Universidad ORT Uruguay
Facultad de Ingeniería

Ingeniería de Software Ágil 2

Informe Final

Clavijo, Tomás (235426)
Barreto, Sofía (258216)
Facello, Bruno (260694)

Docente: Alvaro Ortas

2023

Índice

Gestión del proyecto..... 3
Actividades ejecutadas y herramientas empleadas.....3
Resúmenes de avance..... 4
Reflexiones sobre el aprendizaje.....7
Lecciones aprendidas..... 10
Conclusiones..... 12
Guía de despliegue y ejecución de pruebas.....13
    Restauración de base de datos.....13
    Ejecutar pruebas de SpecFlow..... 14
    Pruebas de integración de selenium..... 14

Gestión del proyecto

El proyecto se basa en la implementación de los principios de DevOps en un código preexistente. Aplicamos prácticas de integración y entrega continua, pruebas automáticas y aseguramiento de calidad.

En cuanto a la gestión del proyecto, utilizamos Kanban como marco ágil. Esta elección se deriva de los requisitos del proyecto. Dentro de Kanban, realizamos tres entregas separadas por un intervalo de dos semanas. Al final de cada entrega, nos aseguramos de cumplir con los objetivos establecidos y llevamos a cabo retrospectivas para mejorar y reflexionar sobre el proceso. Además, generamos informes de progreso.

En términos de ingeniería, adoptamos el enfoque ágil de BDD (Desarrollo Dirigido por Comportamiento). Dentro de este marco, realizamos mejoras en un sistema existente y también implementamos nuevos requisitos. Además, llevamos a cabo revisiones del producto junto con un propietario para aprobar o rechazar los incrementos en el producto. Cada miembro del equipo asumió el rol de desarrollador y Sofía Barreto se desempeñó como “product owner”.

Actividades ejecutadas y herramientas empleadas

Tomamos la decisión de utilizar GitHub como repositorio central para consolidar y compartir la información de manera continua. Esta centralización fomenta el aprendizaje y la experimentación, ya que todos los artefactos relacionados con el producto se encuentran allí y están disponibles para solicitar acceso y realizar modificaciones por parte de todos los miembros del equipo. GitHub ofrece repositorios en la nube basados en Git, es de código abierto, ampliamente utilizado y forma parte de las habilidades iniciales de nuestro equipo. Además, el soporte para el pipeline de CI/CD que aprendimos en el curso también se encuentra en GitHub (a través de GitHub Actions).

Dentro del repositorio de GitHub, albergamos todos los artefactos asociados al producto y al pipeline de CI/CD. Uno de los elementos destacados es el código

fuente y sus respectivas pruebas. El código fuente corresponde a una aplicación full stack, con un servidor desarrollado en C# utilizando .NET y el frontend desarrollado en TypeScript con Angular. Automatizamos las pruebas funcionales de la aplicación utilizando Selenium.

Además, el repositorio contiene otros artefactos relevantes, como archivos de configuración, documentación actualizada y heredada, especificaciones del pipeline, entre otros.

El pipeline es otra herramienta que respaldó las actividades de integración y despliegue continuos. Este fue evolucionando durante todas las entregas. Las versiones están disponibles en el repositorio de Github. El pipeline se basó en un tablero Kanban, comenzó con cuatro columnas y se fue ampliando para poder gestionar nuestra entrega. A medida que implementamos las prácticas DevOps, fuimos agregando nuevos estados para reflejar las actividades que hemos introducido. Las diferentes configuraciones del pipeline se pueden encontrar en las distintas entregas, y la evolución del tablero se registra en la sección “Tablero” de GitHub.

Resúmenes de avance

En la misma carpeta, se encuentran los tres informes de avance que proporcionan información detallada sobre las actividades realizadas en cada una de las entregas. A continuación, resumiremos el contenido de cada informe.

Informe de avance 1: Durante esta etapa, establecimos el marco de trabajo que seguiríamos, optando por implementar el enfoque Kanban. En este sentido, detallamos minuciosamente el manejo de versiones y los estándares aplicados a los commits, issues y nombres de ramas utilizados en el proyecto. Además, documentamos las herramientas específicas que el equipo empleó en el desarrollo. Con el objetivo de abordar la deuda técnica y mejorar la calidad del software, llevamos a cabo un análisis exhaustivo utilizando diversas herramientas, como nDepend, entre otras. También realizamos pruebas manuales para evaluar el funcionamiento del sistema y, en base a estos esfuerzos, identificamos los errores

existentes y los registramos como bugs a ser corregidos. Este proceso nos permitió tener una visión clara de los aspectos técnicos que requerían atención y nos proporcionó información valiosa para orientar nuestro trabajo hacia la mejora continua del proyecto.

Informe de avance 2: A partir de los resultados obtenidos en el primer informe de avance, donde se identificaron diversas issues, procedimos a marcar aquellas que consideramos de mayor relevancia. Para tomar esta decisión, realizamos diversas evaluaciones y análisis. Una vez identificadas estas problemáticas prioritarias, decidimos iniciar el trabajo para solucionarlas. Con el objetivo de garantizar la calidad y el flujo eficiente de desarrollo, establecimos el mantenimiento del pipeline. Para lograr esto, implementamos un proceso automatizado que se activa al realizar una pull request. En este proceso, el sistema compila automáticamente nuestra solución y ejecuta todas las pruebas unitarias correspondientes. En caso de que alguno de estos pasos arrojara un resultado negativo, el pipeline rechazaba automáticamente la pull request, impidiendo así la fusión de los cambios con la rama principal del proyecto.

Informe de avance 3: Durante esta etapa, nos enfocamos nuevamente en mejorar la documentación con el fin de corregir errores anteriores y garantizar su precisión. Además, aprovechamos las nuevas funcionalidades desarrolladas para aplicar ingeniería de requerimientos y proceder con la implementación correspondiente. Como parte de este proceso, actualizamos algunos documentos existentes para adaptarlos a la situación actual del proyecto, brindando un mayor nivel de detalle en relación al tablero utilizado. Es importante destacar que se realizaron casos de prueba exhaustivos para verificar el correcto funcionamiento de las nuevas funcionalidades, siguiendo la metodología de Desarrollo Guiado por Pruebas (TDD).

Informe de avance 4: En la etapa final del proyecto, hemos dedicado esfuerzos adicionales para mejorar la calidad de la documentación, identificando y corrigiendo los errores previamente identificados. Además, hemos llevado a cabo un análisis exhaustivo de las métricas correspondientes a la entrega, evaluando el trabajo realizado y asegurándonos de que cumpla con los estándares establecidos.

También hemos continuado recopilando y analizando la evidencia de las pruebas realizadas, verificando la validez y efectividad de nuestras soluciones. Esta evidencia respalda el cumplimiento de los requisitos establecidos y garantiza la calidad del producto final. En resumen, durante esta última etapa hemos realizado las correcciones necesarias en la documentación, evaluado las métricas de entrega y continuado con la evidencia de pruebas, asegurándonos de cumplir con los objetivos establecidos y entregar un producto de calidad. Toda esta información se incluirá en el informe final del proyecto.

Reflexiones sobre el aprendizaje.

Aplicación de un marco de gestión ágil

El marco ágil no se implementó completamente porque no llevamos a cabo las reuniones Stand Up todos los días, dado que no parecían ser pertinentes debido a que nuestras tareas no eran realizadas diariamente. Por otro lado, aunque no implementamos completamente el marco ágil en su forma tradicional, pudimos adaptar sus principios y prácticas a nuestra situación específica. Aprendimos a ser más flexibles, colaborativos y receptivos al cambio, lo que nos permitió mejorar nuestra gestión del proyecto y alcanzar mejores resultados en cada entrega.

Analizar la deuda técnica

Para analizar la deuda técnica, utilizamos pruebas exploratorias, pruebas de usabilidad y análisis del código fuente. Antes de comenzar, nos aseguramos de comprender el negocio y sus requisitos. Aunque creemos haber encontrado la mayoría de los errores y aplicado los pasos correctos, es posible que hayamos pasado por alto algunos. Sin embargo, hemos hecho esfuerzos significativos para abordar la deuda técnica, utilizando pruebas exhaustivas y análisis detallados.

Implementar un repositorio y procedimientos de versionado

Durante la implementación del repositorio y los procedimientos de versionado, reflexionamos sobre la importancia de la configuración adecuada del archivo ".gitignore" y la necesidad de mantener una estructura de nombres de archivos coherente. Encontramos dificultades al no configurar correctamente el archivo .gitignore, lo que resultó en la inclusión de archivos no deseados y conflictos posteriores.

Esto nos llevó a comprender la relevancia de establecer un archivo .gitignore preciso y completo desde el principio, para evitar la inclusión de archivos innecesarios y mantener un repositorio más limpio y ordenado. También destacamos

la importancia de mantener una estructura de nombres de archivos coherente para facilitar la gestión de versiones y evitar confusiones en el equipo.

Crear un pipeline con eventos y acciones

Al crear el pipeline con eventos y acciones, cometimos el error de incluir pruebas de Speck Flow que dependían de la base de datos sin haberlas mockeado previamente. Después de consultar con nuestros profesores, comprendimos que el problema real fue haber incluido esas pruebas en GitHub Actions. Aprendimos que las pruebas que dependen de la base de datos no deberían formar parte del pipeline debido a problemas de dependencia y rendimiento. A pesar de este error, el resto del proceso de creación del pipeline se llevó a cabo de manera adecuada. Esta experiencia nos enseñó la importancia de evaluar cuidadosamente qué pruebas deben incluirse en el pipeline y cómo gestionar las dependencias de manera adecuada.

Integrar prácticas de QA en el pipeline y gestionar el feedback

Al integrar prácticas de QA en el pipeline y gestionar el feedback, pudimos aprovechar las ventajas de agregar pruebas unitarias y utilizar el pipeline de GitHub Actions. Esto nos permitió identificar posibles fallas en el sistema antes de su implementación. Las pruebas unitarias nos ayudaron a evaluar la funcionalidad y la integridad del nuevo código, garantizando estándares de calidad. El uso de GitHub Actions automatizó el proceso de ejecución de pruebas y proporcionó un feedback rápido, lo que nos permitió tomar medidas proactivas para solucionar los problemas identificados. Esta experiencia nos reafirmó la importancia de integrar prácticas de QA en el desarrollo y utilizar herramientas que aseguren la calidad del software de manera eficiente.

Generar escenarios de testing desde la perspectiva del usuario

Para generar escenarios de testing desde la perspectiva del usuario, utilizamos BDD y Speck Flow como herramientas para abordar este enfoque. Sin embargo,

reconocemos que podríamos haber realizado más pruebas y explorado una mayor diversidad de casos.
Por otro lado, es importante destacar que a pesar de las limitaciones y dificultades encontradas con Speck Flow y con mockear los test, los tests que realizamos funcionaron correctamente.

Automatizar el testing funcional o de caja negra

Al utilizar Selenium para automatizar el testing funcional, pudimos explorar y evaluar el frontend de la aplicación de manera más exhaustiva. Sin embargo, nos enfrentamos al desafío de asegurar la disponibilidad y consistencia de los datos en la base de datos de pruebas. Aprendimos la importancia de planificar y gestionar adecuadamente los datos necesarios para los casos de prueba, estableciendo procesos sólidos para garantizar una evaluación confiable y eficiente.

Reflexionar sobre DevOps

Las técnicas de DevOps tuvieron un impacto significativo en la calidad y productividad del equipo. En comparación a cómo trabajamos antes de aplicar las técnicas de DevOps (en otros obligatorios) observamos mejoras en la calidad del software, una mayor eficiencia en el desarrollo y en mejoras en cada una de las entregas, así como una colaboración más estrecha entre los integrantes del equipo. Sin embargo, también enfrentamos desafíos, como la necesidad de adquirir nuevas habilidades. A pesar de las dificultades, consideramos que los beneficios superan ampliamente los obstáculos y estamos comprometidos a seguir mejorando en esta área clave de desarrollo de software.

Lecciones aprendidas

Evaluar cuidadosamente qué pruebas deben incluirse en el pipeline

Cometimos el error de agregar pruebas de Spec Flow al pipeline sin haberlas mockeado previamente. Aprendimos que deberíamos evaluar cuidadosamente qué tipo de pruebas incluir en el pipeline, especialmente aquellas que dependen de recursos externos como la base de datos. Además, reconocimos la importancia de separar las pruebas que requieren recursos externos de las pruebas unitarias y considerar estrategias alternativas, como el uso de mocks, para evitar problemas de dependencia y rendimiento en el pipeline. Esta experiencia nos enseñó a ser más selectivos y considerados al elegir las pruebas que formarán parte del proceso de integración continua.

Importancia de la comunicacion y organizacion

Cometimos el error de no comunicarnos y organizarnos adecuadamente con las tareas, lo que nos llevó a tener que realizar ciertas actividades de manera apresurada y a última hora. Aprendimos que la comunicación clara y efectiva es esencial para mantener a todos los miembros del equipo informados sobre las tareas pendientes, los plazos y los requisitos específicos. También nos dimos cuenta de la importancia de una organización sólida para asignar responsabilidades de manera clara y asegurarnos de que todas las tareas se realicen de manera oportuna. Esta experiencia nos enseñó la importancia de establecer un sistema de comunicación efectivo y una buena organización para evitar contratiempos y trabajar de manera más eficiente en el futuro.

Importancia de la configuración adecuada del archivo .gitignore

Cometimos el error de no configurar correctamente el archivo .gitignore, lo que resultó en la inclusión de archivos no deseados y conflictos posteriores. Aprendimos que la configuración adecuada del archivo .gitignore es crucial para evitar la inclusión de archivos innecesarios en el repositorio y mantenerlo limpio y ordenado. Reconocimos la importancia de establecer un archivo .gitignore preciso y completo desde el principio, que excluya de manera adecuada los archivos generados

automáticamente, las dependencias y cualquier otro archivo irrelevante para el control de versiones. Esta experiencia nos enseñó a ser más diligentes al configurar el archivo .gitignore y a revisar y actualizar regularmente su contenido para evitar problemas futuros.

Importancia de diferenciar un proceso de ingeniería y un marco de gestión Kanban

Cometimos el error de no aclarar la diferencia entre un proceso de ingeniería y un marco de gestión Kanban. Esto llevó a confusiones dentro del equipo y a una implementación incorrecta de las prácticas. Aprendimos que es fundamental comprender y comunicar claramente la distinción entre estos dos conceptos.

La ingeniería se refiere a la metodología, técnicas y mejores prácticas utilizadas para desarrollar y construir un producto o sistema. Por otro lado, el marco de gestión Kanban es una forma de visualizar y gestionar el flujo de trabajo utilizando herramientas como tableros y tarjetas.

Reconocimos que aunque el marco Kanban puede ser una herramienta valiosa para gestionar el trabajo en el proceso de ingeniería, es importante comprender que no reemplaza las prácticas y principios de ingeniería necesarios para desarrollar un producto de calidad.

Esta experiencia nos enseñó a diferenciar claramente entre el proceso de ingeniería y el marco Kanban, y a utilizar ambos de manera complementaria para lograr una gestión eficiente y un desarrollo exitoso del producto.

Conclusiones

Luego de haber realizado todas las entregas podemos concluir que la gestión de un proyecto es igual o hasta más importante que el proyecto mismo, ya que esta organiza a cada integrante del equipo y mantiene un flujo de trabajo continuo y ordenado.

Como fue hablado en el curso y vivido en experiencia propia, existe una concepto de *experimentación continua y aprendizaje* el cual se refiere a manejar, aprender y dominar herramientas que faciliten y ayuden en el proceso.

Un claro ejemplo fue la automatización de algunas tareas como los test y el compilado antes de cerrar una *pull request* implementada por los *Git Actions.*Con estas tareas automatizadas el equipo puede enfocarse en agregarle valor al producto y evita en su gran mayoría el retrabajo de funciones previamente desarrolladas, también se logra reducir costos, riesgos y ahorrar tiempo.

Estas herramientas fomentan el *Continuous Deployment*, brindándole siempre valor al producto entregado.
A su vez más puestas a producción llevan a más aprendizaje del equipo, el cual si después es trasladado al resto de equipos genera una cultura de aprendizaje del cual todos se ven beneficiados.
Además produce que el equipo se vuelva más eficiente, brindando mayor calidad al software, y rutinizar los deploys, volviendo este aspecto clave del proceso de ingeniería de software parte de cada día.

Kanban nos guió en este proceso de entrega continua, ofreciendo un flujo de entrega continua.
En el contexto de desarrollo de nuevas funcionalidades o productos, el uso de *Sprints* u iteraciones de *Scrum* es de suma ayuda, mientras que en un contexto de mantenimiento, optariamos por utilizar un tablero de *Kanban* el cual nos favorece a la hora del *Continuous Deployment,* aportando valor en el menor tiempo posible al cliente.

Guía de despliegue y ejecución de pruebas

En esta sección vamos a detallar los pasos para ejecutar el obligatorio, también se informará los elementos a tener instalados para el correcto funcionamiento e instalado del sistema.
El sistema operativo utilizado para este obligatorio fue Windows, ya que presentaba mayores compatibilidades con el software utilizado para la creación del obligatorio sobre el cual se parte.
Se requiere poseer el framework *.NET 5.0* , el cual se puede obtener aquí.
Entity Framework que se puede instalar fácilmente con el comando:

```
dotnet tool install --global dotnet-ef
```

El gestor de base de datos *SQL Server.*Se debe utilizar la base provista ya que las pruebas de integración y funcionales modifican datos de la base.

Además se necesita node package manager, disponible aqui.

Angular es necesario para la ejecución del front,obtenible a partir del siguiente comando:

```
npm install -g @angular/cli
```

Para correr las pruebas funcionales de front se debera poseer Selenium IDE.

Restauración de base de datos

En primer lugar se requiere restaurar la base de datos que se encuentra en el repositorio dentro de las carpetas Bases de Datos / ArenaGestorPruebaSnacksSpecFlow.bak

Para hacer esto es necesario abrir el SQL Server, conectarse, ir a la carpeta de *Databases*, hacer click derecho sobre la misma y presionar *Restore Database*.

Luego de esto se seleccionara devices, y se elegirá el archivo nombrado anteriormente. Se espera a que se complete y la restauración quedará lista.

Ejecutar pruebas de SpecFlow

Previamente se debe de setear el connection string, esto se hace modificando el archivo appsetings.json dentro de *Codigo\Backend\ArenaGestor\ArenaGestor.API.*
Para ejecutar las pruebas de SpecFlow debemos tener el sistema corriendo, para ello debemos
Ejecutar los comandos:
Parado en la carpeta *Codigo\Backend\ArenaGestor*

```
dotnet restore
```

Y parado en la carpeta *Codigo\Backend\ArenaGestor\ArenaGestor.API*

```
dotnet run
```

Luego parados en *Codigo\Backend\ArenaGestor\ArenaGestor.SpecFlow*

```
dotnet test
```

Y parado en *Codigo\Backend\ArenaGestor\ArenaGestor.ComprarSnacks*

```
dotnet test
```

Recomendable restaurar la base de datos después de ejecutar las pruebas de SpecFlow ya que impactan en la base de datos y podría influir en los resultados

Para ejecutar los otros tests debemos ejecutar:

parados en *Codigo\Backend\ArenaGestor\Test\ArenaGestor.APITest*

```
dotnet test
```

parados en *Codigo\Backend\ArenaGestor\Test\ArenaGestor.BusinessTest*

```
dotnet test
```

parados en *Codigo\Backend\ArenaGestor\Test\ArenaGestor.DataAccessTest*

```
dotnet test
```

Pruebas de integración de selenium

Se debe de tener ejecutando el Backend, lo cual se hace ejecutando los primeros dos comandos de la parte anterior.

En una consola, estando parado en la carpeta *Codigo\Frontend\ArenaGestorFront*

```
npm install
```

para restaurar los paquetes de npm

En una consola, estando parado en la carpeta *Codigo\Frontend\ArenaGestorFront* , correr el comando

```
npm start
```

para correr el frontend

Ahora al ir a la url http://localhost:4200/ se podrá ver la aplicación corriendo en el ambiente local
Se debe restaurar la base de datos antes de ejecutar las pruebas ya que es necesario para el correcto funcionamiento de las pruebas.

Abrir Selenium IDE, seleccionar "Open an existing project" y seleccionar el archivo *PRUEBAS ARENA GESTOR FRONT.side* que se encuentra en la carpeta *Codigo\Frontend\ArenaGestorFront*

Paso 3: Hacer click en "Run all tests"