

Universidad ORT Uruguay

Facultad de Ingeniería

Informe Académico Final
realizado para el Obligatorio 1 de
Ingeniería de Software Ágil 2

Diego Tenenbaum - 228429

Federico Czarniewicz - 201250

Mateo Costa - 232870

Tutor/es:

Álvaro Ortas

2023

Índice:

Resumen de gestión del proyecto	4
Entrega 1:	4
Entrega 2:	4
Entrega 3:	5
Entrega 4:	6
Justificación de las decisiones tomadas durante el proyecto:	6
Definición de un marco basado en KANBAN:	6
Análisis de deuda técnica mediante analizadores estáticos, pruebas unitarias, test exploratorios:	6
Utilización de Trunk Based Development y One Piece Flow	7
BDD para el desarrollo de funcionalidades:	7
DAKI como método para guiar las retrospectivas:	7
Tableros utilizados en el proceso:	7
Workflows:	8
TDD para el arreglo de bugs en el backend:	8
Issues:	8
Métricas del proyecto:	9
Reflexiones sobre el aprendizaje	10
Drop (¿Qué hicimos mal y deberíamos dejar de hacer?):	10
Add (¿Qué no hicimos y deberíamos comenzar a hacer?):	11
Keep (¿Qué hicimos bien y deberíamos continuar haciendo?):	11
Improve (¿Qué hicimos relativamente bien, pero podríamos mejorar?):	11
Lecciones aprendidas	12
Conclusiones	13
¿Qué significan los resultados de lo investigado, aplicado y/o solucionado?	13
¿Qué consecuencias/implicancias tiene lo anterior?	13
¿Por qué son importantes esas implicaciones o consecuencias?	14
¿A dónde nos conducen?	14
Guía de instalación para desarrollo y despliegue en producción	14
Guía para instalación del despliegue	14
Herramientas necesarias	14
Restaurar la base de datos	14
Despliegue del backend	15
Despliegue del frontend	15
Guía para instalación y uso del ambiente de desarrollo	15
Herramientas	15
Restauración de la base de datos	15
Correr el backend en modo desarrollo	15
Correr el frontend en modo desarrollo	15
Ejecución de las pruebas unitarias y las de Specflow	16
Ejecución de los casos de prueba de selenium	16
Anexos	16
Anexo 1	16
Anexo 2	17
Anexo 3	17
Anexo 4	18

Anexo 5.....	18
Anexo 6.....	19
Anexo 7.....	20
Anexo 8.....	21
Anexo 9.....	22
Anexo 10.....	23
Anexo 11.....	23
Anexo 12.....	24
Anexo 13.....	25
Anexo 14.....	26
Anexo 15.....	27
Anexo 16.....	28

Resumen de gestión del proyecto

Entrega 1:

En el informe 1, hemos abordado una serie de actividades clave en nuestro proyecto, acompañadas de diversos artefactos. En primer lugar, como parte de la definición del marco general de KANBAN, hemos creado una pequeña guía que describe y explica el proceso de ingeniería dentro del contexto de KANBAN. Este artefacto proporciona una referencia clara y concisa para todo el equipo, ayudando a comprender y aplicar correctamente el proceso en nuestro proyecto.

Además, como resultado de completar la primera versión del proceso de ingeniería, hemos generado una pequeña guía que detalla la creación y el mantenimiento del repositorio. Esta guía describe los elementos que deben incluirse en el repositorio y cómo deben ser versionados. Este artefacto es esencial para establecer una estructura organizada en nuestro repositorio y garantizar la correcta gestión del código fuente y otros archivos relacionados. También se definieron los roles del equipo.

En relación a la tarea de análisis de deuda técnica, hemos producido dos artefactos fundamentales. Por un lado, hemos ingresado issues en GitHub, cada uno con su descripción y clasificación correspondiente, como bugs con su nivel de severidad, mejoras o problemas relacionados con estándares. Estos issues nos permiten rastrear y abordar los problemas técnicos identificados en nuestro proyecto. Además, hemos creado un documento resumen que recopila las issues identificadas, brindando una visión general de la deuda técnica presente y facilitando la toma de decisiones para su resolución.

Las siguientes actividades y artefactos fueron comunes en todas las etapas:

Como parte de la gestión y comunicación del progreso del proyecto, hemos elaborado un informe de avance de la etapa. Este documento proporciona una visión general del estado del proyecto, los desafíos enfrentados junto a las lecciones aprendidas de cada etapa.

Para el registro de esfuerzo y control de tiempos, hemos generado un detalle de registro de esfuerzo por tipo de tarea, siguiendo las actividades del proceso de ingeniería o del tablero de KANBAN. Este registro permite evaluar y analizar la distribución del tiempo y los recursos en el proyecto, facilitando la identificación de áreas de mejora. También, permite evaluar la eficiencia y la carga de trabajo en cada etapa y realizar ajustes de ser necesario.

Finalmente, hemos registrado una retrospectiva en forma de video, en la cual el Scrum Master lideró el proceso. Este video nos brinda la oportunidad de reflexionar colectivamente sobre nuestro trabajo pasado, identificar áreas de mejora y establecer acciones concretas para futuros proyectos. La retrospectiva basada en el método DAKI nos permite obtener retroalimentación valiosa y fomentar la mejora continua en nuestro equipo y procesos de trabajo.

Entrega 2:

En nuestra segunda entrega, hemos realizado actividades clave que han contribuido al avance y mejora del proyecto. Primero, creamos la primera versión del tablero en GitHub, basado en el proceso de ingeniería de KANBAN, proporcionando una visualización clara y organizada de nuestras tareas. Configuramos el pipeline para automatizar la construcción, prueba y entrega del software, garantizando la integridad y calidad del código.

Generamos una guía que explica la vinculación del tablero con el proceso de ingeniería, así como otra guía que describe la configuración del pipeline. El pipeline configurado en GitHub Actions automatiza las actividades de construcción, prueba y entrega.

Seleccionamos dos bugs de alta severidad registrados como issues en GitHub y los reparamos utilizando el enfoque de desarrollo guiado por pruebas (TDD). Los artefactos generados incluyen el código reparado y los casos de prueba correspondientes.

El informe elaborado identifica y justifica los bugs seleccionados, documentando los problemas abordados. El registro de esfuerzo por tipo de tarea, el informe de avance y los totales de registro de esfuerzo proporcionan información valiosa sobre la distribución de tiempo y recursos.

Finalmente, grabamos una retrospectiva basada en DAKI, liderada por el Scrum Master (SM). Además llevamos a cabo una review con el Product Owner para validar las soluciones implementadas, mostrando los bugs corregidos al PO. Ambos en formato de video, para obtener retroalimentación.

Estas actividades y los artefactos generados han fortalecido nuestro proyecto y nos han permitido avanzar de manera eficiente y efectiva en nuestro proceso de desarrollo.

Entrega 3:

En la tercera entrega, empezamos por actualizar el proceso de ingeniería basado en BDD, enmarcado en KANBAN. Esta actualización nos permite adoptar las mejores prácticas de BDD para el desarrollo de software en el backend y mejorar la eficiencia de nuestro proceso de trabajo.

Además, creamos la segunda versión del tablero en GitHub, basado en el proceso de ingeniería. El tablero proporciona una visualización clara y organizada de nuestras tareas, facilitando la gestión y seguimiento de nuestro flujo de trabajo.

Configuramos el pipeline en función del tablero mencionado anteriormente. Esta configuración nos permite automatizar el proceso de construcción, prueba y entrega de nuestro software, asegurando la integridad y la calidad del código.

Para las nuevas funcionalidades, creamos escenarios de prueba en BDD, lo cual nos permite especificar el comportamiento esperado y validar el correcto funcionamiento de las nuevas features implementadas.

Desarrollamos el front-end y el back-end de las nuevas funcionalidades de manera separada. Esto nos permite una mejor organización y garantiza una implementación coherente de las funcionalidades.

Entre los artefactos generados en esta entrega, se encuentran las nuevas versiones de las guías que explican el proceso de ingeniería, el tablero y la configuración del pipeline. Estas guías proporcionan una referencia actualizada y clara sobre el uso y la utilidad de cada elemento en nuestro proyecto.

El informe de avance de la etapa y el detalle de registro de esfuerzo por tipo de tarea nos brindan información valiosa sobre el progreso y la distribución del tiempo y los recursos en esta etapa del proyecto.

Finalmente, hemos registrado una retrospectiva en forma de video, donde el Scrum Master (SM) ha liderado el proceso, y también hemos grabado un video de la revisión de las nuevas funcionalidades

con el PO. Estos videos nos permiten obtener retroalimentación y asegurar la alineación con las expectativas del cliente.

Entrega 4:

En nuestro informe 4, hemos realizado una serie de actividades clave junto con los artefactos correspondientes, los cuales respaldan y justifican estas actividades y su utilidad en el proyecto.

Una de las actividades realizadas fue la automatización del test exploratorio funcional de ambos bugs y ambas nuevas funcionalidades. Para esta prueba, utilizamos la herramienta Selenium. La automatización de estos tests nos permite ejecutarlos de manera eficiente y repetible, asegurando la calidad del software y detectando posibles problemas de funcionamiento. Se incluye la evidencia de ejecución de los casos de prueba, que nos permite validar y documentar el correcto funcionamiento de las funcionalidades y la resolución de los bugs.

Además, realizamos la confección y análisis de métricas DevOps. Estas métricas nos proporcionan información valiosa sobre el rendimiento y la eficiencia de nuestro proceso de desarrollo. El informe de métricas DevOps nos ayuda a identificar áreas de mejora y tomar decisiones informadas para optimizar nuestro proceso y flujo de trabajo.

Registramos una retrospectiva en forma de video, donde el Scrum Master (SM) lidera el proceso y se analizan los aspectos relevantes del trabajo realizado. Además, grabamos un video de la revisión de los bugs con el Product Owner (PO), lo cual nos permite validar y discutir las soluciones implementadas, asegurando que cumplan con los requisitos y expectativas del cliente.

Justificación de las decisiones tomadas durante el proyecto:

Definición de un marco basado en KANBAN

- Está diseñado para mostrar una visualización clara del flujo de trabajo, lo que nos ayuda a identificar cuellos de botella y tomar decisiones rápidas para optimizar la eficiencia y la productividad
- Compartir todos un tablero ayuda a colaborar entre miembros del equipo y la transparencia del mismo ayuda la mejora continua
- KANBAN está enfocado en la entrega continua y reducir los tiempo de ciclos, que es fundamental en el enfoque de DevOps del curso

Análisis de deuda técnica mediante analizadores estáticos, pruebas unitarias, test exploratorios

- La ejecución de las pruebas unitarias definidas previamente sirve para determinar la integridad del build actual, ya que en teoría las pruebas unitarias deberían pasar. Si fallan, se puede deber a deuda técnica identificada por el equipo de desarrollo anterior, o también puede representar un problema de configuración.
- Realizar test exploratorio es esencial, ya que es la manera más simple, eficaz y directa de evaluar si el sistema cumple con los requisitos definidos en la letra.
- Los analizadores de código estático sirven para asegurar que el proyecto cumple con ciertos estándares de codificación. Estos estándares no son funcionales, por lo que no se pueden

encontrar a través de los tests exploratorios ni las pruebas unitarias, por lo que es necesario incorporar herramientas de análisis de código estático.

Utilización de Trunk Based Development y One Piece Flow

- One Piece Flow permite tener un enfoque de desarrollo ágil y continuo, se centra en completar una sola tarea a la vez antes de pasar a la siguiente. Minimiza el tiempo de espera y los cuellos de botella, ya que cada tarea se completa de manera rápida y eficiente. Mantiene un flujo constante de entregas y reduce el riesgo de acumular trabajo incompleto.
- Trunk Based Development permite gestionar el repositorio de manera eficiente al trabajar directamente en la rama principal (trunk) del repositorio. Brinda una visibilidad clara y facilita la colaboración entre los miembros del equipo, ya que todos trabajan en el mismo código base en tiempo real. Al evitar ramas largas y complejas, se reduce la complejidad y el riesgo de integración, lo que permite implementar cambios y mejoras de manera más rápida y segura.

BDD para el desarrollo de funcionalidades

- Nos permite centrarnos en el comportamiento y los resultados esperados de nuestra aplicación. Al definir los escenarios de prueba en lenguaje natural, queda de manera explícita y clara que se interpretó por las funcionalidades pedidas y que se espera de ellas.
- Definir los escenarios con el formato “Dado-Cuando-Entonces” no deja lugar a ambigüedades y trivializa su pasaje a pruebas automatizadas. También fuerza a orientar el desarrollo al cumplimiento de las historias de usuario, y le da un cierto nivel de certeza a los desarrolladores que están cumpliendo con los criterios de aceptación de los usuarios.

DAKI como método para guiar las retrospectivas:

- DAKI proporciona una estructura clara para la reflexión y el aprendizaje durante la retrospectiva. Ayuda a identificar tanto los aspectos positivos como los desafíos y las áreas de mejora, lo que a su vez brinda una base para la toma de decisiones y la implementación de mejoras.
- Fomenta la participación y la colaboración de todo el equipo en la retrospectiva. Promueve un ambiente inclusivo y facilita la generación de ideas y soluciones creativas para abordar los problemas identificados.
- Va de la mano con el enfoque KANBAN. Al identificar lo que se debe dejar, agregar, mantener y mejorar, permite ajustar y optimizar el proceso de trabajo dentro del marco de KANBAN. Ayuda a abordar los desafíos y aprovechar las oportunidades de mejora de manera sistemática y enfocada.

Si bien existen otras técnicas para realizar retrospectivas, elegimos el método DAKI debido a su enfoque estructurado, inclusivo y alineado con el enfoque KANBAN.

Tableros utilizados en el proceso:

Están debidamente justificados en cada una de las entregas correspondientes:

[Justificación tablero entrega 2](#)

[Justificación tablero entrega 3](#)

[Justificación tablero entrega 4](#)

Workflows:

La configuración de dos archivos YAML como workflows en nuestro pipeline de GitHub, uno para automatizar el build y los tests unitarios del backend y otro para el build del frontend de nuestra aplicación web, se justifica por las siguientes razones:

- Automatización del proceso: Ahorra tiempo y esfuerzo en la ejecución manual de ejecutar los builds o tests.
- Modularidad: Al tener dos archivos YAML separados, se mantiene una separación clara y concisa entre el build y las pruebas del backend, y el build del frontend.
- Facilidad de mantenimiento: Permite realizar modificaciones y mejoras de manera más sencilla sin afectar innecesariamente el otro workflow.
- Flexibilidad: Brinda la flexibilidad de adaptar y escalar cada componente de forma independiente. Permite agregar funcionalidades a futuro de manera más ágil y controlada, sin afectar el otro componente.

TDD para el arreglo de bugs en el backend:

- Escribir los tests primero ayuda a delimitar el alcance del bug en la aplicación, además que el hecho de escribir un test que pruebe un bug es una manera efectiva y eficiente (los tests unitarios corren de manera rápida) de asegurarse que el bug no vuelva a ocurrir.
- Tener una suite de tests que verifiquen la integridad de la funcionalidad de todo el código permite refactorizar el código de la aplicación sin tener la preocupación de que por arreglar un bug se rompa otra funcionalidad.

Issues:

En el correr de las entregas hemos abordado diferentes actividades relacionadas con las issues que se fueron reportando, algunas fueron cerradas y otras quedan abiertas como deuda técnica pendiente para el futuro.

A continuación, se presenta un resumen de cada entrega y las acciones realizadas:

Entrega 1: En esta etapa inicial, al haber recibido un proyecto realizado por otras personas, nos enfocamos en buscar, analizar y clasificar las issues detectadas en el sistema. Realizamos un análisis detallado, donde encontramos 20 issues relacionadas principalmente con los requerimientos y necesidades del cliente en relación al negocio. Estas issues fueron consideradas en su mayoría de prioridad alta, ya que estaban directamente vinculadas a las expectativas y objetivos del cliente.

En cuanto a las issues detectadas por las herramientas de análisis de código estático, las hemos agrupado en 9 tipos generales. Estos tipos incluyen errores de sintaxis, problemas de rendimiento, malas prácticas de codificación, entre otros. Esta clasificación nos ha brindado una visión general de los problemas más comunes y nos ha permitido clasificar según severidad y prioridad, donde consideramos que eran de menor prioridad que las issues mencionadas anteriormente debido al alcance del proyecto, ser issues heredados por el código en sí, entre otras cosas.

Hemos detectado un total de 1773 issues en el código del backend utilizando la herramienta ReSharper. Por otro lado, hemos encontrado 1652 issues en el código del frontend utilizando la herramienta ESLint. Si bien no se cerraron dichas issues, de ahí en adelante se trató de seguir los estándares y reglas adecuadas al codificar para no seguir agrandando la cantidad de issues.

Entrega 2: En esta fase, nos centramos en la elección y reparación de dos bugs específicos que fueron registrados previamente como issues, considerando su severidad y prioridad para el proyecto.

Estos bugs fueron seleccionados cuidadosamente, ya que tenían un impacto significativo en el sistema. La resolución de estos bugs contribuyó a mejorar la integridad y la calidad del código.

Entrega 3: En esta entrega, nos dedicamos al desarrollo de dos nuevas funcionalidades que fueron representadas como tarjetas en el tablero Kanban de GitHub. Asociamos a cada funcionalidad una issue específica, lo que nos permitió tener un seguimiento claro y ordenado del progreso. En total, creamos dos nuevas issues y las cerramos satisfactoriamente al completar el desarrollo de las features correspondientes.

Entrega 4: En esta etapa final, nos enfocamos en automatizar las pruebas exploratorias. Testeamos los dos bugs reparados en la segunda entrega y las dos funcionalidades desarrolladas en la tercera entrega. Al igual que en entregas anteriores, representamos estas tareas como tarjetas en el tablero Kanban y les asociamos una issue a cada una. En resumen, creamos cuatro nuevas issues y las dejamos completadas al finalizar las pruebas correspondientes.

Imágenes de las issues cerradas en los tableros:

[Anexo 15](#)

[Anexo 16](#)

Métricas del proyecto:

Referencia:  **Metricas**

Los valores exactos del leadtime, cycletime, touchtime, throughput, y flow efficiency de cada entrega/tarjeta están definidos de manera explícita en la referencia, por lo que consideramos absurdo volver a escribirlas en este documento. En cambio nos enfocamos en analizar y reflexionar sobre las mismas a continuación.

El lead time varió considerablemente de entrega a entrega, pero esto se debe a que no se realizaron la misma cantidad de tarjetas por entrega. Si analizamos esta relación, que sería el throughput, se ve que la entrega 2 y 4 tuvieron el mismo throughput (de 1) mientras que la entrega 3 tuvo un throughput de 0,15. Reflexionamos y consideramos que esto reflejó la pobre eficiencia que tuvimos durante la entrega 3, ya que estuvimos 2 semanas para realizar 2 tarjetas.

En cambio el cycle time varió solo de 1 a 4 días (comparando la fecha en la que se comenzó a realizar la tarjeta con la fecha en la que se terminó). Consideramos que el lead time varió considerablemente por que en su momento estuvimos cargados con otros asuntos; otros obligatorios, parciales y actividades personales de cada uno ya que a la hora de realizar las tarjetas no hubo un cambio tan considerable.

En términos de eficiencia de flujo, un outlier considerable fue la tarjeta "Comprar snacks", que tuvo una eficiencia de flujo de 60,7%, mientras que las tarjetas restantes van de 12% a 25%. Concluimos que esto se debe a que el touch time fue altísimo comparado con el resto de las tarjetas (15,8 hs), lo que contrarrestó el tiempo perdido que tienen las tarjetas esperando por la review con el product owner (uno de los cuello de botella que analizamos en la retrospectiva de la entrega 4).

Estas conclusiones se realizaron en la retrospectiva de la entrega 4.

Frecuencia de entrega:

Para calcular la frecuencia de entrega, dividimos el número total de entregas (8) por el período de tiempo transcurrido (del 14/4 al 23/5).

El número de días entre el 14/4 y el 23/5 es de 40 días (incluyendo ambos extremos).

La frecuencia de entrega se calcula dividiendo el número de entregas (8) entre los días transcurridos (40):

Frecuencia de entrega = número de entregas / días transcurridos = $8 / 40 = 0.2$ entregas por día

Durante el período analizado, la frecuencia de entrega promedio fue de aproximadamente 0.2 entregas por día. Teniendo en cuenta la duración de 40 días, es importante considerar que este lapso incluye días feriados y otros en los que posiblemente no se haya trabajado. Para facilitar la comprensión, podemos estimar que esta frecuencia equivale a realizar aproximadamente 8 entregas por mes, tomando como referencia un mes típico de 30 días.

Reflexiones sobre el aprendizaje.

Drop (¿Qué hicimos mal y deberíamos dejar de hacer?)

Entrega 1:

- La retrospectiva fue muy general y por lo que no aportó tanto valor, entre otras cosas porque no se definió una lista clara de acciones a seguir.
- La convención para nombrar las ramas y su definición no fue la más eficiente pensando en el proyecto a nivel general. Eso llevó a que no se respeten adecuadamente en el correr de las entregas..
- El análisis de deuda técnica apoyado en las herramientas de código estático no las detallamos muy bien.

Entrega 2:

- En la definición del proceso de ingeniería, se incluye la definición de requerimientos aunque se trataba solamente de arreglar bugs.

Entrega 3:

- No aclaramos cómo se tenían que mover las tarjetas de bug en el nuevo tablero.
- No cumplimos el orden indicado de desarrollo en la letra. Se realizó una charla entre el desarrollador de frontend y de backend para definir la API de las nuevas funcionalidades y luego cada uno las desarrolló en paralelo.

Entrega 4:

- En el análisis de métricas, se registra el lead time y el cycle time en días pero el touch time en horas trabajadas.
- Durante el proyecto, implementamos tableros Kanban para mejorar la gestión del mismo. Sin embargo, es necesario mejorar el manejo del tablero, específicamente en la 4ta entrega. Aunque cumplimos con el principio de "One Piece Flow" (OPF), nos dimos cuenta de que no estábamos moviendo las tarjetas a medida que las completábamos. En lugar de eso, las movimos todas al mismo tiempo al percatarnos de nuestro error. Esta situación fue única y se debió principalmente a la velocidad con la que podíamos realizar las tarjetas. Como conocíamos bien las tareas de antemano, no consideramos necesario revisar constantemente las tarjetas, lo que provocó un descuido en la manipulación adecuada del tablero Kanban.

Add (¿Qué no hicimos y deberíamos comenzar a hacer?)

- En las funcionalidades implementadas con BDD, se identificó una mejora necesaria relacionada con la presentación del precio total de la compra en el frontend. Específicamente, se detectó que no se mostraba correctamente el cálculo que incluía el precio de los tickets y el precio de los snacks. Para brindar una experiencia más completa y transparente al usuario, se recomienda realizar las modificaciones necesarias en la interfaz para mostrar el precio total de la compra de manera clara y concisa, incluyendo tanto los

tickets como los snacks seleccionados. Esto ayudará a evitar confusiones y brindará una visión más precisa de los costos totales para los usuarios.

Keep (¿Qué hicimos bien y deberíamos continuar haciendo?)

- El proceso de ingeniería se definió con gran éxito en la teoría en todas las entregas. Desde la fase de planificación, se estableció como una guía invaluable para orientar nuestro trabajo durante el resto de la entrega.
- Se realizaron excelentes elecciones en cuanto a los bugs que debíamos mejorar. Esto se debió a una meticulosa clasificación de los mismos durante la primera entrega, lo cual facilitó enormemente la selección de los bugs a abordar en la segunda entrega.
- Es fundamental mantener el correcto uso de TDD (Desarrollo Guiado por Pruebas) en la realización del código. Nos complace observar que los miembros del equipo han adoptado con facilidad esta técnica de programación, ya que resultó muy simple de utilizar durante la etapa obligatoria. Continuemos aprovechando los beneficios de TDD para garantizar un desarrollo de código más robusto y confiable.
- La automatización del test exploratorio con Selenium se llevó a cabo de manera excelente y resultó muy cómoda de implementar, gracias a la herramienta utilizada. La ejecución del test exploratorio se simplificó significativamente gracias a las capacidades y funcionalidades que ofrece Selenium. Esto nos permitió ahorrar tiempo y esfuerzo al automatizar las pruebas, mejorando la eficiencia en el proceso de detección de errores.

Improve (¿Qué hicimos relativamente bien, pero podríamos mejorar?)

- Debemos mejorar la adhesión a la guía de mantenimiento del repositorio y nombrar las ramas de manera más efectiva. Los cambios que realizaremos serían los siguientes:
 1. Asegurarse de seguir estrictamente la guía de mantenimiento del repositorio en todo momento. Esto garantiza la coherencia y la facilidad de colaboración con otros miembros del equipo.
 2. Al nombrar las ramas, utilizar nombres descriptivos y significativos que reflejen claramente el propósito y el contenido de la rama. Evitar nombres genéricos o ambiguos que puedan generar confusión. Esto facilita la identificación y el seguimiento de las ramas en el repositorio.
 3. Utilizar convenciones de nomenclatura consistentes para las ramas, esto ayuda a organizar y categorizar las ramas de manera más clara.
 4. Documentar adecuadamente cada rama, proporcionando una descripción clara y concisa de su propósito, objetivos y alcance. Esta información es útil para que otros miembros del equipo comprendan rápidamente el contenido y la intención de cada rama.
- Mejorar la división de secciones del proyecto por parte del equipo para realizar pruebas exploratorias de la aplicación y detectar errores de una manera más ordenada y eficaz.
- Refinar la descripción de nuestro proceso para identificar los bugs, evitando limitarnos únicamente a mencionar el error. Aplicaremos un mayor rigor al documentar los bugs en los issues correspondientes, asegurando así un registro detallado de cómo se descubrió cada bug.
- El análisis de las métricas no se realizó de manera óptima durante el proyecto, ya que no se tuvieron en cuenta todos los detalles necesarios para recopilar información de manera precisa sobre el proceso de trabajo. En este sentido, se sugiere que, en futuros proyectos, se comience desde cero utilizando las métricas de DevOps y se enfrente directamente a los

desafíos que conlleva su implementación. Además, se recomienda automatizar el cálculo de las métricas siempre que sea posible. Abordar este problema lo antes posible permitirá encontrar formas más efectivas de recopilar datos y ayudará al equipo a acostumbrarse a documentar detalladamente sus acciones y su cronología. Asimismo, es fundamental asegurarse de que los tableros reflejen de manera automática todo el trabajo realizado, siempre y cuando el equipo realice una manipulación adecuada de los mismos.

- Durante el proyecto, las revisiones con el Product Owner (PO) fueron realizadas de manera impecable. Sin embargo, se considera que habría sido aún más beneficioso tener una reunión con el PO después de completar cada tarjeta, en un formato de mano a mano entre el desarrollador y el PO. Esto habría permitido que las tarjetas se movieran más rápidamente a la etapa de "Done" y se mantuviera el principio de "One Piece Flow" (OPF). Al tener estas reuniones individuales, el desarrollador podría recibir retroalimentación directa y aclarar cualquier duda o inquietud específica sobre la tarjeta completada. Esto facilita una mejor comprensión de los requisitos y expectativas del PO, lo que a su vez aceleraría el proceso de validación y aprobación de las tarjetas. Además, esta comunicación más cercana y directa fomentaría una colaboración más efectiva entre el equipo de desarrollo y el PO, permitiendo una alineación más rápida y eficiente en el proyecto. Considerando estas mejoras, se promovería una mayor agilidad en el flujo de trabajo, reduciendo el tiempo que las tarjetas permanecen en progreso y maximizando la eficiencia en la entrega de las funcionalidades al PO.

Lecciones aprendidas.

Las lecciones las enfocamos como tips a futuros cursantes de la materia, por lo que las escribimos como si les estuviéramos hablando a un futuro equipo. Desarrollamos estos tips en función de nuestras reflexiones sobre el aprendizaje.

Tips:

- Luego de terminar el contenido de la entrega, dejarla "pronta" toma más tiempo del que parece. Dejen aproximadamente 5 horas de margen para emprolijar el documento, realizar la retrospectiva y realizar la review con el product owner (si es que realizan una sola al final de la entrega).
- A medida que se realizan las actividades junto a los correspondientes artefactos, es recomendable ir documentando para que sea más natural, eficiente y productivo que documentar recién al final del todo.
- Cuando vayan a configurar el pipeline, hacerlo sobre develop y no main, ya que ensucia la rama main que nosotros al menos la usamos para las entregas. Como estrategia complementaria configurar las acciones para ambas ramas main y develop, lo que aporta un espacio seguro como la rama develop para probar definitivamente el pipeline de Github.
- Cuando realicen el análisis de deuda técnica, registren como replicar el bug de una manera no vaga (nosotros lo registramos, para que sea explícito, con un "Dado, cuando, entonces").
- Si dejan la review con el P.O para al final de la iteración, luego de realizar todas las tarjetas, tengan en cuenta que por definición es un cuello de botella y les va a afectar negativamente a la eficiencia de flujo. Es mejor llevar a cabo varias mini revisiones con el Product Owner y el desarrollador encargado de la tarjeta, para marcarla como 'Done', y posteriormente realizar una revisión más exhaustiva que abarque todo lo visto durante la entrega, en la cual participe todo el equipo.

- Tener una idea medianamente clara del pipeline antes de implementarlo, ya que si no se generan muchos commits innecesarios e inconsistencias entre ramas. A nosotros en particular nos pasó que hubo una descoordinación en los tests que se ejecutaban entre ramas, entonces nos fallaba el pipeline cuando en realidad no había errores en el código.
- Realicen el registro y análisis de las métricas básicas de devops (las que se incluyen en los temas del RAT 1) lo antes posible. Esto permite realizar 2 cosas:
 - Una retrospectiva fundamentada en evidencia empírica - y por lo tanto más útil que una basada en sesgos.
 - Encontrar errores conceptuales en el uso y registro de estas métricas. Por ejemplo, a pesar de que registramos fecha de inicio y de fin desde la entrega 2, nosotros nos dimos cuenta a la hora de analizar las métricas en el informe 4 que el lead time y el cycle time se definen suponiendo que uno tiene un horario fijo de trabajo, lo cual no se aplica para nada al proyecto.
- Intenten familiarizarse con las herramientas a usar en la entrega lo antes posible. Realizar una estimación de tiempo sin tener clara la herramienta lleva por definición a imprevistos. Por ejemplo, para la entrega de BDD nosotros consideramos de antemano que iba a ser prácticamente lo mismo que escribir los test unitarios de clase de materias pasadas. Sin embargo, en BDD la unidad es la user story, lo cual no se mapea 1 a 1 con las clases que hay que desarrollar, por lo que qué mockear y que no mockear por cada test de Specflow no es obvio y conlleva un ligero cambio de mentalidad.
- Usar labels en los issues para marcar la severidad y prioridad de cada bug encontrado es una de las cosas más útiles que hemos usado para clasificar los errores ya que quedan centralizados en el repositorio.

Conclusiones.

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

Los resultados de nuestro trabajo demuestran la aplicación y beneficios de utilizar las metodologías ágiles, como KANBAN y DevOps, junto con otras técnicas y conceptos, en el desarrollo de proyectos de software. Hemos logrado implementar prácticas de CI/CD, QA y testing automático en proyecto de código heredado, mejorando la calidad y eficiencia del proceso de desarrollo.

¿Qué consecuencias/implicancias tiene lo anterior?

Las consecuencias de nuestra implementación incluyen una mayor productividad, mejor colaboración entre los miembros del equipo, detección temprana de errores, reducción de la deuda técnica y entrega más rápida y confiable del software. También hemos promovido una cultura de mejora continua y aprendizaje en nuestro equipo.

¿Por qué son importantes esas implicaciones o consecuencias?

Estas implicaciones son importantes porque nos permiten desarrollar software de alta calidad, adaptarnos rápidamente a los cambios y requisitos del cliente, y brindar un valor agregado. Además,

la detección temprana de problemas y la reducción de la deuda técnica evitan costos y retrasos en el futuro, asegurando la mantenibilidad del proyecto.

¿A dónde nos conducen?

Nuestro trabajo nos lleva hacia una mejora continua en la forma en que desarrollamos software, adoptando prácticas ágiles, automatización y enfoques colaborativos. Nos proporciona una base sólida para abordar proyectos futuros y nos equipa con herramientas y conocimientos para enfrentar los desafíos de desarrollo de software en el mundo real de manera eficiente.

En resumen, nuestras conclusiones destacan la importancia y los beneficios de las metodologías ágiles y las técnicas aplicadas en el desarrollo de software. Nuestro enfoque en la calidad, la eficiencia y la mejora continua nos permite obtener resultados positivos y nos prepara para enfrentar futuros desafíos en el desarrollo de proyectos de software.

Guía de instalación para desarrollo y despliegue en producción.

Guia para instalación del despliegue

Herramientas necesarias

- Docker
- Microsoft SQL Server Management Studio

Restaurar la base de datos.

1. Ejecutar la aplicación Microsoft SQL Server Management Studio.
2. Conectarse al servidor por defecto (no es relevante a qué servidor se conecta).
3. Hacer click derecho en la carpeta "Databases", y luego en "Restore Database"
[Anexo 1](#)
4. Apretar la opción de device y apretar el botón de los 3 puntitos.
[Anexo 2](#)
5. Apretar el botón de "Add". Si le sale de antemano otras opciones en "Backup media", remueva las, ya que pueden ocasionar problemas.
[Anexo 3](#)
6. Copiar la ruta de acceso a la carpeta donde están guardados los .bak del servidor
[Anexo 4](#)
7. Pegar la ruta en el buscador de archivos
8. Ir a la carpeta /Despliegue/database y copiar el .bak que se encuentra en esa carpeta.
9. Pegarlo en la carpeta donde están guardados los .bak del servidor
10. Refrescar la pestaña en la que nos quedamos en el paso 6. Ahora debería aparecer nuestro .bak
[Anexo 5](#)
11. Seleccionarlo y apretar "OK"
[Anexo 6](#)
12. Seleccionar nuestro .bak en "Backup media" (debería ser la única opción) y volver a apretar "OK"
[Anexo 7](#)

13. Apretar "OK"

[Anexo 8](#)

14. Luego de unos segundos, nos debería saltar este anuncio lo que indica que se restauró nuestra base de datos con éxito.

[Anexo 9](#)

Despliegue del backend

1. Ir a la carpeta ./Despliegue/backend/

2. Ejecutar el archivo "ArenaGestor.API.exe"

Esto va a levantar el backend en el puerto 5000:5001. No se puede cambiar el puerto porque está hardcodeado en el frontend.

Despliegue del frontend

1. Parados en la carpeta ./Despliegue/, ejecutar el comando: docker-compose up

Esto va a levantar el frontend en el puerto 8080, ya que así está definido en el archivo docker-compose.yml. Para cambiarlo, basta con cambiar el puerto definido en este archivo con un editor de texto y volver a ejecutar el comando.

Guia para instalación y uso del ambiente de desarrollo

Herramientas

- Microsoft SQL Server Management Studio
- Visual Studio
- Angular versión 13.2.6

Restauración de la base de datos

Seguir los pasos de la guía de instalación del despliegue, ya que la base de datos es la misma en ambos ambientes.

Correr el backend en modo desarrollo

1. Abrir con Visual Studio el archivo "ArenaGestor.sln" que se encuentra en ./Obligatorio/codigo/ArenaGestor
2. Iniciar la aplicación sin depurar

Correr el frontend en modo desarrollo

1. Parado en ./Obligatorio/codigo/ArenaGestorFront, ejecutar el comando "npm install"
2. Luego ejecutar el comando "ng serve --open"

Ejecución de las pruebas unitarias y las de Specflow

1. Abrir con Visual Studio el archivo "ArenaGestor.sln" que se encuentra en ./Obligatorio/codigo/ArenaGestor
2. En el menú de "Prueba", apretar "Ejecutar todas las pruebas"

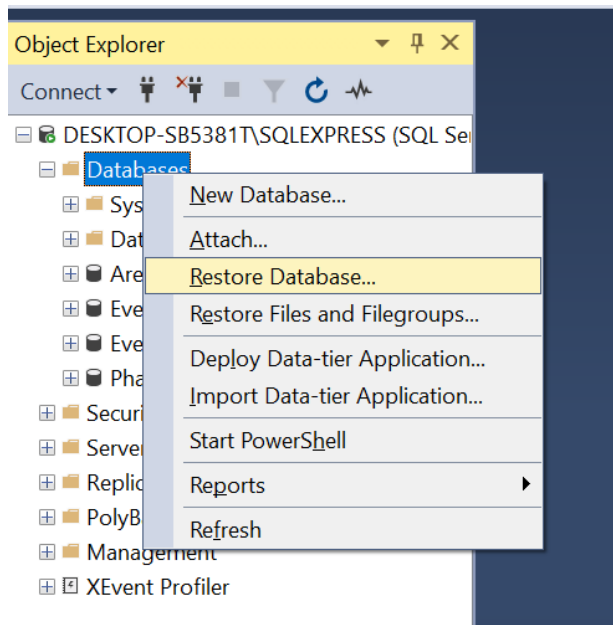
[Anexo 10](#)

Ejecución de los casos de prueba de selenium

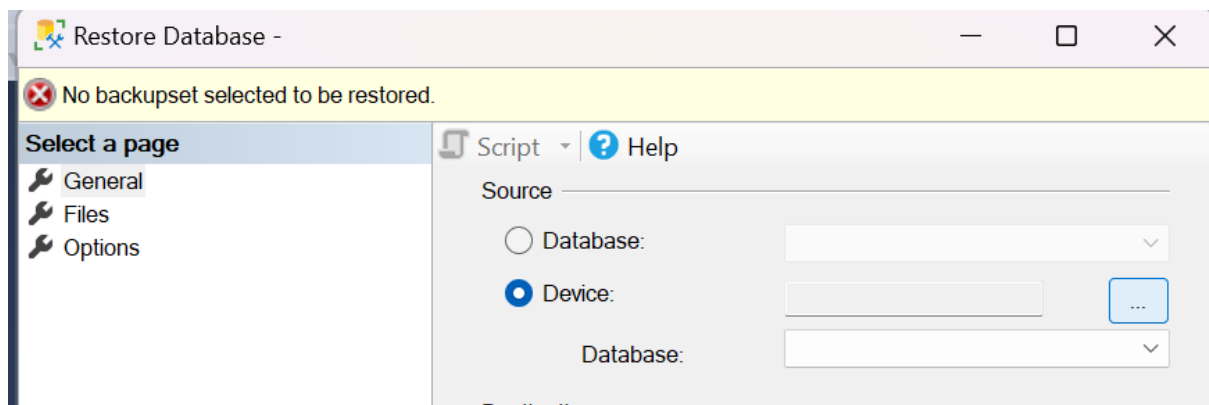
1. Mirar el documento de la entrega 4: [Evidencia de ejecucion de casos de prueba.pdf](#)
2. Se debe de tener el back y el front corriendo, el back debe de correr contra la base de datos llamada BaseDeDatosSelenium.back. La aplicación entera se levanta como se explicó anteriormente [Restaurar la base de datos](#), [Despliegue del backend](#) y [Despliegue del frontend](#).
3. Procedemos en el navegador de Chrome a buscar la siguiente extensión que nos permite usar Selenium IDE y agregarla. [Acá encuentras la extensión](#). [Anexo 12](#).
4. Es momento de abrir selenium con el navegador y abrir nuestra colección de test, mirar [Anexo 13](#).
5. Debemos correr los test, para ello es esencial que el usuario use una base de datos sin alterar del repositorio y se este en Home de la página web [Anexo 14](#).

Anexos

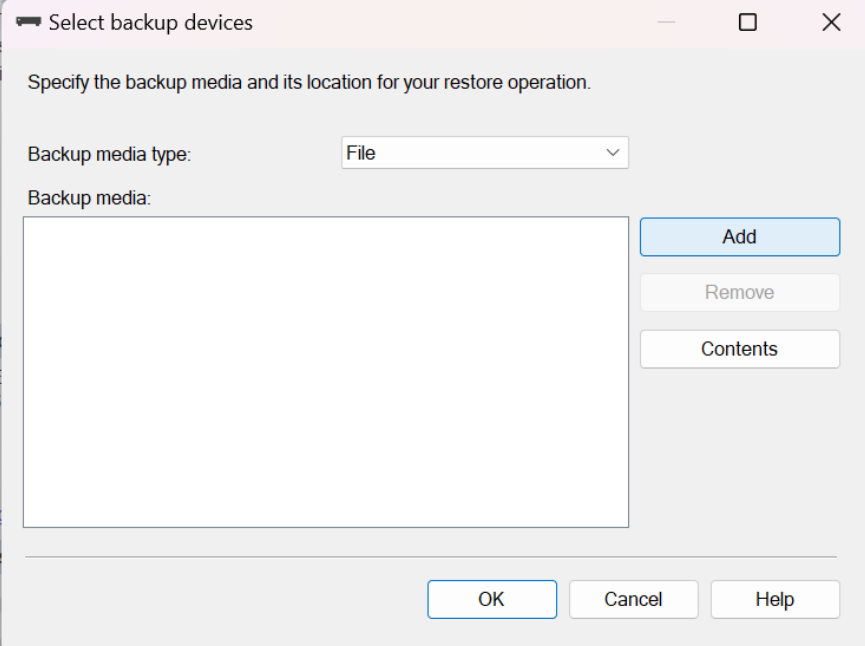
Anexo 1



Anexo 2



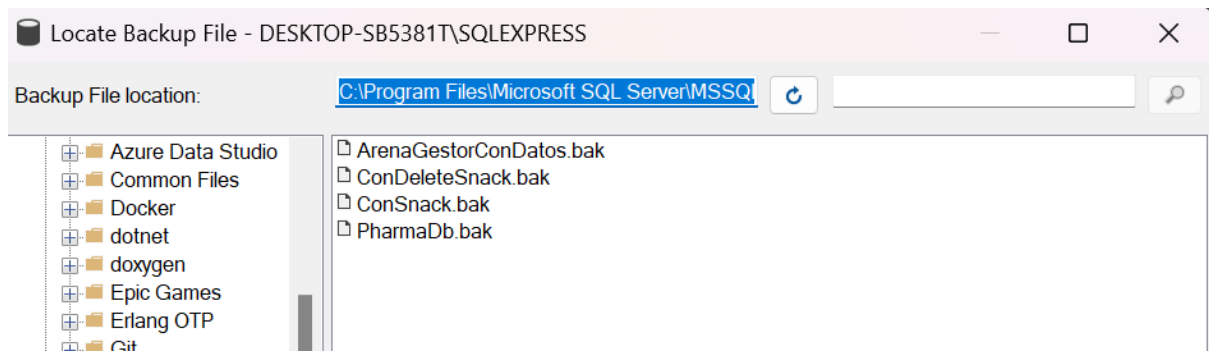
Anexo 3



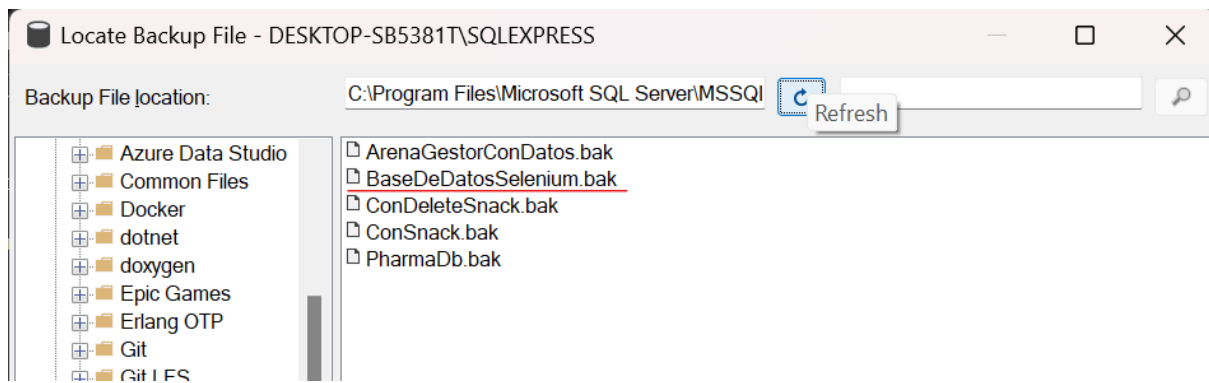
The image shows a Windows-style dialog box titled "Select backup devices". The title bar includes a minimize button, a maximize button (disabled), and a close button. The main content area has a light gray background and contains the following elements:

- A header text: "Specify the backup media and its location for your restore operation."
- A label "Backup media type:" followed by a dropdown menu currently showing "File".
- A label "Backup media:" followed by a large, empty rectangular list box.
- To the right of the list box are three buttons: "Add" (highlighted in blue), "Remove" (disabled), and "Contents" (disabled).
- At the bottom of the dialog are three buttons: "OK" (highlighted in blue), "Cancel", and "Help".

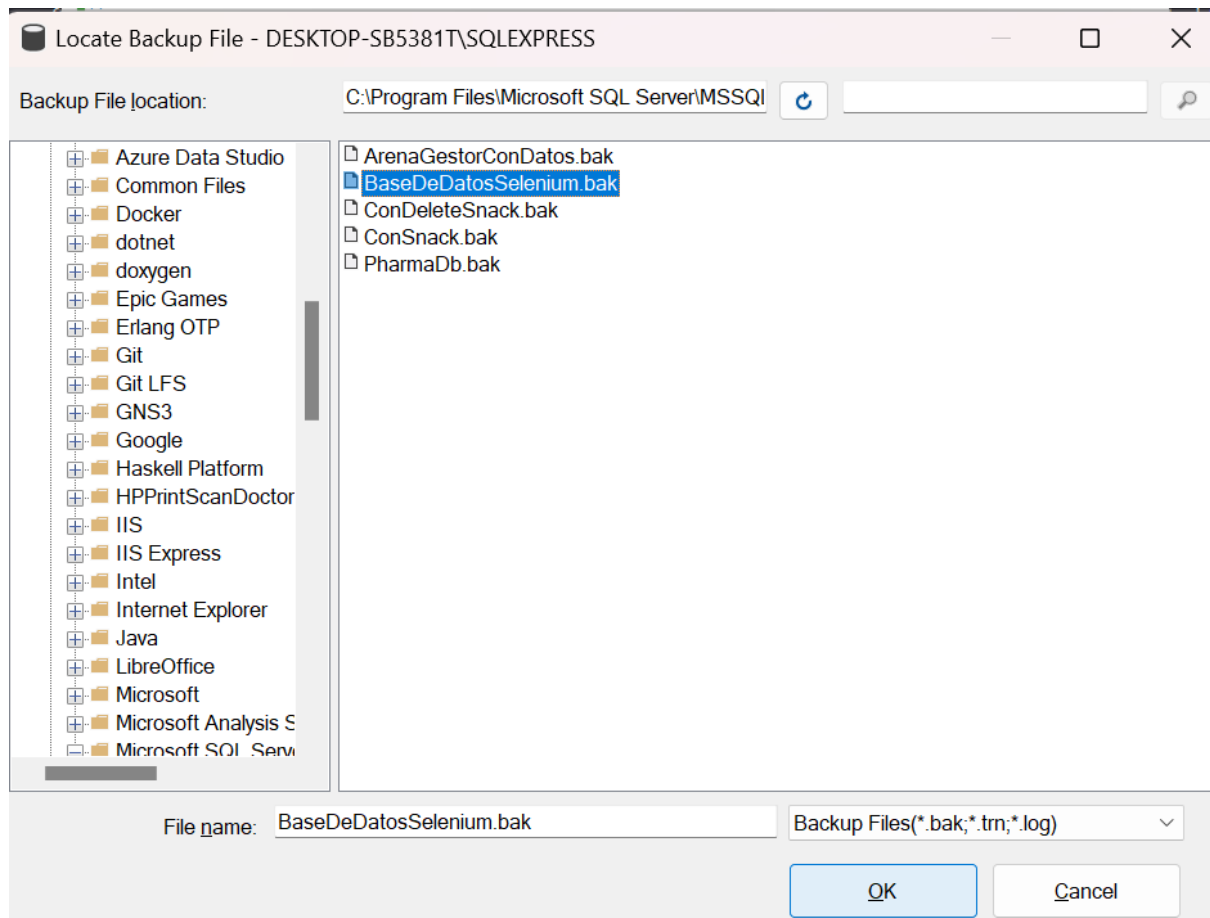
Anexo 4



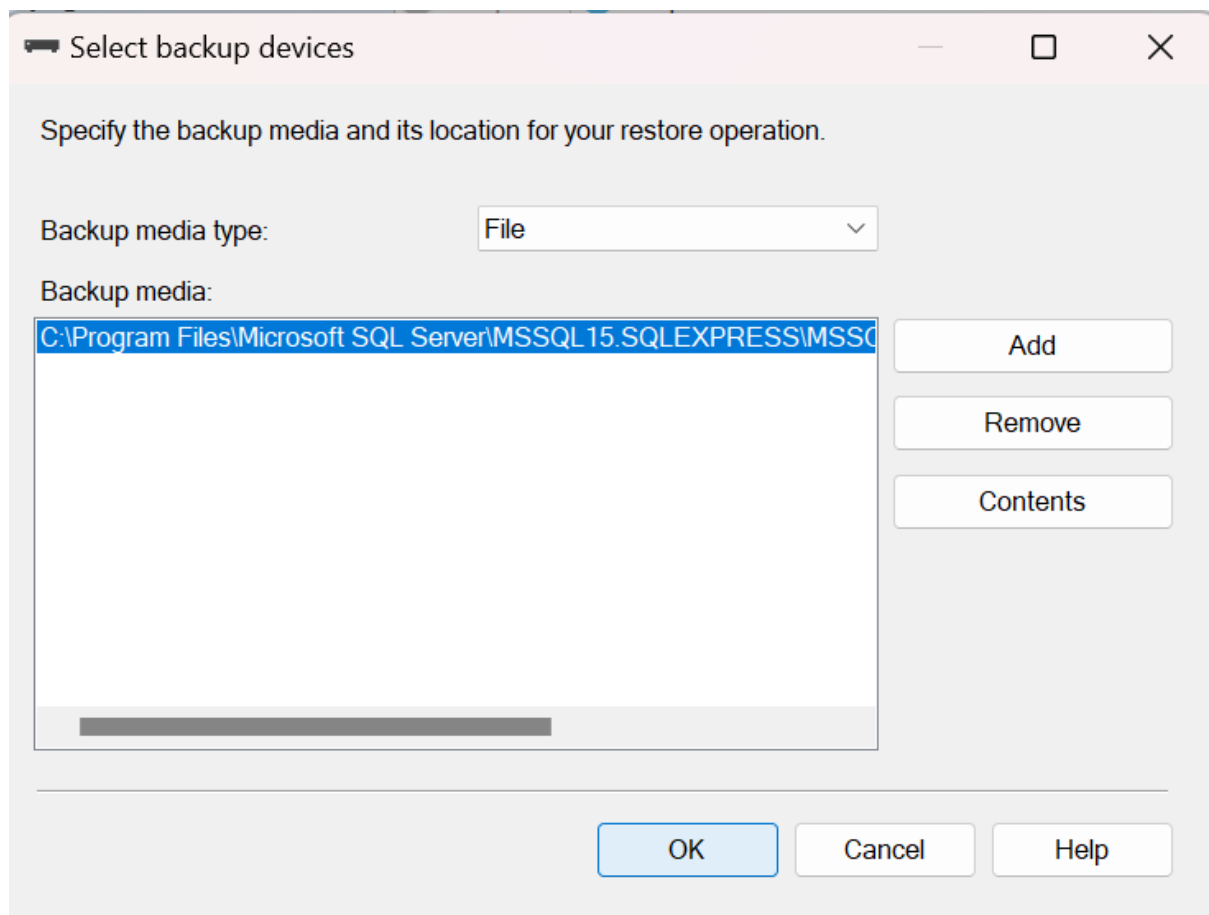
Anexo 5



Anexo 6



Anexo 7



The image shows a Windows-style dialog box titled "Select backup devices". The title bar includes a minimize button, a maximize button (disabled), and a close button. The main area of the dialog contains the instruction "Specify the backup media and its location for your restore operation." Below this, there is a label "Backup media type:" followed by a dropdown menu currently set to "File". Underneath, the label "Backup media:" is followed by a list box. The list box contains a single entry: "C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSO". To the right of the list box are three buttons: "Add", "Remove", and "Contents". At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

Select backup devices

Specify the backup media and its location for your restore operation.

Backup media type: File

Backup media:

- C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSO

Add

Remove

Contents

OK Cancel Help

Anexo 8

Restore Database - ArenaGestorDB

Ready

Select a page

General

Files

Options

Connection

DESKTOP-SB5381T\SQLEXPRESS [DESKTOP-SB5381T\Usuario]

[View connection properties](#)

Progress

Done

Script Help

Source

Database:

Device:

C:\Program Files\Microsof

...

Database:

ArenaGestorDB

Destination

Database:

ArenaGestorDB

Restore to:

The last back

Timeline...

Restore plan

Backup sets to restore:

Restore	Name	Component	Type
<input checked="" type="checkbox"/>	ArenaGestorDB-Full Database Backup	Database	Full

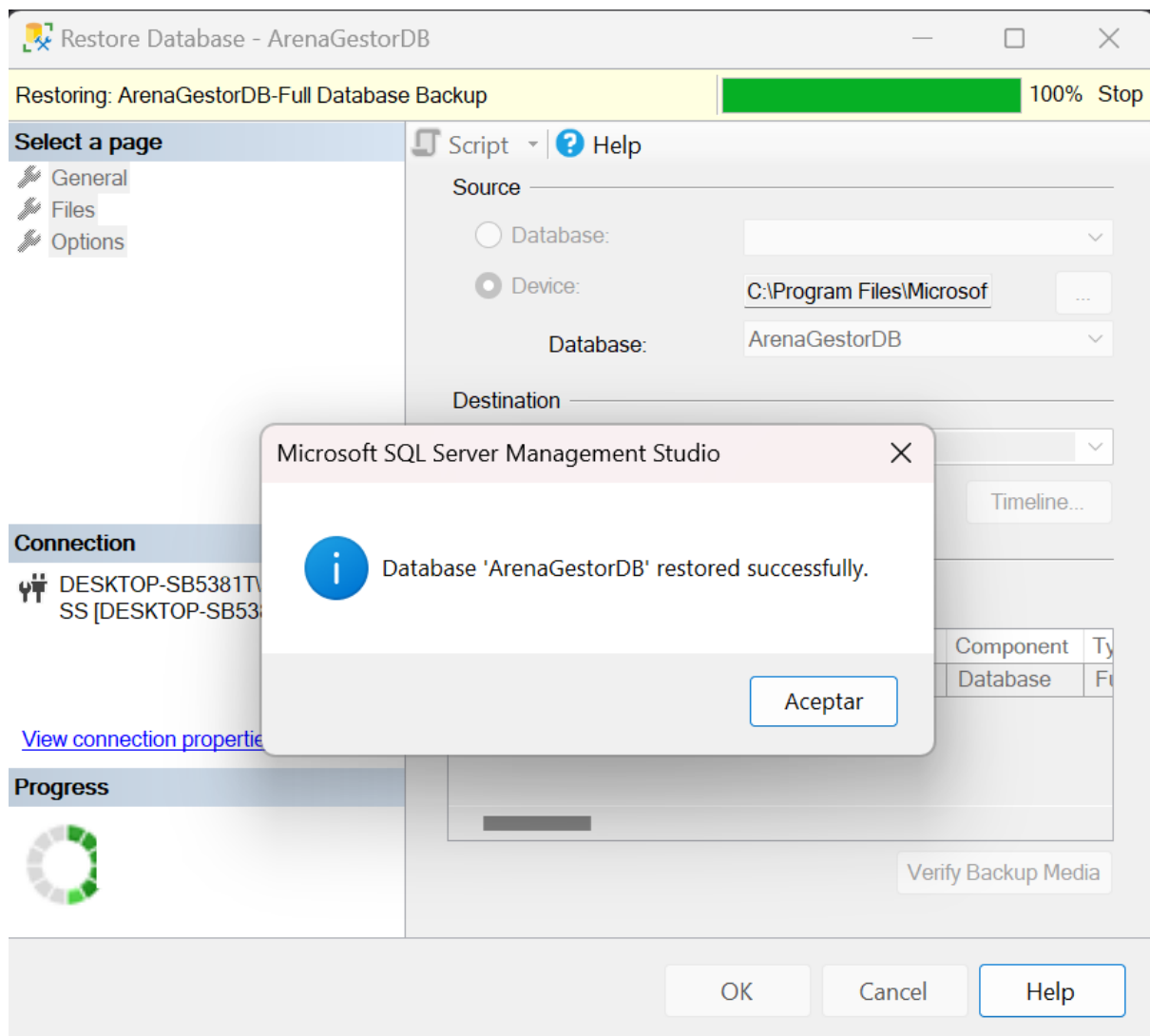
Verify Backup Media

OK

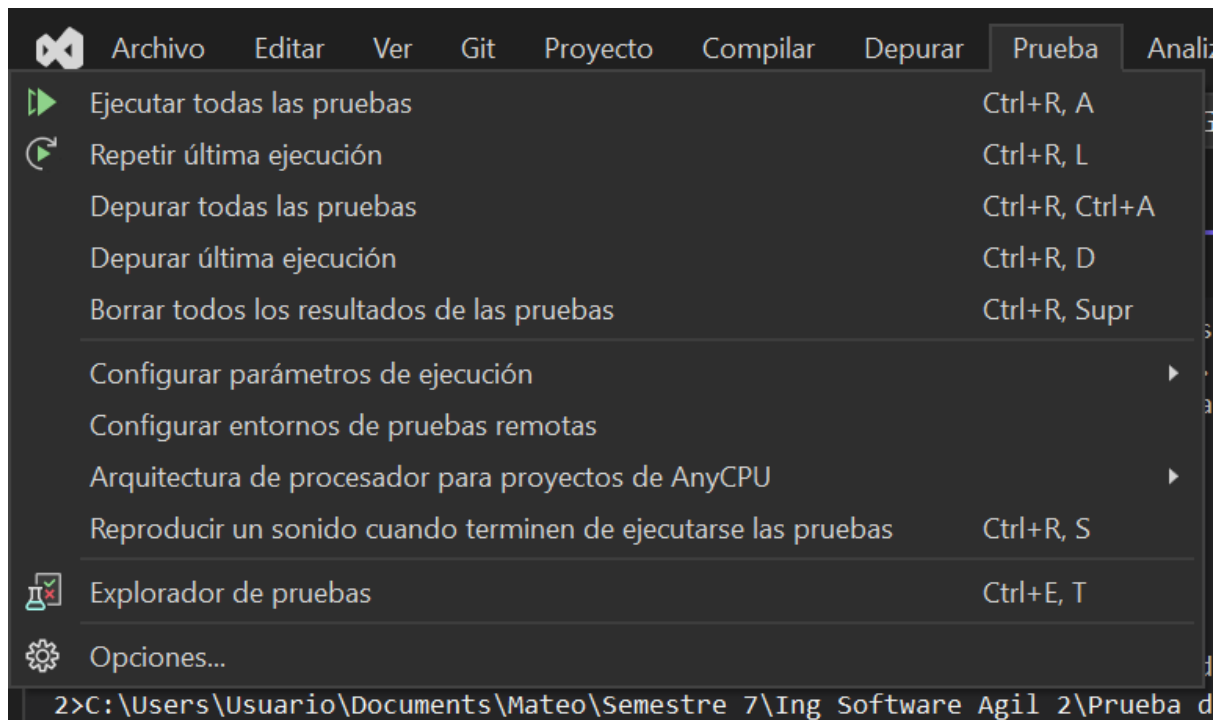
Cancel

Help

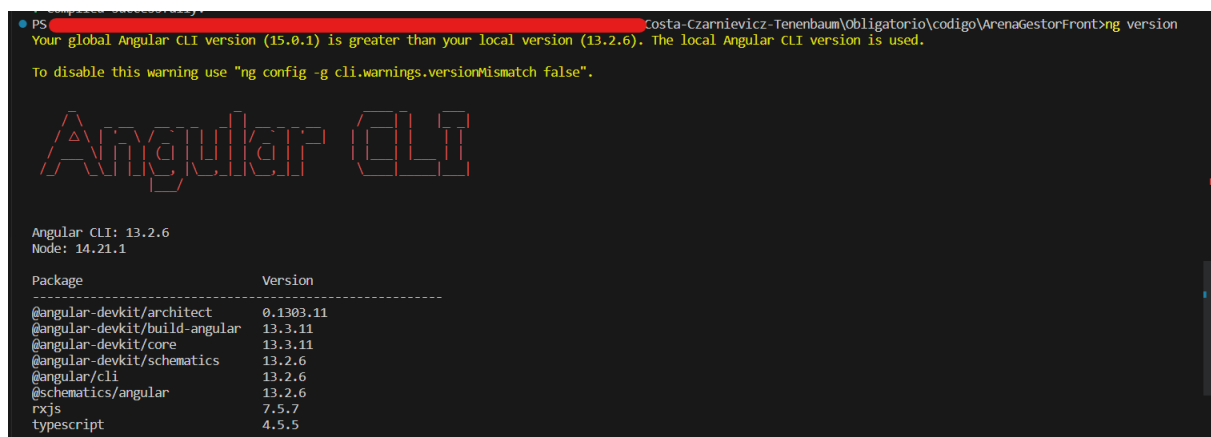
Anexo 9



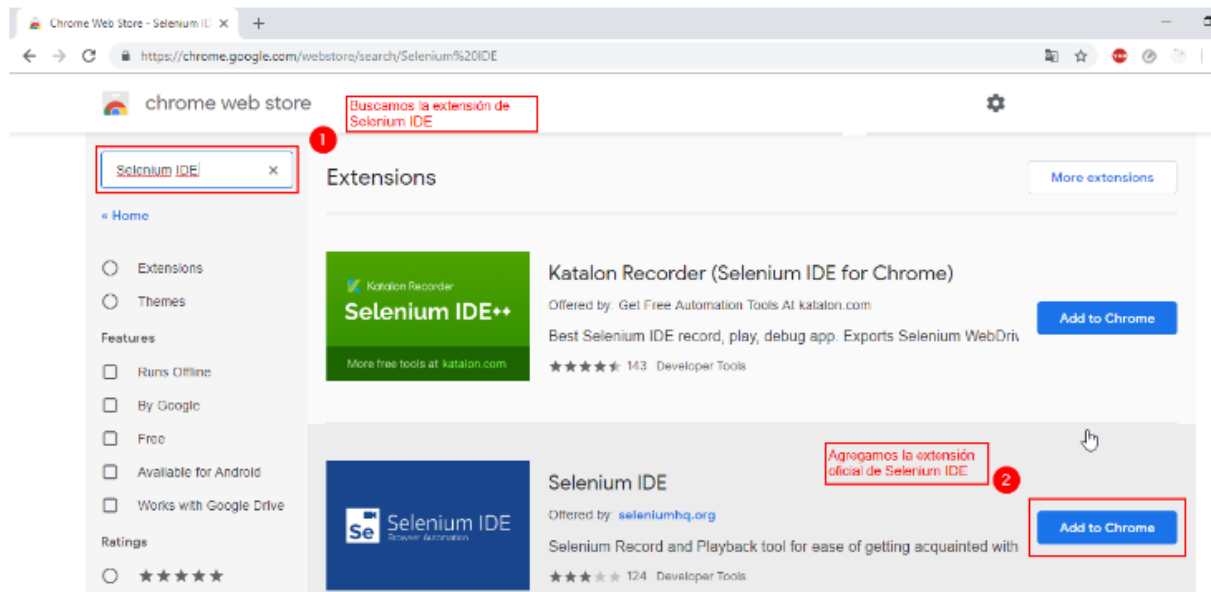
Anexo 10



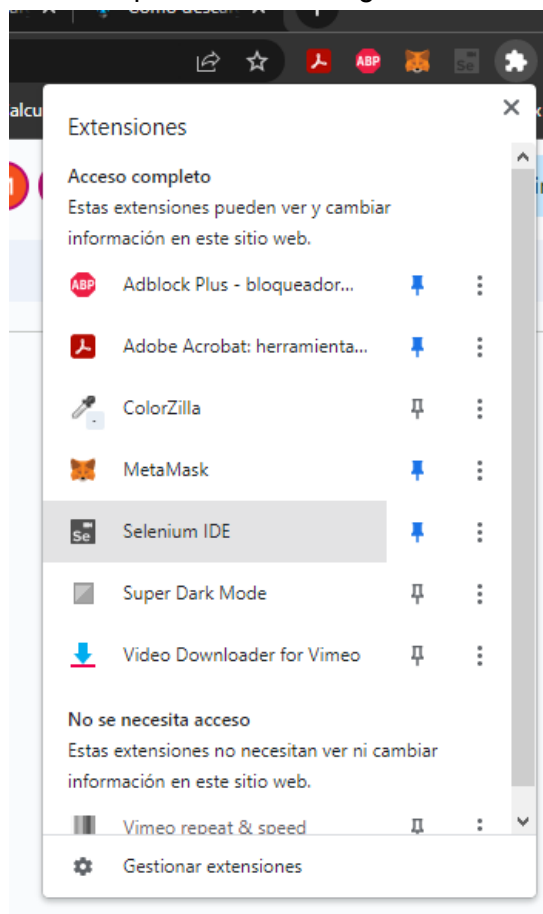
Anexo 11



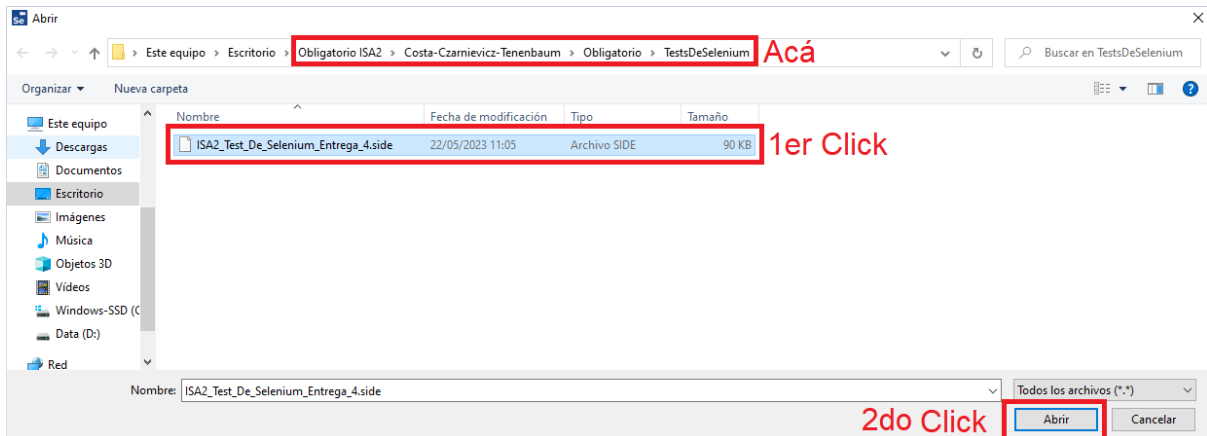
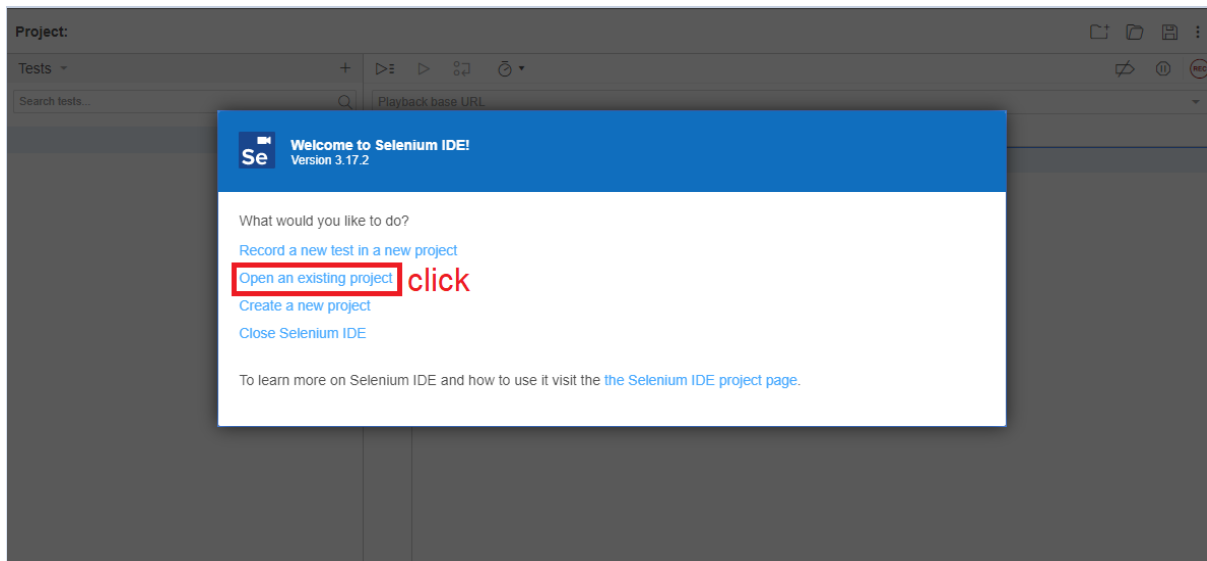
Anexo 12



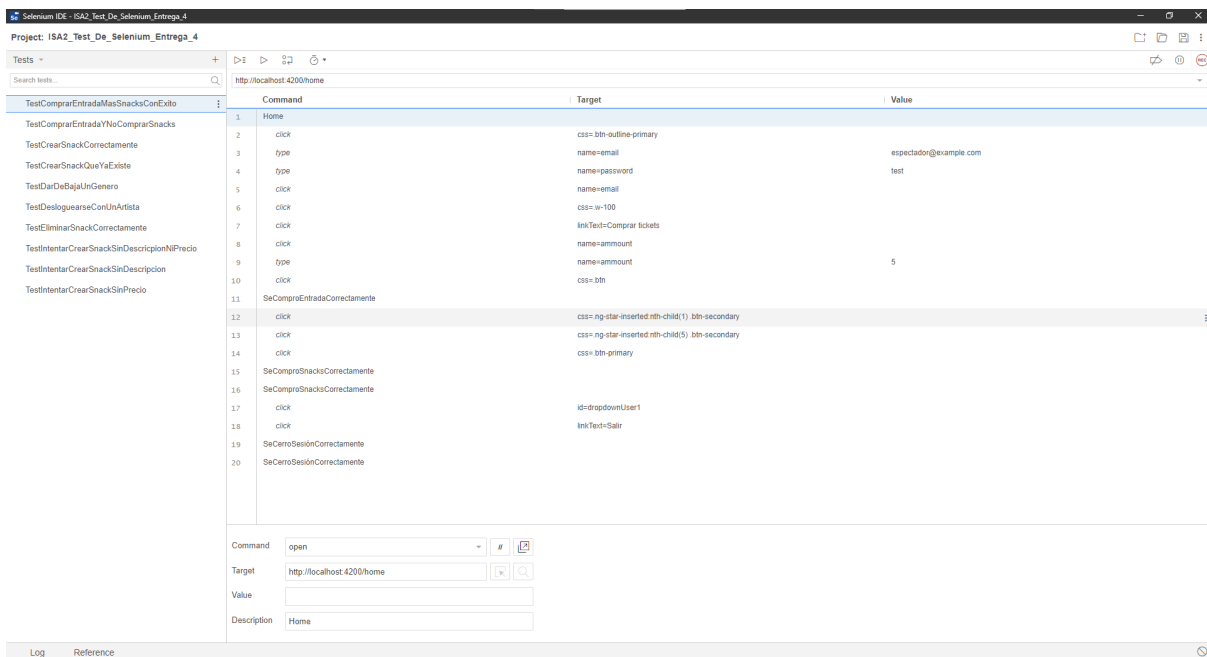
Debe de quedar en el navegador así:



Anexo 13

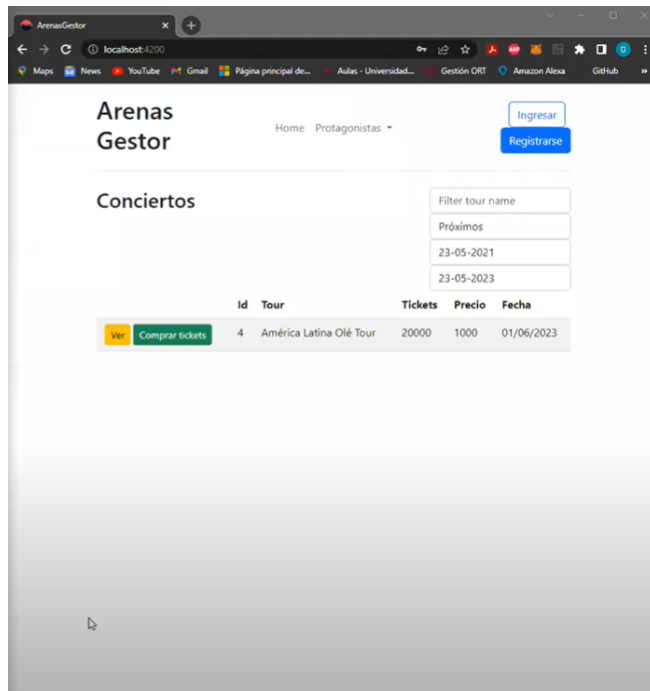


Vemos esto:

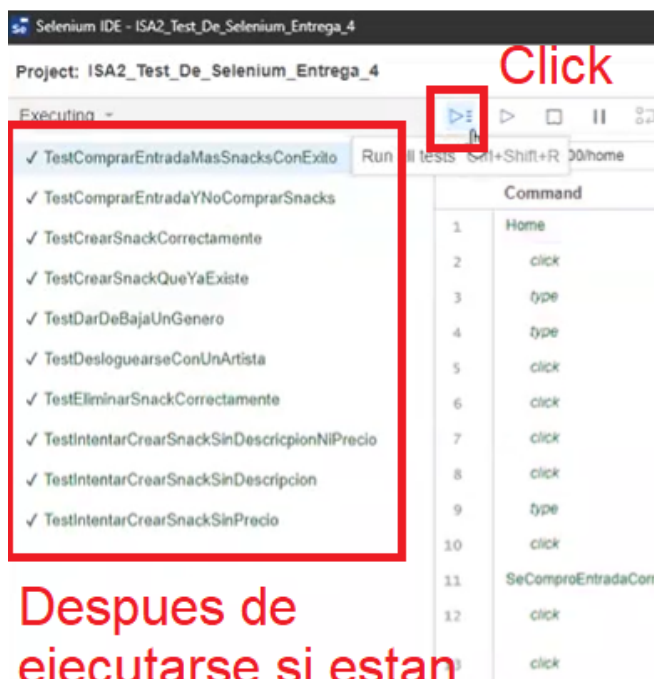


Anexo 14

Estamos acá:



Vamos al IDE:



Despues de
ejecutarse si estan
todas verdes
funcionaron bien todos
los test

Anexo 15



Anexo 16

