Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio Ingeniería de Software Ágil 2
Entrega 5, Equipo 6

Link al repositorio

Fabio Orlinski Nº 173630

Andrés Fraga Nº 151351

Santiago Irazoqui Nº 234730

Profesores: Álvaro Ortas, Carina Fontán
Grupo: N7A

Informe académico final

1. Resumen de gestión del proyecto.

Dado que el proyecto realizado este semestre fue subdividido en cuatro entregas, la gestión, actividades y productos entregados fueron cambiando en cada una de ellas.

Para cada entrega en primer lugar se realizó un pequeño informe de avance con un resumen de los objetivos propuestos para la misma, y el equipo escribió también las conclusiones y aprendizajes que pudimos sacar en cada una de ellas. Si bien la gestión fue cambiando, algunos de los entregables se mantuvieron en casi todas las entregas.

Por ejemplo, en cada iteración se documentó la “definición y uso del proceso de ingeniería en el contexto de Kanban”, en el cual se detallan las decisiones tomadas en cuanto a la gestión ágil. Incluye información, como las decisiones tomadas sobre los tableros, y metodologías usadas para trabajar con BDD. Por otro lado, en cada entrega se entregó también un registro de esfuerzo con las tareas realizadas por cada integrante y el esfuerzo realizado para cada una de ellas. Otro producto que se mantuvo a lo largo del proyecto es la retrospectiva. Al final de cada entrega el equipo realizó una reunión con el objetivo de analizar y mejorar el funcionamiento del equipo en el marco de gestión ágil. Para realizarlas, seguimos el formato DAKI para identificar las cosas a mejorar, mantener, agregar y dejar. Para terminar identificamos las acciones que el equipo consideró importante tomar luego de compartir opiniones de lo anterior. Por último, también se documentó en cada oportunidad el funcionamiento del tablero ya que este cambió bastante dado que cambiaron los objetivos y metodologías de desarrollo.

La primera entrega se centró en el análisis de la deuda técnica, por lo que intentamos identificar bugs y posibles mejoras, estas últimas separadas en mejoras de diseño de la arquitectura y problemas con estándares (cómo podría ser Clean Code). Todos estos fueron reportados como issues en el repositorio de Github. Para los bugs, se les dio una descripción, y creamos nuevas tags para clasificar estos bugs según su Prioridad (el tiempo en el que el bug debería ser resuelto) y Severidad (que tan crítico es el error, si impide el funcionamiento de la app o algo menor). En total fueron creadas 8 tags, 4 para cada clasificación. También se crearon dos nuevas tags para asignarle a las issues de problemas de estándares y los problemas de diseño. Además de reportar las issues en el repositorio, también se entregó un documento escrito con las mismas.

Una de las primeras cosas que tuvimos que definir fue la estructura del repositorio y mantenimiento del mismo. Por ejemplo, una de las discusiones que se planteó fue si utilizar GitFlow o Trunk Based Development en el proyecto. Dado que todos teníamos conocimientos

previos de GitFlow, y poca familiarización con el desarrollo Trunk Based, decidimos optar por la primera. Sobre esta decisión también pesó que en nuestro equipo hay una diferencia de conocimiento en cuanto a las estrategias de codificación, lo que podría significar problemas si estamos haciendo merge a una rama principal, y consideramos que tener cada cosa encapsulada en su rama sería mejor. En este mismo documento también se encuentran definidas otras cosas importantes como la creación de PR, la nomenclatura de ramas, etc. Para la nomenclatura, dividimos en 5 categorías las posibles ramas: docs, devops, feature, fix, hotfix. Esto lo hicimos para poder reconocer rápidamente que cambios tiene cada una, y poderlos ordenar y encapsular mejor.

En cuanto al tablero, esta fue la única entrega en la que utilizamos un tablero ágil básico. Dado que las tareas en este caso varían mucho en su flujo de trabajo (documentación es muy diferente a proceso de ingeniería por ejemplo), utilizar un tablero sustentable podría llevar a confusiones. El tablero ágil en cambio, facilita la lectura de estas ya que no nos interesa en que “etapa” se encuentra.

En la segunda entrega el enfoque paso en primer lugar a la corrección de bugs identificados en la primera entrega. Para esto, se seleccionaron tres bugs considerando su clasificación de prioridad y severidad, y se justificó por qué la elección de los mismos. El desarrollo en este caso fue realizado siguiendo TDD. Para demostrar esto, se documentó también el código de los nuevos tests, junto con fotos de las fases rojas y verdes.

Otro de los objetivos de esta entrega fue la automatización del pipeline utilizando Github Actions. Esto nos permitió automatizar algunas partes del flujo, lo que nos dió más seguridad y feedback. El equipo decidió automatizar dos aspectos para esta entrega: el build y los tests, los cuales se ejecutan cuando se dispara una acción de "push" o un "pull request" en las branch main, develop o release. Todo esto, junto con otra información, fue documentado en un artefacto.

Dado que los objetivos de esta entrega pasaron a ser de desarrollo, lo primero que se replanteó es el tablero de Kanban, el cual pasó a ser un tablero sustentable. La razón de esto fue principalmente que este tablero es muy bueno para observar claramente la etapa de desarrollo en la que se encuentra una tarjeta. Esto es también importante para el equipo ya que no todos los integrantes tienen los mismos conocimientos en el área de desarrollo y TDD, por lo que saber si una tarea se encuentra trancada y en dónde pasó a ser importante. Por otro lado, gracias a que las tarjetas se definieron bien como US, y el flujo de trabajo sería el mismo para todas, utilizar el tablero sustentable fue posible.

Algo que se sumó en esta entrega y se va a mantener por el resto del proyecto, es la revisión con el PO. Estas reuniones las hicimos por Teams, en la que uno de los integrantes simulaba ser el PO. En esta etapa, la reunión fue para realizar la revisión de los bugs y que los cambios sean aceptados. Este video fue grabado y documentado como artefacto.

Para la tercera entrega, nos enfocamos en la implementación del CRUD de un producto, como además la compra de uno por parte de un cliente final, que fueron indicadas por los profesores. Sin embargo, en esta oportunidad utilizamos BDD, una nueva metodología ágil, junto con la herramienta Specflow y Gherkin para definir y ejecutar pruebas de comportamiento.

SpecFlow nos permite escribir escenarios de prueba en lenguaje natural y automatizar las pruebas en función de esos escenarios. Esto garantiza que nuestras características se alineen con las expectativas del usuario y que el software cumpla con los requisitos definidos. Además, esto significó un cambio en el tablero, ya que varias columnas del tablero sustentable pasaron a ser las etapas de BDD.

En vez de tener las columnas del tablero sustentable diseño, desarrollo y review (más el To Do y Done), ahora tenemos una por cada etapa de BDD: CCC, Test case implementation, App implementation, Testing, Refactoring.

Finalmente, para la cuarta entrega, nuestro objetivo paso a ser la automatización de testing exploratorio utilizando la herramienta Selenium. Por otro lado, también realizamos un análisis de métricas de DevOps con las métricas que hemos ido guardando a lo largo del proyecto.

Con esto, pudimos fácilmente automatizar muchos de los tests que anteriormente teníamos que hacer de forma manual, ahora teniendo que realizarlos solo una vez para grabar las pruebas. Estas pruebas exploratorias están en el nivel más alto de la pirámide de testing que vimos en el curso, y tenerlas automatizadas es un gran beneficio ya que realizar paso a paso cada escenario definido para cada feature toma mucho tiempo. Tener estas pruebas no solo agiliza y brinda más confiabilidad a los desarrolladores, sino que también es útil para el cliente, ya que pueden ver como las pruebas se ejecutan y ver que todos los requerimientos funcionales se están cumpliendo.

Finalmente, se realizaron las métricas de la entrega 2 y 3 en las cuales se realizaron las correcciones de bugs y creación de nuevas funcionalidades, cuyo detalle se puede encontrar en los Anexos 6.1 y 6.2 y cuyo análisis en profundidad se deja dentro del detalle de la entrega 4 en github.

2. Reflexiones sobre el aprendizaje

2.1 Aplicar un marco de gestión ágil

Sobre las cosas a mejorar, en primer lugar están las retrospectivas. Si bien algunas las hicimos realizamos como deberíamos, nos paso en algunas entregas de no tener mucho tiempo y tener que hacerlas apurados, lo que no termina sirviendo mucho. Por otro lado está el mantenimiento

del tablero, que no siempre mantuvimos muy bien. En las primeras entregas fue uno de los errores que destacamos en la retrospectiva y que intentamos mejorar.

2.2 Analizar la deuda técnica

Consideramos que hicimos bien la separación de tareas para este análisis. Primero identificamos y separamos cada requerimiento del software detallado en la letra para encontrar los bugs. Para los estándares separamos en Frontend y Backend, y el Backend luego en cada proyecto. Esto nos aseguró que no nos faltara ningún requerimiento ni ningún área del código sin revisar.

Algo que resultó difícil es separar las mejoras personales personales de lo que realmente son bugs o problemas de diseño / estándares. A veces miramos algo que no nos convence, o consideramos que falta algo, y nos sentimos impulsados a reportarlo, cuando quizás eso no es algo que se hubiera pedido o simplemente es una perspectiva diferente. Es importante mejorar nuestra habilidad para reconocer cuando algo está rompiendo con alguno de los requerimientos para que efectivamente sea un bug.

2.3 Crear un pipeline con eventos y acciones

Aprendiendo cómo utilizar github actions. Aprender a utilizar github actions fue algo muy interesante, y algo que nos gustaría continuar.

Explorar más formas de automatización del pipeline, adaptarlo mejor a cada etapa
Consideramos que algo que nos faltó es explorar más formas en las que las github actions (u otras formas de automatización del pipeline) pueden ser aplicadas en el flujo de desarrollo, y cómo los cambios en el proyecto (como metodologías de desarrollo utilizadas, estándares utilizados, etc.) puede introducir nuevas formas de automatización.

2.4 Integrar prácticas de QA en el pipeline y gestionar el feedback

Debido al alcance de la materia no pudimos profundizar en la integración de los tests exploratorios dentro del pipeline, es una de las cosas que nos hubiera gustado haber tenido de realizar porque le daba un cierre a este tipo de práctica.

A la hora de realizar los test exploratorios nos dimos cuenta de la dificultad a la hora de seleccionar un elemento específico dentro del DOM, no tuvimos el tiempo suficiente para agregar etiquetas html para que nos sirvan a la hora de identificar un tag específico y hacer que las pruebas sean más fáciles de comprender solamente leyendo el código.

2.5 Generar escenarios de testing desde la perspectiva del usuario

Algo que nos resultó difícil a la hora de codificar los tests siguiendo escenarios es sacar nuestra cabeza de TDD. Veníamos muy acostumbrados a tener un método de test con una parte de Arrange, Act y Assert. Pero utilizando Specflow, cada “step” del escenario está separado, de tal manera que se encapsula el comportamiento esperado. Esto generó confusiones a la hora de programar porque esperábamos que funcionara igual que TDD. Más de una vez miramos un step y nos preguntamos “qué se supone que ponga acá”, o queremos meter todo el test en solo un step, etc.

Por esto algo que nos parece importante mejorar es que los tests detrás de los escenarios reflejen verdaderamente cada una de sus partes, de tal forma de que podamos asegurarnos de que está brindando valor al usuario y se testea acorde a ello.

2.6 Automatizar el testing funcional o de caja negra

Al crear los tests para que demostrar que los controles de escenarios de falla estaban contemplados, nos encontramos con la dificultad de que se cree un nuevo id de 5 dígitos nuevo e irrepetible, nos hubiera gustado haber encontrado alguna manera de encontrar algún id único (como lo hace al crear un UUID en otro tipo de tests), enfocado a las limitaciones de 5 números que requería el producto

2.7 Reflexionar sobre DevOps

Creemos haber logrado una comprensión suficiente sobre DevOps y las posibilidades que hay al aplicarlo enfocado a los requisitos de un proyecto específico, que a lo largo de este obligatorio tuvimos que ir moldeando para cada entrega.

3. Lecciones aprendidas

“Explorar en más profundidad como los cambios en el proyecto afectan al pipeline de desarrollo”
Esto es algo que nos costó, y se vio reflejado en que nuestro pipeline no cambió mucho a lo largo del proyecto.

“Las retrospectivas son importantes, no hay que esperar al último día para hacerlas”
Fallamos un poco con algunas de las retrospectivas, y gran parte de la razón de esto es que siempre esperamos a estar cerca del final de una entrega para realizarla (último día en general). Sin embargo, perfectamente se puede hacer un poco antes, ya que el funcionamiento del equipo a lo largo de la etapa no va a cambiar en los últimos días.

“Los bugs son requerimientos del software que no se comportan como deberían”

Si encontramos algo que nos parece mal, pero no es un requerimiento del software, entonces o es una mejora de diseño, o un problema con los estándares, o una mejora que uno personalmente considera mal.

“Para analizar la deuda técnica, reconocer y separar los requerimientos detallados en la letra primero”
Como un bug es un requerimiento que no funciona como debería, ponerse a buscarlos sin conocer bien los requerimientos no tiene sentido. Nosotros lo primero que hicimos fue listar cada uno de ellos, y luego creamos una tarea en el tablero para cada uno. De esta manera pudimos estar seguros de que cada requerimiento fue chequeado.

4. Conclusiones

4.1 Conclusiones del equipo

Gracias a nuestras distintas experiencias, cada uno pudo aportar en su área de mayor expertise, pudiendo entre nosotros siempre evacuar dudas y aplicar los conocimientos de la mejor forma posible, ya sea sobre documentación, estándares y formas de codificación y testing y uso óptimo del repositorio. Un punto muy importante a destacar, fué el poder dividirnos las tareas de forma equitativa, uniendo en ciertos casos dos integrantes del equipo, uno con mayor conocimiento que el otro, para poder finalizar la tarea haciendo pair programming, lo cuál nos daba un sentido de completitud, ya que no fue solamente tomar tareas según conocimiento, sino también por desconocimiento; y alcanzar la meta no se tomaba sólo como un punto terminado para la próxima entrega, sino donde el compañero pudo aprender y/o fortalecer el conocimiento, de cómo realizar ese procedimiento y el mas experimentado en el tema también lograba de esta manera afianzarlo, para poder aplicarlo y a su vez explicarlo de una forma fácilmente comprensible.

4.2 ¿Qué significan los resultados de lo investigado, aplicado y/o solucionado?

Los resultados reflejan un enfoque metódico en la gestión del proyecto. Desde la identificación de deudas técnicas hasta la implementación de pruebas automatizadas y métricas de DevOps, cada fase ha contribuido al desarrollo y mejora continua del software que nos fue asignado. Se abordaron aspectos clave, como la corrección de bugs, la automatización del pipeline y la integración de prácticas de QA, evidenciando un compromiso con el producto final.

4.3 ¿Qué consecuencias/implicancias tiene lo anterior y por que son importantes?

Las consecuencias de este enfoque son varias, como por ejemplo, la identificación temprana de deudas técnicas permitió abordar problemas fundamentales en las primeras etapas, evitando así que se acumulen y perjudiquen el avance de los entregables. La implementación de Github Actions automatizó procesos cruciales, que nos brindaron mayor seguridad y eficiencia en el desarrollo en cada commit/pull request. La introducción de BDD y Selenium mejoró y facilitó las prácticas de testing, asegurando un software más robusto y confiable.

4.4 ¿A dónde nos conducen?

Estos resultados nos conducen a una comprensión más profunda de la importancia de la gestión ágil en el desarrollo de software. Se destaca la necesidad de mantener un equilibrio entre la flexibilidad de adaptarse a cambios y la rigurosidad en la planificación y ejecución. Además, nos orientan hacia la continuación de prácticas exitosas, como la automatización del pipeline, la integración de prácticas de QA y la reflexión constante a través de retrospectivas.

5. Guía de instalación para desarrollo y despliegue en producción

Las releases de Frontend, backend y backups de la db se encuentran dentro de la carpeta *Aplicación*. El código dentro de la carpeta *código*.

1. Restaurar la base de datos con los .bak y .sql provistos
2. Correr el backend desde el ejecutable del release
3. Correr el front. Hay dos formas de hacerlo:
    a. Utilizando el comando ng serve
        i. Abrir una terminal en la carpeta del código del front
        ii. Ejecutar el comando *ng serve*.
        iii. Ir a la dirección de localhost indicada en la terminal
    b. Levantar un web server local (Vamos a detallar usando IIS Express. Probamos con otras formas de levantar servers locales como live server pero no pudimos hacerlo funcionar)
        i. Copiar la carpeta con el build de la aplicación en *C:\inetpub\wwwroot*
        ii. Abrir IIS Express
        iii. Crear un nuevo sitio. La ruta de acceso física debe ser la carpeta que copiamos en wwwroot.
        iv. Examinar el sitio en el puerto seteado

Si hay que cambiar la dirección de la ApiUrl del front (y se está corriendo desde el release), hay que cambiar la variable Fi_apiUrl en el archivo main.js (buscando localhost se encuentra también).

Observación (Explicación de un bug y solución):
Haciendo el deploy encontramos un bug importante que nunca nos había saltado y no nos dimos cuenta mirando el código. Las consultas de los servicios llamados en la página /home se realizan todas con un token de autorización que viene del login. Esto ya venía del obligatorio original y seguimos con la misma lógica sin darnos cuenta. El problema es que un usuario anónimo no hace login, por lo que el token no se encuentra seteado, y se envía vacío en las consultas. La consulta por *drugs* funciona sin problemas ya que en el back, el filtro de autorización no se corre a nivel de controlador, solo algunos métodos que lo necesitan. Sin embargo, al no darnos cuenta de esto nuestro controlador de *products* sí tiene este filtro a nivel de controlador. Esto significa que cuando se entra a la página, se intenta hacer un *GetAll* de los *products*, pero este falla en el filtro ya que no tiene el token.

Este error nunca salió a luz ya que localhost no borra lo que haya en el localstorage, no tiene fecha de expiración. Dado que nosotros ya habíamos hecho logins desde el principio del proyecto, el token siempre estuvo seteado. Por eso nunca nos saltó el error.

La solución es fácil, hay que cambiar que el filtro de autorización sea a nivel de los métodos que lo requieren. Sin embargo, para no estar haciendo cambios a última hora (y que no corresponden en esta entrega), la solución fácil es hacer un login antes de probar algo en *home* (por ejemplo: Usuario: “employee20”, Contraseña: “Aqwertyu2.”). De esa manera ya queda en localstorage un token guardado, y luego se puede entrar a *home* como siempre, ya que el token en localstorage no impacta (más allá del error) la experiencia del usuario anonimo.

5.1 Ejecución de casos de prueba

5.1.1 Ejecución de tests unitarios

1. Abrir una terminal en la carpeta del código del backend
2. Ejecutar el comando *dotnet test*

Se muestran por separado los tests unitarios realizados siguiendo TDD y los tests realizados siguiendo BDD con SpecFlow.

5.1.2 Guía de instalación de Selenium IDE

1. Ingresar a www.selenium.dev
2. Hacer click en la pestaña Descargas (Downloads)

3. Desplazarse hasta la sección de Selenium IDE y descargar la versión correspondiente al navegador que se va a utilizar para grabar los casos de prueba.
4. Luego se añadirá la extensión al navegador.

Observación: Es recomendable también descargar la extensión SelectorsHub, la cual brinda más opciones a la hora de buscar identificadores de elementos, pudiendo así no utilizar los asignados por defecto por el IDE, ya que por ejemplo en ciertos casos estos identificadores pueden ser dinámicos y causar problemas a la hora de la ejecución de las pruebas.

6. Anexos

6.1 Métricas

Entrega 2

Lead time, Cycle Time & Esfuerzo

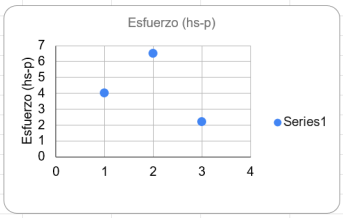
US	Todo Start Date	Date	Done	Lead Time	Cycle Time	Esfuerzo (hs-p)
trackeo inexistente muestra	26-Sep-2023	27-Sep-2023	28-Sep-2023	2	1	4
negativa de medicamentos al	26-Sep-2023	27-Sep-2023	28-Sep-2023	2	1	6.5
cantidades negativas a una	26-Sep-2023	27-Sep-2023	28-Sep-2023	2	1	2.2
					Promedio	
				2	1	4.233333333

Flow Efficiency

Cycle Time	1
Lead Time	2
Flow Efficiency	50.00%

Throughput

3



Entrega 3

Lead time, Cycle Time & Esfuerzo

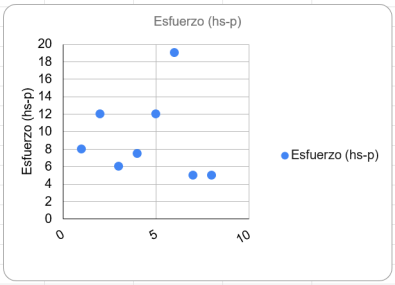
US	Todo Start Date	Definition Start	Done	Lead Time	Cycle Time	Esfuerzo (hs-p)
FE - Modificacion Producto#56	17-Oct-2023	21-Oct-2023	23-Oct-2023	6	2	8
BE - Modificacion Producto#55	17-Oct-2023	23-Oct-2023	23-Oct-2023	6	1	12
FE - Baja Producto#51	17-Oct-2023	21-Oct-2023	23-Oct-2023	6	2	6
BE - Baja producto#52	17-Oct-2023	19-Oct-2023	23-Oct-2023	6	4	7.5
FE - Alta de Producto#50	17-Oct-2023	17-Oct-2023	23-Oct-2023	6	6	12
BE - Alta de Producto#49	17-Oct-2023	17-Oct-2023	23-Oct-2023	6	6	19
FE - Purchase products#58	17-Oct-2023	23-Oct-2023	23-Oct-2023	6	1	5
BE - Purchase products#59	17-Oct-2023	23-Oct-2023	23-Oct-2023	6	1	5
					Promedio	
				6	2.875	9.3125

Flow Efficiency

Cycle Time	2.875
Lead Time	6
Flow Efficiency	47.92%

Throughput

8



6.2 Métricas medidas con esfuerzo

Entrega 2

Lead time, Cycle Time & Esfuerzo

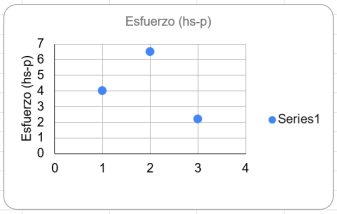
US	Todo Start Date	Date	Done	Lead Time	Cycle Time	Esfuerzo (hs-p)
trackeo inexistente muestra	26-Sep-2023	27-Sep-2023	28-Sep-2023	7	4	4
negativa de medicamentos al	26-Sep-2023	27-Sep-2023	28-Sep-2023	9.5	6.5	6.5
cantidades negativas a una	26-Sep-2023	27-Sep-2023	28-Sep-2023	5.2	2.2	2.2
					Promedio	
				7.233333333	4.233333333	4.233333333

Flow Efficiency

Cycle Time	4.233333333
Lead Time	7.233333333
Flow Efficiency	58.53%

Throughput

3



Entrega 3

Lead time, Cycle Time & Esfuerzo

US	Todo Start Date	Definition Start	Done	Lead Time	Cycle Time	Esfuerzo (hs-p)
FE - Modificacion Producto#56	17-Oct-2023	21-Oct-2023	23-Oct-2023	20	8	8
BE - Modificacion Producto#55	17-Oct-2023	23-Oct-2023	23-Oct-2023	30	12	12
FE - Baja Producto#51	17-Oct-2023	21-Oct-2023	23-Oct-2023	18	6	6
BE - Baja producto#52	17-Oct-2023	19-Oct-2023	23-Oct-2023	13.5	7.5	7.5
FE - Alta de Producto#50	17-Oct-2023	17-Oct-2023	23-Oct-2023	12	12	12
BE - Alta de Producto#49	17-Oct-2023	17-Oct-2023	23-Oct-2023	19	19	19
FE - Purchase products#58	17-Oct-2023	23-Oct-2023	23-Oct-2023	23	5	5
BE - Purchase products#59	17-Oct-2023	23-Oct-2023	23-Oct-2023	23	5	5
					Promedio	
				19.8125	9.3125	9.3125

Flow Efficiency

Cycle Time	9.3125
Lead Time	19.8125
Flow Efficiency	47.00%

Throughput

8