Índice

ET.01. Problema Primer Trabajo Teórico

Grupo de Trabajo	
Componentes	1.Pablo Jimenez Parada 2. Alejandro Muñoz Gonzalez 3. Daniel Sanchez Rodriguez 4. George Felician Soldubanu 5. Jun Hao Lin 6. Alejandro Rodriguez Saludador

Fecha		15/11/2024
Titulo:	Trabajo Teórico	

PROCESO UNIFICADO DE DESAROLLO

Análisis de Requisitos

El equipo capturo un total de 9 requisitos, ambos pertenecientes tanto a cliente como a servidor:

1) Login
2) Gestión de Metadatos
3) Gestión de Proyectos
4) Acceso a metadatos
5) Administrar permisos de Accesos
6) Realizar Evaluación
7) Realizar reportes
8) Solicitar informe de calidad
9) Proponer Datos

Para estos requisitos hemos definido una aproximación 1:1
1 requisito -> 1 caso de uso -> 1 iteración.

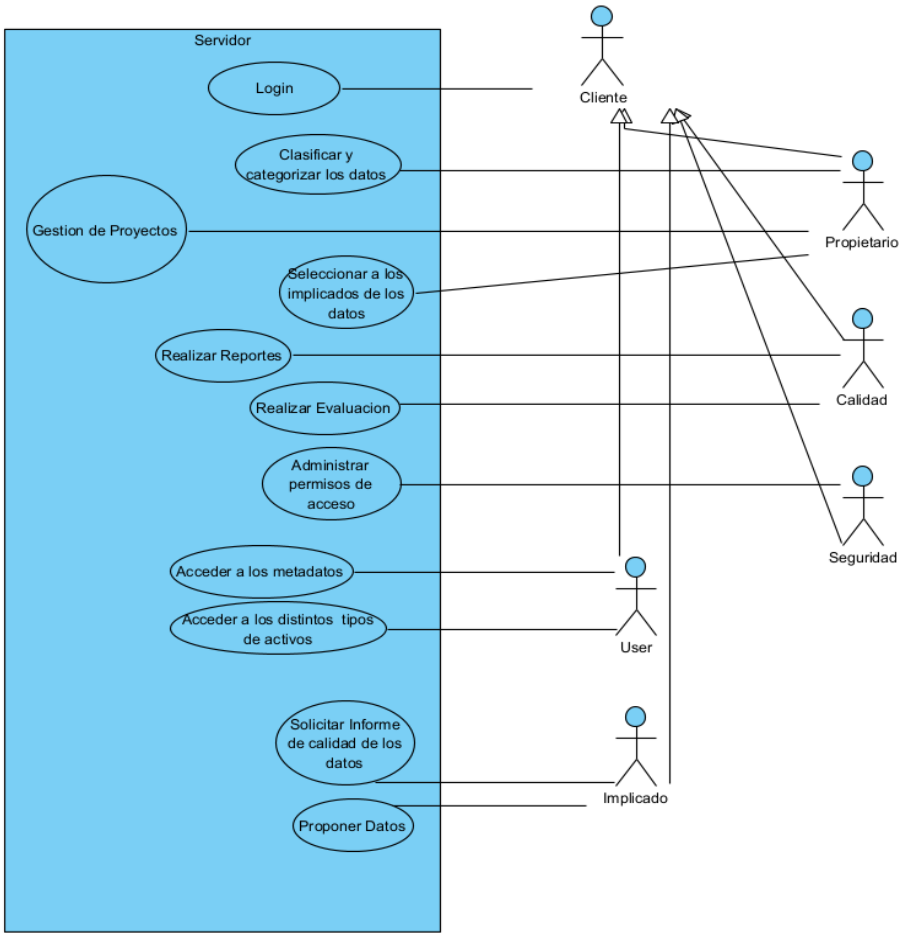
Además, hemos asignado una prioridad para cada uno de los casos de uso:

Rf	CDU	Prioridad	Iteración	Función
RF1	CDU 1	1	1	Login
RF2	CD2	1	2	Gestión de metadatos
RF3	CDU3	1	3	Gestión de proyectos
RF4	CDU 4	2	4	Acceso a metadatos
RF5	CDU 5	3	5	Administrar permisos de accesos
RF6	CDU 6	3	6	Realizar evaluación
RF7	CDU 7	4	7	Realizar reportes
RF8	CDU 8	4	8	Solicitar informe de calidad
RF9	CDU 9	5	9	Proponer Datos

CASOS DE USO

Tras analizar el problema y analizar los requisitos funcionales, hemos definido un total de cinco actores, los cuales todos sufren herencia de un actor general que denominaremos “Cliente”, cada uno de estos actores contara con unos casos de uso específicos para él.

Para los casos de uso tampoco encontrar algún tipo de relación entre ellos.



Nuestro caso en específico cuenta con 9 iteraciones más una "iteración 0" y una final, este desarrollo se realizará mediante un desarrollo en paralelo cliente-servidor. Además, también definimos que la jornada laboral seria de 8h diarios durante 5 días semanales. Es importante indicar que tenemos un personal compuesto por 6 personas. De estas 6: 2 son especialistas en requisitos, que pueden hacer las veces de analistas, 2 son diseñadores, que pueden hacer las veces de implementadores y 2 testers.
Por último, el inicio de desarrollo se marcó el día 4/11/2024.

RQ#	CDU	PRIORIDAD	REQUISITOS	ANALISIS	DISEÑO	IMPLEMENTACION	PRUEBAS
RF1	CDU.1.	1	3	3	7	18	5
RF2	CDU .2.	2	4	4	10	19	5
RF3	CDU .3.	2	3	4	8	24	9
RF4	CDU .4.	3	3	4	9	26	6
RF5	CDU .5.	3	3	6	8	13	6
RF6	CDU .6.	4	3	2	3	9	2
RF7	CDU .7.	4	3	4	6	13	5
RF8	CDU .8.	4	2	2	3	7	1
RF9	CDU .9.	5	1	2	3	3	1

Tabla 1: Casos de uso, prioridad y esfuerzo estimado por cada tarea

Planificación en Paralelo

Fase	Inicio	Elaboración	Construcción	Transición
Iteración	It0	It1	[it2, it9]	F
Coste	1000€	1630€	11840€	3000€
Duración	40h	36h	284h	120h
Agenda	5dias(28oct-1nov)	5dias(4Nov-8Nov)	35dias(11Nov-2Enero)	15dias(3Enero-23Enero)

Tabla 2: Tabla con planificación en paralelo

RRHH	GF1					GF2					GF3				
	R	A	D	I	P	R	A	D	I	P	R	A	D	I	P
R1	2	1				2	2				1	2			
R2	1	2				2	2				2	2			
R3			3	9				5	10				4	12	
R4			4	9				5	9				4	12	
R5					3					2					4
R6					2					3					5
Total Horas	36					42					48				
Total Coste	1630					2030					2040				

RRHH	GF4					GF5					GF6				
	R	A	D	I	P	R	A	D	I	P	R	A	D	I	P
R1	2	2				1	3				1	1			
R2	1	2				2	3				2	1			
R3			5	13				4	7				1	4	
R4			4	13				4	6				2	5	
R5					3					3					1
R6					3					3					1
Total Horas	48					35					19				
Total Coste	2100					1830					970				

RRHH	GF7					GF8					GF9				
	R	A	D	I	P	R	A	D	I	P	R	A	D	I	P
R1	1	2				1	1				1	1			
R2	2	2				1	1					1			
R3			3	6				1	3				2	1	
R4			3	7				2	4				1	2	
R5					2										1
R6					3					1					
Total Horas	31					15					10				
Total Coste	1510					790					570				

Agenda

MES	SEMANA	LUN	MAR	MIER	JUE	VIER
11	1	4	5	6	7	8
11	2	11	12	13	14	15
11	3	18	19	20	21	22
11	4	25	26	27	28	29
12	5	2	3	4	5	6
12	6	9	10	11	12	13
12	7	16	17	18	19	20
12	8	23	24	25	26	27
12	9	30	31			
1	9			1	2	3
1	10	6	7	8	9	10
1	11	13	14	15	16	17

Si se siguiese sin complicaciones la agenda, el desarrollo finalizaría el día 2/01/2024.

Esta planificación es la más óptima para este desarrollo, nos permite trabajar en varias funciones a la vez, funciones que son equivalentes en ambos apartados (cliente y servidor).

De todas formas, esta planificación no es perfecta pues hay desperdicio de horas en ciertas partes del desarrollo.

CONTROL DE CALIDAD

Cuando hablamos de la calidad del software, tenemos que ver con la norma ISO 25000. De una manera más concreta estaremos utilizando de las divisiones, ISO 250Xn que habla sobre el modelo de calidad de los productos de software.

Adecuación funcional

En esta característica se incluyen:

Integridad funcional

Podemos alcanzar esta calidad asegurando que las acciones que puede realizar el programa se corresponden con los requisitos funcionales.

Corrección funcional

Para asegurar esto tenemos que demostrar que cada característica que ofrece el sistema se realiza correctamente

Fiabilidad

La Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados sin interrupciones o fallos.
Nosotros controlamos la fiabilidad de esta manera:

Capacidad de recuperación

capacidad para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallo. Para ello nosotros tenemos una base de datos secundaria que es una copia de la original, esta hace la función de backup que remplazaria a la original en caso de ocurriese un problema.

Tolerancia a fallos

Capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software. Nuestra aplicación implementa sentencias try-catch que capturan las posibles excepciones o situaciones de error y lanzan un mensaje de error discreto que, una vez cerrado, permite continuar con la operación normal de la aplicación.

Seguridad

Capacidad de protección de la información y los datos de manera que las personas u otros productos tengan el grado de acceso a los datos adecuado a sus tipos y niveles de autorización, y para defenderse de los patrones de ataque de agentes malintencionados.

No repudio

Capacidad de demostrar las acciones o eventos que han tenido lugar, de manera que dichas acciones o eventos no puedan ser repudiados posteriormente. Para ello usaremos una de las formas mas comunes para capturar esto, el uso de firmas digitales. Con esto creamos un vinculo único entre el autor y el documento

o mensaje. La firma digital está vinculada tanto al documento como a la clave privada del firmante. Si el documento se modifica en cualquier forma después de haber sido firmado, el hash del documento cambiará y la firma ya no será válida. Esto asegura que el firmante no solo no puede negar haber firmado, sino que también garantiza que el contenido no ha sido alterado después de la firma.

Compatibilidad

a compatibilidad es la capacidad de compartir información entre varios sistemas y realizar sus funciones. Gracias a utilizar una base de datos basada en mysql en la nube, otras aplicaciones que necesiten acceder a los datos generados por la aplicación podrán acceder sin problema bajo previa autorización por nuestra parte, incluso podrían añadir datos nuevos.

Mantenibilidad

Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.

Modularidad

Capacidad de un producto para evitar que los cambios en un componente afecten a otros componentes. Para lograr esta característica utilizamos orientación a objetos y modelado multicapa, además repartimos el programa en módulos y submódulos que se realizaran entre si, de forma que podremos realizar variaciones en el código sin preocupación a un cambio no esperado.

Eficacia de desempeño

Esta característica representa el desempeño de un producto en la realización de sus funciones dentro de unos parámetros de tiempo y rendimiento especificados y con un uso eficiente de recursos (CPU, memoria, almacenamiento, energía...) utilizados bajo determinadas condiciones.

Comportamiento temporal

Grado en que un producto realiza sus funciones de forma que el tiempo de respuesta y el ratio de rendimiento cumple los requisitos especificados. Para cumplir esta característica se realizo una gestión de optimización de código mediante pruebas y actualizaciones pertinentes en caso de código mal optimizado.

Utilización de recursos

Grado en que la cantidad y tipos de recursos utilizados por el producto al llevar a cabo su función bajo condiciones determinadas no exceden lo especificado. La utilización de recursos es el número de medios usados cuando se ejecuta el software en unas condiciones específicas. No se ha detectado anomalías de rendimiento en ninguna de las pruebas, todos los equipos hasta los que menos memoria tenían han ejecutado la aplicación sin agotar los recursos.

Capacidad de uso

La usabilidad es la capacidad en la que un usuario interactúa de forma eficiente con el software. Para ello hemos citado a dos usuarios que no han codificado la aplicación para que la prueben y nos den feedback.

Reconocibilidad

La reconocibilidad es la capacidad por la que un usuario decide si el software cubre sus necesidades. En esta característica nos encontramos trabajando ya que aún no hay suficiente feedback por parte de los usuarios para saber si realmente las necesidades están cubiertas de forma amena y amigable para el usuario.

GESTIÓN DE LA CONFIGURACIÓN

Arquitectura

Para este proyecto propondremos una arquitectura cliente-servidor. Esta arquitectura permite centralizar la gestión de los datos y metadatos en un único servidor, mientras que diferentes tipos de usuarios acceden a través de un cliente web que adaptaremos en función de su rol (propietario de datos, implicado en los datos, gestor de seguridad, gestor de calidad de datos y usuarios de los metadatos).

Branching en la Herramienta Github

Respecto al uso de Github usaremos un esquema simple, Se crearán dos Ramas main, una para el cliente y otra para el servidor, en ambas ramas es donde se realizará el merge una vez acabada cada iteración, cada una de estas será una subrama de su respectivo Main (ya sea cliente o servidor), estas subramas contendrán únicamente su iteración. Tanto las subramas como la Main pueden sufrir variaciones en sus versiones, para el control de estas versiones seguiremos la semántica **Major.Minor. Patch.** Usaremos la sección *Major* para incluir cambios que introduzcan nuevas funcionalidades o rompan la compatibilidad con versiones anteriores del sistema. La sección *Minor* se utilizará para añadir mejoras o cambios que mantengan la compatibilidad. Finalmente, la sección *Patch* se destinará exclusivamente a la corrección de errores.

Los merges de las ramas con el main se asignaron así:

1. 27/11/2024 → Fin de GF3- Pull Request Main (Alpha) (Contiene [GF1-GF3])
2. 16/12/2024 → Fin de GF6- Pull Request Main (Beta) (Contiene [GF1-GF6])
3. 2/01/2025 → Fin de GF9- Pull Request Main (Final) (Contiene [GF1-GF9])

Es importante indicar que en cada uno de estos pull la documentación también queda actualizada, estas GF se añadirán al main en su versión final, una vez realizado su testing correspondiente. Además, cada una de las subramas de iteración se completará dentro de sus respectivas fechas marcadas.

VERSIONADO

En este proyecto vamos a realizar una única iteración, en la cual vamos a tener cuatro versiones:

-V0.0.0: Es la versión inicial del proyecto, en la cual definimos los requisitos y desarrollamos, en un plazo de 17 días, tres grupos funcionales.

-V1.0.0: Es la versión alfa, la cual no tiene toda la funcionalidad implementada. En esta versión también desarrollamos tres grupos funcionales, esta vez en un plazo de 14 días.

-V2.0.0: Es la versión beta, ya con la mayoría de la funcionalidad implementada, que sirve para hacer las pruebas necesarias para comprobar si el proyecto funciona correctamente. En esta versión, desarrollamos tres grupos funcionales y empezamos otros dos en un plazo de 11 días.

-V3.0.0: Es la versión final del proyecto, en la que terminamos los dos grupos funcionales que empezamos en la versión beta. Una vez terminados, es la versión del proyecto que entregamos al cliente.

Gestion de Realeses

En la liberación del sistema llevaremos a cabo una **release mayor** cuando introduzcamos una nueva versión completa del software, que incluirá funcionalidades y mejoras significativas.

En caso de detectar errores críticos después de una release mayor, gestionaremos estos problemas a través de una **release menor**, utilizando la rama Hotfix. Esta liberación menor servirá para aplicar un parche que corrija fallos específicos sin alterar las funcionalidades principales del sistema.

En la liberación del sistema realizaremos una liberación del Software masivo, y por defecto una release Mayor siendo esta una nueva versión del producto. En caso de encontrar fallos que se resolverían en la rama Hotfix tendremos una release menor para corregir el sistema con un parche.