

[Universidad de Burgos]



# [Informe sobre aplicación KANBAN]

---

[DISEÑO Y MANTENIMIENTO DEL SOFTWARE]

Mario Flores Espiga  
Sergio López Bueno

## Contenido

OBJETIVOS.....	3
REALIZACIÓN.....	5
BACKLOG.....	5
DEFECTO .....	5
HISTORIA DE USUARIO.....	5
MIEMBRO DE EQUIPO .....	5
PRODUCT BACKLOG.....	5
REQUISITO .....	5
SPRINTBACKLOG .....	6
TAREA.....	6
KANBAN .....	6
PERSISTENCIA .....	7
ACLARACIONES .....	7

# OBJETIVOS

El proyecto consistirá en la elaboración de una **pequeña aplicación de tableros kanban para scrum**, a realizar en **grupos de dos personas**. El lenguaje de implementación será **Java**.

Se realizarán **dos entregas** a través de un **repositorio online** (GitHub, GitLab...), los días **25 de noviembre** y **23 de diciembre**, con un peso del **15%** y el **25%** respectivamente sobre la nota final de la asignatura. Si los commits están marcados como *primera* y *segunda entrega* (ambos anteriores a sus respectivas fechas límite), se usarán éstos como entregas. Si no hay ninguno etiquetado, se usará el último commit antes de su vencimiento (aunque se trate de commits intermedios de desarrollo). **La memoria debería estar también disponible en el repositorio**. En la primera entrega se valorará la implementación de las funcionalidades esenciales y el proceso de diseño hasta llegar a ellas (10 primeras historias de usuario; ver más abajo). En la segunda entrega se valorarán el diseño e implementación finales (todos los requisitos, además de las correcciones sugeridas en la entrega anterior). En ella se implementará el punto 11 o el 12 (a elegir por cada grupo); de ambos 11 y 12 se documentará el diseño y por qué facilita su futura extensión, y además se dará la implementación de uno de ellos. Se tomará como punto de partida el repositorio <https://github.com/Kencho/ubu-gii-dms-po1c>, del que cada equipo hará un *fork* cuando lo indique el profesor para poder trabajar sobre su propia copia independiente.

**Sobre la lógica de negocio:** La aplicación debería permitir un uso simplista de la metodología/orientaciones *Scrum*, en el que se definen **tareas** asociadas a **historias de usuario y/o defectos** a corregir, representadas físicamente en tarjetas. Cada una de estas tareas se encuentra en un estado concreto dentro de un *backlog*. Hay dos tipos de backlogs: **Product backlog**, con las tareas definidas, pero no asignadas a un ciclo concreto (*sprint*), y los **Sprint backlog**, o *backlogs* de cada iteración específica. Las tareas en el *Product backlog* sólo estarán en un estado *pendiente*, mientras que en los *Sprint backlog* pueden estar como *pendientes* ("To do"), *en proceso* ("Doing"), *en validación* ("QA/Testing"), o *completadas* ("Completed" o "Finished"). Al planificar la siguiente iteración, se extraen tareas del *Product backlog* y se colocan en el *Sprint backlog* que se está definiendo, como *pendientes*. Durante el *sprint*, estas tareas se irán moviendo entre estados hasta que se completen. Todos los *sprints* tendrán una **duración equivalente**, y cada uno tendrá asociada una **fecha de inicio**. Además del requisito asociado, cada tarea debe tener al menos los siguientes campos: **Título, descripción, coste, beneficio, y miembro asignado**.

### **Requisitos de la aplicación** en forma de historias de usuario:

1. Como **evaluador** quiero **acceso al código y su historial** para **ver la implementación**
2. Como **evaluador** quiero **una memoria** para **evaluar el uso de buenas prácticas de diseño y patrones de diseño**
3. Como **evaluador** quiero **instrucciones** para **poder construir y lanzar la aplicación**
4. Como **cliente** quiero **poder añadir miembros al equipo**
5. Como **cliente** quiero **poder añadir sprint backlogs** para las nuevas iteraciones
6. Como **cliente** quiero **poder añadir tareas al product backlog**
7. Como **cliente** quiero **poder mover tareas del product backlog al sprint backlog**
8. Como **cliente** quiero **poder mover tareas entre las fases del ciclo de vida**
9. Como **cliente** quiero **poder editar las tareas**
10. Como **cliente** quiero **que los backlogs y tareas se mantengan cuando vuelva a abrir la aplicación**
11. Como **cliente** quiero **que la aplicación pueda tener nuevas formas de guardar el modelo en el futuro (implementar una si no se implementa el punto 12)**
12. Como **cliente** quiero **que la aplicación pueda tener nuevas formas de interacción en el futuro (implementar una si no se implementa el punto 11)**

# REALIZACIÓN

Se ha seguido el esquema subido en github, con alguna modificación de carácter leve.

## BACKLOG

Clase padre de la que heredan SprintBacklog y ProductBacklog

## DEFECTO

Clase que hereda de Requisito

## HISTORIA DE USUARIO

Clase que hereda de Requisito

## MIEMBRO DE EQUIPO

En esta clase se ha codificado a un usuario de la aplicación KANBAN.

Como atributos tiene su nombre, edad y DNI, que actuará de cara a la persistencia como clave primaria.

Los métodos de la clase son simplemente getters y setters para los distintos atributos.

## PRODUCT BACKLOG

Hereda de Backlog y representa el lugar donde están las tareas antes de ser movidas a un sprint.

Las tareas se almacenan en una lista.

Tiene métodos para devolver la lista completa de las tareas o en su defecto una tarea específica, pasando como parámetro la referencia a la tarea.

## REQUISITO

Clase que representa un requisito asociado a una tarea.

Se compone simplemente de un String que almacena el texto del requisito.

## SPRINTBACKLOG

Clase similar a ProductBacklog.

Contiene un método para imprimir las distintas tareas almacenadas y su estado, puesto que en SprintBacklog a diferencia de ProductBacklog las tareas pueden estar en diferentes estados (TODO, DOING, TEST, FINISHED)

## TAREA

En esta clase se codifican las tareas del tablero KANBAN.

Una tarea tiene los atributos título, descripción, coste, beneficio, asignadoA (Miembro al que esta asignada la tarea) y requisito.

Además, para la implantación del estado de las tareas se usa un tipo enumeración.

Tiene getters y setters para todos los atributos, y un método toString que imprime el título de la tarea a efectos de identificación.

## KANBAN

Clase raíz del sistema desde la que se lanza la aplicación.

Para el menú principal se utiliza un bucle con las distintas opciones a escoger.

Se utilizan impresiones por pantalla y un reader para recoger la interacción con el usuario.

# PERSISTENCIA

Para la persistencia se ha optado por utilizar lectura y escritura en archivos CSV debido a su sencillez tanto para su programación como su uso frente a soluciones más complejas como el acoplamiento de una base de datos.

En concreto se utiliza un archivo para cada tipo de entidad, con una línea para cada entidad concreta, guardando sus atributos separados por un carácter definido dentro del código.

# ACLARACIONES

En esta primera entrega hemos buscado implementar todas las funcionalidades sin tener en cuenta otros aspectos como la robustez del código frente a entradas de usuario incorrectas, sin tener en cuenta tampoco el refinamiento de la interfaz de texto. Este tratamiento de excepciones y mejora de la interfaz se mejorará para la segunda entrega del proyecto.

También se ha obviado la aplicación de patrones de diseño para tener una base sobre la cual aplicarlos de cara a la segunda entrega.

# INSTRUCCIONES DE USO

Para ejecutar la aplicación basta con importar el proyecto java y ejecutarlo. La clase Kanvan.java es la que contiene el main principal del programa.

Una vez ejecutado solo hay que seguir los pasos que indica el programa en modo texto.

Al iniciar el programa te da una serie de opciones para poder introducir los datos, simplemente se debe introducir el número correspondiente a la opción elegida. (En esta primera entrega hay que tener cuidado con el orden en el que se realizan las cosas ya que aún no se ha implementado correctamente el control de errores. Por ejemplo, fallaría al crear una tarea antes de crear al menos un miembro ya que no hay miembros a los que asignar la tarea.)

Para salir del programa tienes dos opciones, la primera es salir guardando los datos en dos archivos CSV que se crearán en la raíz del proyecto y la segunda que es salir sin guardar. Para cargar los datos de los archivos basta con utilizar la opción 0 "Cargar datos".