Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de Software Ágil 2 – Entrega 5

Fiorella Macedo (251662)

Matías Mazziotti (250896)

Nicolás Torres (251420)

Link al repositorio:
https://github.com/IngSoft-ISA2-2023-2/obligatorio-mazziotti-macedo-torres

Profesores: Carina Fontán, Álvaro Ortas

Noviembre, 2023

Informe Académico Final: Proyecto DevOps

Resumen de las Etapas del Proyecto

Este informe encapsula los esfuerzos y aprendizajes del equipo a lo largo del proyecto, que tuvo como meta principal la incorporación efectiva de prácticas de CI/CD, QA y testing automático en un contexto ágil utilizando el marco de trabajo Kanban.

A lo largo del proyecto, se lograron los siguientes hitos:

- Implementación de 4 nuevas funcionalidades críticas.
- Corrección de 3 bugs fundamentales.
- Desarrollo y ejecución de numerosas pruebas automatizadas.

Etapa 1: Fundamentos y Exploración

En la fase inicial, el equipo se dedicó al Testing Exploratorio y al Análisis de Código. Se identificaron problemas críticos, se documentaron issues pertinentes y se estableció una base sólida para las etapas subsiguientes del proyecto. Aquí surgieron 15 Issues del testing exploratorio y 19 sobre el análisis estático de código.

En esta etapa se crearon 62 issues y drafts, de los cuáles se completaron 27 (y, al ser una etapa temprana del proyecto, no estábamos acostumbrados a usar Kanban, y por esto nos olvidamos de pasar 4 de estas historias a done), 1 draft fue despriorizado del proyecto (la instalación de ambiente de Fiorella) y el resto de Issues quedaron en el Todo para ser tomados en etapas posteriores.

Para tener información más detallada de esta entrega se puede acceder al siguiente link.

Etapa 2: TDD y Solución de Bugs

Con el foco en la mejora del código existente, se abordaron y solucionaron tres bugs significativos utilizando la metodología TDD, asegurando así la robustez y fiabilidad del software.

En esta etapa se crearon 13 issues, los cuáles se completaron todos, y a su vez se completaron 3 de las issues que habían quedado abiertas en la etapa anterior.

Para tener información más detallada de esta entrega se puede acceder al siguiente link.

Etapa 3: BDD y Desarrollo de Nuevas Funcionalidades

La tercera etapa se centró en la adición de cuatro nuevas funcionalidades, desarrolladas con un enfoque BDD utilizando SpecFlow. Esto se complementó con pruebas de integración manuales, consolidando la calidad de las implementaciones.

Aquí se completaron completamente las 11 issues que se crearon en esta etapa.

Para tener información más detallada de esta entrega se puede acceder al siguiente link.

Etapa 4: Pruebas Funcionales y Análisis de Métricas DevOps

Durante la última fase, se realizó un análisis de las métricas abarcando todas las funcionalidades y correcciones de bugs. Se implementaron pruebas funcionales automáticas con Selenium y se efectuó una evaluación detallada de los procesos y herramientas empleados.

Las métricas claves que obtuvimos del proyecto incluyen:

- Tiempo de Ciclo (Cycle Time): 7-9 días para US, 1 día para bugs.
- Tiempo de Entrega (Lead Time): 12-16 días para US, 2-3 días para bugs.
- Eficiencia del Flujo (Flow Efficiency): 53%-59% para US, 33%-50% para bugs.
- Throughput: 4 funcionalidades y 3 bugs en el período de la etapa.

En la cuarta y última etapa que utilizamos Kanban, se crearon 15 Issues, de las cuales 3 fueron canceladas y reemplazadas por otras, ya que no estaban pasando por las columnas que estaban pasando las user stories. El resto de Issues sí fueron finalizadas sin problema.

Para tener información más detallada de esta entrega se puede acceder al siguiente link.

Reflexiones sobre el Aprendizaje

Reflexionar sobre el proceso de aprendizaje dentro de un proyecto de desarrollo de software es una tarea esencial para comprender cómo las acciones tomadas se han alineado con los objetivos planteados y para identificar oportunidades de mejora. A continuación, se presenta una reflexión detallada, basada en los objetivos de la rúbrica y el método DAKI.

Análisis de Deuda Técnica

¿Qué hicimos mal y deberíamos dejar de hacer?

- Empezamos tarde en la etapa, lo cual nos retrasó en el manejo proactivo de la deuda técnica.
- Las reuniones extensas fueron contraproducentes.
- El uso de Excel para llevar un registro de horas resultó ineficiente.

¿Qué no hicimos y deberíamos comenzar a hacer?

- Comenzar con la etapa lo antes posible para tener más margen de maniobra.
- Establecer horarios fijos para mejorar la organización y la planificación.
- Optimizar el ambiente de trabajo para cada integrante.

¿Qué hicimos bien y deberíamos continuar haciendo?

- La motivación y el apoyo constante entre los miembros del equipo fueron fundamentales para superar los desafíos.
- La buena comunicación y el compromiso de cada uno contribuyeron a un ambiente de trabajo positivo y productivo.

¿Qué hicimos relativamente bien, pero podríamos mejorar?

- Implementar un sistema de seguimiento de tiempo más eficiente, como Clockify, para mejorar la precisión en el registro de horas.
- Dividir las tareas desde un inicio para asegurar una distribución equitativa y eficiente del trabajo.

Reparación de Bugs

¿Qué hicimos mal y deberíamos dejar de hacer?

- La distribución desigual de la carga de trabajo en la reparación de bugs creó desequilibrios en el equipo.

¿Qué hicimos bien y deberíamos continuar haciendo?

- El uso de herramientas de seguimiento de tiempo como Clockify ayudó a mantener la organización y la responsabilidad individual.
- La colaboración y el apoyo entre los integrantes del equipo al enfrentar desafíos técnicos.

¿Qué hicimos relativamente bien, pero podríamos mejorar?

- Ajustar las columnas del tablero Kanban para que reflejen más precisamente el flujo de trabajo y mejorar la distribución de tareas.
- Considerar la implementación de badges en los workflows para una visualización más clara del progreso y responsabilidades.

Desarrollo de Nuevas Funcionalidades

¿Qué hicimos mal y deberíamos dejar de hacer?

- Comenzar las etapas del proyecto más temprano para evitar acumulación de trabajo hacia el final.

¿Qué no hicimos y deberíamos comenzar a hacer?

- Mejorar la organización en la gestión de tiempos, especialmente cuando se presentan desafíos externos como otras obligaciones académicas.

¿Qué hicimos bien y deberíamos continuar haciendo?

- El apoyo grupal y la dinámica de equipo positiva que permite enfrentar los retos de desarrollo con confianza y colaboración.

¿Qué hicimos relativamente bien, pero podríamos mejorar?

- Optimizar el manejo de tiempos y la división de tareas para adaptarse a las circunstancias cambiantes y mantener un ritmo de trabajo sostenible.

Análisis de Métricas DevOps

¿Qué hicimos bien y deberíamos continuar haciendo?

- El enfoque en el apoyo grupal y en ayudarse mutuamente cuando se presentan dificultades técnicas.

¿Qué hicimos relativamente bien, pero podríamos mejorar?

- Mejorar el manejo de tiempos para adaptarse a las fluctuaciones en la carga de trabajo y optimizar la eficiencia del equipo.

Lecciones Aprendidas

Gestión del Tiempo y Organización

La experiencia nos enseñó la importancia de una gestión del tiempo efectiva y una organización sólida. A pesar de las fluctuaciones en la carga de trabajo y las circunstancias imprevistas, logramos adaptarnos y mantener la productividad. Sin embargo, reconocemos que iniciar las etapas del proyecto con antelación y emplear herramientas de seguimiento de tiempo desde el comienzo habría mejorado significativamente nuestra eficiencia.

Ejemplo: Nos enfrentamos a una acumulación de trabajo al final de la etapa debido a un inicio tardío. Esto nos llevó a reevaluar nuestros métodos y a adoptar Clockify para una mejor gestión del tiempo.

Equilibrio y Distribución de Tareas

Aprendimos que una distribución equitativa de las tareas es esencial para mantener el equilibrio del equipo y evitar la sobrecarga de trabajo. Los distintos integrantes del equipo deben intentar hacer esfuerzos similares y fomentar un entorno de trabajo saludable.

Ejemplo: Observamos que uno de los miembros del equipo tenía una carga de trabajo desproporcionadamente alta, lo cual fue evidente durante la reparación de bugs. Ajustamos nuestra estrategia y mejoramos la división de tareas para las etapas posteriores.

Comunicación y Colaboración

La comunicación abierta y la colaboración son pilares de un equipo de desarrollo eficiente. Mantener una dinámica de equipo positiva y un fuerte apoyo grupal es fundamental para superar desafíos y alcanzar objetivos comunes.

Ejemplo: La colaboración efectiva fue clave cuando uno de nuestros miembros se atascó en una compleja funcionalidad de BDD. Trabajando juntos, pudimos encontrar una solución creativa y eficiente.

Flexibilidad y Adaptabilidad

La flexibilidad y la capacidad de adaptarse a los cambios son cualidades indispensables en un entorno de desarrollo ágil. Aprender a manejar los tiempos de forma adaptable, especialmente cuando factores externos afectan la carga de trabajo, nos permitió continuar progresando sin comprometer la calidad.

Ejemplo: La necesidad de adaptar nuestro horario de trabajo durante la semana de entregas y parciales fue un recordatorio de que la flexibilidad es crucial para mantener el flujo de trabajo y cumplir con nuestros compromisos.

Mejora Continua y Uso de Métricas

El enfoque en la mejora continua y la utilización de métricas DevOps ha sido un aspecto revelador. Las métricas nos brindaron enseñanzas valiosas sobre nuestro rendimiento y nos ayudaron a identificar áreas de mejora.

Ejemplo: Aunque las métricas se recopilaron al final del proyecto, su análisis nos reveló áreas de eficiencia e ineficiencia que no habíamos notado durante el desarrollo. Por ejemplo, identificamos que el tiempo dedicado a ciertas tareas no se correlacionaba con su valor o complejidad percibida. Esta comprensión nos ha llevado a la conclusión de que, para proyectos futuros, debemos implementar un seguimiento de métricas desde el inicio para ajustar nuestras estrategias y procesos de trabajo de manera más dinámica y efectiva.

Conclusiones

Interpretación de los Resultados

Los resultados obtenidos reflejan el éxito en la integración de prácticas DevOps y la adopción de un enfoque ágil para la gestión de proyectos de software. Mostraron que es posible mejorar la calidad del software y la eficiencia del equipo aplicando métodos ágiles y de revisión y mejora continuas.

Consecuencias de las Acciones

Las acciones emprendidas llevaron a un producto más robusto y a un equipo mejor alineado. Las prácticas implementadas ayudaron a reducir errores y a incrementar la capacidad del equipo para responder a cambios y requerimientos inesperados.

Importancia de los Resultados

La importancia de estos resultados radica en su capacidad de demostrar que la metodología y las herramientas adecuadas pueden tener un impacto significativo en el desarrollo de software, mejorando tanto el producto final como la experiencia del equipo de desarrollo.

Futuras Direcciones

Los resultados orientan hacia un desarrollo de software más dinámico y eficiente, enfatizando la mejora continua y la adaptabilidad como claves para el éxito en futuros proyectos tecnológicos.

Guía de instalación para desarrollo y despliegue en producción.

Antes de iniciar con la instalación, se debe tener instalado SQL Express y SQL Server Management Studio para el backend, así como NodeJS y Selenium IDE para el frontend.

El primer paso que se necesita para comenzar a realizar los tests y poder correr la aplicación es restaurar la base de datos desde el .bak que se encuentra en el repositorio en la carpeta *\Entrega 5\backup*.

El connectionString está configurado para funcionar en cualquier sistema que use SQL Express de forma local, sin embargo, la configuración del usuario puede causar problemas y que esto no funcione correctamente, en cuyo caso el usuario deberá cambiar dicho string, adaptándolo a su base de datos.

Instrucciones para el Backend

Para compilar el backend, puede utilizar Visual Studio en modo Release o ejecutar el comando *dotnet build --configuration Release* desde la línea de comandos en la raíz del proyecto backend. Esto generará el archivo ejecutable PharmaGo.WebApi.exe en la carpeta *Entrega 5\Codigo\Backend\PharmaGo.WebApi\bin\Release\net6.0*.

Para el despliegue local del backend, ejecute el archivo .exe que se encuentra en Entrega 5\Despliegue\Backend (es el mismo que se generó anteriormente).

Instrucciones para el Frontend

Dentro del directorio del proyecto frontend, ejecute npm install seguido de npm run build. Esto generará la carpeta de distribución dist\pharma-go. Ejecute finalmente npm install -g http-server para instalar el servidor HTTP globalmente.

Para desplegar el frontend localmente, ejecute http-server dentro de la carpeta Entrega 5\Despliegue\Frontend, que contiene el contenido previamente generado en dist\pharma-go.

Ejecución de Tests Automáticos

Para ejecutar los tests del backend, recomendamos abrir el test explorer de Visual Studio con la solución y ejecutarlos desde allí.

Es posible que, al ejecutar los tests desde el IDE, algunas de las pruebas de specflow lancen una excepción de tipo DbUpdateConcurrencyException. Esto se debe a problemas de concurrencia que no supimos darnos cuenta qué los causa. La solución a esto es correr cada test que falle de manera individual. Al hacer esto, no se dará esa excepción y el test será exitoso.

Debido a este problema es que decidimos correr estos tests por este medio y no desde consola, para poder apreciar todos los tests funcionando. Con más tiempo nos hubiera gustado arreglar este problema.

En cuanto a los tests de Selenium para el frontend, primero se debe tener corriendo el frontend en modo debug, ya que no quisimos modificar la URL de inicio de los tests para mantener coherencia con lo que hicimos anteriormente y tener separados los ambientes de producción y desarrollo. Para esto, en lugar de utilizar http-server, corremos directamente el comando npm run start estando parado en la carpeta del código del frontend. Luego de esto hay que abrir la extensión de Chrome Selenium IDE y cargar el archivo de pruebas Entrega 5\Selenium\Pruebas Selenium.side. Para los tests de compra de producto, se debe vaciar el carrito después de cada ejecución para evitar conflictos. El resto de los tests pueden ejecutarse en secuencia sin intervención adicional. Para volver a correr los tests, se debe volver a cargar la base de datos de prueba inicial.