



Universidad ORT Uruguay

Facultad de Ingeniería

# Informe de avance 2

## Ingeniería de Software Ágil 2

[Repositorio en Github](#)

Los distintos documentos que forman parte de la entrega se encuentran dentro del repositorio  
Documentación/Entrega 2

Melissa Molinari – 229896

María Agustina Lamanna – 223040

Sebastian Daners - 255927

Profesores: Álvaro Ortas, Carina Fontán

Grupo: M7A

Septiembre 2023

# Definición del proceso de ingeniería en el contexto de KANBAN

Según lo planteado por el marco de trabajo ágil KANBAN no se tienen roles, ceremonias ni artefactos definidos. Pero decidimos adaptarlo a las necesidades de este proyecto y en específico para esta entrega. Por ello, se generó una *definition of done* y se definieron *roles*.

## Definition of done

Para esta entrega, en donde se realizaron modificaciones de código consideramos pertinente plantear una *definition of done*. De esta forma al momento de arreglar los bugs seleccionados, los desarrolladores tenían claro cuáles eran las condiciones para poder considerar el mismo como solucionado.

En este caso, la definition of done que seleccionamos fue la siguiente:

“El sistema cumple con la funcionalidad propuesta en el cuerpo del bug correspondiente y además, pasa correctamente todos los tests.”

## Designación de roles

Para esta entrega, al igual que para la anterior, se definieron los roles que asumiría cada uno de los integrantes.

Se definió el rol de SM (Scrum Master), el mismo lo adoptó Melissa Molinari. Este rol ya había sido definido para la entrega anterior y consideramos que fue muy beneficioso a la hora de realizar la *ceremonia de retrospectiva* ya que cumplía un rol de moderador. Además de aportar ideas sobre cómo podemos mejorar nuestro enfoque y aplicación del marco KANBAN a nuestro proyecto.

Por otra parte, se definió el rol de PO (Product Owner), el mismo lo adoptó Agustina Lamanna. Para esta entrega se realizó una *ceremonia de revisión*, en donde se le mostró al PO los cambios realizados para la solución de los bugs seleccionados. Allí el PO fue quién se encargó de aceptar aquellos cambios que cumplieran con la definition of done que se definió en la reunión de planificación.

Asimismo, los tres integrantes adoptaron roles de desarrolladores y testers. Cada uno se encargó de solucionar uno de los bugs seleccionados.

## Diseño del tablero

Utilizamos un tablero vinculado al proceso de ingeniería dentro de Github. Para esta entrega, elegimos utilizar un tablero sustentable con siete columnas.

El tablero consta de una columna Backlog en donde se ubicaron los bugs seleccionados para su corrección.

La columna To do contiene todas las actividades pendientes. Luego, se dividió la columna doing en 4. Las columnas Designing, Programming y Testing fueron utilizadas para indicar de forma más clara las actividades que se estaban desarrollando específicamente en relación a los bugs. Las tarjetas se fueron moviendo entre las mismas dependiendo de las actividades que se estaban realizando. Consideramos que esta separación sería beneficiosa ya que tomamos la decisión de crear para cada bug dos tarjetas. Una correspondiente a la realización de TDD (en los casos que fuera necesario) y otra correspondiente al código fuente. De esta forma sería más claro visualizar entre las columnas y las tarjetas las actividades que se estaban llevando a cabo.

Finalmente, la columna *In progress* se agregó para las tarjetas correspondientes a tareas complementarias que se fueron realizando. Consideramos que esta separación entre las columnas permitiría mantener un claro registro del progreso a medida que se avanzó con el proceso de ingeniería.

La columna *Done* contiene todas aquellas actividades finalizadas.

El tablero se puede visualizar en el siguiente [link](#).

## Configuración del pipeline y su vínculo con el tablero

Utilizando Github Actions se implementaron acciones que compilar y corren las pruebas automáticamente en el pipeline.

Para realizar la configuración se añade un archivo YAML al repositorio github. El archivo se configuró de forma que al realizar un push o pull request en cualquier rama, se ejecuten las acciones especificadas. En nuestro caso las acciones que decidimos ejecutar son: la instalación de dependencias, la compilación del código y la ejecución de los unit Tests encontrados en *PharmaGo.Test.csproj*.

El pipeline se configuró con el objetivo de asegurar que cualquier cambio efectuado en el repositorio sigue permitiendo la compilación de la aplicación y a su vez sigue cumpliendo con el estándar definido para su funcionamiento en las pruebas unitarias. Asimismo, la automatización de la compilación y las pruebas permite agilizar la detección de problemas y errores en el código.

# Guía del repositorio

## Elementos que contiene

El repositorio cuenta con una carpeta Código la cuál incluye el código fuente de la aplicación junto con las modificaciones de código realizadas durante esta entrega. Por otra parte, se encuentra la carpeta Documentación donde se creó una subcarpeta correspondiente a esta entrega. Dentro de la misma se podrán encontrar los artefactos recomendados para esta entrega.

Otros elementos incluidos dentro del repositorio son el tablero Kanban. Se creó un nuevo tablero denominado *Entrega 2*, allí se detallaron todas las tareas realizadas durante esta entrega, junto a los responsables de la misma, el tiempo empleado para realizar dicha actividad, el esfuerzo (medido en horas-persona) y una pequeña descripción de lo realizado.

Finalmente, se generó la etiqueta tag con el nombre v.2 correspondiente a esta entrega.

## Estrategia de flujo

Para esta nueva entrega se eligió una nueva estrategia de branching. En este caso, decidimos que sería más conveniente adoptar una estrategia trunk-based. Se decidió generar una rama por cada uno de los bugs a solucionar. Tomamos como estándar para su nomenclatura *bug-nombre\_descriptivo* para así lograr identificarlas con facilidad.

Por otra parte, al elegir esta estrategia de branching decidimos determinar que cada 2 días se realizaría un merge de las ramas creadas para los bugs con main. De esta forma, como las ramas fueron creadas entre los días domingo/lunes, el día martes se realizó el merge de las mismas a la rama main.

# Identificación y justificación de los bugs seleccionados

Para esta entrega, se seleccionaron aquellos bugs que consideramos debían ser solucionados lo antes posible. Por ello, se seleccionaron los bugs de severidad *Crítica*, lo que significa que son errores que impiden o bloquean completamente el uso de una función. Además, se tomó en cuenta la prioridad establecida. Decidimos seleccionar aquellos bugs con un nivel de prioridad *Inmediato* ya que son aquellos con un plazo máximo para su solución de 24hs.

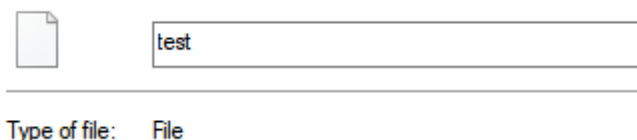
Por lo tanto, se seleccionaron los bugs:

- Exportador - archivo generado (#1) → con severidad Crítica y prioridad Inmediato
- Funcionalidad Login/Signup no accesible desde home (#5) → con severidad Crítica y prioridad Inmediato
- Solicitudes de stock negativas (#8) → con severidad Leve pero prioridad Inmediato

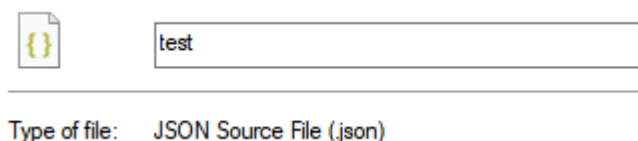
Este último bug fue seleccionado, aunque tenga una severidad Leve, ya que tiene una prioridad Inmediata y no habían más bugs con severidad Crítica.

## Exportador

Este bug refiere a que al intentar usar el exportador de JSON, el archivo no se crea con la extensión adecuada. Esto significa que a la hora de intentar utilizarlo, no es posible. Si intenta exportar con un nombre de archivo "test", el resultado de la exportación es lo siguiente:



Cuando lo que se debería ver en realidad es lo siguiente:



Este problema se solucionó agregando el siguiente código en el JSON exporter:

```
absolutePath = Path.ChangeExtension(absolutePath, "json");
```

No se utilizó la metodología TDD para este bug fix ya que estamos tratando con Reflections y es un bug que se soluciona con la librería System.IO, la cual es extremadamente robusta. También porque no existen pruebas previas para estas funciones.

## Login / Signup

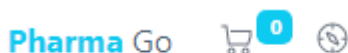
Este bug refiere a la ausencia de un botón que lleve al usuario desde la página *Home* a las pestañas de Login o Signup, lo cual significa que cualquier usuario que desconozca las url correspondientes no podrá hacer ninguna de estas dos acciones. Dado que el login y registro de los usuarios es esencial para el funcionamiento de la aplicación se determinó como crítico y se decidió resolverlo en esta iteración. Se comenzó por la etapa de diseño donde se evaluó como resolver el problema. Dado que se trata de un error exclusivamente del frontend se omite la fase de TDD, ya que no hay tests para el proyecto de frontend. Por último, en la etapa de programación se resolvió el problema según lo establecido en la etapa de diseño.

Código añadido:

Archivo modificado: `src\app\custom\custom-header\custom-header.component.html`

```
<c-nav-item title="Tracking">
  <a routerLink="/home/tracking" cNavLink>
    <svg cIcon name="cil-compass" size="lg"></svg>
  </a>
</c-nav-item>
<c-nav-item title="Login">
  <button class="customButton">
    <a routerLink="/login" cNavLink>
      Login
    </a>
  </button>
</c-nav-item>
<c-nav-item title="Register">
  <button class="customButton">
    <a routerLink="/register" cNavLink>
      Sign Up
    </a>
  </button>
</c-nav-item>
</c-header-nav>
</div>
```

Antes:



Después:



## Solicitudes de stock negativo

Este bug está relacionado con la actividad realizada por los empleados de generar una solicitud de stock para un cierto medicamento. Al crear una solicitud de stock, se le solicita al empleado que para cada medicamento indique la cantidad necesaria. Identificamos que al momento de seleccionar la cantidad que se desea no se validaba dentro del backend que no se ingresaran valores negativos, ya que esto carecería de sentido en la realidad del dominio. Por lo tanto, el sistema permitía crear solicitudes de stock para medicamentos con cantidades negativas.

En este caso, al tratarse de una corrección de código dentro del backend se siguió el enfoque de programación TDD. Por lo tanto, se comenzó realizando el test necesario y una vez finalizado el mismo se agregó el código mínimo indispensable para que el test funcionara.

### Código de caso de prueba

Para este caso, se agregó un único test en dónde se verifica que en caso de que se intente crear una solicitud de stock en donde alguno de los componentes contenga un valor de cantidad negativo, el sistema envía un error de tipo `InvalidResourceException`.

Archivo modificado:

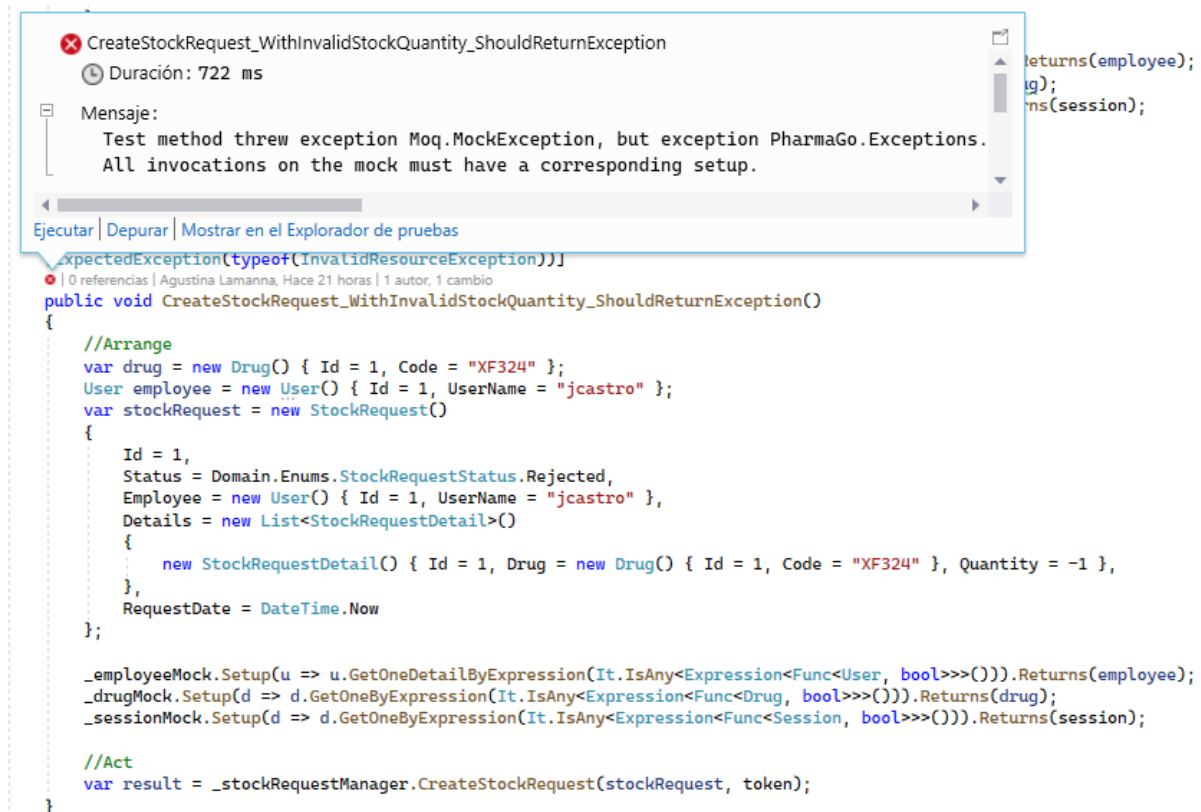
Código\Backend\PharnaGo.Test\BusinessLogic.Test\StockManagerTest.cs

La siguiente imagen muestra el código que se agregó para dicho test:

```
312 [TestMethod]
313 [ExpectedException(typeof(InvalidResourceException))]
314 // 0 referencias | Agustina Lamanna, Hace 21 horas | 1 autor, 1 cambio
315 public void CreateStockRequest_WithInvalidStockQuantity_ShouldReturnException()
316 {
317     //Arrange
318     var drug = new Drug() { Id = 1, Code = "XF324" };
319     User employee = new User() { Id = 1, UserName = "jcastro" };
320     var stockRequest = new StockRequest()
321     {
322         Id = 1,
323         Status = Domain.Enums.StockRequestStatus.Rejected,
324         Employee = new User() { Id = 1, UserName = "jcastro" },
325         Details = new List<StockRequestDetail>()
326         {
327             new StockRequestDetail() { Id = 1, Drug = new Drug() { Id = 1, Code = "XF324" }, Quantity = -1 },
328         },
329         RequestDate = DateTime.Now
330     };
331     _employeeMock.Setup(u => u.GetOneDetailByExpression(It.IsAny<Expression<Func<User, bool>>>()))).Returns(employee);
332     _drugMock.Setup(d => d.GetOneByExpression(It.IsAny<Expression<Func<Drug, bool>>>()))).Returns(drug);
333     _sessionMock.Setup(s => s.GetOneByExpression(It.IsAny<Expression<Func<Session, bool>>>()))).Returns(session);
334
335     //Act
336     var result = _stockRequestManager.CreateStockRequest(stockRequest, token);
337 }
```

Evidencia de ejecución de casos de prueba → Etapa RED

La siguiente imagen muestra la etapa RED dentro de TDD. En donde se tiene el test, pero aún no se agregaron las líneas de código necesarias para que el test pase y de GREEN.



```
ExpectedException(typeof(InvalidResourceException)))
0 referencias | Agustina Lamanna, Hace 21 horas | 1 autor, 1 cambio
public void CreateStockRequest_WithInvalidStockQuantity_ShouldReturnException()
{
    //Arrange
    var drug = new Drug() { Id = 1, Code = "XF324" };
    User employee = new User() { Id = 1, UserName = "jcastro" };
    var stockRequest = new StockRequest()
    {
        Id = 1,
        Status = Domain.Enums.StockRequestStatus.Rejected,
        Employee = new User() { Id = 1, UserName = "jcastro" },
        Details = new List<StockRequestDetail>()
        {
            new StockRequestDetail() { Id = 1, Drug = new Drug() { Id = 1, Code = "XF324" }, Quantity = -1 },
        },
        RequestDate = DateTime.Now
    };

    _employeeMock.Setup(u => u.GetOneDetailByExpression(It.IsAny<Expression<Func<User, bool>>>()))Returns(employee);
    _drugMock.Setup(d => d.GetOneByExpression(It.IsAny<Expression<Func<Drug, bool>>>()))Returns(drug);
    _sessionMock.Setup(d => d.GetOneByExpression(It.IsAny<Expression<Func<Session, bool>>>()))Returns(session);

    //Act
    var result = _stockRequestManager.CreateStockRequest(stockRequest, token);
}
```

Código de software reparado

En este caso, se modificó el método *CreateStockRequest* que se encarga de crear el objeto correspondiente a la solicitud de stock y lo agrega dentro de la tabla correspondiente. Por lo tanto, dentro de las validaciones que se realizan antes de enviar el objeto para agregarse a la correspondiente tabla, se agregó una validación para la cantidad. La misma se encarga de ver dentro de todas las solicitudes de medicamentos que va a contener la solicitud de stock, si hay alguna de ellas que contenga un valor de cantidad negativo. En caso de que haya alguna se envía un error de tipo *InvalidResourceException* con un mensaje indicando a qué se debe el error.

Archivo modificado: Código\Backend\PharmaGo.BusinessLogic\StockRequestMangaer.cs



En la siguiente imagen se encuentran resaltadas las líneas de código agregadas:

```
69 public StockRequest CreateStockRequest(StockRequest stockRequest, string token)
70 {
71     User existEmployee = null;
72     if (stockRequest.Details == null) throw new InvalidResourceException("Invalid stock details.");
73     if (stockRequest.Details.Count == 0) throw new InvalidResourceException("Invalid stock details.");
74     var stocks = stockRequest.Details;
75     foreach (StockRequestDetail s in stocks)
76     {
77         if (s.Quantity < 0)
78         {
79             throw new InvalidResourceException("Invalid stock quantity");
80         }
81     }
82 }
```

Evidencia de ejecución de casos de prueba → Etapa GREEN

La siguiente imagen muestra el test ejecutado una vez agregado el mínimo código necesario para que el mismo pase.

✓ CreateStockRequest\_WithInvalidStockQuantity\_ShouldReturnException  
Duración: 255 ms

Ejecutar | Depurar | Mostrar en el Explorador de pruebas

ExpectedException(typeof(InvalidResourceException))

0 referencias | Agustina Lamanna, Hace 21 horas | 1 autor, 1 cambio

```
public void CreateStockRequest_WithInvalidStockQuantity_ShouldReturnException()
{
    //Arrange
    var drug = new Drug() { Id = 1, Code = "XF324" };
    User employee = new User() { Id = 1, UserName = "jcastro" };
    var stockRequest = new StockRequest()
    {
        Id = 1,
        Status = Domain.Enums.StockRequestStatus.Rejected,
        Employee = new User() { Id = 1, UserName = "jcastro" },
        Details = new List<StockRequestDetail>()
        {
            new StockRequestDetail() { Id = 1, Drug = new Drug() { Id = 1, Code = "XF324" }, Quantity = -1 },
        },
        RequestDate = DateTime.Now
    };

    _employeeMock.Setup(u => u.GetOneDetailByExpression(It.IsAny<Expression<Func<User, bool>>>()))Returns(employee);
    _drugMock.Setup(d => d.GetOneByExpression(It.IsAny<Expression<Func<Drug, bool>>>()))Returns(drug);
    _sessionMock.Setup(d => d.GetOneByExpression(It.IsAny<Expression<Func<Session, bool>>>()))Returns(session);

    //Act
    var result = _stockRequestManager.CreateStockRequest(stockRequest, token);
}
```

# Sobre la entrega

## Alcance de la entrega

Durante este intervalo de trabajo se tenía como objetivo redefinir el tablero de tareas así como la definición del proceso de ingeniería. Así como configurar el pipeline mediante Github Actions. Por último, se buscaba seleccionar bugs críticos para el funcionamiento de la aplicación y resolverlos siguiendo el proceso establecido previamente.

## Conclusiones

Se logró cumplir satisfactoriamente con los objetivos planteados al inicio de la iteración durante la etapa de planificación. Se resolvieron los bugs cumpliendo con la definición de done definida logrando así un incremento de valor sobre el proyecto. Además, la implementación de un pipeline de deployment efectiviza el proceso de desarrollo avisando sobre errores en el código antes que puedan causar problemas en el resto de proyecto.

## Métricas

### Lead Time

El lead time mide el tiempo que le lleva a una tarea pasar desde la columna *To Do* hasta *Done*. En esta entrega, todas las tareas se agregaron al *To Do* el día domingo, y fueron finalizadas el día martes. Por lo tanto el **Lead Time fue de 2 días**.

### Cycle time y Touch Time

El cycle time mide el tiempo desde que se comienza la tarea *Doing* o, según la nueva definición del tablero *Designing/In Progress* hasta *Done*. El touch time mide el tiempo en que realmente se trabajó una tarea. Es decir, el cycle time menos el tiempo que estuvo bloqueada. En este caso ninguna tarea fue bloqueada por otra por lo que el touch time es el mismo que el cycle time. En promedio, cada tarea tomó **2,3 horas** en realizarse.

### Esfuerzo total

Cantidad de trabajo total medido en horas-persona.

El esfuerzo total para esta iteración fue de **16,1 horas-persona**.

### Flow Efficiency

Touch time / Lead time = **2,3 horas / 48 horas = 0,05 %**

## Throughput

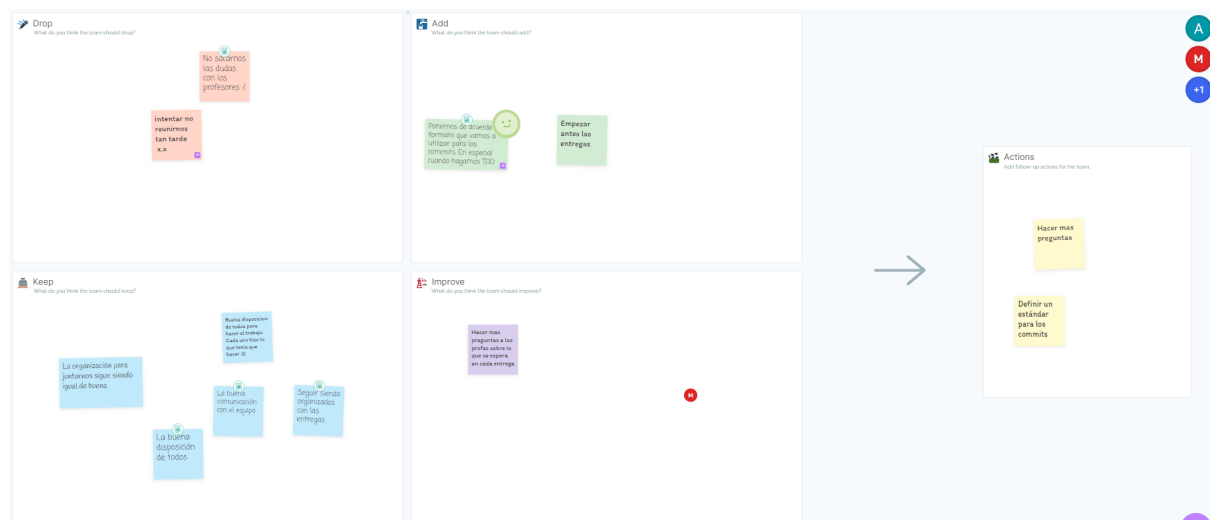
Cantidad de trabajo que se completa en un periodo de tiempo determinado.

Se completaron **9 actividades** que se corresponden con las tarjetas en el tablero, en **3 días**.

## Retrospectiva

Se realizó una reunión entre los tres integrantes del equipo. La misma estuvo guiada por el SM, en este caso Melissa Molinari.

Se llegó a la siguiente learning matrix:



Las principales conclusiones a las que llegamos son:

- Debemos establecer un estándar para los commits de Github, en especial para cuando utilicemos TDD
- Sacarnos las dudas de letra que puedan surgir, con los profesores.

En el siguiente [link](#) se puede encontrar el video de la reunión.

## Review

Se realizó una reunión en donde cada desarrollador le mostró al PO, en este caso Agustina Lamanna, cómo solucionó el bug. En este caso, todas las soluciones fueron aceptadas por el PO ya que cumplían con la definición de done planteada y solucionan correctamente los problemas mencionados en los issues.

En el siguiente [link](#) se puede encontrar el video de la reunión.