Informe de Análisis

Integrantes del grupo C2.026:

* Ignacio Blanquero Blanco ([ignblabla@alum.us.es](mailto:ignblabla@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))

Fecha: Sevilla, 8 de Julio 2024

Tabla de contenido

[Resumen del Informe 3](#_Toc170979110)

[Historial de Versiones 4](#_Toc170979111)

[Introducción 5](#_Toc170979112)

[Registro de análisis 6](#_Toc170979113)

[Requisito suplementario 1 6](#_Toc170979114)

[Descripción 6](#_Toc170979115)

[Análisis y decisiones 6](#_Toc170979116)

[Validación por el profesor 7](#_Toc170979117)

[Requisito obligatorio 1 7](#_Toc170979118)

[Descripción 7](#_Toc170979119)

[Análisis y decisiones 7](#_Toc170979120)

[Validación por el profesor 9](#_Toc170979121)

[Requisito obligatorio 2 9](#_Toc170979122)

[Descripción 9](#_Toc170979123)

[Análisis y decisiones 9](#_Toc170979124)

[Validación por el profesor 10](#_Toc170979125)

[Requisito obligatorio 3 11](#_Toc170979126)

[Descripción 11](#_Toc170979127)

[Análisis y decisiones 11](#_Toc170979128)

[Validación por el profesor 11](#_Toc170979129)

[Requisito obligatorio 4 12](#_Toc170979130)

[Descripción 12](#_Toc170979131)

[Análisis y decisiones 12](#_Toc170979132)

[Validación por el profesor 13](#_Toc170979133)

[Requisito suplementario 2 13](#_Toc170979134)

[Descripción 13](#_Toc170979135)

[Análisis y decisiones 14](#_Toc170979136)

[Validación por el profesor 14](#_Toc170979137)

[Conclusiones 15](#_Toc170979138)

[Bibliografía 16](#_Toc170979139)

# Resumen del Informe

El objetivo de este informe es proporcionar detalles sobre el análisis de todos los tipos de requisitos especificados en esta segunda 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 |
| V2.1 | Actualización de requisitos implementados: cambios y adiciones. Conclusiones. | 08/03/2024 | Mª Salud Carrera Talaverón |
| V3.0 | Actualización de requisitos implementados tras la segunda convocatoria. | 04/07/2024 | 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 tanto para estos objetos como 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.

En este informe se comentarán todas las tareas realizadas excepto los informes producidos, que se pueden consultar en la carpeta correspondiente.

# 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.

Además, tras la entrega de la primera convocatoria, este requisito fue validado por el profesor.

## Requisito obligatorio 1

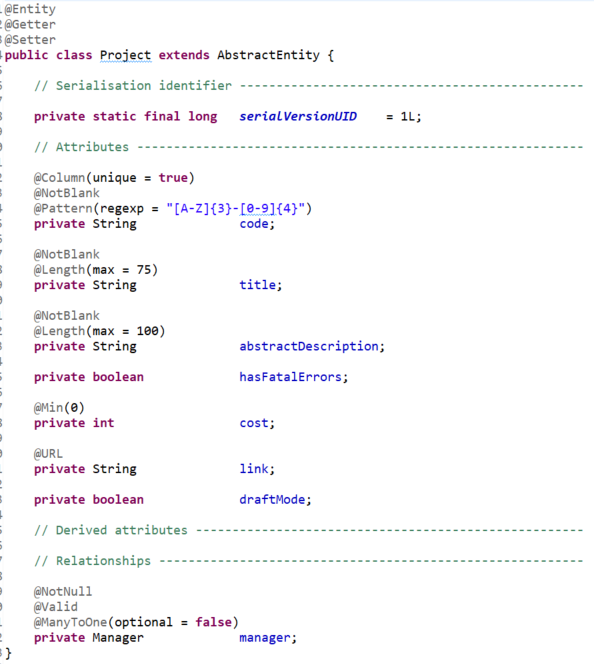
### Descripción

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:

La clase Java se encuentra en el paquete acme.entities.projects y 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 o, en caso necesario, a usar tipos primitivos de Java que no admiten valores nulos. 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 o de tipo Money, análogamente al atributo estimatedCost de las historias de usuario (explicado en el requisito siguiente).

Además, se ha añadido un atributo que no se pedía expresamente: un booleano llamado draftMode. En el enunciado del requisito se indica que un proyecto es una agregación de historias de usuario, 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, siempre y cuando estas historias de usuario tampoco estén 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 no se han implementado en este entregable, debido a que deben ir en los servicios, y eso corresponde al siguiente entregable. 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 publicadas 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 de no haberse añadido aún 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. El día 06/03/2024 indicó que era más lógico utilizar tipos primitivos en aquellos atributos que no pudieran ser nulos, en lugar de usar tipos envolventes y añadirles la anotación @NotNull.

Además, tras la entrega de la primera convocatoria, este requisito fue validado por el profesor.

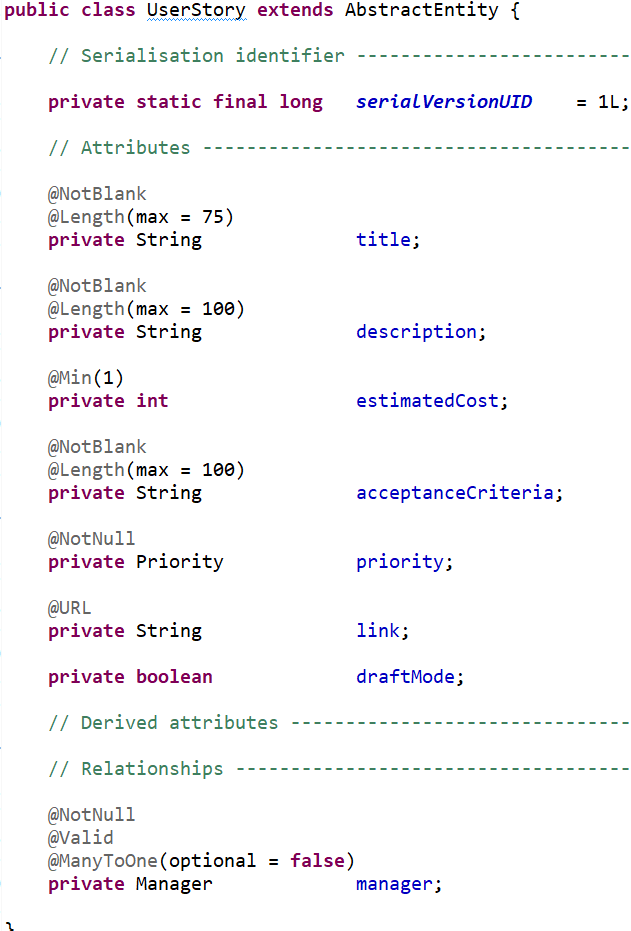
## Requisito obligatorio 2

### 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:

La clase Java se encuentra en el paquete acme.entities.userstories y 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 o, en caso necesario, a usar tipos primitivos de Java que no admiten valores nulos. 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 genera confusión si se usan números decimales ya que las horas siguen un sistema hexadecimal y no decimal, que sería lo que estaría indicando un atributo de tipo double. También se consideró crear un DataType llamado Hour, pero se descartó ya que era esencialmente lo mismo que implementar el atributo como un entero.

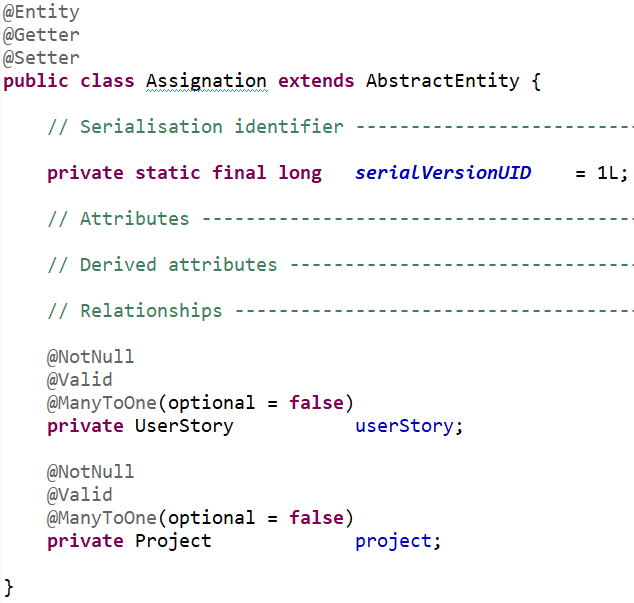
Imagen que contiene Texto

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.

Además, se ha añadido un atributo que no se pedía expresamente: un booleano llamado draftMode. En el enunciado del requisito se indica que un proyecto es una agregación de historias de usuario, 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. Además, servirá para implementar la operación de publicación de una historia de usuario, que se verá en el entregable D03.

En cuanto a las relaciones de user story, tiene dos: una relación unidireccional many-to-one con manager y otra unidireccional many-to-many con project. Un manager crea cero o muchos user stories, mientras que un user story es creado obligatoriamente un manager. Algo distinto sucede con project: un project está compuesto por uno o muchos user stories; sin embargo un user story puede ser compartido por varios projects.

Para implementar la relación many-to-many se podría haber indicado con la anotación @ManyToMany en un atributo de tipo lista de projects. Sin embargo, esto se crearía en la base de datos mediante una tabla intermedia, que guardase cada proyecto con cada una de sus HUs, inaccesible por el desarrollador. Por tanto, se ha optado por implementar una nueva entidad llamada Assignation.java, que haga la misma función que esta tabla pero que sí puede ser accedida por el desarrollador.

Vemos que esta entidad se encuentra en el paquete acme.entities.assignations y que también extiende a la clase AbstractEntity.

Sólo tiene por atributos sus dos relaciones: una relación unidireccional many-to-one con project y otra igual con user story. De esta forma, se soluciona la problemática de la traducción a la tabla intermedia en la base de datos.

### 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 agregación proyecto – historia de usuario.

El día 06/03/2024 aclaró que una HU puede ser compartida por varios proyectos, pues este requisito se había malinterpretado, y refirió la información proporcionada por las transparencias de la asignatura: que debía crearse una entidad intermedia.

Además, tras la entrega de la primera convocatoria, este requisito fue validado por el profesor.

## 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. Algunos atributos están definidos con tipos primitivos ya que se sabe que nunca van a tomar un valor nulo; estos son los atributos que representan un total. Sin embargo, los atributos que representan cálculos están definidos con tipos envolventes para que, cuando la consulta se haga sobre un conjunto de datos vacío, puedan tomar valores nulos.

### Validación por el profesor

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

Además, tras la entrega de la primera convocatoria, este requisito fue validado por el profesor.

## Requisito obligatorio 4

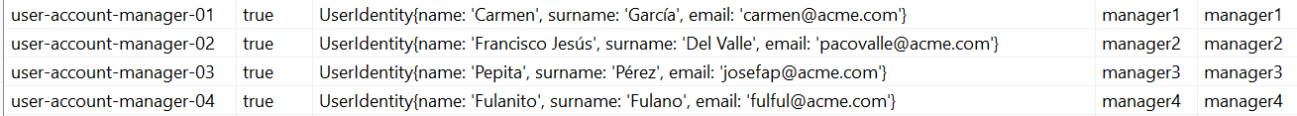
### Descripción

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”.

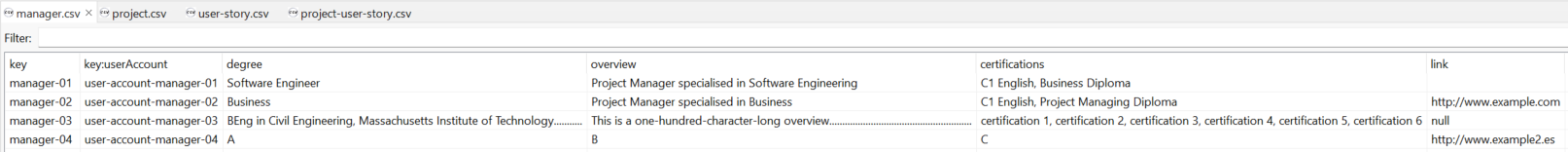
### Análisis y decisiones

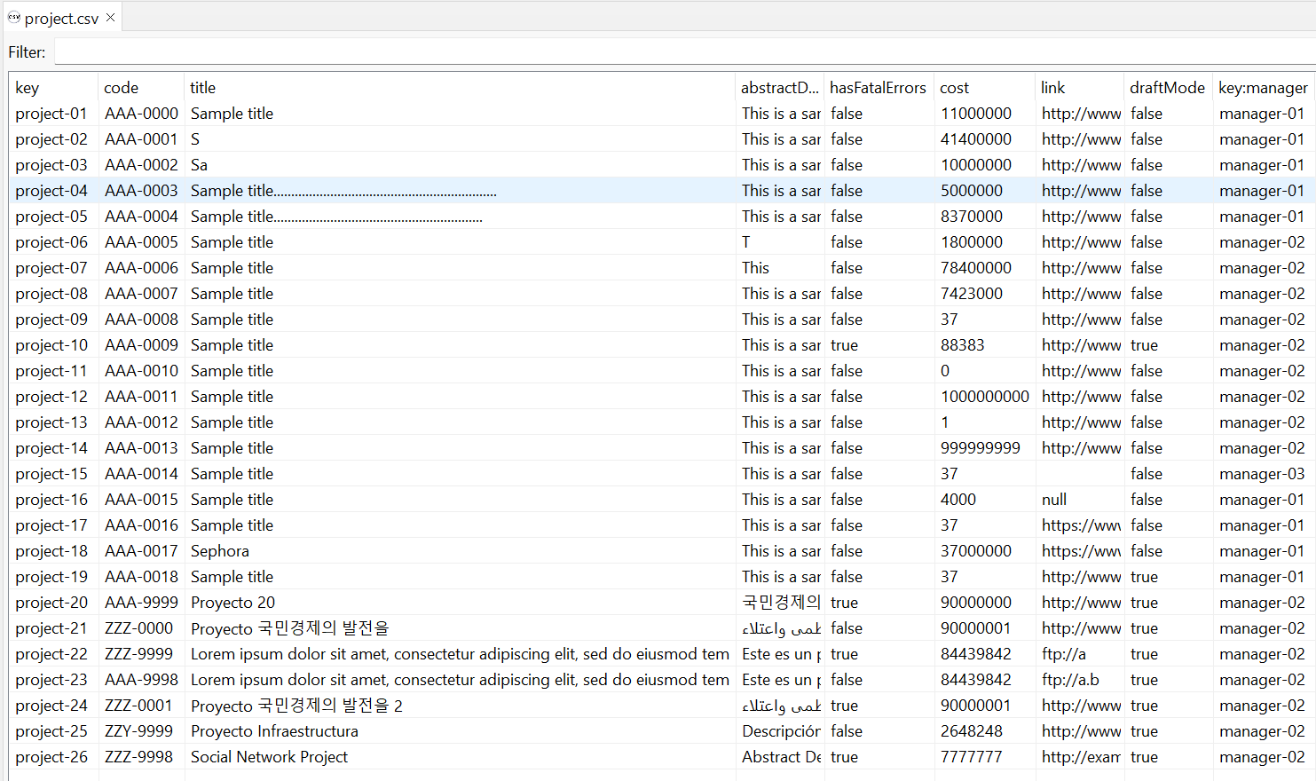
Tras haber atendido a las lecciones de clase, y haber consultado en caso necesario las transparencias correspondientes, se han realizado las acciones descritas a continuación:

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



Después, se han creado archivos .csv para cada rol y entidad en la ruta “/src/main/webapp/WEB-INF/resources/sample-data/” y se ha rellenado con la siguiente información:





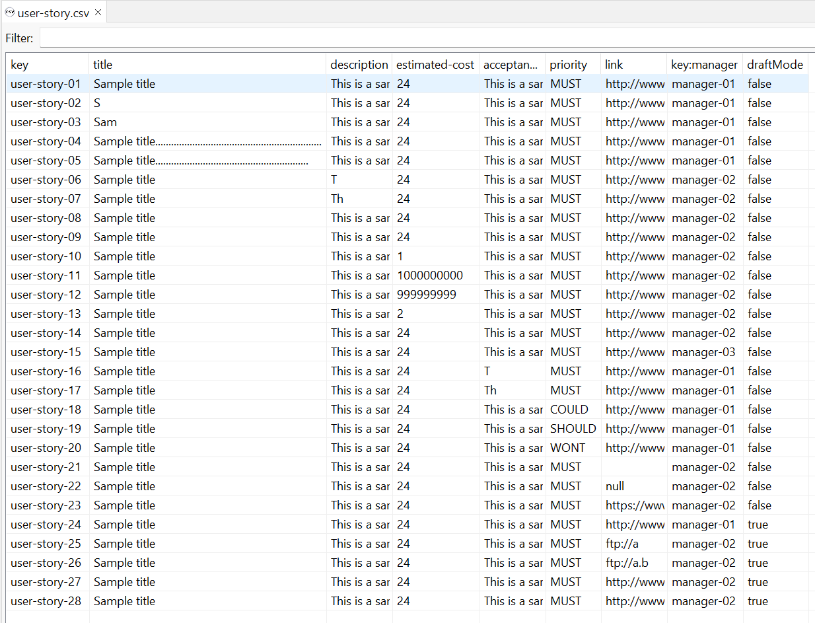


Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza bajaComo se indica en las transparencias de la asignatura, se han generado casos de prueba primero con un objeto base, que cumpla todas las restricciones, y luego creando variaciones del objeto base para comprobar los límites de las restricciones de cada atributo. Para una mejor visualización del contenido de estos archivos, especialmente para ver el contenido de atributos con valores más largos y todas las filas del archivo assignation.csv, dirigirse a la ruta indicada en el proyecto.

### Validación por el profesor

Tras la entrega final de la convocatoria de junio, el profesor no solicitó cambios en este requisito; por tanto, se asume que se ha validado.

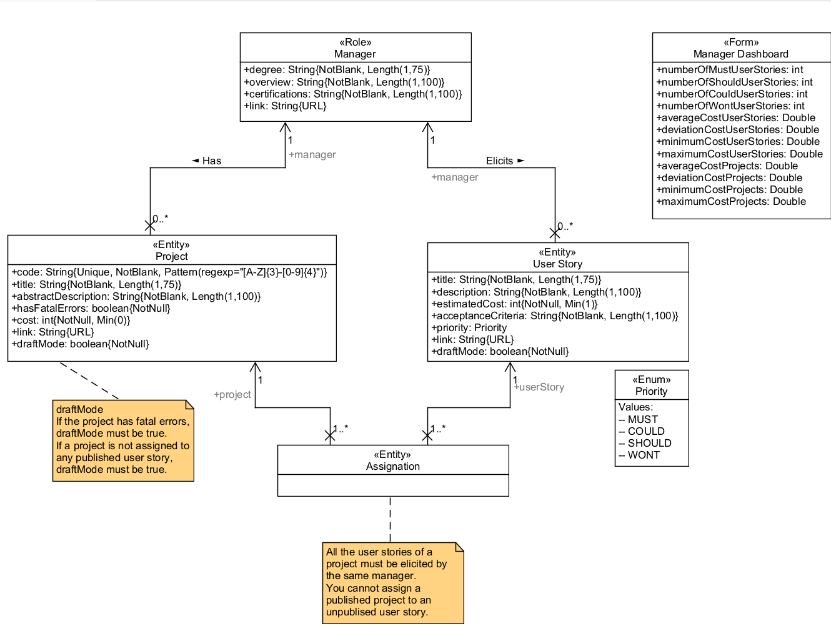
## Requisito suplementario 2

### 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:

En este diagrama vemos claramente el rol Manager, la entidad Project, la entidad User Story y el formulario Manager Dashboard con sus atributos definidos con sus tipos y sus restricciones. Las restricciones personalizadas se han indicado mediante notas unidas a la clase correspondiente.

Vemos las relaciones entre las entidades explicadas anteriormente, y la nueva entidad intermedia llamada Assignation. Todas las relaciones tienen la multiplicidad, la dirección y el atributo por el que se relacionan indicados. Además, se ha añadido el enumerado Priority con sus posibles valores.

### Validación por el profesor

Tras la entrega final de la convocatoria de junio, el profesor no solicitó cambios en este requisito; por tanto, se asume que se ha validado.

# Conclusiones

Tras este análisis, concluimos que la implementación de todos los requisitos no ha supuesto mayor complicación. El mayor obstáculo ha sido determinar correctamente el tipo a utilizar en cada atributo y la naturaleza del mismo (primitivo o envolvente), pero gracias a las indicaciones del profesor se ha realizado sin problemas.

Consideramos exitoso el desarrollo de las tareas individuales de este entregable.

# Bibliografía

Transparencias de DP2 – L02, S01 – A foundation.

Transparencias de DP2 – L02, S02 – Sample data.