Universidad ORT Uruguay

Facultad de Ingeniería

Informe de avance de etapa: Informe final

Ingeniería de Software Ágil 2

Entregado como requisito para la obtención del título de Ingeniero en Sistemas

Sofía Montero - 250618

Romina Pardo - 253445

Andrés Wilchinski - 193694

Tutor: Carina Fontan y Alvaro Ortas

2023

https://github.com/IngSoft-ISA2-2023-2/250618_193694_253445

Índice

1. Resumen de gestión del proyecto

1.1. Actividades realizadas y productos entregados

A lo largo de las entregas hemos realizado varias actividades que han ayudado a mejorar el proyecto. Para comenzar hemos realizado un análisis de deuda técnica, donde hallamos y documentamos los problemas técnicos en el código, como malos diseños, código redundante, bugs, entre otros para luego realizar mejoras al sistema. Dentro del análisis de deuda técnica realizamos un reporte de issues donde estos fueron clasificados por prioridad y severidad para poder trabajar ordenadamente y eficientemente en la próxima etapa. En la siguiente etapa resolvimos tres de estos issues, documentando y actualizando las distintas guías relacionadas. Estos issues fueron seleccionados debido a que eran los de mayor prioridad respecto a la clasificación previamente mencionada. También agregamos acciones de github para que cada vez que se realiza un push de un commit o se cree una pull request se compile el backend y se corran los unit test en el mismo. Esto nos permite identificar si los cambios efectuados rompen alguna funcionalidad prevista, facilitando la modificabilidad y mantenibilidad del proyecto. En la tercera etapa, ampliamos el sistema inicial al agregar cuatro nuevos requerimientos por medio de BDD y la tecnología Specflow. Finalmente, en la cuarta etapa, verificamos el correcto funcionamiento del sistema utilizando Selenium, esto resultó en un archivo .side con pruebas de integración automatizadas que aseguran el correcto funcionamiento del sistema actual y a futuro. Gracias a estas pruebas en esta etapa logramos corregir algunos bugs del sistema.

A su vez, a lo largo de las entregas, se implementaron métricas que nos dan una representación gráfica del proceso realizado, el rendimiento del equipo y la calidad del sistema. De igual manera, para cada entrega fue realizado un informe de avance para la descripción del proceso y evaluación de la productividad del equipo.

Podrá ver las documentaciones de cada entrega aquí: Entrega 1, Entrega 2, Entrega 3, Entrega 4

1.2. Issues

Aquí puede ver el reporte de issues completo. A continuación se podrá observar un resumen de la cantidad de bugs por categorías al finalizar este proyecto.

Prioridad / Severidad	*Crítico*	*Mayor*	*Menor*	*Leve*
Inmediata	0	0	0	0
Alta	0	0	0	0
Media	0	0	2	0
Baja	0	0	7	2

1.3. Métricas

Bugs

Bugs (entrega2)

Nro sub tarea	Tipo	Descripción	Horas/Esfuerzo persona	Responsable	Fecha In Todo
6	Bug	Fix Bug 3	1:30:00	Andres	20/9/2023
7	Bug	Fix Bug 4	1:30:00	Sofia	20/9/2023
8	Bug	Fix Bug 6	1:00:00	Romina	20/9/2023

Nro sub tarea	Cycle Time (dias)	Lead Time (dias)	Flow efficiency	Throughput	Duración de la entrega (dias)
6	0.00	3.00	0	0.33	9
7	1.00	3.00	0.3333333333		
8	0.00	3.00	0		
	0.33	3.00	0.1111111111		

User Stories

User Story (entrega 3)

Nro sub tarea	Tipo	Descripción	Horas/Esfuerzo persona	Responsable	Sprint Backlog
6	User Story	User Story 1 - Backend y Frontend	15:00:00	Todos*	11/10/2023
7	User Story	User Story 2 - Backend y Frontend	3:30:00	Sofia	11/10/2023
8	User Story	User Story 3 - Backend y Frontend	10:00:00	Romina	11/10/2023
9	User Story	User Story 4 - Backend y Frontend	5:00:00	Andy	11/10/2023

Nro sub tarea	Cycle Time (dias)	Lead Time (dias)	Flow efficiency	Throughput	Duración de la entrega (dias)
6	8.00	11.00	0.7272727273	0.1666666667	24
7	9.00	12.00	0.75		
8	8.00	11.00	0.7272727273		
9	9.00	12.00	0.75		
	8.50	11.50	0.7386363636		

Comparativas

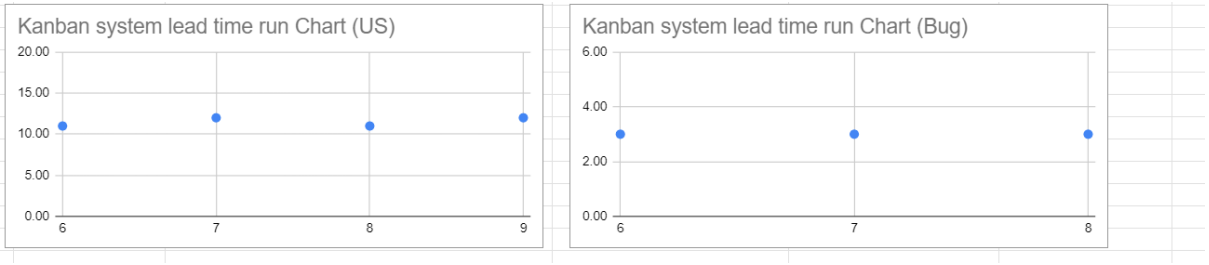
Cycle Time: Los Bugs tienen un Cycle Time mucho más corto en comparación con las User Stories. Los Bugs se resuelven en cuestión de horas (0.33 días), mientras que las User Stories días (8.50-9 días).

Lead Time: El Lead Time de las User Stories es significativamente más largo que el de los Bugs, con una diferencia de alrededor de 8 días. Esto va relacionado con lo dicho en la parte de cycle time.

Flow Efficiency: Los Bugs tienen un Flow Efficiency mucho más bajo (11.11%) en comparación con las User Stories (73.87%). Esto indica que las User stories se están moviendo de manera más eficiente a través del proceso. Esta métrica nos indica que en proporción los bugs pasan más tiempo en el ToDo que las user stories

Throughput: La Throughput de los Bug Fixes es más alta en comparación con las User Stories, osea, se demora menos en entregar el arreglo de un bug, que la implementación de una nueva user story. Esto es porque el esfuerzo de corregir un bug es menor que el de implementar una nueva user story.

Kanban system lead time chart



En esta gráfica se puede observar el tiempo que llevó cada user story y bug desde que se definió hasta completarse. El promedio de resolución de los bugs es de 3 días, a diferencia de las user stories que son entre 11 y 12 días. Por lo tanto, las user stories requieren más esfuerzo en ser resueltas que los bugs.

Como conclusión, debido a la gran diferencia de esfuerzo entre los Bugs y las User Stories sus métricas no son comparables para la obtención de un resultado concluyente respecto a la performance del equipo. A su vez, el periodo de realización de las mismas también varía drásticamente lo que afirma que estas no son comparables.

Registro del trabajo y Esfuerzo

- Para la primera entrega contamos con 14 tareas y el total del registro de esfuerzo fue de 18 horas y media, podrá ver el seguimiento aquí.
- Para la segunda entrega contamos con 13 tareas y bugs, el total del registro de esfuerzo fue de 13 horas y media, podrá ver el seguimiento aquí.
- Para la tercera entrega contamos con 9 tareas y user stories, el total del registro de esfuerzo fue de 21 horas con 30 minutos. Podrá ver el seguimiento realizado aquí.
- Para la cuarta entrega contamos con 9 tareas y el total del registro de esfuerzo fue de 17 horas. Podrá ver el seguimiento de esfuerzo y métricas realizadas aquí.

A su vez, para este informe final, se realizaron 10 horas persona de esfuerzo. Por lo tanto, entre todas las entregas el esfuerzo total es de 80 horas y media.

2. Reflexiones sobre el aprendizaje

2.1. Sobre los objetivos

Aplicar un marco de gestión ágil

Aplicar un marco de gestión ágil nos permitio adoptar un enfoque flexible a nuestro proyecto. Este marco no solo nos permitió organizar las tareas de forma organizada, sino que también nos permitió llevar a cabo un mecanismo de trabajo más consciente de nuestras acciones pasadas y futuras. Esto provocó que nuestro trabajo fuera más eficiente y organizado.

Consideramos que hicimos bien el manejo y creación del tablero de kanban lo cual nos facilito ver que tareas estaba haciendo cada uno y en qué etapa estaban. Aunque a veces nos olvidábamos de tomar el tiempo que nos llevaba cada tarea y aproximamos. También respecto a las ceremonias creo que hicimos bien las daily standups y tratamos de siempre sacar conclusiones en las retrospectives para luego implementar o corregir aspectos del proceso en entregas siguientes.

Analizar la deuda técnica

Consideramos que este análisis ayuda tremendamente a visualizar las fallas del sistema para futura mejora, es una buena forma de determinar el orden de prioridad para la realización de mejoras o bug fixes. También nos permitió familiarizarnos con el código y el sistema.

Implementar un repositorio y procedimientos de versionado

La implementación de un repositorio y procedimientos de versionados es fundamental para cualquier proyecto. Esto permite retroceder en el tiempo, dejando registrada cada modificación y, por lo tanto, manteniendo un registro detallado de la evolución del proyecto.

Utilizamos correctamente el trunk base para manejar el versionado tratando de mergear rápido a main.

Crear un pipeline con eventos y acciones, integrar prácticas de QA y gestionar el feedback

Crear una pipeline de acciones nos permite agregar y correr pruebas automáticas lo cual nos da una mayor confianza de que nuestro código es correcto y no rompe nada de lo anteriormente desarrollado. Además también nos permite darle feedback directo a nuestros desarrolladores si un test no pasa. Otra cosa que no hicimos pero pudiéramos agregar en estas acciones es el formateo de archivos garantizando así que todo quede en un mismo formato y estilo.

Generar escenarios de testing desde la perspectiva del usuario

Al realizar este tipo de escenarios logramos garantizar que el producto final satisfaga las necesidades y expectativas de los usuarios, lo que a su vez mejora la calidad del producto y la satisfacción del cliente.

Esto lo logramos mediante el uso de specflow. Una cosa para mejorar en estos test es que algunos no eran 100% independientes de los otros o repetibles, haciendo que solo pasen si existía X elemento en la

base de datos. Esto aunque no es un problema grave, podría hacer el test fallar por razones equivocadas si el tester no sabe esto. Además quizás hubiese estado bueno agregar dichos test a las github actions para asegurar el funcionamiento de dichas funcionalidades.

Automatizar el testing funcional o de caja negra.

Al automatizar el testing funcional logramos obtener pruebas de integración automatizadas para asegurar el correcto funcionamiento del sistema a la hora de la entrega y a futuro por si se realiza algún otro cambio.

Esto lo hicimos correctamente ya que mediante la herramienta selenium fuimos capaces de testear la funcionalidad de principio a fin. Lo que podríamos haber realizado mejor es el hacer que dichos test sean independientes entre sí y de la base de datos ya que nos paso que para que los test anduviesen se deben de correr en determinado orden lo que implica que si en un futuro se sacan algun test o algún test se rompe va a romper otro y no necesariamente porque la funcionalidad no ande correctamente. También nos pasó que un test depende del orden en el que están los datos mostrandose en la aplicación y como la base de datos que usamos el local cada uno de los desarrolladores posee distintos datos haciendo que dicho test solo anduviese en una computadora y no en todas.

Reflexionar sobre DevOps

La distribución de roles, el registro de esfuerzos y la utilización de tableros bajo la filosofía de DevOps no solo nos brindaron organización, sino también una mayor conciencia sobre nuestras tareas y en qué estado se encuentran. Esta metodología nos ha permitido ser eficientes, fomentar la colaboración y alcanzar con éxito nuestros objetivos en esta entrega.

2.2. Uso de DAKI

A lo largo de las entregas hemos usado el método DAKI: Drop (¿Qué hicimos mal y deberíamos dejar de hacer?), Add (¿Qué no hicimos y deberíamos comenzar a hacer?), Keep (¿Qué hicimos bien y deberíamos continuar haciendo?) y, finalmente, Improve (¿Qué hicimos relativamente bien, pero podríamos mejorar?) para las retrospectivas del equipo, para aprender más sobre la metodología hemos usado distintos templates que nos ayudaron a otorgarle distintos enfoques a la retrospectivas. A su vez, consideramos que esta metodología nos ha ayudado a reflexionar sobre el proceso, encontrar sencillamente aspectos a mejorar y concluir en las distintas lecciones que nos llevamos gracias al trabajo realizado.

2.3. Lecciones aprendidas

A lo largo de las distintas entregas hemos reflexionado sobre distintas lecciones resultantes de las etapas del trabajo relacionadas con los productos entregados. A su vez, gracias a las retrospectivas y reflexiones, llegamos a otras lecciones que se relacionan a recomendaciones para otros equipos DevOps que desean realizar trabajos similares.

Lección 1 - Análisis de deuda técnica

Enunciado: Aprendimos a cómo reportar bug y a utilizar un linter.

Ejemplo práctico: Este análisis ayuda tremendamente a visualizar las fallas del sistema para futura mejora. Como podemos ver en las primeras dos entregas, realizar un buen análisis facilita la organización de prioridad para la resolución de issues o bugs para resultar en un mejor producto para el usuario. También nos permite familiarizarnos con el código del sistema y cómo funciona el mismo.

Lección 2 - Ceremonias

Enunciado: Al hacer los stand ups diarios, reviews y la retrospectiva nos dimos cuenta de cosas que deberíamos mejorar para las siguientes entregas, a su vez de mejoras más pequeñas a lo largo del trabajo para esta entrega.

Ejemplo: Al realizar la retrospectiva de la entrega 2, una de las cosas mencionadas fue “Tener un poco más de cuidado con la ortografía a la hora de realizar commits y en la documentación ”. Esta mención a una etapa tan temprana del proyecto nos permitió mantener una estructura más prolija y correcta en las siguientes entregas. Si esta no hubiera sido comunicada en esa instancia, se hubieran dado errores de ortografía en el proyecto sin darle la prioridad correcta para corregirlos.

Lección 3 - Automatización de pruebas, SpecFlow

Enunciado: Aprendimos a utilizar specFlow, a su vez de la implementación del frontend y backend a través de APIs.

Ejemplo: Un ejemplo concreto fue la implementación de la user story “Alta de productos”. En la cual se crearon sus escenarios y a partir de eso se crearon las pruebas con SpecFlow para probar el correcto funcionamiento de la implementación de esta user story. Luego de probado esto se implementó el frontend con Angular, probando después desde la perspectiva del usuario que todo funcione correctamente. Esta automatización de pruebas nos permitió ahorrar mucho tiempo a la hora de corregir fallos y la verificación de funcionamiento, además de que nos aseguro que todo estuviera funcionando de la manera deseada.

Lección 4 - Automatización de pruebas, Selenium

Enunciado: Se adquirió experiencia en la utilización de Selenium con BDD para realizar pruebas automatizadas que verifican el comportamiento del sistema. También se comprendió la importancia de la automatización de pruebas en el proceso de desarrollo para garantizar la calidad y el funcionamiento correcto del sistema.

Ejemplo: Se crearon las pruebas con Selenium a partir de los escenarios creados con BDD. Se crean las pruebas de “Alta de producto” para cada escenario y luego se corre. Se puede observar en la pantalla como la máquina interactúa con la aplicación como si de un usuario se tratase. Poder visualizar esto fue de suma importancia, ya que nos dio una visión más profunda de cómo sería la interacción del usuario con nuestro código y que fallas se podrían corregir o mejoras se podrían implementar.

Lección 5 - Organización y comunicación

Enunciado: Gracias a las retrospectivas hemos notado la importancia de la organización y comunicación dentro del equipo para lograr un buen ambiente y producto final. Esto nos da un sentido más profundo a lo trabajado y el esfuerzo hecho.

Ejemplo práctico: Como ejemplo de buenas prácticas de comunicación y organización consideramos los daily stand ups o reuniones de equipo al igual que una buena asignación de tareas y registro de esfuerzo como las que fueron realizadas a lo largo del trabajo. Estas lograron facilitar el trabajo de las etapas finales.

Lección 6 - Métricas

Enunciado: Hemos también descubierto que el automatizar el cálculo de las distintas métricas desde las primeras etapas facilita visualizar el rendimiento del equipo así como ahorrar el esfuerzo de recalcular para cada entrega las distintas métricas que se quieren analizar de forma manual.

Ejemplo práctico: Al revisar los excel de registro de esfuerzo y métricas se puede visualizar como desde la segunda entrega el equipo ha estado experimentando con la automatización del cálculo de distintas métricas lo cual facilitó enormemente el cálculo y análisis de métricas para la cuarta entrega.

Estas métricas como por ejemplo el lead time o cycle time nos permitieron analizar con mayor profundidad nuestro trabajo, cuánto se demora nuestro equipo en terminar una user story o en corregir un bug. Así como también utilizando gráficos como el cumulative flow diagram ver cuáles fueron los cuellos de botella en el proceso. Estas métricas nos permiten saber mejor cómo funciona el equipo y así poder planear con mayor precisión a futuro así, como poder encontrar y solucionar posibles problemas como el hecho de los cuellos de botella.

Lección 7 - La importancia de automatizar los test y github actions

Enunciado: Encontramos que utilizando github actions para compilar el proyecto y correr los test de manera automática, nos daba una mayor garantía y seguridad que el código que llegaba a la rama main iba a andar como lo esperábamos y si no lo hacía era que teníamos que agregar más test para que ataquen esos casos. También nos daba un feedback rápido de los errores y nos permite detectar errores de manera muy temprana ya que las actions no sólo corren al realizar pull request sino que cada vez que se realiza push de un commit.

Ejemplo: Al tener una pipeline de github actions que corra los unit test cuando se crea un pull request nos garantiza que al menos en el backend los cambios que se efectuaron no rompieron nada. En un futuro faltaría agregar los test de integración en estas acciones para así poder garantizar con mayor certeza que todo el sistema funciona y que los cambios introducidos no destruyen el funcionamiento del sistema. También nos permite probar para distintos sistemas operativos ahorrándonos mucho trabajo. Además de otorgarnos feedback casi inmediatamente si el cambio introducido rompió algún test.

Lección 8 - La importancia de que los test sean independientes y no dejen modificaciones en la base de datos

Enunciado: Los test deben ser independientes uno de los otros y poderse repetir infinidad de veces dando el mismo resultado. Los test deben ser first (rapidos, independientes, replicables, auto validados y escribirse antes del código)

Ejemplo: un problema que nos pasó es que los test que hicimos de selenium no eran independientes 100%, ejemplo uno de ellos selecciona un elemento que se va a borrar, el 5to en la lista por ejemplo y chequea que aparezca el mensaje de que se eliminó correctamente el producto shampoo. Que pasa que al correr de nuevo o si otro miembro del equipo con una base de datos distintas (con otros datos) corría el test este cae porque se elimina otro producto ya que no tiene los mismos productos en la base de datos. Otra cosa que nos pasó con los test de selenium es que los mismos no son independientes entre sí por lo cual se deben correr en cierto orden lo que los hace más difíciles de mantener.

Lección 9 - Tiempo

Enunciado: Al comenzar las entregas con tiempo y dejar uno o dos días para revisar el trabajo realizado permite ofrecer buen feedback a los otros desarrolladores y conseguir un mejor producto final.

Ejemplo: Gracias a las retrospectivas hemos encontrado que uno de los errores que más hemos repetido es no siempre comenzar con tiempo cada una de las entregas que resulta en nosotros realizando cosas a último momento y sin tiempo para revisar el trabajo de los demás y ofrecer feedback.

3. Conclusiones

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

Los resultados de las lecciones aprendidas indican que hemos mejorado significativamente como equipo en la organización, comunicación y eficiencia a lo largo de las entregas. Se ha fortalecido la comprensión y aplicación de herramientas como SpecFlow, Selenium y el análisis de deuda técnica, al igual que la comprensión del funcionamiento de los equipos DevOps y la aplicación de un marco de gestión ágil. La implementación de prácticas como el análisis de deuda técnica, la realización de ceremonias, y la automatización de pruebas ha contribuido a la evolución positiva del equipo.

3.2. ¿Qué consecuencias/implicancias tiene lo anterior?

Las consecuencias de estas mejoras son una mayor calidad en el producto final, una reducción de errores y una optimización en nuestro rendimiento. La implementación de métricas automatizadas también ha simplificado el seguimiento del progreso realizado, proporcionando una visión más clara y detallada de nuestro trabajo.

3.3. ¿Por qué son importantes esas implicancias o consecuencias?

Estas implicancias son importantes porque garantizan un desarrollo más efectivo, aumentan la satisfacción del cliente al ofrecer un producto de mayor calidad y proporcionan una base sólida para futuros proyectos. La cultura de trabajo más colaborativa y organizada resulta en un ambiente positivo y eficiente.

3.4. ¿A dónde nos conducen?

Estas mejoras nos conducen hacia una cultura de desarrollo más ágil y eficiente. Las lecciones aprendidas ofrecen una guía valiosa para equipos DevOps similares, apuntando hacia la mejora continua y el logro de objetivos de manera más efectiva en el futuro. La evolución constante en la forma en que abordamos el desarrollo de software se traduce en un enfoque más estructurado, automatizado y centrado en la mejora continua.

4. Guía de instalación para desarrollo y despliegue en producción

4.1. Build

Para correr el proyecto se deben hacer lo siguiente:

1. Abrir SQL Management Studio y crear o restaurar la base de datos utilizando los scripts ubicados en Proyecto/Implementacion/Base de Datos/EntregaAgil2/ ya sea con datos o vacía.
2. Abrir una terminal la ruta del proyecto y correr los siguientes comandos:
    a. dotnet restore Proyecto/Implementacion/Codigo/Backend/PharmaGo.sln
    b. dotnet build --configuration Release --no-restore Proyecto/Implementacion/Codigo/Backend/PharmaGo.sln
    c.
3. Este último comando genera un .exe en la carpeta Proyecto/Implementacion/Codigo/Backend/PharmaGo.WebApi/bin/Release/net6.0
4. Hacer doble click en el archivo PharmaGo.WebApi.exe o correr en la terminal el comando

    Proyecto/Implementacion/Codigo/Backend/PharmaGo.WebApi/bin/Release/net6.0/PharmaGo.WebApi.exe

5. Abrir una nueva terminal en el path (Proyecto/Implementacion/Codigo/Frontend)
    a. Correr el comando npm install
    b. Correr el comando npm start
    c. Abrir la url que se imprime en la terminal

4.2. Casos de prueba unitarios (SpecFlow)

6. Abrir SQL Management Studio y conectarse con la base de datos
7. Cargar los datos en la base de datos. Dichos datos se encuentran en: Proyecto/Implementacion/Base de Datos/EntregaAgil2/ConDatos/PharmaGoDb.bak
8. Para correr los test desde consola
    a. Abra una terminal y muévase al path del proyecto
    b. Corra el backend (ver 4.1)
    c. Ejecute en la terminal el comando:

    dotnet test Proyecto/Implementacion/Codigo/Backend/SpecFlowProducts

4.3. Casos de prueba funcionales (Selenium)

Para correr las pruebas en Selenium se debe (por consola):
Pre - requisitos:

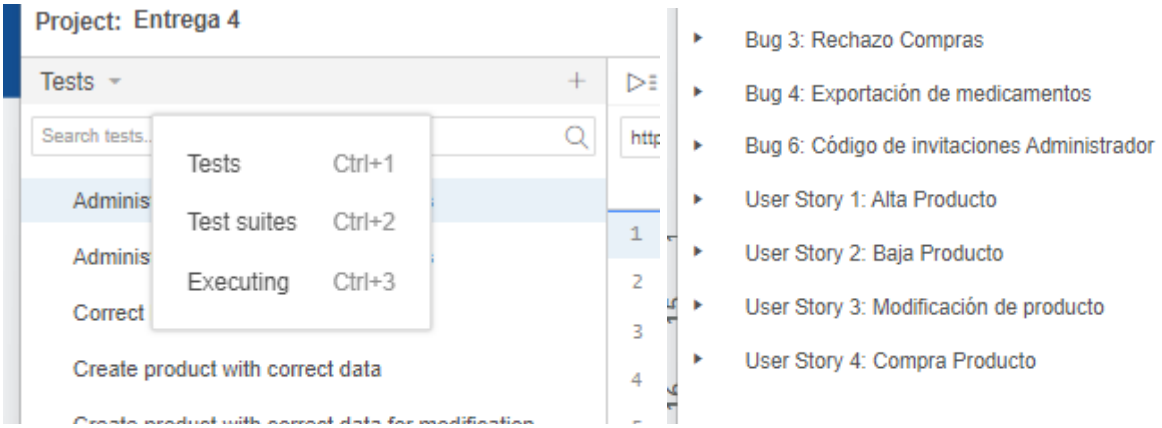
- node

- npm
- selenium-side-runner
- browser driver (chromedriver / edgedriver / geckodriver)

1. Instalar node y selenium-side-runner:
   > brew install node
   > npm install -g selenium-side-runner
2. Instalar driver del navegador
   > npm install -g geckodriver
   > npm install -g edgedriver
3. Ejecución del proyecto completo
   >selenium-side-runner Proyecto/Entrega_4.side

Para correr las pruebas en Selenium se debe (con IDE):

1. Descargarse Selenium (extensión de Chrome)
2. Abrir Selenium desde la zona de extensiones de Chrome
3. Poner “Open an existing project”
4. Seleccionar el archivo .side (Proyecto/Entrega_4.side)
5. Ir a la zona de “Test suites” y seleccionar el conjunto de test deseado



6. Apretar la flecha con barras para correr toda la Test Suit o la flecha con solo una barra para correr solo un escenario (debe seleccionar el escenario específico para eso).

Recordar que el orden de ejecución recomendado es el siguiente:

- Crear producto
- Modificar producto
- Eliminar producto
- Intentar aprobar una compra con producto eliminado
- Exportar medicamentos

- Modificar código de una invitación
- Hacer compra

Esto se recomienda ya que hay cambios en la base de datos que pueden afectar el desarrollo positivo de alguna de las pruebas anteriores.

4.4. Pipeline (Git Actions)

Las actions se corren cuando se pushea un commit a cualquier branch o sino cuando se realiza una pull request a main en la que se cambia un archivo con terminación .cs o .csproj.