Informe de Análisis

Integrantes del grupo C1.026:

* Ignacio Blanquero Blanco ([ignblabla@alum.us.es](mailto:ignblabla@alum.us.es))
* Adrián Cabello Martín ([adrcabmar@alum.us.es](mailto:adrcabmar@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))
* Natalia Olmo Villegas ([natolmvil@alum.us.es](mailto:natolmvil@alum.us.es))

Fecha: Sevilla, 8 de Marzo 2024

Tabla de contenido

[Resumen del Informe 4](#_Toc160301892)

[Historial de Versiones 5](#_Toc160301893)

[Introducción 6](#_Toc160301894)

[Registro de análisis 7](#_Toc160301895)

[Requisito suplementario 1 7](#_Toc160301896)

[Descripción 7](#_Toc160301897)

[Análisis y decisiones 7](#_Toc160301898)

[Validación por el profesor 8](#_Toc160301899)

[Requisito obligatorio 1 8](#_Toc160301900)

[Descripción 8](#_Toc160301901)

[Análisis y decisiones 8](#_Toc160301902)

[Validación por el profesor 9](#_Toc160301903)

[Requisito obligatorio 2 9](#_Toc160301904)

[Descripción 9](#_Toc160301905)

[Análisis y decisiones 10](#_Toc160301906)

[Validación por el profesor 10](#_Toc160301907)

[Requisito obligatorio 3 11](#_Toc160301908)

[Descripción 11](#_Toc160301909)

[Análisis y decisiones 11](#_Toc160301910)

[Validación por el profesor 12](#_Toc160301911)

[Requisito obligatorio 4 12](#_Toc160301912)

[Descripción 12](#_Toc160301913)

[Análisis y decisiones 12](#_Toc160301914)

[Validación por el profesor 12](#_Toc160301915)

[Requisito suplementario 2 13](#_Toc160301916)

[Descripción 13](#_Toc160301917)

[Análisis y decisiones 13](#_Toc160301918)

[Validación por el profesor 13](#_Toc160301919)

[Requisito suplementario 3 13](#_Toc160301920)

[Descripción 13](#_Toc160301921)

[Análisis y decisiones 13](#_Toc160301922)

[Validación por el profesor 13](#_Toc160301923)

[Requisito suplementario 4 13](#_Toc160301924)

[Descripción 13](#_Toc160301925)

[Análisis y decisiones 13](#_Toc160301926)

[Validación por el profesor 14](#_Toc160301927)

[Conclusiones 15](#_Toc160301928)

[Bibliografía 16](#_Toc160301929)

# Resumen del Informe

El objetivo de este informe es proporcionar detalles sobre el análisis de todos los tipos de requisitos especificados en esta primera 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 obligatorios y suplementarios implementados. Bibliografía. | 02/03/2024 | Mª Salud Carrera Talaverón |
|  |  |  |  |

# Introducción

En esta segunda entrega cada estudiante debe implementar obligatoriamente dos entidades y un formulario además de crear datos de prueba para un nuevo rol. Entre las tareas suplementarias se encuentran la de implementar un nuevo rol (el mismo para el que hay que crear los datos de prueba), la de generar un diagrama UML del modelo de dominio y la producción de dos nuevos informes.

A pesar de que la tarea de implementar un rol nuevo es suplementaria, por lo que debería dársele menor prioridad que a las tareas obligatorias, se ha decidido implementar esta la primera debido a que el resto de los objetos tenían relaciones directamente con dicho rol. A continuación, se describirá este proceso con más detalle.

# Registro de análisis

Los requisitos se describirán en el orden de implementación y no necesariamente en el que se presentan en el documento de requisitos del estudiante 1; es decir, si el primer requisito que se ha implementado ha sido uno suplementario por las razones comentadas anteriormente, ese es el que se comentará primero, a pesar de ser de menor prioridad o de estar solicitado posteriormente en el documento de requisitos.

## Requisito suplementario 1

### Descripción

Implementar un nuevo rol abstracto para el proyecto, llamado manager (gerente en español). El perfil de un gerente debe contar con la siguiente información: grado o carrera (no puede estar en blanco ni ocupar más de 75 caracteres), una vista general o descripción (no en blanco, 100 caracteres o menos), una lista de certificaciones (no en blanco, 100 caracteres o menos) y opcionalmente un enlace con más información.

### Análisis y decisiones

Tras haber atendido a las lecciones de clase, y haber consultado en caso necesario las transparencias correspondientes, se implementó el rol manager de la siguiente manera:

Tabla

Descripción generada automáticamenteVemos que la clase Java se encuentra en el paquete acme.roles y que extiende a la clase AbstractRole heredada de Acme Framework.

Tiene una serie de anotaciones que indican que se trata de una entidad y que se deben generar automáticamente los métodos para consultar y modificar todos los atributos de la clase.

El primer atributo no tiene ninguna función; sin embargo, se debe implementar igualmente debido a necesidades de versiones antiguas de Java.

A continuación, están implementados los atributos especificados en el enunciado del requisito: degree, overview, certifications y link. Todos tienen las restricciones pedidas, implementadas gracias a las anotaciones correspondientes. El atributo certifications, además, se ha implementado como una cadena de texto en lugar de una lista de cadenas de texto debido a la restricción de no poder ocupar más de 100 caracteres.

Por haberlo descrito en este primer requisito y al tener la misma explicación, no se va a volver a comentar la existencia del atributo serialVersionUID ni las anotaciones @Entity, @Getter y @Setter cuando estas aparezcan posteriormente en otras implementaciones.

### Validación por el profesor

El profesor dio el visto bueno a esta implementación el día 21/02/2024, y pidió que se eliminases otras clases creadas para este requisito (repositorio, servicio y controladores), debido a que no se esperaba que fueran implementadas para este entregable.

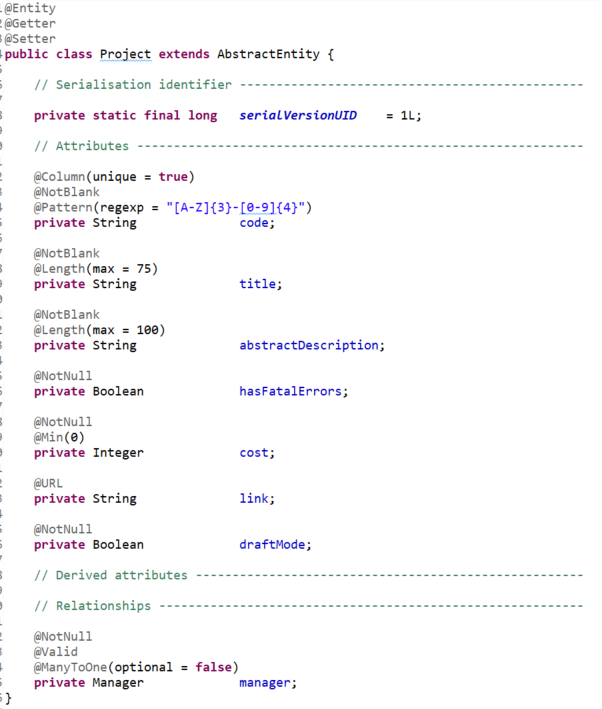
## Requisito obligatorio 1

### Descripción

Implementar una entidad llamada project (proyecto en español). Un proyecto es una agregación de historias de usuario creadas por un mismo manager. Un proyecto debe contar con la siguiente información: un código único (no en blanco, debe seguir el patrón “[A-Z]{3}-[0-9]{4}”), un título (no en blanco, 75 caracteres o menos), una descripción abstracta (no en blanco, 100 caracteres o menos), una indicación de si tiene errores fatales, un coste (positivo o cero) y opcionalmente un enlace con más información. Si el proyecto tiene errores fatales, debe ser rechazado por el sistema.

### Análisis y decisiones

Tras haber atendido a las lecciones de clase, y haber consultado en caso necesario las transparencias correspondientes, se implementó la entidad project de la siguiente manera:

Vemos que la clase Java se encuentra en el paquete acme.entities.projects y que extiende a la clase AbstractEntity heredada de Acme Framework.

A continuación, están implementados los atributos especificados en el enunciado del requisito: code, title, abstractDescription (que debía llamarse sólo abstract, pero dado que se trata de una palabra reservada en Java se ha decidido añadirle la palabra Description a continuación), hasFatalErrors, cost y link. Todos tienen las restricciones pedidas, implementadas gracias a las anotaciones correspondientes. Aquellos atributos que no estuvieran explícitamente mencionados como opcionales se han implementado como obligatorios. Se ha decidido que el atributo cost sea un número entero en lugar de uno decimal, análogamente al atributo estimatedCost de las historias de usuario (requisito siguiente); no obstante, se cambiará en el futuro en caso necesario.

Además, se ha añadido un atributo que no se pedía expresamente: un booleano llamado draftMode. A pesar de que en el enunciado del requisito se indica que un proyecto es una agregación de historias de usuario, se ha considerado que es más lógico implementar esta relación como una composición, ya que la existencia de un proyecto no tiene sentido sin la existencia de las historias de usuario que lo compongan, y viceversa. Por tanto, este atributo se hacía necesario. Su función es indicar si un proyecto se encuentra finalizado o no: se creará un proyecto en el sistema antes de crear sus historias de usuario y, para evitar que se genere un error por no tener asignadas aún ninguna de sus partes, se indica que el proyecto se encuentra en modo borrador. Cuando se creen historias de usuario y se le asignen a este proyecto, podrá dejar de estar en modo borrador. Adicionalmente, este atributo tiene otra función: cuando el proyecto tenga errores fatales, deberá estar también en modo borrador hasta que estos se solucionen.

En cuanto a las relaciones de project, sólo tiene una: una relación unidireccional many-to-one con manager. Un manager tiene cero o muchos projects, mientras que un project tiene obligatoriamente un manager.

Las restricciones más complejas aún no se han implementado, debido a que deben ir en los servicios, y todavía no se ha dado en clase de teoría. Estas restricciones son: las historias de usuario de un proyecto deben ser todas del mismo gerente; un proyecto con errores fatales debe tener su draftMode a true; un proyecto que aún no tiene historias de usuario debe tener su draftMode a true.

De ahora en adelante, sólo se mencionarán las restricciones complejas que estén sin implementar de entidades posteriores, pero no se explicará el motivo debido a que es el mismo que aquí descrito.

### Validación por el profesor

El profesor dio el visto bueno a esta implementación el día 21/02/2024. Sugirió darle esa doble funcionalidad al atributo draftMode.

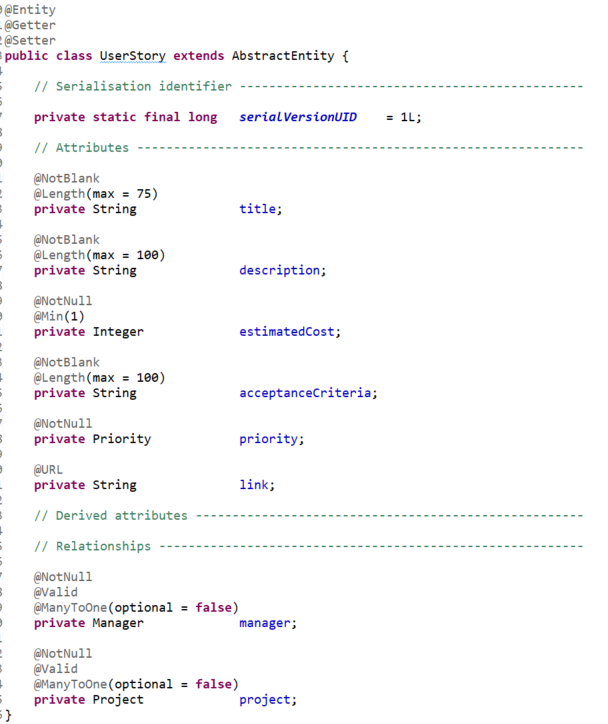
## Requisito obligatorio 2

### Descripción

Implementar una entidad llamada userStory (historia de usuario en español). Una historia de usuario, HU, es un documento utilizado por un gerente para representar la unidad mínima de trabajo en un proyecto. En el dominio del problema, es la parte de la que se compone un proyecto. Una HU debe contar con la siguiente información: un título (no en blanco, 75 caracteres o menos), una descripción (no en blanco, 100 caracteres o menos), un coste estimado en horas (positivo mayor de cero), un criterio o criterios de aceptación (no en blanco, 100 caracteres o menos), su prioridad (“Must”, “Should”, “Could”, o “Won’t”) y opcionalmente un enlace con más información.

### Análisis y decisiones

Tras haber atendido a las lecciones de clase, y haber consultado en caso necesario las transparencias correspondientes, se implementó la entidad user story de la siguiente manera:

Vemos que la clase Java se encuentra en el paquete acme.entities.userstories y que extiende a la clase AbstractEntity heredada de Acme Framework.

A continuación, están implementados los atributos especificados en el enunciado del requisito: title, description, estimatedCost, acceptanceCriteria, priority y link. Todos tienen las restricciones pedidas, implementadas gracias a las anotaciones correspondientes. Aquellos atributos que no estuvieran explícitamente mencionados como opcionales se han implementado como obligatorios.

Se ha decidido que el atributo estimatedCost sea un número entero debido a que representa un número de horas, que no pueden ser decimales. También se consideró crear un DataType llamado Hour, pero se descartó ya que era esencialmente lo mismo que implementar el atributo como un entero. No obstante, se cambiará en el futuro en caso necesario.

Imagen que contiene Texto

Descripción generada automáticamentePara el atributo priority se ha creado un enumerado con los posibles valores pedidos en el enunciado del requisito, que vemos en la siguiente imagen:

Se encuentra en el mismo paquete que user story.

En cuanto a las relaciones de user story, tiene dos: una relación unidireccional many-to-one con manager y otra igual con project. Un manager crea cero o muchos user stories, mientras que un user story es creado obligatoriamente un manager. Lo mismo sucede con project: un project está compuesto por uno o muchos user stories, mientras que un user story sólo es de un project.

### Validación por el profesor

El profesor recordó que la relación recomendada a implementar entre dos objetos, en caso de duda, era una relación unidireccional many-to-one; así se ha implementado la composición proyecto – historia de usuario.

Falta la validación del día 6 de marzo.

## Requisito obligatorio 3

### Descripción

Implementar un formulario llamado dashboard. El dashboard de un manager debe proporcionar la siguiente información: el número total de historias de usuario de cada prioridad; la media, la desviación, el mínimo y el máximo del coste estimado de las historias de usuario creadas por el manager; la media, la desviación, el mínimo y el máximo del coste de los proyectos del manager.

Un dashboard es un formulario ya que proporciona información que no persiste explícitamente en la base de datos, sino que la calcula a partir de datos sí persistidos en esta.

### Análisis y decisiones

Tras haber atendido a las lecciones de clase, y haber consultado en caso necesario las transparencias correspondientes, se implementó el formulario manager dahsboard de la siguiente manera:

Vemos que la clase Java se encuentra en el paquete acme.forms y que extiende a la clase AbstractForm heredada de Acme Framework.

A continuación, están implementados los atributos especificados en el enunciado del requisito: numberOfMustUserStories, numberOfShouldUserStories, numberOfCouldUserStories, numberOfWontUserStories, averageCostUserStories, deviationCostUserStories, minimumCostUserStories, maximumCostUserStories, averageCostProjects, deviationCostProjects, minimumCostProjects y maximumCostProjects. Todos estos atributos están implementados en la parte de atributos de respuesta del formulario, y con tipos primitivos ya que se sabe que nunca van a tomar un valor nulo.

Adicionalmente, se ha añadido un atributo llamado manager en la parte de atributos de consulta del formulario, que especifica sobre qué manager queremos ver el dashboard (es decir, sobre quién estamos haciendo la consulta).

### Validación por el profesor

El profesor sugirió usar tipos primitivos para aquellos atributos que nunca fueran a ser nulos.

Falta la validación del día 6 de marzo.

## Requisito obligatorio 4

### Descripción

Añadir datos de prueba para testear la aplicación de manera informal. Los datos deben incluir dos cuentas de manager con credenciales “manager1/manager1” y “manager2/manager2”.

### Análisis y decisiones

Tras haber atendido a las lecciones de clase, y haber consultado en caso necesario las transparencias correspondientes, se han realizado acciones descritas a continuación:

Primero, se han añadido dos filas al archivo “/src/main/webapp/WEB-INF/resources/sample-data/user-account.csv” con las credenciales de los dos managers:

Texto, Aplicación

Descripción generada automáticamente

Después, se ha creado el archivo “/src/main/webapp/WEB-INF/resources/sample-data/manager.csv” y se ha rellenado con la siguiente información:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Como se indica en las transparencias de la asignatura, aquel atributo que podía ser nulo se ha dejado en blanco en una de las instancias para comprobar que la restricción funciona correctamente.

### Validación por el profesor

Falta la validación del día 6 de marzo.

## Requisito suplementario 2

### Descripción

Crear un diagrama UML del modelo de dominio.

### Análisis y decisiones

Tras haber atendido a las lecciones de clase, y haber consultado en caso necesario las transparencias correspondientes, se ha creado el siguiente diagrama UML utilizando la aplicación UMLet:

Foto diagrama.

Descripción diagrama.

### Validación por el profesor

Falta la validación del día 6 de marzo.

## Requisito suplementario 3

### Descripción

Desarrollar un informe de análisis.

### Análisis y decisiones

Este presente informe es el informe de análisis. Se ha decidido comentar únicamente las tareas individuales del estudiante 1, para que el informe de análisis grupal describa a su vez únicamente las tareas grupales.

### Validación por el profesor

Debido a la sencillez de la tarea y, al mismo tiempo, la longitud del informe producido, no se requiere validación por parte del profesor.

## Requisito suplementario 4

### Descripción

Desarrollar un informe de planificación y progreso.

### Análisis y decisiones

El informe de planificación y progreso del estudiante 1 se encuentra dentro de la misma carpeta en la que se halla el presente informe de análisis. En este caso, se ha decidido comentar tanto las tareas individuales del estudiante 1 como las grupales que se le han asignado, de manera que la evaluación de rendimiento tenga sentido hacerse en este documento (puesto que dicha evaluación depende tanto de las tareas grupales como de las individuales).

### Validación por el profesor

Debido a la sencillez de la tarea y, al mismo tiempo, la longitud del informe producido, no se requiere validación por parte del profesor.

# Conclusiones

En este análisis, la implementación del requisito funcional que consistía en modificar el menú anónimo se llevó a cabo de manera eficiente y sin complicaciones significativas. Los conocimientos adquiridos en las clases previas han permitido identificar rápidamente los archivos a modificar y la validación por parte del profesor no ha sido necesaria dada la simplicidad del cambio.

No obstante, hemos encontrado más dificultades a la hora de preparar todo el entorno sobre el que vamos a estar trabajando a lo largo de este proyecto, un tema sobre el que se tratará en más profundidad en el informe correspondiente (tarea 008).

En resumen, la compresión de la estructura de este proyecto se refleja en la exitosa implementación del requisito; tanto por mi parte como por la de mis compañeros.

# Bibliografía

Transparencias de DP2 – L02, S01 – A foundation.

Transparencias de DP2 – L02, S02 – Sample data.