Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de software ágil 2
Documentación de la entrega 5
Informe académico

https://github.com/IngSoft-ISA2-2023-2/obligatorio-grupo11-bandera-caffarena-gonzalez

Sebastián Bandera-220417

Martín Caffarena-230351

Nicolas Gonzalez-231843

Entregado como requisito de la materia Ingeniería de software ágil 2

Resumen de gestión del proyecto.

Marco de gestión ágil

En la primera iteración se planteó un marco de gestión ágil sobre el cual se iba a trabajar a lo largo del proyecto, sin embargo este a medida que se avanzaba se iba perfeccionando en base a necesidades o mejoras durante el avance del proyecto.

Kanban

El proceso de ingeniería que se planteó en la primera iteración del proyecto se basa en la metodología ágil llamada Kanban.

Kanban plantea que el trabajo a realizarse o en proceso tiene que ser visible en todo momento para los integrantes del equipo, por lo que se decidió tener en todas las iteraciones un tablero Kanban, el cual mutaba según el planteo de la etapa (Es decir que la configuración del tablero cambiaba por etapa según el trabajo que se realizaría en la misma). Al comienzo de cada iteración se pasaban las tarjetas correspondientes al trabajo a realizarse en la etapa a la columna de “To Do” y los integrantes tenían completa visibilidad sobre estas para así poder visualizar el proceso durante cada etapa, el trabajo restante y el trabajo completado hasta el momento.

Ceremonias

Retrospectiva: Esta ceremonia se realizó un total de cuatro veces antes de cada entregable con el fin de afinar el proceso de trabajo y así ser más eficientes y eficaces en la entrega de valor. El método utilizado fue DAKI, que consiste en identificar los elementos a quitar, agregar, mantener y mejorar sobre la experiencia hasta el momento.

Revisión con el product owner: Esta ceremonia se realizó un total de tres veces antes de cada entregable con el fin de presentar el resultado a nivel de producto. Esto es mostrar al usuario los cambios o nuevas funcionalidades incluidas en el sistema, de modo de obtener feedback y validar el resultado.

Configuración del tablero

El equipo decidió utilizar la herramienta de GitHub projects dentro del repositorio de GitHub del proyecto sobre otras herramientas (Como podían ser Trello, Jira, entre otros.) para implementar el tablero Kanban porque así se mantenían todos los elementos correspondientes al proyecto en un repositorio único del equipo. Pudiendo así conectar issues con tarjetas y el tablero al repositorio.

El equipo decidió que para tener mayor transparencia sobre el trabajo todas las tarjetas las cuales no habían sido realizadas se colocaron en el Backlog y aquellas las cuales se irían a realizar en la iteración correspondiente se pasarían a la columna del “To Do”, es decir que las tarjetas correspondientes a tareas, historias de usuario o bugs que se irían a realizar en determinada etapa estarían en el To Do esperando ser realizadas. Luego todo el resto de columnas variaba según el alcance y el trabajo a realizar en la etapa.

Informe de avance: primera iteración

Análisis de deuda técnica

Para la primera iteración el equipo realizó un análisis técnico de las diferentes partes del proyecto, de esta forma poder tener una idea sobre la deuda técnica del mismo. Es decir que el equipo pueda saber a grandes rasgos los bugs/issues que deberán ser resueltos por el equipo de desarrolladores.

La herramienta utilizada fue SonarQube que además de analizar el código fuente de ambas aplicaciones, permitirá comparar el estado actual del programa con el estado futuro luego de las correcciones e incorporaciones nuevas de funcionalidades.

De esta actividad se obtuvo un total de 21 issues, los cuales 16 corresponden a Issues de tipo análisis de código, los cuales 4 son de gravedad media y 12 de gravedad leve y los restantes 5 issues corresponden a bugs encontrados, siendo 3 de severidad mayor y 2 de severidad menor.

Creación y posterior mantenimiento del repositorio

El repositorio tiene una carpeta “Código”, en la cual se encuentra todo el código correspondiente a la aplicación separado por dos subcarpetas, una carpeta llamada “Frontend” la cual se compone por el proyecto de Angular y otra carpeta llamada “Backend” que contiene la solución .NET y las pruebas unitarias.

Otra carpeta “Base de Datos” en la cual se encuentra una base de datos de prueba y otra vacía.

Por último una carpeta “Documentación” donde se encuentra la documentación de cada etapa en conjunto con la documentación general de la aplicación. En la subcarpeta “Original” se encuentra la documentación entregada por los autores originales del obligatorio. Así como la letra del obligatorio original.

Definimos un formato para los commits el cual se basa en poner como título del commit que acción se realizó (ADD, DELETE, FIX, UPDATE) seguido de una pequeña descripción de la acción en sí (Esto puede ser sobre que archivo fue acción, que se intentó con la acción, entre otros.).

Informe de avance: segunda iteración

Para esta etapa se comenzó primero que nada con la designación de roles, de esta manera ya establecer una estructura clara dentro del equipo desde un inicio. Luego se comenzó de forma inmediata con el desarrollo de un Pipeline en GitHub, el cual primero el equipo debió aprender a desarrollar el mismo. Luego se determinó que issues serían los corregidos en esta etapa, sin embargo como se explica en el apartado “Elección de Bugs”, estos fueron cambiando a medida que avanzamos en la etapa. Por lo que también se debió de hacer una nueva instancia de Test Exploratorios por parte de todo el equipo, esto con el fin de poder encontrar nuevos Bug´s relevantes, apuntando más a Bug´s en la parte del Backend. A medida que estos fueron encontrados se fueron documentando de manera correspondiente y se agregaron al tablero. En total se resolvieron 3 Bug´s los cuales todos fueron resultados utilizando TDD (Especificado en apartados “Elección de Bugs” y “Reparación de Bug´s con TDD”).

Pipeline en GitHub

En GitHub se realizaron dos pipelines en la rama main para realizar el build en distintos ambientes, ubuntu, macOS y windows. Si una pipeline falla, se debe atender el error inmediatamente, ya sea revirtiendo el commit a un histórico correcto, o realizando la corrección. Además es ideal aprovechar esa oportunidad para aprender del error.

La pipeline para el backend, se ejecuta cuando algún archivo .cs o .csproj es modificado. Con .NET 6.0.x ejecuta la compilación y testeo en los tres sistemas antes mencionados. Fue necesario indicar la ubicación del proyecto en .NET con 'Codigo/Backend/PharmaGo.sln' dado que en caso contrario se buscaba en la raíz. Además se colocó '--property WarningLevel=0' porque encontraban más de 350 advertencias que detenían la pipeline.

La pipeline para el frontend se ejecuta cuando hay algún cambio en 'Codigo/Frontend/**', con esto se practicó dos formas de restringir la pipeline (según subcarpeta o según extensión de archivo en el caso de la pipeline del backend). En el caso del frontend, no se encontraron pruebas unitarias, por lo cual la pipeline sólo realiza el build. Fue necesario realizar un 'cd Codigo/Frontend' antes de ejecutar los comandos para que el build se realice en el directorio correcto. La versión de node utilizada es la 16.x. Se removió el control del caché de dependencias 'if: steps.cache-nodemodules.outputs.cache-hit != 'true'' sobre 'Install Dependencies' porque provocaba error en las ejecuciones de la pipeline que no fueran la primera, lo cual era un gran problema de reproducibilidad.

Por lo tanto el pipeline se ejecutará siempre que se realice un push sobre la rama "main" la cual es nuestra rama de producción, con esto nos aseguramos que cada integración sobre nuestro ambiente de trabajo está controlado, es decir que se tiene feedback inmediato sobre la integración por si algún cambio genera un problema sobre alguna parte del sistema. Esto se ve en el tablero cuando las issues están en fase test estando pasando por el pipeline.

A su vez para esta etapa se decidió el resolver 3 BUG´s, teniendo en cuenta la prioridad de estos asignada por el PO en la etapa anterior. En un principio se habían elegido 3 bugs los cuales tenían la prioridad más alta, sin embargo el equipo luego descubrió que estos eran bugs únicamente de la parte de Frontend y como se debía de resolver bugs de backend los cuales se desarrollaran con el uso de TDD, debimos de buscar nuevos bugs a resolver.

Reparación de bugs mediante TDD

El equipo decidió utilizar la técnica de TDD para la reparación de bugs, la misma consiste en 3 etapas (RED, GREEN, REFACTOR). La primer etapa se crea el test para el código que se va a implementar pero sin implementarlo todavía por lo tanto el mismo falla (se ve como el test está en estado fallido en rojo), luego la siguiente etapa consiste en realizar la implementación mínima del código para que la prueba pase (se ve con estado verde la prueba ya que esta fue implementada como se esperaba) y por último se trata de mejorar el código sea con mejoras funcionales o sintácticas.
Para identificar estas etapas se decidió que luego de la acción en el commit (ya explicado en la anterior entrega) se indica en qué fase se encuentra este y por último sobre qué funcionalidad. Ejemplo: **UPDATE - RED - CreateStockRequest.**

El desarrollo bajo esta metodología de trabajo le permite al equipo avanzar de forma consistente y constante, ya que TDD si realizado de forma correcta permite el siempre estar

“Parados en suelo firme” es decir, como para todo el código que desarrollamos ya hay test, es más difícil cometer errores y que estos sean pasados por altos, pudiendo así llegar a producción. Por lo tanto el equipo puede seguir desarrollando sobre esto sabiendo que las pruebas pasan y que no hay errores o al menos no errores catastróficos ocultos que podrían “explotar” luego en producción.

Informe de avance: tercera iteración

Para esta etapa se comenzó primero que nada con la designación de roles como ya se había repetido en etapas anteriores, de esta manera ya establecer una estructura clara dentro del equipo desde un inicio. Como ya mencionamos en esta ocasión se distinguió entre desarrolladores frontend y Backend.

Luego se hizo una definición de todas las tareas/issues que debían ser realizadas para esta entrega en el nuevo tablero del cual hablaremos más adelante. A su vez se asignó a cada miembro las tareas de las cual cada uno era responsable, teniendo en cuenta sus roles, como desarrollador frontend o backend, etc. Aclarar que en este momento fue cuando se definieron los diferentes escenarios los cuales debían de cumplir cada funcionalidad, esto con el fin de que luego el desarrollo de cada funcionalidad con el uso de SpecFlow y BDD sea más sencillo, claro y estructurado, cosa que vinimos que es muy importante al estar trabajando con BDD.

En un inicio el plan del equipo era primero realizar la parte del desarrollo de frontEnd para luego pasar al backend. Sin embargo por temas de tiempo y habilidades de cada integrante se decidió por dividir entre el equipo las tareas y comenzar con la parte de backend por más que algunas partes del front fueron terminadas más adelante, una vez que el backend ya estaba terminado. Aclara a su vez que a finales del proceso de desarrollo el equipo realizó varias pruebas exhaustivas de las nuevas funcionalidades para corroborar su correcta funcionalidad y a su vez se realizaron pruebas generales de la aplicación, incluso de partes anteriores, ya que el equipo quería asegurarse de que los nuevos cambios no repurquiesen negativamente en el resto de la aplicación. Por esta razón una vez terminado este proceso se debieron de realizar algunos cambios menores referenciados como BugFix para solucionar/mejorar algunos problemas que se encontraron en esta etapa de algunas funcionalidades, tanto nuevas como anteriores funcionalidades.

Algo que se puede notar si hacemos un test de cobertura del código, vemos que este bajo ligeramente, esto se debe a que se agregaron archivos al proyecto los cuales son tenidos en cuenta para dicha cobertura pero no entraba dentro de esta entrega el realizar ningún test sobre estos, como por ejemplo la parte de la WebApi de Productos, por esta razón vemos esta baja en los niveles de cobertura de código del proyecto.

Nuevas funcionalidades utilizando BDD

Para la implementación de las nuevas funcionalidades se utilizó la metodología de BDD, la cual el desarrollo de las mismas son guiados por los comportamientos y para hacer uso de la misma se utilizó la herramienta de SpecFlow, en la cual se definió la narrativa de las historias de usuario, en nuestro caso cada funcionalidad tenía su propia historia de usuario porque nos pareció que quedaba con mayor claridad si se manejaba de esta manera, y también los escenarios de las mismas. Dentro de BDD identificamos la fase RED, que al igual que en TDD corresponde a la creación de los casos de prueba, en este caso la narrativa con sus escenarios, pero sin implementar la misma, luego la fase GREEN donde se implementa el código para que pasen todos los escenarios dentro de las pruebas de la

historia de usuario correspondiente y por último la fase de REFACTOR, la cual no siempre está presente pero corresponde a hacer cambios sobre el código y las pruebas para mejorar estos.

Informe de avance: cuarta iteración

Luego de la designación de roles rotados con respecto a la entrega anterior, se procedió a realizar las pruebas con la herramienta selenium sobre la mayoría de los escenarios especificados con BDD.A su vez se realizaron cálculos sobre los registros de esfuerzo de las entregas pasadas para elaborar varias métricas de Kanban, a nivel de actividad, de entrega y global.

Selenium

En esta etapa se generaron test en base al frontend con la herramienta de Selenium. Se generaron test para las nuevas funcionalidades implementadas en la etapa anterior. El objetivo de utilizar esta herramienta fue ver el correcto funcionamiento de los escenarios planteados en la etapa de implementación mediante BDD y si estos se cumplían en la aplicación web.

Análisis Métricas de Kanban

Otra actividad realizada en esta etapa fue el análisis de métricas de Kanban utilizando el registro de esfuerzos de las iteraciones anteriores que el equipo llevó a cabo mediante el uso de una bitácora, en la cual cada integrante era responsable de registrar las tareas/actividades realizadas y el esfuerzo de la misma. Se calcularon Lead Time, Cycle Time, Esfuerzo de una tarea, Flow efficiency y Throughput y en base a estos se sacaron conclusiones sobre el trabajo realizado por el equipo y se justificó el mismo.

Entrega 1	Throughput	Lead Time Promedio (En Dias)	Cycle Time Promedio (Dias)	Esfuerzo promedio de una tarea (En horas)	Flow efficiency
Entrega 1 (04/09/23 - 18/09/2023)	0,07	13,00	2,00	4,50	15,38%
Entrega 2 (18/09/2023 - 27/09/2023)	0,44	7,00	1,00	1,50	14,29%
Entrega 3 (27/09/2023 - 23/10/2023)	0,35	24,33	1,00	3,28	4,11%

Imagen 1 - Métricas por entrega

Metricas de todas las entregas	
Lead Time Promedio (En Dias)	14,77
Cycle Time Promedio (Dias)	1,30
Esfuerzo promedio de una tarea (En horas)	3,00
Flow efficiency	11,27%
Throughput	0,29

Imagen 2 - Métricas en todas las entregas

En las métricas se ve que el Lead Time es alto, pero el Cycle Time da bajo en comparación. Esto es debido a que el inicio de la mayoría de las tareas se daba en general cuando más de la mitad del tiempo disponible ya había pasado. Pero una vez iniciadas se completan en pocos días. La eficiencia de flujo nos dió 11% aproximadamente. Para mejorar esta métrica se podría iniciar y terminar varias tareas temprano, de modo que el Lead Time de estas tareas baje, y afecte al promedio a la baja. Este Lead Time menor, aumenta la eficiencia de flujo ya que está métrica se usa como divisor en el cálculo. Se puede ver que el Cycle Time es relativamente constante en las tres entregas, pero el Lead Time está directamente correlacionado según cuánto tiempo teníamos en dicha entrega. Entonces lo que más afectó/disminuyó a nuestro Flow Efficiency es el alto Lead Time.

Reflexiones sobre el aprendizaje

El primer bloque entregable, se encargó de realizar un análisis del estado del proyecto, revisando su deuda técnica. Para esto se utilizó la herramienta de análisis SonarQube con la cual pudimos identificar 21 issues de diversos niveles de severidad. La herramienta despliega los resultados en una web, lo que permitió a los compañeros acceder desde sus máquinas al análisis detallado. Esto se traduce en que si se despliega un servidor con SonarQube, se puede centralizar los análisis de los proyectos de una organización y disponibilizarlos según los permisos adecuados.

A nivel de proceso aumentamos la coordinación y comunicación del equipo para evitar problemas de duplicación de tarjetas y documentación que nos ocurrió en la entrega uno. También adelantamos la primera reunión del equipo para mejorar esta coordinación en las siguientes entregas.

A nivel de repositorio, utilizamos por primera vez la metodología trunk-based. Es muy interesante la ventaja en la cual los “merge” o integraciones, son muy sencillas dado que no hay muchas ramas que puedan divergir demasiado. Además de estar todo centralizado en un único repositorio, también hace que esté centralizado en una única rama, facilitando la búsqueda de los artefactos.

El segundo bloque, se encargó de la reparación de varios bugs identificados en la parte anterior. El uso de TDD de nuestra parte nos permitió que todo el código desarrollado este testeado, y el uso de TDD en el proyecto base, nos permitió aprovechar esas pruebas para revisar si la corrección de los bugs afectó alguna prueba pasada. Por esto siempre logramos estar en “suelo firme”, estado compatible con trunk-based dado que la rama en uso necesitar estar en un estado deployable todo el tiempo. El uso de Github Action para ejecutar el proceso de compilación y prueba sobre el estado de la rama fue ideal para reducir los problemas asociados a “en mi pc funciona” y poder estar seguros de que lo necesario para que la función desarrollada se comporte como se espera, está en el repositorio.

El tercer bloque, se encargó del desarrollo de nuevas funcionalidades. El uso de BDD permitió generar los casos de prueba en función de lo que le da valor al usuario, osea en base a varios casos de ejemplo que el usuario podrá recorrer en la aplicación. Para utilizar BDD en la práctica, se empleó el lenguaje Gherkin para escribir los escenarios, y la

herramienta SpecFlow para poder enlazar los escenarios en Gherkin con el código de la aplicación. Por la inexperiencia con SpecFlow, creemos que nos quedaron varios muchos escenarios juntos en un sólo archivo, cuando la herramienta de alguna forma debe permitir una mejor separación para facilitar la lectura y mantenimiento de estos casos. Como mejora, se debería investigar cómo separar mejor los escenarios en más archivos aunque sean de la misma funcionalidad. También puede ser una mejora notable generar alguna clase de utilidad para generalizar algunos bloques de prueba repetidos que se dieron en la implementación.

El cuarto bloque, se encargó de la implementación de pruebas a nivel de usuario, osea pruebas que simulen las acciones que puede realizar el usuario desde el navegador, sobre las funciones nuevas desarrolladas en el tercer bloque. Estas pruebas tienen como ventaja que realmente ensayan sobre la aplicación en su estado que le da valor al usuario, osea la aplicación corriendo. Este ensayo permite identificar si hay algún punto del escenario que esté dando problema, ya sea por un problema en el FrontEnd o por integración, porque los problemas específicos de una funcionalidad ya deberían estar contenidos por las pruebas unitarias.

La herramienta Selenium IDE permitió realizar una grabación de las acciones del usuario, aunque fue necesario luego ajustar esta grabación para realizar los pasos correctos de la prueba. Dado que seleccionar los elementos de la página es una parte importante en este tipo de pruebas, se generaron nuevos identificadores HTML en el FrontEnd de la aplicación, para facilitar la ubicación de los elementos por parte de la herramienta Selenium.

Lecciones aprendidas.

En esta sección del documento detallamos diferentes aspectos del proyecto sobre los cuales el equipo consideró que se podrían concluir lecciones que sirvan como sugerencias o consejos al momento de encarar proyectos similares y podrían de ayudar a nuevas generaciones que fueran a enfrentarse a situaciones similares en un proyecto DevOps parecido a este.

Registro de Esfuerzo del Equipo

El seguimiento del esfuerzo de cada miembro se realizó mediante una bitácora en una hoja de cálculo de Google Drive accesible para todos. Los integrantes registraban diariamente sus actividades, indicando la tarea, la fecha, el issue resuelto y el tiempo dedicado. En la primera iteración, hubo olvidos en el registro, abordados en la retrospectiva, llevando a un compromiso mayor con la bitácora. Aunque se discutió el uso de herramientas externas, se optó por continuar con la actual debido a la familiaridad del equipo. La combinación de la bitácora, el tablero de GitHub y su organización contribuyó a la transparencia y eficiencia del trabajo.

Métricas de Desarrollo

El registro permitió calcular métricas como Lead Time, Cycle Time, Esfuerzo de una tarea, Flow efficiency y Throughput. A pesar de resultados inesperados en ciertos casos, explicados por inicios tardíos, se extrajo la lección de comenzar las etapas más temprano

para evitar acumulaciones. Aunque no se enfrentó a la falta de tiempo, iniciar antes habría mejorado la comodidad en cada etapa.

Lección 1: **Registrar Detalladamente el Trabajo**

La lección principal es la importancia de registrar detalladamente el trabajo para promover la transparencia. La bitácora sirvió para reflexionar sobre las cargas de trabajo, identificar posibles desequilibrios y adaptar el proceso según las capacidades y asignaciones de los integrantes. Las horas y la división de tareas al finalizar un entregable son fundamentales para ajustar el desarrollo, maximizando la efectividad y promoviendo un flujo constante. La separación del tiempo en diferentes categorías facilita métricas efectivas y la mejora continua del equipo, analizando posibles cuellos de botella y áreas de desarrollo.

Reuniones del equipo

El proyecto destacó por sus diversas reuniones, desde planificación y organización hasta ceremonias más estructuradas, como las retrospectivas de etapa y encuentros con el Product Owner (PO). Estas reuniones promovieron la transparencia sobre las tareas de cada miembro, permitiendo identificar necesidades de apoyo y evaluar el progreso conjunto. Las retrospectivas, facilitadas por DAKI, impulsaron mejoras continuas al analizar las sugerencias del equipo para modificar, añadir, mantener o descartar prácticas. El equipo aprendió que la preparación previa para cada reunión, ya sea una retrospectiva o planificación, agilizar el proceso, permitiendo presentar ideas más desarrolladas y mejorar constantemente en cada etapa del proyecto.

Coordinación y organización del equipo

La coordinación y organización del equipo fueron vitales para el proyecto. Detalles como mantener actualizados el tablero y la bitácora, comunicar las tareas en curso y trabajar de manera ordenada resultaron fundamentales. En una instancia, la implementación de BDD subrayó la importancia de la organización al evitar solapamientos de trabajo entre miembros. Hubo un caso puntual de falta de comunicación donde dos miembros trabajaron en partes similares, generando un problema leve que se resolvió posteriormente.

Lección 2: **Ceremonias, Coordinación y Organización del Equipo**

La lección extraída destaca la necesidad de enfocarse en las ceremonias, coordinación y organización del equipo. Las reuniones efectivas y la preparación previa agilizan el proceso y fomentan mejoras continuas. La importancia de mantener actualizados los registros y trabajar de manera ordenada se refleja en la implementación de BDD. Además, se subraya la crucial importancia de las reuniones a lo largo de cada etapa para conocer las actividades individuales, favoreciendo la organización del equipo. Aunque hubo casos aislados de falta de comunicación, se aprendió que una coordinación efectiva evita problemas y contribuye al éxito del proyecto.

Manejo de Nuevas Herramientas

En el contexto del manejo de nuevas herramientas y técnicas en ingeniería de software ágil, la eficacia de estudiar a fondo estas herramientas antes de implementarlas se revela como un principio fundamental. La premisa es clara: aprender mediante la prueba y error o la intuición puede resultar menos eficiente. La importancia de no reinventar la rueda se enfatiza, abogando por aprovechar el conocimiento existente y evitar esfuerzos redundantes. Un ejemplo concreto de esta lección se experimentó durante la implementación de BDD en el proyecto, donde el equipo reconoció la necesidad de una investigación más profunda antes de aplicar la metodología al desarrollo.

Compartir Aprendizajes en el Equipo

En el ámbito de compartir aprendizajes en el equipo, se subraya la relevancia de la comunicación y la transferencia de conocimiento. Cuando un miembro adquiere nuevos conocimientos o supera desafíos específicos, compartir estas experiencias no solo evita que otros cometan errores similares, sino que también facilita a los compañeros trabajar de manera más efectiva. Este intercambio continuo de información contribuye a la creación de un entorno colaborativo y de aprendizaje constante. Un ejemplo palpable de este principio se manifestó nuevamente en la implementación de BDD, donde un miembro del equipo compartió exitosamente sus conocimientos, facilitando el aprendizaje del resto del equipo en su aplicación.

Ajuste del Proceso

En cuanto al ajuste del proceso de ingeniería, se destaca la importancia asignada a la flexibilidad y la adaptabilidad. La lección aprendida es clara: no hay que adherirse a un proceso rígido. En lugar de eso, es crucial ajustar el proceso según los objetivos actuales y permitir que varíe si dichos objetivos cambian. Esta flexibilidad no solo permite una respuesta más ágil a las necesidades cambiantes del proyecto y del equipo, sino que también fomenta la mejora continua a medida que el proyecto avanza, permitiendo pequeños ajustes en cada etapa. Este enfoque dinámico se revela como un motor para el progreso constante del equipo.

Lección 3: **Dinamismo y Colaboración en la Ingeniería de Software Ágil**

La lección conjunta resalta la gran importancia de un enfoque flexible y colaborativo en la ingeniería de software ágil. La previsión al estudiar nuevas herramientas antes de implementarlas, la comunicación constante de aprendizajes dentro del equipo y la capacidad para ajustar el proceso según los objetivos actuales conforman un ciclo esencial de mejora continua. Este enfoque activo y colaborativo no solo impulsa la eficiencia y la calidad en el trabajo, sino que también fortalece la capacidad del equipo para afrontar desafíos y adaptarse a cambios en el entorno del proyecto. La adaptabilidad y la retroalimentación constante se revelan como elementos clave para el éxito sostenible en proyectos de ingeniería de software ágil.

Conclusiones

Con este proyecto se consiguió poner en prueba los conceptos contemplados en el curso de Ingeniería de Software Ágil 2 sobre el marco de gestión ágil.

Se aplicaron gran cantidad de principios sobre la entrega continua como fue el trabajo por lotes y las mejoras continuas dentro de cada etapa con objetivos específicos y simulaciones de reuniones con un PO (Product owner) para verificar cada integración o cambio sobre el producto.

A lo largo de todo el trabajo se trabajó con la práctica de mantener un único repositorio del equipo en el cual se encontraba la totalidad del proyecto, sea código, documentación e incluso el tablero de Kanban de cada etapa.

Todos los principios y prácticas mencionadas anteriormente en este documento se pueden ver plasmadas en el trabajo realizado con cada etapa y se aprendió de las mismas al realizarlas. A su vez en cada etapa se realizaban actividades de feedback con las cuales el equipo podía mejorar y validar lo realizado en base a estos. Como resultado se obtiene un incremento en el valor del producto, viendo nuevas funcionalidades y arreglando bugs sobre el producto. El uso de métricas de Kanban proporciona una visión clara del rendimiento del equipo, permitiendo la identificación de áreas de mejora y la toma de decisiones informadas

En conclusión el equipo destaca el trabajo realizado como una manera exitosa para poner en práctica los aspectos de la metodología kanban a lo largo del proyecto. También ven el proyecto valioso con respecto al trabajo en equipo y guiado y que el mismo aportó valor para el futuro profesional de los integrantes del grupo, ya que muchas de estas prácticas se utilizan en el desarrollo de software hoy en día y tener contacto con estos previamente se ven como una ventaja a futuro.

Guía de despliegue en producción

Se necesita mysql, .NET 6 y node 16.13.0

Backend:

1. Clonar el repositorio con
   ```
   git clone
   https://github.com/IngSoft-ISA2-2023-2/obligatorio-grupo11-bandera-caffarena-gonzalez.git
   ```
2. Crear una nueva base de datos vacía de nombre ‘PharmaGoDb’.
3. Sobre la base anterior ejecutar el script SQL ubicado en el repositorio: /Base de Datos/Base con datos de prueba/PharmaGoDb.sql (En el repositorio clonado)
4. Cambiar de directorio
   ```
   cd obligatorio-grupo11-bandera-caffarena-gonzalez\Codigo\Backend
   ```
5. Ejecutar sobre el directorio anterior el build
   ```
   dotnet restore
   dotnet build -c Release --no-restore
   ```
6. (Opcional) Ejecutar las pruebas. Este paso es opcional porque se puede revisar el estado de la pipeline en el repositorio para ver si el commit descargado pasó con éxito las pruebas.
   ```
   dotnet test PharmaGo.sln --verbosity normal
   ```
7. Ejecutar la aplicación

```
dotnet run --project PharmaGo.WebApi -c Release --no-build
--no-restore
```

8. Prueba de humo
   `curl https://localhost:7186/api/Drug`
   Si devuelve un json, la conexión a la base de datos y el backend están ok.
   Si demora en dar respuesta y no devuelve un json, es probable que exista un problema de conexión con la base de datos. La posible solución es la siguiente:
   a. Detener el backend con control + C en la consola o similar.
   b. En 'PharmaGo.WebApi\appsettings.json' => "ConnectionStrings":"PharmaGo" colocar lo siguiente como valor
   server=localhost;database=PharmaGoDb;Trusted_Connection=False;MultipleActiveResultSets=True;uid=<USUARIO>;Password=<CONTRASEÑA>
   c. Reemplazar <USUARIO> y <CONTRASEÑA> con el usuario y contraseña asociada a la instalación en uso del mysql.
   d. Volver a levantar el backend con el paso 7

Frontend:

1. (Si se realizó en el bloque del backend, saltear este paso) Clonar el repositorio con
   `git clone https://github.com/IngSoft-ISA2-2023-2/obligatorio-grupo11-bandera-caffarena-gonzalez.git`
2. Cambiar de directorio
   `cd obligatorio-grupo11-bandera-caffarena-gonzalez\Codigo\Frontend`
3. Ejecutar
   `npm install =>(luego) npm run build =>(luego) ng serve`
4. (Opcional) Se pueden ejecutar las pruebas con
   `npm install -g selenium-side-runner`
   `selenium-side-runner Codigo\Frontend\PharmaGo.side`

Página de inicio: http://localhost:4200/home
El puerto de la página de inicio puede variar si se agrega "--port otroPuerto" al ng serve