# Decisiones de diseño bloque 2

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 1: Añadir preguntas a un examen

### Descripción del problema:

Nos planteamos la adición de preguntas a un examen en la página. El problema reside en que no podemos saber de antemano el número de preguntas que tendrá ni de qué tipo serán, ya que no hay una respuesta fija.

### Alternativas de solución evaluadas:

Alternativa 1.a: Añadir las preguntas en la página de creación o edición del examen.

En la página de creación/edición del examen se colocaría un botón de añadir pregunta. Mediante Javascript se irían añadiendo secciones al formulario y se pasarán a la base de datos desde un único formulario.

**Ventajas**:

* Es la forma más rápida si se quieren añadir varias preguntas.
* Se realizan menos accesos a la base de datos.

**Inconvenientes**:

* Encontramos, el procedimiento, complejo de implementar, ya que no tenemos todos los conocimientos necesarios de las tecnologías requeridas para llevarlo a cabo.
* Puede resultar confuso para la navegabilidad del usuario medio que va a utilizar la aplicación.

Alternativa 1.b: Crear una sección enfocada exclusivamente a las preguntas del examen, en las que se puedan añadir y editar las preguntas de una en una.

**Ventajas:**

* Para un usuario promedio, al estar dividido en secciones más específicas, puede ser más sencillo de comprender.
* La implementación es sencilla.
* Permite editar una pregunta sin tener que modificar el resto del examen.
* Se pueden añadir preguntas de distinto tipo a un examen más fácilmente.

**Inconvenientes:**

* Múltiples accesos a la base de datos.
* Puede ser lento si se desean añadir muchas preguntas.

### Justificación de la solución adoptada

Alternativa 1b: Creemos que gran parte del público estará conformada por personas que no están acostumbradas a trabajar con sistemas de información de uso avanzado. Por tanto, prima que el funcionamiento de nuestra aplicación sea sencillo. Además, debido al plazo del que disponemos, nos hemos decantado por la solución que podíamos conocer mejor a primera vista.

## Decisión 2: Añadir opciones a una pregunta

### Descripción del problema:

El problema es similar que surge al añadir preguntas a un examen. Queremos añadir opciones a una pregunta. El problema reside en que no tienen un número fijo de opciones.

### Alternativas de solución evaluadas:

Alternativa 1.a: Añadir las opciones en el momento en el que se crea o edita una pregunta.

En el formulario de creación de pregunta, habrá un botón “Añadir opción” que, cada vez que sea pulsado, añadirá de manera asíncrona un formulario para una nueva opción.

**Ventajas**:

* Es la forma más rápida si se quieren añadir varias opciones.
* Se realizan menos accesos a la base de datos.

**Inconvenientes**:

* Al igual que con la decisión de añadir preguntas a exámenes, encontramos, el procedimiento, complejo de implementar. Presenta dificultades similares a la anterior decisión.
* Puede resultar confuso para la navegabilidad del usuario medio que va a utilizar la aplicación.

Alternativa 1.b: En la página donde se listan las preguntas del examen, al lado de cada pregunta, se muestra el botón “Añadir opción”. Este botón llevará a un formulario exclusivo para esa opción.

**Ventajas:**

* Para un usuario promedio, puede ser un procedimiento fácil de entender.
* La implementación es sencilla.
* Permite editar una opción sin alterar el resto de preguntas, de manera que es menos probable modificar otras opciones o preguntas por error.
* Es una solución más coherente a la ya decidida para añadir preguntas al examen.

**Inconvenientes:**

* Múltiples accesos a la base de datos.
* Puede ser lento si se desean añadir muchas opciones.

### Justificación de la solución adoptada

Alternativa 1b: Creemos que gran parte del público estará conformada por personas que no están acostumbradas a trabajar con sistemas de información de uso avanzado. Por tanto, prima que el funcionamiento de nuestra aplicación sea sencillo. Además, debido al plazo del que disponemos, nos hemos decantado por la solución que podíamos conocer mejor a primera vista.

## Decisión 3: Realización de examen

### Descripción del problema:

Nos planeamos el problema de, si tenemos un examen con sus correspondientes preguntas, cómo se puede realizar este, de forma que sea cómodo para el usuario y quede constancia de lo que respondió.

### Alternativas de solución evaluadas:

Alternativa 1.a: Poner todas las preguntas en una misma página.

Las preguntas se dispondrían en una única página y a la base de datos, una vez pulsado un botón “enviar respuesta”, se le pasa una lista con las respuestas seleccionadas.

**Ventajas**:

* Se realizan menos accesos a la base de datos.

**Inconvenientes**:

* La dificultad, a nivel programático, aumentaría a la hora de gestionar las respuestas.
* Ante un error de conexión, como el envío no se realiza hasta que no se finaliza el examen, se podrían perder todas las respuestas que el usuario había dado hasta el momento.

Alternativa 1.b: Mostrar las preguntas de una en una, cada una en una página diferente. En la base de datos, las respuestas se irían añadiendo al intento de una en una.

**Ventajas:**

* La implementación es más sencilla que la anterior alternativa.
* El usuario es más consciente del número de pregunta que ya ha respondido y no tiene que hacer scroll para ver todas las preguntas.
* Ante un error de conexión, las preguntas que ya se habían contestado estarán almacenadas en la base de datos y, por tanto, no desaparecerán las respuestas.

**Inconvenientes:**

* Múltiples accesos a la base de datos.

### Justificación de la solución adoptada

Alternativa 1b: Hemos seleccionado esta opción principalmente porque nos facilita el tratamiento de datos y porque disminuye las posibilidades de pérdida de información. Además, creemos que, de esta manera, podemos hacer la vista del usuario más amigable que mediante la opción 1.a.

## Decisión 4: Tipos de pregunta ampliables

### Descripción del problema:

A pesar de que principalmente se plantean preguntas de tipo test o de redactar una corta respuesta, nuestra cliente pide que se puedan añadir otro tipo de preguntas en el futuro.

### Alternativas de solución evaluadas:

Alternativa 1.a: Poner un atributo a las preguntas de tipo enum, que podrá tomar los valores de los distintos tipos de preguntas existentes en la aplicación.

**Ventajas**:

* Se puede conocer de manera mucho más rápida el tipo de la pregunta tan sólo comprobando el valor del enum.

**Inconvenientes**:

* En la clase pregunta habría que poner numerosos atributos para que pudiera ajustarse a todos los tipos de pregunta, no usándose la mayoría y reservando más espacios de la cuenta que estarán a null.
* Para añadir nuevos tipos de pregunta, hay que añadir todos los atributos distintos de esta nueva pregunta a la clase, teniendo que modificar mayor cantidad de código.
* Hay que modificar la lógica que siguen todas las preguntas cada vez que se quiera añadir un nuevo tipo de pregunta.

Alternativa 1.b: Hacer una jerarquía de clases de tipos de pregunta con una única clase, “TipoTest”. De esta forma, será sencillo añadir otro tipo de pregunta. Por defecto, se entiende que, si la pregunta no tiene ningún tipo de la jerarquía asociado, es de redacción.

**Ventajas:**

* Para añadir un nuevo tipo de pregunta, tan sólo hay que añadir una nueva clase a la jerarquía de tipos de pregunta con los atributos exclusivos de su tipo.
* Es más eficiente, ya que la lógica que hay que implementar es más simple.

**Inconvenientes:**

* Hay más relaciones intermedias y esto puede ser más complejo la hora de implementar nuevas funcionalidades.

### Justificación de la solución adoptada

Alternativa 1b: Hemos seleccionado esta opción porque presenta más facilidades a la hora de ampliar los tipos de pregunta disponibles y resulta más ventajosa que la primera alternativa.

## Decisiones diseño (no bloque 2)

## Decisión X: Borrado de usuarios

### Descripción del problema:

Queremos que un usuario pueda cancelar su cuenta en la página cuando lo considere necesario sin que esto cause problemas al resto de las entidades.

### Alternativas de solución evaluadas:

Alternativa 1.a: Eliminar el usuario de la base de datos

**Ventajas**:

* Se ahorra espacio en la base de datos, de manera que no hay sobrecarga de información innecesaria.
* Es fácil de implementar, ya que no sigue ninguna lógica de negocio compleja.

**Inconvenientes**:

* Un usuario en la base de datos tiene muchas dependencias, lo cual puede causar muchos problemas durante el borrado.
* Borrar un usuario implica que se pueda borrar información útil de otras entidades.
* Hay que gestionar el flujo de eliminación para que no se alteren otras partes de la aplicación.

Alternativa 1.b: Añadir un atributo al usuario que indique si tiene la cuenta habilitada para su uso.

**Ventajas:**

* Si un usuario decide eliminar su cuenta, podría retomarla en un futuro en el mismo estado en el que la deshabilitó.
* Permite mantener coherencia con las dependencias del usuario, de forma que no se pierde información.
* Aunque se deshabilite la cuenta, se puede aprovechar la información para aplicar algoritmos estadísticos para sacar conclusiones que ayuden a tomar decisiones a la organización, por ejemplo, el uso de big data.
* Puede ayudar, aparte de a hacer cuentas invisibles, a penalizar a usuarios por comportamiento indebido.

**Inconvenientes:**

* Mayor carga de información en la base de datos.
* Mayor complejidad de implementación debido a la lógica de negocio compleja.

### Justificación de la solución adoptada

Alternativa 1b: Hemos seleccionado esta opción porque hay una gran diferencia entre la cantidad de ventajas e inconvenientes que ofrece con respecto a la otra alternativa.

## Decisión X1: Validaciones

### Descripción del problema:

Queremos validar que los datos de los formularios contienen campos correctos para la creación o edición de nuevas entidades. Teniendo en cuenta que existen varias estrategias para hacerlo, el problema reside en cuál de ellas escoger.

### Alternativas de solución evaluadas:

Alternativa 1.a: Únicamente a través de anotaciones definidas por Spring.

**Ventajas**:

* Soporta validaciones sintácticas
* Requiere de poca implementación, ya que vienen preconfiguradas por Spring.

**Inconvenientes**:

* No soportan validaciones semánticas, es decir, no podríamos implementar reglas de negocio complejas.

Alternativa 1.b: Crear una clase que realice la validación de los distintos campos y, además, utilizar las anotaciones predefinidas de Spring.

**Ventajas:**

* Soporta validaciones tanto semánticas como sintácticas.
* Es fácil de implementar con Spring.
* Podemos utilizar los métodos del BindingResult
* Favorece la cohesión de la aplicación

**Inconvenientes:**

* Un controller sólo puede tener asociado una clase validación, por tanto habría que planear previamente cómo validar todos los campos sin se utilizan en varias entidades a la vez.
* Hay que incluir de manera manual la clase validación de una entidad en cada una de las clases en las que se use esta.

Alternativa 1.c: Validación a través de excepciones personalizadas.

**Ventajas:**

* Soporta validaciones tanto semánticas como sintácticas.
* Lanzar excepciones en lenguajes orientados a objetos, en este caso Java, es fácil y común.
* Facilita el poder proporcionarle al usuario el motivo detallado de los fallos de las validaciones.

**Inconvenientes:**

* Puede complicar los códigos, ya que el tratamiento de las excepciones se debe intercalar en el código donde requiere utilizar la validación.
* No puede haber excepciones simultáneas, por lo que podrían quedarse campos sin validar.
* Ejecutar diferentes métodos sin comprobar excepciones puede dar lugar a inconsistencias en la base de datos.

### Justificación de la solución adoptada

Alternativa 1b: En general, hemos seleccionado esta opción a la hora de validar porque hemos decidido dar peso a la cohesión en nuestra aplicación y porque es sencilla de implementar.

Sin embargo, debido a la poca complejidad de algunas clases (como PDF), en ocasiones hemos usado la primera opción.