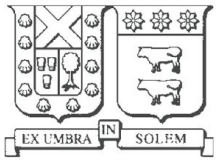
Departamento de Informática
Universidad Técnica Federico Santa María



Entregable IV

Proyecto: Kolbfinder

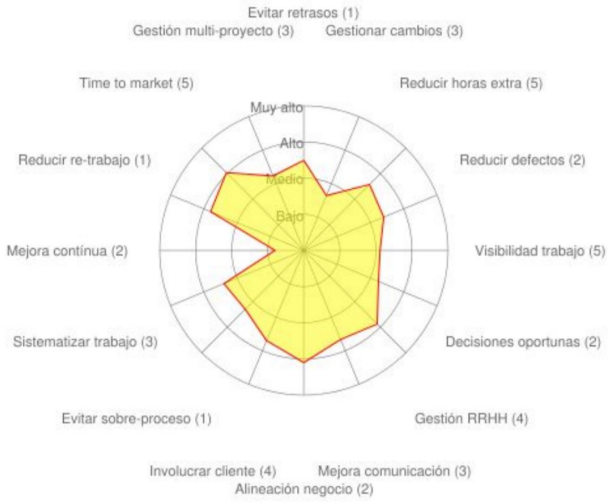
Integrantes:

Nombres y Apellidos	Email	ROL USM
Felipe Avaria	felipe.avaria@alumnos.usm.cl	2923547-3
Sebastián Torrico	sebastian.torrico.12@sansano.usm.cl	201330061-8
Andrés Cifuentes	andres.cifuentesv@alumnos.usm.cl	201004652-4

Post-Mortem Metodológico

Evaluacion por Areas :

Evaluación por Objetivos:

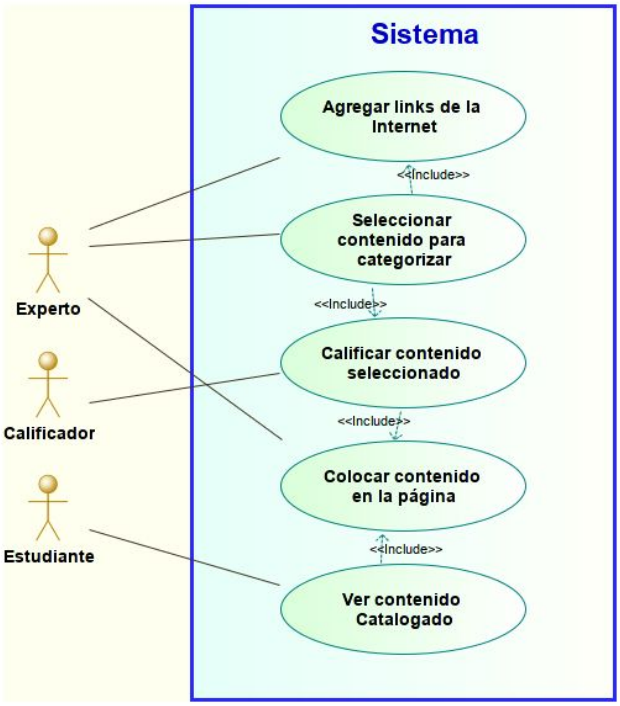


Analizando los resultados entregados por la encuesta, podemos ver que nuestras fortalezas están relacionadas con el trabajo en equipo, el espacio de trabajo, el liderazgo y el producto, lo que refleja nuestra intención de llevar a cabo el trabajo en un grato ambiente centrándonos en nuestro equipo como los principales gestores de los resultados.

Podemos ver también nuestras principales debilidades en los ejes de procesos, cliente y reuniones, estas pueden provenir principalmente de situaciones de interrupciones presentadas dentro del marco temporal del proyecto, como 'el paro' y la poca disponibilidad y presencia del cliente en las presentaciones.

Como equipo debemos enfocarnos en mejorar la comunicación con el cliente y la definición de reuniones para plantear aspectos a mejorar y dar mayor énfasis a objetivos como mejora continua y la reducción de horas extra.

Diagrama de Casos de Uso



Se mantuvo el diagrama de casos de uso inicial, ya que no se vio la necesidad de agregar más casos de uso que los presentados anteriormente.

Patrones de diseño y Frameworks

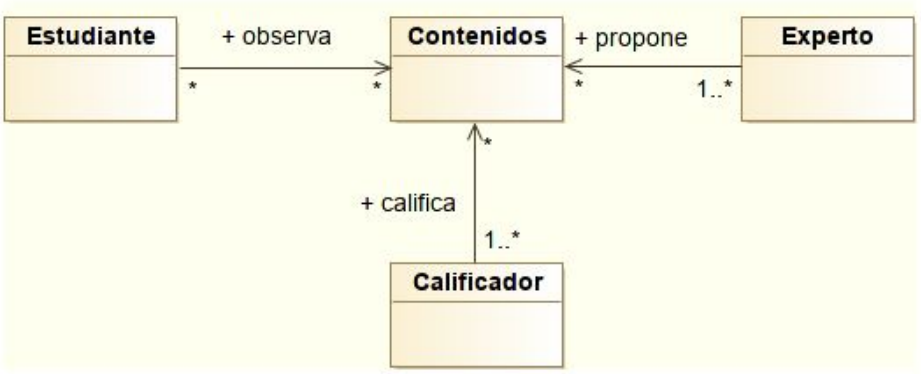
Intención	Patrón de Diseño	Razonamiento
Utilizar objetos a lo largo del proyecto, donde una clase necesite datos de otra clase, pero con cierta forma y estructura específica, para cada sección.	Builder (Adonis Js)	Para la creación de una aplicación, se tiene que utilizar diversos objetos que extienden a clases. En este caso, se utiliza Builder. Esto creando a partir de lo que se tiene en la base de datos, y de lo que necesita el usuario, la implementación del objeto necesaria para cumplir

		con las respectivas partes de la aplicación.
Los usuarios, utilizan los objetos. Pero este no observa los atributos internos del objeto. Simplemente, interactúan con estos, y hace selección de si utilizar o no aquel.	Factory (Adonis Js)	Se debe bajar la complejidad de la aplicación, por lo que el uso de extensión de objetos, debe ser una práctica común para reducir la complejidad. Con factory, se permite la creación de un objeto, con atributos que se entregan en su creación, logrando así, el uso práctico de la clase.
Separar el uso de la Base de Datos, respecto a usarlo en el código. Esto, por qué usar directamente, puede producir cierto problemas.	Repository (Adonis Js)	Se utiliza diversas clases en el proyecto, para poder implementar comunicación con la base de datos. Para ello, se utiliza los modelos de Adonis, los cuales permiten que estos tengan métodos de clase, para poder comunicarse con la base de datos, sin tener que usar Querys de SQL en forma natural.
La aplicación no debe ser complicada de utilizar, y de la menor forma posible al programar el código. Por lo que se requiere modularizar funciones, o reducirlas a uso de una clase.	Facade (Adonis Js)	Facade, permite que a través de utilizar un solo objeto, se puede interactuar con una estructura de objeto mucho más compleja, y así lograr modularizar código en diversas secciones del software.
Se desea crear una estructura en el Diagrama de Clases, separando los datos de la aplicación, la lógica de funcionamiento y la interfaz de usuario.	MVC (Adonis Js)	Las clases que poseen datos importantes como el usuario y los contenidos, deben ser representados por un modelo que trabaje con ellos, tanto como para acceder a la información y actualizar su estado.

		Los datos deben ser visualizados en una interfaz de usuario a través de clases que contienen el código para ello. Estas permitirán mostrar la salida. El controlador se encargará en enlazar el modelo y la vista para responder a las acciones que se solicitan en el sistema, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno. El patrón MVC cumple exactamente con lo anterior, ya que este es un estilo de arquitectura de software que separa los componentes en la estructura deseada (Modelo-Vista-Controlador).
En la View, o interfaz del cliente, se requiere que sus interacciones (a través de clicks), se vean observados los cambios que este realiza.	Observer (Vue.Js)	Vue Js, permite crear una interfaz dinámica en el explorador Web, dónde propiedades de objetos desplegados en la página, se pueden cambiar dinámicamente a través de acciones del usuario. En este caso, es utilizado para seleccionar material que calificador desea agregar, o que experto quiera colocar en contenido público.
Se busca mostrar en el Diagrama de Clases la transformación del objeto contenido, propuesto por	Adapter (Implementación propia en Javascript)	El objeto creado a partir de la clase experto contiene información que solo ella maneja y es poco

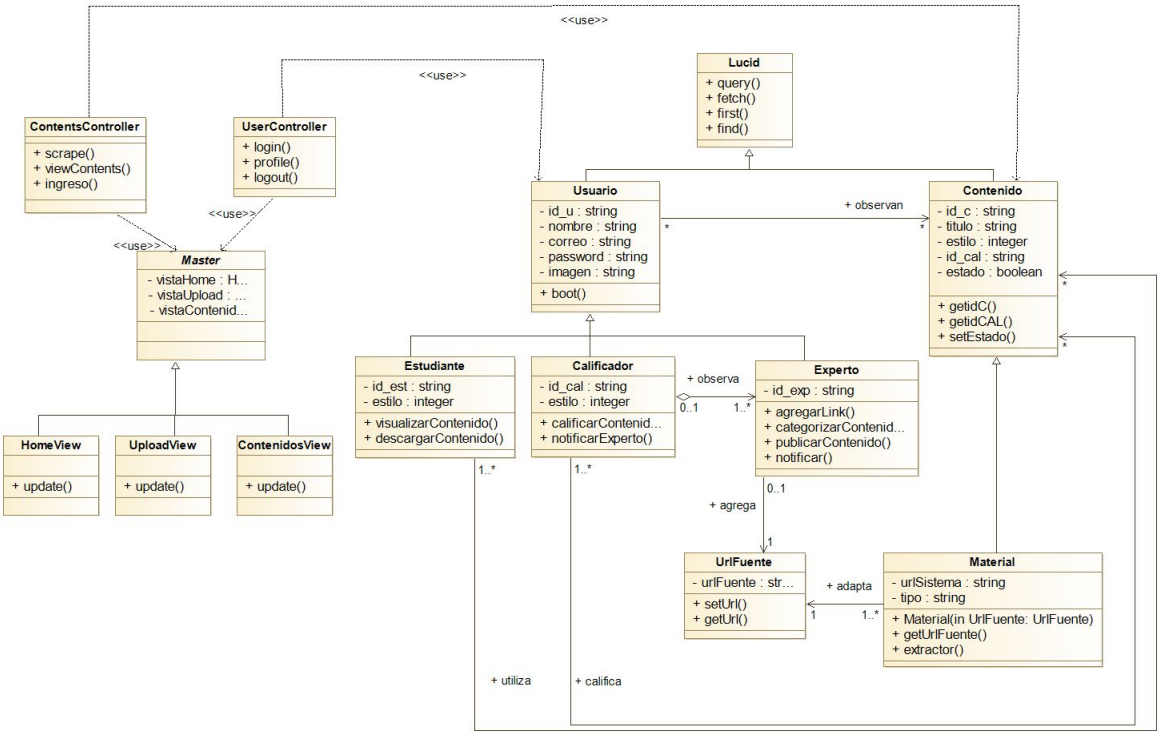
el experto, a una nueva clase que pueda ser reconocida por todos los demás usuarios.		precisa, ya que es una página web general. Por tanto, debe ser adaptada a una clase que atomice el contenido de ella, separandola en atributos de información (como el título o su tipo) y que contenga específicamente lo que se desea mostrar, ya sea una imagen, documento o video. El patrón Adapter transforma una interfaz en otra, de modo que una clase que no pueda utilizar la original, lo haga a través de una adaptación. Esto cumple con la necesidad involucrada en el Sistema.

Modelo de Dominio



Se mantuvo el modelo de dominio, ya que es suficientemente representativo y no se realizaron cambios significativos en el enfoque.

Diagrama de Clases



No se modificó el diagrama de clases, ya que fue lo suficientemente representativo como para llevar a cabo el proyecto, muchas de las entidades son parte del framework y no es necesario cambiarlo.

Pruebas de Software (actualización)

Defecto encontrado	Mitigación	Resultado obtenido	Observaciones
Login sin detalles y restricciones.	Cambio total del login: se modificó el diseño por uno con más espacios, se crearon funciones para mostrar casos de autentificación incorrectos y se añaden los mensajes de alerta.	Login funcional y estandarizado. En caso de ingresar mal los datos, el sistema muestra un mensaje de error. En caso contrario; se accede al sistema.	Por problemas con la vista, fue necesario una reconstrucción total del login. Sin embargo, se utilizaron las mismas funciones para la autentificación.

Íconos sin funcionalidad.	Dependiendo del ícono, se decidió si había que agregarle una función o eliminarlo.	El 30% de los íconos revisados fueron conservados y se le añadieron funciones. El resto; eliminados.	Los íconos eran parte del template implementado, por tanto, habían algunos que no tenían aplicaciones para las funcionalidades del sistema o eran muy complicados para implementar alguna.
Sin confirmaciones/errores al realizar algún CRUD.	Se añadieron mensajes de confirmación o alertas para aquellas funcionalidades que involucran a la base de datos. Si bien, la acción se realizaba, no mostraba una alerta diciendo que se había agregado algo, por tanto, simplemente se agregó una línea para imprimir luego de finalizar la función. Algunas requerían ciertas condiciones para aparecer.	Al agregar nuevos usuarios como administrador, el sistema puede responder con distintos mensajes: 1. "Usuario añadido correctamente". 2. "Correo ya existente". 3. "Nombre de Usuario ya existente". Los mensajes mostrados anteriormente son ejemplos de las alertas.	Implementar una alerta no es complejo, sin embargo, al no conocer muy bien el framework costó tiempo encontrar una forma para mostrar los mensajes ya que no sigue la forma tradicional como sería AngularJS o un simple "print".. En este caso, existe un arreglo general de errores y este se muestra en la vista dependiendo del caso.
Al seleccionar contenido no se muestra su cambio de estado (seleccionado o descartado). Esto se debe a un defecto del framework.	Se investigó sobre una funcionalidad que tiene el framework para solucionar este tipo de problema (reactividad) y hacer interactivas las acciones del usuario.	Al hacer click, se cambian funciones del código a lo que la documentación del framework señala.	Al usar un framework, se debe asegurar que tenga una amplia documentación y comunidad para resolver estas situaciones.