

Universidad ORT Uruguay

Ingeniería De Software Ágil 2

Obligatorio

Entrega 2



Estudiantes:

- Emiliano Ruiz - 263136
- Facundo Jaume - 239695
- Marcel Bejerez - 242484

Docentes:

- Álvaro Ortas
- Carina Fontan

Link al repositorio:

<https://github.com/IngSoft-ISA2-2023-2/obligatorio-bejerez-jaume-ruiz>

Índice

Índice.....	2
Informe de avance de la etapa.....	4
Actividades realizadas.....	4
Justificación de decisiones.....	5
Resultados obtenidos en el período.....	5
Dificultades encontradas y formas de solución.....	6
Lecciones aprendidas y mejoras en el proceso.....	6
Anexo.....	7
Definición/uso del proceso de ingeniería en el contexto de KANBAN.....	7
Explicación del tablero y su vínculo con el proceso de ingeniería.....	7
Pequeña guía de "Configuración del pipeline y su vínculo con el tablero".....	8
Identificación y justificación de los bugs en función la clasificación proporcionada.....	8
Código de software reparado.....	8
Bug: Búsqueda de farmacias por espacio en blanco.....	8
Bug: Mensajes de notificación persistentes.....	8
Bug: Solicitud de stock con cantidad negativa.....	9
Código de casos de prueba.....	10
Evidencia de ejecución de casos de prueba.....	11
Fase roja: Crear una request de un medicamento negativo.....	11
Fase verde: Crear una request de un medicamento negativo.....	12
Registro de esfuerzo por tipo de tarea.....	12
Totales de registro de esfuerzo por la entrega.....	13
Video de la retrospectiva con el "SM"	13
Video de la revisión de los bugs con el PO.....	13
Análisis de la retrospectiva.....	14
Drop.....	14
Add.....	15
Keep.....	15
Improve.....	16
Actions.....	16

Informe de avance de la etapa

Actividades realizadas.

En esta entrega en particular, se busca seleccionar tres bugs de los seleccionados en la entrega anterior y resolverlos.

Comenzamos creando un tablero en GitHub. Para esto, primero fue necesario definir cómo íbamos a adaptar el marco de Kanban a nuestro proyecto según nuestras prioridades o más bien necesidades que consideramos para la entrega. La definición del marco general se puede encontrar más detallada en: [Definición de marco general de KANBAN](#).

Sobre el contenido del tablero, desarrollamos con mejor detalle en la sección de [Explicación del tablero y su vínculo con proceso de ingeniería enmarcado en KANBAN](#). Enlistando los atributos que contiene cada tarjeta, junto con una explicación de ciertos puntos que creímos necesarios registrar.

Por otra parte, para llevar a cabo el análisis y selección de bugs para resolver, comenzamos viendo la severidad de estos y a su vez las recomendaciones del Product Owner. Los bugs elegidos y su respectiva justificación se encuentran en la sección de [Identificación y justificación de los bugs en función de la clasificación proporcionada](#).

Para esta entrega, las actividades a resolver se hicieron predominantemente de forma individual. Esto se debe a que, como se mencionó anteriormente, esta entrega constaba de resolver bugs, específicamente uno por integrante.

De todas formas, en ningún momento los integrantes se desentendieron de las otras tareas del proyecto. Es decir, todos estaban pendientes de las otras tareas que se estaban realizando gracias al tablero y a las reuniones que armábamos regularmente. Esto creo que ayudó a lograr un trabajo de gran calidad, complementando y asegurándonos de cubrir muchos puntos y detalles. Esto lo aplicamos para todo: registro de horas, gestión del tablero, etc. Esto probablemente disminuya en ciertas áreas para la próxima entrega, dado que buscaremos separar ciertos roles, sin perder esa ayuda mutua que nos benefició.

En paralelo a todas estas actividades, el equipo se encargó de registrar las horas trabajadas, para poder convertirlas en esfuerzo y así, luego, poder extraer métricas como Lead time o Cycle time de cada tarjeta. Esto está evidenciado con mayor detalle en [Registro de esfuerzo por tipo de tarea y totales de registro de esfuerzo total](#).

El proceso para este registro fue armar una hoja de cálculo con las tareas que se iban realizando y junto a ellas la siguiente información:

- Cant Personas: Indica la cantidad de personas involucradas para la actividad a realizar
- Integrantes: Muestra los integrantes del equipo involucrados
- Fecha: Indica la fecha en la que se realizó la actividad
- Hora Inicio
- Hora Fin
- Duración: Cantidad de tiempo invertido en la tarea
- Horas Persona: Cantidad de tiempo invertido en la tarea por persona

- Total Horas Persona: Es el total de las horas trabajada en el proyecto por persona

Para calcular la Duración, Horas Persona y Total Horas Persona, en vez de calcularlas a mano, optamos por asignar una función para cada columna, y de esta manera obtenerlas automáticamente a partir de los otros campos. Consideramos que llevar este registro y en particular esta solución nombrada anteriormente para no calcular campos a mano va contribuir en gran manera a la hora de calcular las métricas en futuras entregas.

Justificación de decisiones.

En todas las secciones anteriormente mencionadas, siempre justificamos todas las decisiones. En particular, desarrollamos sobre aquellas que tuvimos que adaptar en mayor medida a nuestras necesidades para lograr los mejores resultados.

Continuamos con Kanban como marco de trabajo ágil que no busca ser un contrato, sino una herramienta. Es por esto, que la prioridad siempre va a ser trabajar acorde a lo que el equipo (u organización) considere más beneficioso.

En base a esto, también es necesario hablar sobre la Definición de roles del equipo que tuvimos que llevar a cabo como parte del proyecto para poder trabajar sobre una base organizada con roles y responsabilidades claras. Como indica la letra, todos cumplimos los roles de desarrolladores y testers. Además, definimos un Scrum Master y un Product Owner. En el Anexo justificamos dicha decisión y aclaramos que para futuras entregas vamos a mejorar dicho aspecto y asegurarnos de rotar los roles para que el aprendizaje sea mejor.

Resultados obtenidos en el período.

En cuanto a los resultados obtenidos, los consideramos satisfactorios en base al objetivo planteado para la entrega. Usamos el tablero de gran forma para identificar las tareas y agregarlas al “To do”, así como también para llevarlas hasta “Done”. En esta segunda entrega la distribución entre tareas y bugs fue más pareja. Al igual que en la entrega anterior, siempre estuvieron muy bien organizadas: con atributos claros y útiles, con los responsables asignados, y actualizadas en la columna correspondiente.

Con respecto a esto, creo que también ayudó lo cuidadosos que fuimos al tomar una tarjeta y pasarla a “In progress”, ya que buscamos no pasarnos de un cierto límite en el WIP (Work in Progress). Si bien por momentos pasamos la cantidad de integrantes (que es lo recomendado para que nadie esté trabajando sobre más de una tarjeta a la vez), siempre fue por motivos válidos. En esta entrega, al tener tareas de gestión, muchas veces estas se llevaban a cabo más de una a la vez. Por ejemplo, la redacción del informe y creación del tablero fueron tareas que se fueron llevando a cabo en forma paralela a otras.

En cuanto a la resolución de bugs, se puede encontrar un resumen en la sección Código de software reparado del Anexo.

Dificultades encontradas y formas de solución.

La mayor dificultad que encontramos fue a la hora de automatizar los tests en GitHub, donde nos encontramos con un test que pasaba localmente en Visual Studio, mientras que desde GitHub no. De esto, desarrollamos con mayor detalle en la sección de *Pipeline en GitHub Actions configurado*.

Por otra parte, seguimos con la dificultad de que los IDs no se asignan automáticamente en las tarjetas del tablero. Mantenemos que es de gran importancia identificar las tareas con un código identificador, que en este caso, bastaba con un número.

Por último, en términos más generales, otro de los mayores obstáculos fue el tiempo. Para algunos integrantes más que otros, pero a todos en general nos limitó el tiempo como para llegar al resultado deseado. Una semana donde hay otros compromisos, varias tareas y mucho que investigar con el fin de elaborar una entrega de calidad, nos resultó acotado. Estamos conformes dentro del marco dado, pero no con respecto a lo solicitado.

Lecciones aprendidas y mejoras en el proceso.

Como conclusión del proyecto, llevamos a cabo la retrospectiva como sugiere el marco de Kanban. Buscamos analizar el proceso de trabajo en el proyecto, con el fin de identificar puntos de fortaleza y puntos de mejora.

Con todos los integrantes del equipo, realizamos la ceremonia, donde el Scrum Master se encargó de grabar y elegir la plantilla, de modo que todos pudieran brindar sus opiniones y aportar para crecer como equipo. Todo el análisis de este proceso se puede encontrar en la sección del Anexo, *Análisis de la retrospectiva*. Ahí, también se encontrará la evidencia.

Anexo

Definición/uso del proceso de ingeniería en el contexto de KANBAN.

El proceso de ingeniería correspondiente a esta entrega consiste en: Selección de bugs a resolver, configuración del pipeline en github actions, diseño de testing unitario, diseño de solución, codificación y prueba.

La ejecución de este proceso de ingeniería se da en el contexto de Scrum como marco de gestión del proyecto.

El equipo se compone de tres integrantes, de los cuales Emiliano adopta el rol de PO, DESA y TES, Facundo toma el rol de SM, DESA y TES y Marcel cumple el rol de DESA y TES. Consideramos importante destacar que esa definición de roles surge de que a lo largo de las entregas intentamos todos ocupar todos los roles, para así lograr un aprendizaje más homogéneo a lo largo del proceso.

Los métodos y artefactos utilizados en esta entrega son: TDD para el estilo de codificación y testing, y github actions para la automatización del pipeline.

Explicación del tablero y su vínculo con el proceso de ingeniería

En cuanto a este aspecto, hemos realizado una redefinición del tablero ágil para que se adapte mejor a los objetivos de esta entrega, centrada en la corrección de errores. En este sentido, hemos decidido expandir la columna 'InProgress' en varias columnas explicadas más adelante, no obstante de esto quisimos mantener esta columna, ya que las tasks transitan por esta etapa, pero también hemos considerado la posibilidad de añadir columnas adicionales para los issues.

Las columnas 'ToDo', 'Done' y 'Product Backlog', que fueron explicadas en entregas anteriores, se mantienen sin cambios.

Entre las nuevas incorporaciones, destacamos la columna 'Design & TDD', donde las tarjetas indican que se está discutiendo la forma de diseñar la implementación del cambio. También es posible que esta etapa haya concluido y se esté avanzando a la primera fase de TDD, antes de pasar a la siguiente columna.

La columna siguiente es 'Programming', que, como su nombre indica, tiene las tarjetas en pleno proceso de codificación e implementación.

Finalmente, se ha introducido la columna 'Testing', que básicamente recoge las tarjetas que han pasado por las etapas anteriores, donde se ha implementado o corregido un error. En esta fase, se llevan a cabo pruebas por parte del equipo para verificar los nuevos cambios.

En sintonía con la evolución del proceso de ingeniería para esta entrega, hemos realizado ajustes significativos en nuestro tablero ágil descritos arriba. La redefinición tuvo como objetivo reflejar de manera más precisa las etapas del proceso, así como facilitar la colaboración y la visibilidad de cada tarea.

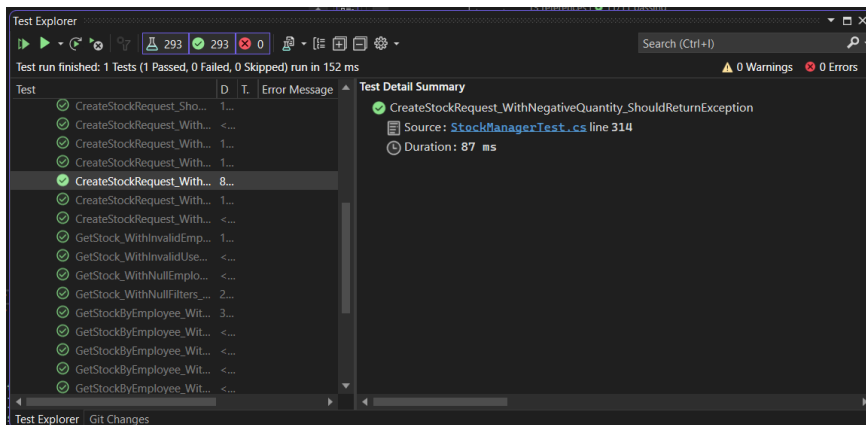
Pipeline en GitHub Actions configurado

Con respecto a este punto, pudimos lograr la automatización del build, y los test. Tuvimos la dificultad de que hay un test que no nos corre, este sería el `GetOneByExpression`.

```
441 Starting test execution, please wait...
443 A total of 1 test files matched the specified pattern.
444 Failed Test_GetOneByExpression [54 ms]
445 Error Message:
446 Assert.IsNotNull failed.
447 Stack Trace:
450 at PharmaGo.Test.DataAccess.Test.PurchasesDetailRepositoryTest.Test_GetOneByExpression() in D:\a\obligatorio-bejerez-jaume-ruiz\obligatorio-bejerez-jaume-ruiz\Implementacion\Codigo\Backend\PharmaGo.Test\DataAccess.Test\PurchasesDetailRepositoryTest.cs:line 92
451
450 Failed! - Failed: 1, Passed: 292, Skipped: 0, Total: 293, Duration: 3 s - PharmaGo.Test.dll (net6.0)
451 Error: Process completed with exit code 1.
```

Este punto va a quedar a revisión para la siguiente entrega debido a que no nos dio el tiempo para revisarlo para esta.

De igual modo dejamos adjunto la evidencia que de modo local nos corren todos los test:



Del proceso que tuvimos que hacer para lograr esto se habla en el siguiente punto.

Pequeña guía de "Configuración del pipeline y su vínculo con el tablero"

Para esta parte creamos un archivo `.yaml` dentro de la carpeta `.github/workflows` que contenía el script para la automatización del build y testing unitario.

Configuramos el script para que se realice cada vez que se hacía un push en cualquier rama. De esta manera logramos automatizar la columna de testing del tablero.

De todas maneras, como se mencionó anteriormente, tuvimos dificultades para lograr que el script funcione 100% bien, ya que el resultado indica que un test falla. Esto se puede ver mejor en [Pipeline en GitHub Actions configurado](#).

Identificación y justificación de los bugs en función la clasificación proporcionada

Elegimos tres bugs teniendo en cuenta la severidad de cada uno y las prioridades definidas por el PO.

Finalmente optamos por resolver:

- **Búsqueda de farmacias por espacio en blanco:** Este bug generó muchos problemas ya que, sin intención, algunos admins agregaban farmacias con un nombre en blanco al sistema, lo que no es correcto.

- **Mensajes de notificación persistentes:** Este bug fue indicado como prioridad por el PO, ya que era muy molesto tener que eliminar manualmente cada mensaje de notificación.
- **Solicitud de stock con cantidad negativa:** Este bug fue seleccionado debido a que se daba la opción de solicitar un stock de cantidad negativa, lo que en el contexto del negocio no tiene sentido, por lo que se entendió prioritaria su resolución.

Código de software reparado

Para la tarjeta de agregar una farmacia con el nombre en blanco, el cambio que se hizo es agregar una validación más, que sería la siguiente.

Bug: Búsqueda de farmacias por espacio en blanco

```
2 referencias
public void ValidOrFail()
{
    if (string.IsNullOrEmpty(Name) || Name.Length >= 50 || string.IsNullOrEmpty(Address)
        || Users == null || Drugs == null || Name.Equals(" "))
    {
        throw new InvalidResourceException("The Pharmacy is not correctly created.");
    }
}
```

```
2 referencias
public void ValidOrFail()
{
    if (string.IsNullOrEmpty(Name) || Name.Length >= 50 || string.IsNullOrEmpty(Address)
        || Users == null || Drugs == null)
    {
        throw new InvalidResourceException("The Pharmacy is not correctly created.");
    }
}
```

Bug: Mensajes de notificación persistentes


```

constructor(
    private commonService: CommonService
) {
    var self = this;
    this.commonService.onToastDataUpdate.subscribe((data: any) => {
        if ((Object.keys(data).length !== 0) && data.message !== "") {
            self.visible = true;
            self.message = data.message;
            self.color = data.type;
            self.title = data.title;
            setTimeout(function() {
                self.visible = false;
                self.commonService.updateToastData("", "", "");
            }, 7000);
        }
    });
}

```

Bug: Solicitud de stock con cantidad negativa

A continuación mostramos el código anterior donde item a item de los detalles valida que la solicitud sea sobre un medicamento válido.

```

foreach (StockRequestDetail item in stockRequest.Details)
{
    var drug = _drugRepository.GetOneByExpression(d => d.Code == item.Drug.Code && d.Pharmacy.Id == existEmployee.Pharmacy.Id);
    if (drug == null) throw new InvalidResourceException("Stock request has invalid drug.");

    item.Drug = drug;
}

```

Por lo tanto, lo que agregamos en este punto fue la validación de que la cantidad que se solicita sea positiva, de lo contrario, lanza una excepción con el mensaje “Stock request has negative quantity”. Esto se puede ver en la siguiente foto. Cabe aclarar que esto es una validación de respaldo como para asegurarnos que no se ingresen datos en la base de datos incoherentes. Es decir, esto ya es validado en el front, pero consideramos oportuno agregar esto para tener un sistema más robusto.

```

foreach (StockRequestDetail item in stockRequest.Details)
{
    var drug = this._drugRepository.GetOneByExpression(d => d.Code == item.Drug.Code && d.Pharmacy.Id == existEmployee.Pharmacy.Id);
    if (drug == null)
    {
        throw new InvalidResourceException("Stock request has invalid drug.");
    }
    if (item.Quantity < 1)
    {
        throw new InvalidResourceException("Stock request has negative quantity.");
    }

    item.Drug = drug;
}

```

Como mencionamos, para este bug fue necesario arreglar el front, mostrando un mensaje de advertencia cuando se escribe un número negativo y mostrando un mensaje de error cuando se intenta agregar. Por lo tanto, ni siquiera deja agregarlo a la lista de requests.

```

addRequest(): void {
  if(this.quantity<1){
    this.commonService.updateToastData("Quantity must be greater than 0", "danger", "Error");
  }else{
    let exist: boolean = false;
    for(let item of this.requests){
      if(item.name === this.name){

```

```

<c-col class="mb-3">
  <c-input-group class="mb-3">
    <label cInputGroupText for="quantityInput">
      Quantity
    </label>
    <input
      cFormControl
      [(ngModel)]="quantity"
      id="quantityInput"
      value="1"
      min="1"
      type="number"
    />
  </c-input-group>
  <span class="custom-error" *ngIf="quantity<1">Quantity must be greater than 0</span>
</c-col>

```

Código de casos de prueba

Para la tarjeta de agregar una farmacia con el nombre en blanco, se hizo el siguiente test.

```

112
113 [TestMethod]
114 [ExpectedException(typeof(InvalidResourceException))]
115 0 referencias
116 public void CreateBlankNamePharmacy()
117 {
118     pharmacy.Name = " ";
119     var pharmacyReturned = _pharmacyManager.Create(pharmacy);
120 }

```

Evidencia de ejecución de casos de prueba

Para la tarjeta de agregar una farmacia con el nombre en blanco, se hizo el test para el cual adjuntamos la evidencia de que este test dio verde.

The screenshot displays the Visual Studio IDE with a C# code file and the Test Explorer window.

Code File:

```

112
113     [TestMethod]
114     [ExpectedException(typeof(InvalidResourceException))]
115     public void CreateBlankNamePharmacy()
116     {
117         pharmacy.Name = " ";
118         var pharmacyReturned = _pharmacyManager.Create(pharmacy);
119     }
120
121     [TestMethod]
122     [ExpectedException(typeof(InvalidResourceException))]
123     public void CreateLargeNamePharmacy()

```

Test Explorer Window:

The Test Explorer window shows a summary of test results:

- Pruebas de pruebas finalizada:** 1 pruebas (Superadas: 1; Con errores: 0; Omitidas: 0) ejecutadas en 774 ms
- Resumen del grupo:**
 - PharmaGo.Test
 - Pruebas en grupo: 292
 - Duración total: 695 ms
- Salidas:**
 - 291 No ejecutada
 - 1 Correcta

The Test Explorer also lists individual test methods, including `CreateBlankNamePharmacy` which is marked as passed (green checkmark).

Fase roja: Crear una request de un medicamento negativo

Para esta sección empezamos creando la prueba, definiendo el stock request con sus respectivos datos (Arrange). En el primer detalle de Details, definimos un StockRequestDetail con Quantity negativa (-50).

Luego, establecimos los mocks para aislar el comportamiento de las dependencias y probar simplemente el método que nos interesaba.

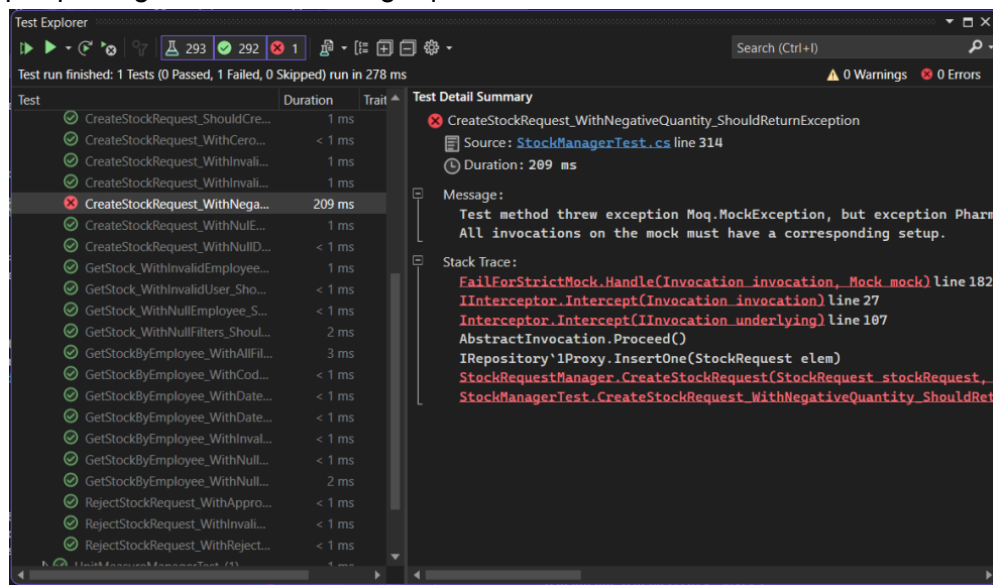
Por último, llamamos al método (Act), esperando que en cierto punto lanzara una excepción de tipo `InvalidResourceException` al chequear si la `Quantity` de todos los detalles eran mayores que cero.

```
[TestMethod]
[ExpectedException(typeof(InvalidResourceException))]
// 0 references
public void CreateStockRequest_WithNegativeQuantity_ShouldReturnException()
{
    //Arrange
    Drug? 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 = -50 },
            new StockRequestDetail() { Id = 2, Drug = new Drug() { Id = 2, Code = "RS546" }, Quantity = 25 }
        },
        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);

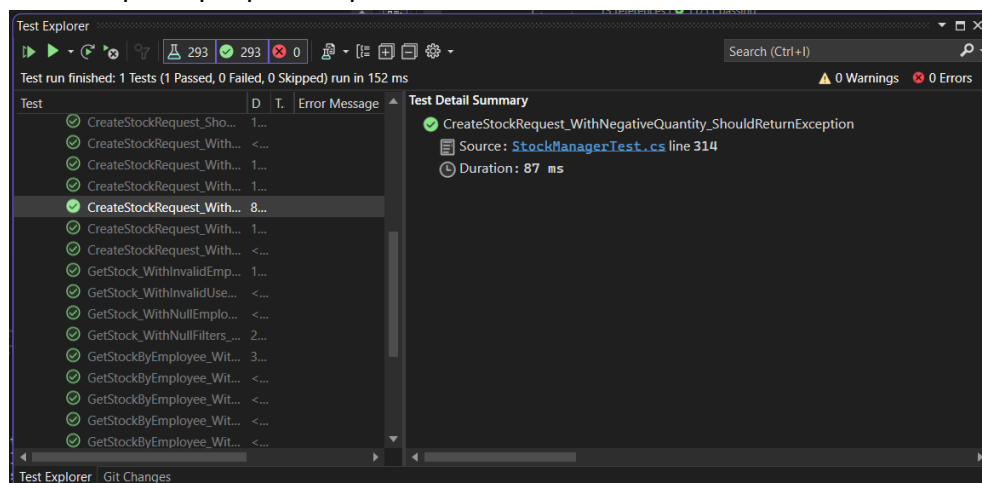
    //Act
    var result = _stockRequestManager.CreateStockRequest(stockRequest, token);
}
```

Acá se puede ver como el test comienza tirando MockException porque llega a líneas de código que no deseamos.



Fase verde: Crear una request de un medicamento negativo

Luego, al aplicar los cambios que ya mencionamos anteriormente llegamos a la fase verde. Es decir, implementamos los mínimos cambios para que pase la prueba.



Por último, aplicamos refactor para mejorar la legibilidad y calidad del código, sin alterar la funcionalidad.

Registro de esfuerzo por tipo de tarea

Para registrar el esfuerzo para cada tarea decidimos continuar con la misma hoja de cálculo en Google Sheets para anotar la tarea que realizamos, pero separamos por tablas para cada entrega, esta decisión la tomamos ya que nos va a ser más sencillo ver las métricas por entrega más adelante.

Quedando así esta segunda entrega:

Tarea	Esfuerzo(HP)
Agregar prioridad a las tarjetas	0:30
Crear tabla para el registro de horas	0:30
Configuración del Pipeline	2:00
Selección de bugs a resolver	0:30
Subir proyecto al repositorio de github	0:30
Reparación bug mensajes persistentes	2:00
Automatización del pipeline	8:00
Redacción Informe	27:30
Reparación bug Farmacia nombre en blanco	0:30

Totales de registro de esfuerzo por la entrega

En total, como se observa en la tabla, el equipo alcanzó un esfuerzo de 47 horas-persona.

Video de la retrospectiva con el “SM”

<https://youtu.be/WTH2Fwt2Y8g>

Video de la revisión de los bugs con el “PO”

https://youtu.be/_psYgtvPYtM

Análisis de la retrospectiva.

Al igual que la entrega pasada, nos reunimos para llevar a cabo esta ceremonia y nuevamente discutir, evaluar y tomar acciones sobre los procesos realizados durante esta entrega. Otra vez, utilizamos la herramienta Metro Retro y elegimos la plantilla DAKI (Drop, Add, Keep, Improve).

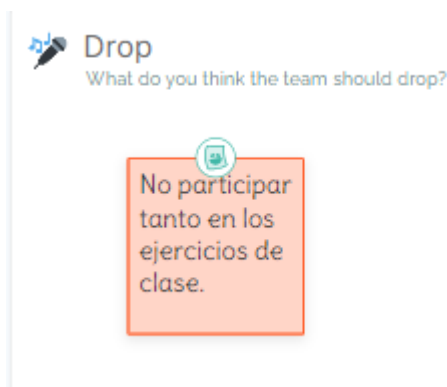
Siguiendo la misma metodología que la anterior retrospectiva, nos conectamos por zoom utilizando la herramienta metro retro. Y otra vez nos dimos tiempo para escribir las tarjetas en privado, al cabo de un tiempo las dimos vuelta y discutimos de cada punto, empezando con Drop.

Después de la retrospectiva, quedó como resultado una lista de acciones para potenciar el rendimiento del equipo en las próximas entregas del proyecto. En resumen, consideramos que llevamos a cabo la ceremonia de manera exitosa; se siguió manteniendo el mismo ambiente, en todo momento, los integrantes se sintieron a gusto para compartir comentarios sobre mejoras o contribuciones al proceso de trabajo.


Para la próxima entrega, entonces, vamos a intentar: Tener una planning al principio ya sea para organizarnos mejor tanto en lo que vamos hacer o en métricas y estándares como por ejemplo de codificación o de branching, también queremos estar más participativos en los ejercicios de clase que son para entendimiento del obligatorio, hacer más preguntas si tenemos dudas, y por último tomar más en cuenta las validaciones para los merge.

Por último, dejamos la evidencia de los distintos aspectos de la ceremonia.


Drop




Add

 **Add**
What do you think the team should add?

Una planning mas formal y organizada





Definir algunos estandares mas de mensajes de commit y cosas de esas para cuando escale



Hacer las cosa con un poco mas de orden.


Keep

 **Keep**
What do you think the team should keep?



Algo que esta muy bueno para seguir haciendo, es el registro de horas bien especifico en excel.

El buen manejo del tablero



mantener el buen trabajo en equipo y la buena distribución de tareas

Improve




Improve

What do you think the team should improve?

Validar con
mas tiempo
dudas

Validar un
poco mas el
trabajo del
resto

 cuidar un
poco mas la
estrategia de
branching

Estudiar
bien los
artefactos
dados


Hacer
más
dudas


Actions





Actions

Add follow-up actions for the team.

 Intentar
hacer una
planning

 Prestar
mas
atención a
clase

 Hacer mas
dudas.

 Tomar mas
en cuenta
la
validación.