inconvenientes**:

* No había garantía de que esta solución fuese a funcionar.
* No habíamos visto esta alternativa entre las múltiples propuestas que hay en foros en internet, por lo que seguramente habría algún inconveniente más.

Alternativa 5.c: Hacer llamadas a un mismo archivo jsp cambiando los valores del model para distinguir las distintas llamadas, emulando un efecto de recursividad.

**Ventajas**:

* Solución simple y eficaz al problema.
* Es la que más se recomienda en internet.

**Inconvenientes**:

* Algunas de las características que tendría una función recursiva en un lenguaje que admita esta funcionalidad, por ejemplo, Java, se pierden, dando lugar a menor manejo de la situación de recursividad.

### Justificación de la solución adoptada

Seleccionamos la alternativa 5.c, porque era la más recomendada en internet (de hecho, en internet con una rápida búsqueda se puede copiar la estructura básica de la recursividad siguiendo este método) y rápidamente entendimos cómo se realizaba la recursividad sin necesidad de adaptarnos a una nueva librería o a aventurarnos a usar más lenguajes de programación de forma innecesaria (javascript).

## Decisión 6: Mostrar número de notificaciones del usuario autenticado a tiempo real.

### Descripción del problema:

Nuestra aplicación da soporte a un sistema de notificaciones. Para que este sea útil, vimos necesario que un usuario registrado pudiera ver el número de notificaciones que tiene en cualquier momento de la página, y que este se actualizase de forma constante. Por ello, decidimos mostrar este número en el header de la página, cuyo código se encuentra en el archivo menu.tag. El problema vino al intentar cambiar este número a tiempo real.

### Alternativas de solución evaluadas:

Alternativa 6.a: Hacer uso de anotaciones de Spring que ejecutasen un método java cada cierto tiempo.

**Ventajas**:

* Solución simple que soporta nuestro sistema Spring.

**Inconvenientes**:

* Como prácticamente cualquier funcionalidad de Spring, el principal inconveniente es que no es fácil qué está ocurriendo detrás de cada anotación, y tampoco se puede editar ese código para adaptarlo a nuestra situación.
* El código no se ejecutaba en el orden que teníamos pensado, y daba lugar a bastantes errores.
* Por algún motivo, hacer uso de esta anotación impedía que funcionasen los Junit tests.

Alternativa 6.b: Hacer uso de javascript y ajax para hacer llamadas a una función java a tiempo real.

**Ventajas**:

* Solucionaría el problema de forma eficaz.
* El uso de javascript para cambios a tiempo real en la página sin necesidad de recargar la página es lo más recomendado.

**Inconvenientes**:

* No había garantía de que esta solución fuese a funcionar.
* No sabíamos si podría provocar conflictos con Spring.

Alternativa 6.c: Crear un atributo en la clase del usuario que fuese el número de notificaciones, y hacer una llamada a este atributo en el código de menu.tag, de igual forma que se coge el nombre de usuario del usuario autenticado.

**Ventajas**:

* Solución sencilla.

**Inconvenientes**:

* No habría forma de hacer que se actualizase a tiempo real a menos que el usuario recargase la página.
* Tendría que haber un intercambio de información entre las creaciones de las notificaciones y el número que representa el número de notificaciones del usuario.

### Justificación de la solución adoptada

Probamos las tres soluciones, empezando por 6.a, y terminando por 6.b, que es la que finalmente elegimos. 6.a quedó rechazada al dar problemas con las pruebas unitarias. La alternativa 6.c tampoco nos valía porque no se podía hacer una llamada a cualquier atributo del usuario con tanta facilidad como pensábamos que se podría. La solución 6.b es la última que se nos ocurrió, y fue sencillo de implementar y no dio ningún problema.

## Decisión 7: Niveles en los comentarios.

### Descripción del problema:

Relacionada con la decisión 5, este problema vuelve a tener que ver con comentarios y citas. En este caso, el problema estaba en cómo distinguir en qué nivel del árbol se encontraba cada comentario (donde la raíz del árbol sería el comentario que no citaba a ninguno, las ramas los comentarios que no han sido citados, y los demás nodos los comentarios que citan a otro comentario y que luego son citados por otro).

### Alternativas de solución evaluadas:

Alternativa 7.a: En la llamada recursiva, aumentar en uno el nivel de un comentario por cada llamada.

**Ventajas**:

* Solución simple y lógica. Prácticamente igual que como hemos programado siempre las funciones recursivas.

**Inconvenientes**:

* Debido a que la alternativa escogida en la decisión 5 fue la de hacer llamadas recursivas a un mismo archivo jsp, muchas funcionalidades lógicas de una función recursiva no funcionaban como uno se esperaría, dificultando en gran medida esta alternativa.

Alternativa 7.b: Crear un atributo en comentario llamado nivel, que le es asignado en su creación en función de si cita o no a un comentario, y en función del nivel del comentario que esté citando, si es que cita a algún comentario.

**Ventajas**:

* Solucionaría el problema sin tener que tocar la función recursiva.

**Inconvenientes**:

* No es la respuesta más lógica al problema.
* Hay que añadir atributos que en principio no serían necesarios ya que son derivados de otros atributos de la clase (la clase comentario tiene un atributo que representa una relación de cita entre comentarios, a través del cual se podría deducir el nivel del comentario sin necesidad de hacer un atributo nuevo).

### Justificación de la solución adoptada

En primer lugar, elegimos aplicar la alternativa 7.a, pero el inconveniente de esa alternativa no nos permitió conseguir los resultados que queríamos. La alternativa 7.b fue un poco más laboriosa de lo que hubiera sido la 7.a (si no hubiese sido por su inconveniente), pero funcionó bien, así que fue la que finalmente escogimos.

## Decisión 8: Implementación de los distintos tipos de usuario.

### Descripción del problema:

Nuestra aplicación hace uso de distintos roles para los usuarios. Estos roles permiten acceder a determinadas partes de la página e incluso a ciertas funcionalidades. La implementación de esta funcionalidad se podía hacer de distintas formas.

### Alternativas de solución evaluadas:

Alternativa 8.a: Crear un atributo en la clase del usuario que determinase su rol.

**Ventajas**:

* Solución simple, de fácil implementación y eficaz.

**Inconvenientes**:

* No podrían aplicarse operaciones demasiado complejas a los distintos privilegios que otorga cada rol.

Alternativa 8.b: Crear una clase para cada atributo, y crear una relación entre la clase usuario y esas clases.

**Ventajas**:

* Solucionaría el problema y permitiría hacer operaciones más complejas para el comportamiento de cada rol.

**Inconvenientes**:

* La implementación no es trivial.

Alternativa 8.c: Hacer uso de la clase Authorities ya existente en el proyecto petclinic, que además ya tiene una relación con User (que a su vez tiene una relación con nuestra clase Usuario).

**Ventajas**:

* Solución más sencilla que la 6.b y no mucho más compleja que la 6.a.
* Permite hacer uso de ciertas funcionalidades que Spring tiene reservadas para esta clase, que está estrechamente ligada al sistema de seguridad de Spring.
* Hay soporte en línea para los distintos problemas que pudieran surgir.

**Inconvenientes**:

* Podría haber comportamiento inesperado si no supiéramos manejar bien las funcionalidades de Spring.
* No hay garantía de que pudiéramos implementar operaciones de mayor complejidad para los permisos de los distintos roles.

### Justificación de la solución adoptada

La solución 8.c fue la que elegimos, porque no era demasiado difícil de implementar y podíamos buscar soluciones en línea para los posibles problemas que pudiese dar. Además, creíamos que podíamos hacer uso de ciertas funcionalidades de Spring para cambiar la vista con facilidad en función del rol del usuario, pero al final cuando intentamos usar estas funcionalidades no nos sirvieron, pero encontramos una forma alternativa de hacer lo mismo.