



2131119919 - López Delgado Luis Antonio

2131120207 - Medina Licona Gerardo de Jesús

2131120358 - Rodríguez canales Yasmin

2131120079 - Xithe Pintado Daniel

Software 07_02

Programación concurrente

Ingeniería en software

Universidad Politécnica de Pachuca

Jazmín Rodríguez Flores

Septiembre-diciembre 2023

Índice

Introducción.....	3
Justificación.....	3
Requisitos del sistema.....	4
Historias de usuario:.....	4
Organización de tareas.....	8
Medina Licon Gerardo de Jesús.....	8
Rodríguez canales Yasmin.....	8
López Delgado Luis Antonio.....	8
Xithe Pintado Daniel.....	8
Casos de pruebas de funcionalidad.....	9
Introducción al Caso de Prueba - Registro de pedidos:.....	9
Introducción al Caso de Prueba 2 - consulta de pedidos:.....	11
Introducción al Caso de Prueba 3 - Orden de preparación de pedidos:.....	13
Introducción al Caso de Prueba 4 - Registro de pedidos en archivo de texto:.....	15
Pruebas de Software.....	16
Alcance del Proyecto:.....	16
Características a Ser Probadas:.....	18
Requisitos Funcionales:.....	19
Análisis de riesgos y Contingencias:.....	21
Matriz de trazabilidad.....	23
Pruebas Funcionales:.....	24
· Prueba funcional 1 - Agregar comanda válida:.....	24
Salida en consola:.....	25
Prueba funcional 2 - Verificación del Algoritmo de planificación Round-Robin:.....	25
Salida en consola:.....	27
Prueba funcional 3 - Mostrar comandas no completadas:.....	27
Salida en consola:.....	29
Prueba funcional: Cálculo de tiempo de preparación según cantidad de un plato.....	29
Herramientas:.....	29
Descripción:.....	29
Herramientas.....	31
Entidades de los campos.....	32
Patrón de concurrencia.....	33
Round-robin.....	33
Uso de hilos.....	35
Función escribir_base_de_datos.....	35
Función menu.....	36
Lock y sincronización.....	37
Commits en Github.....	39
Gráfica de contribución.....	40

Insights.....	41
Evaluación 😊.....	42
Información del proyecto:.....	42
GitHub:.....	42
Drive:.....	42

Introducción

La tecnología es algo con lo que se están familiarizadas hoy en día las personas. Podemos mirar a nuestros alrededores y entender que está más presente de lo que creemos, implementada en diversas cosas que facilitan la vida cada vez más, por ello, se busca aplicar lo mejor posible en todos los ámbitos de la vida cotidiana.

Un ejemplo claro de lo bien que se puede aplicar la tecnología, es haciendo uso de ella en un restaurante. Ver cómo los meseros meten órdenes, muchas veces suele ser algo enredado al momento de atenderlas y es ahí donde surge la necesidad de explorar nuevas herramientas y enfoques que no sólo resuelvan los desafíos presentes, sino que también transforman la experiencia en restaurantes en algo más eficiente y satisfactorio.

Entre las posibles soluciones, se destaca la implementación de hilos, bases de datos y otras herramientas tecnológicas, que, en conjunto, prometen revolucionar la forma en que los restaurantes gestionan sus operaciones diarias. La idea es trascender las limitaciones de los métodos tradicionales y abrazar la innovación digital para crear un entorno donde la toma de órdenes se convierta en un proceso fluido y sin complicaciones.

Justificación

El crear este programa podrá ser de gran ayuda para restaurantes pequeños que quieran modernizar su negocio y aumentar la eficiencia, esto permitirá traer más clientes y por lo mismo se verá un incremento en sus ganancias.

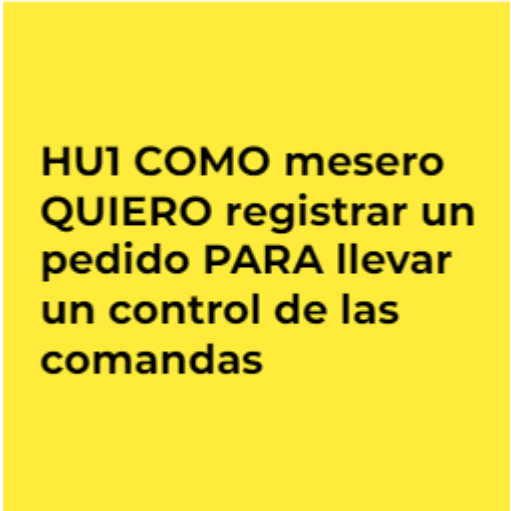
El programa se basa principalmente en tener ordenando los pedidos para que meseros y cocineros puedan realizar sus tareas de una mejor manera. El programa permitirá al mesero realizar comandas exactas de una manera más rápida y con poca probabilidad de error, en segunda le permitirá tener un mejor control de lo solicitado por el cliente al tener bien anotado todo y en tercera porque le permitirá entregarle al cocinero una comanda que el pueda entender fácilmente y no existan dudas en alguna preparación.

Este programa también ayudará a los cocineros a ordenar los pedidos y a darles un orden de preparación peculiar. En lugar de ir preparando pedido por pedido hasta terminar, se implementará un método el cual le permite ir preparando por partes cada pedido hasta terminar. Esto permitirá mantener a los clientes de un mejor humor y evitando problemas por la alta espera de sus alimentos, ya que estos van a ir llegando por partes a todos.

Requisitos del sistema

Historias de usuario:

HU1: Esta historia de usuario se enfoca en la necesidad del mesero de registrar pedidos para mantener un control eficiente de las comandas. Permitirá al personal tomar y gestionar órdenes de manera organizada, agilizando el servicio y garantizando la precisión en la entrega de los platillos solicitados por los clientes. Este registro de pedidos actuará como un sistema centralizado para supervisar y administrar eficazmente las solicitudes de comida dentro del restaurante, mejorando la eficiencia operativa y la satisfacción del cliente



**HU1 COMO mesero
QUIERO registrar un
pedido PARA llevar
un control de las
comandas**

HU2: La Historia de Usuario 2 se centra en la necesidad del mesero de supervisar el proceso de los pedidos con el fin de mantener un control efectivo sobre el tiempo de cada solicitud. Este requerimiento permitirá al personal monitorear el avance de los pedidos desde su registro hasta su entrega, garantizando tiempos de preparación óptimos. Al visualizar el progreso de los pedidos, el restaurante puede optimizar los tiempos de entrega, identificar posibles cuellos de botella en la cocina y mejorar la eficiencia operativa para proporcionar un servicio más rápido y de calidad a los clientes.

**HU2 COMO mesero
QUIERO ver el
proceso de los
pedidos PARA llevar
el control de tiempo
de cada pedido**

HU3: La Historia de Usuario 3 aborda la necesidad del cocinero de realizar modificaciones en un pedido para indicar cuando un pedido a sido preparado, este podrá ser true(en caso de que ya se haya preparado) o false(en caso de que aún no sea preparado).

**HU3 COMO cocinero
QUIERO poder
editar el estado del
pedido PARA indicar
cuando un pedido
ya ha sido
terminado**

HU4: La Historia de Usuario 4 se enfoca en la necesidad del cocinero de optimizar el flujo de pedidos en la cocina. La solicitud es que las comandas se presentan secuencialmente en función del orden de llegada. Esta funcionalidad busca evitar la acumulación excesiva de órdenes que podrían sobrecargar la cocina, al mismo

tiempo que garantiza que los pedidos se atienden en el orden en que se registraron. La implementación de este mecanismo de gestión de pedidos optimizará el proceso de preparación, asegurando la eficiencia y la prontitud en el servicio.

**HU4 COMO cocinero
QUIERO que las
comandas se
ordenando
secuencialmente,
PARA llevar un
orden de
preparación**

HU5: La Historia de Usuario 5 se centra en la necesidad del cocinero de tener la capacidad de eliminar o cancelar comandas del sistema en caso de que surjan problemas durante la preparación de un platillo. Esta funcionalidad brinda la flexibilidad de gestionar los pedidos registrados, permitiendo al personal de cocina tomar decisiones basadas en la disponibilidad de ingredientes, capacidad de preparación o cualquier otro inconveniente que pueda surgir. La capacidad de cancelar comandas garantiza un proceso de gestión de pedidos más ágil y efectivo, evitando problemas potenciales al evitar la preparación de platillos que no pueden ser atendidos o se encuentran fuera de stock.

**HU5 COMO cocinero
QUIERO que las
comandas se
puedan eliminar
PARA evitar
problemas en caso
de no poder
preparar algún
platillo**

HU6: Esta historia de usuario hace referencia que es necesario llevar un registro de los pedidos ya terminados, para esto utilizaremos un archivo txt en el cual imprimimos la información de los pedidos en el orden que se hayan terminado, todos en un mismo archivo para un fácil acceso.

**HU6 COMO dueño
QUIERO que las
comandas
terminadas se
guarden en un
archivo PARA llevar
un control de los
pedidos ya vendidos**

Organización de tareas

Medina Licon Gerardo de Jesús.

- Conexión a base de datos.
- Implementación de función “Crear”.
- Implementación de función “Modificar”.
- Implementación de función “Eliminar”.
- Introducción.
- Captura de commits y gráficas.

Rodríguez canales Yasmin.

- Implementación de función “mostrar_datos”.
- Integración de opción salir.
- Implementación de función “actualizar_completado”.
- implementación de función “mostrar_comandas”.
- Organización de tareas
- Descripción del patrón de concurrencia
- Uso de hilos

López Delgado Luis Antonio.

- Inclusión de funciones para el tiempo de cada comanda generada.
- Ejecución de pruebas.
- justificación
- Pruebas de software
- Insights

Xithe Pintado Daniel.

- Implementación de Round-Robin para la preparación de pedidos.
- Registro de pedido en archivo externo.
- Validaciones de registros.
- Requisitos del sistema
- Casos de prueba de funcionalidad
- Entidades de los campos
- Lock y sincronización

Casos de pruebas de funcionalidad

Introducción al Caso de Prueba - Registro de pedidos:

El caso de prueba "Registro de pedidos" se centra en validar la funcionalidad fundamental de registrar pedidos por parte de los meseros, asegurando que los datos ingresados sean enviados de manera íntegra y precisa sin experimentar modificaciones inesperadas. Este caso de prueba se ejecutará para garantizar que el proceso de registro de pedidos se realice sin inconvenientes, permitiendo una interacción sin errores entre el sistema y el mesero al generar las comandas.

La precondition establecida para este caso de prueba es que el mesero debe iniciar la generación de una comanda. La intención principal es confirmar que el sistema acepte y procese adecuadamente la información proporcionada por el mesero para crear y registrar pedidos con precisión.

Escenario de casos de uso

Escenario 1: Registro exitoso

- 1.-Abrir el programa
- 2.-Seleccionar la opción de registrar comanda
- 3.-Ingresar los datos (solo números)
- 4.-Después de registrar todos los datos se enviará automáticamente.

Resultado esperado:

Un mensaje de validación de que el registro fue exitoso

Datos de prueba:

barbacoa=1
consome=1
quesadilla=1
bebida=1

resultado:

tiempo_preparacion=10

Escenario 1: Registro no exitoso

- 1.-Abrir el programa
- 2.-Seleccionar la opción de registrar comanda
- 3.-Ingresar los datos incorrectos (como letras)
- 4.-Después de registrar todos los datos se enviará automáticamente.

Resultado esperado:

Se imprimirá un mensaje donde se indique que los datos son erróneos.

Datos de prueba:

barbacoa=1
consome=1

quesadilla=1

bebida="cuatro"

Resultado obtenido

Por favor, ingrese un número válido.

Id de caso de prueba	1
Nombre de caso de prueba	Registro de pedidos
Descripción	Se comprobará el correcto funcionamiento de realizar los registros de pedidos de los clientes y que los datos enviados se manden de manera correcta sin ninguna modificación no esperada.
Precondiciones	•El mesero tiene que generar una comanda
Relaciones de casos de uso	NA
Pasos y condiciones para su ejecución	•Abrir el programa •En el menú principal seleccionar la opción de registrar pedido •Registrar todo lo solicitado por parte del cliente •Presionar el botón de enviar pedido para su preparación
Resultado Esperado	Saldrá un mensaje de validación de que el registro a sido exitoso y nos indicará su tiempo total de preparación
Estado Caso de prueba	En construcción
Resultados obtenidos	-
Errores asociados	-
Responsable del diseño	Daniel Xithe Pintado
Responsable ejecución	-
Comentarios	-

Introducción al Caso de Prueba 2 - consulta de pedidos:

El segundo caso de prueba, denominado "Consultar pedidos ya registrados", hace mención al apartado para acceder a una lista de pedidos registrados y verificar su estado de preparación.. Se considera como precondition que existan registros previos de pedidos.

Escenario de casos de uso

Escenario 1: Tener algún registro en la base de datos (todas las comandas)

- 1.-Abrir el programa.
- 2.-Seleccionar la opción de consulta.
- 3.-Seleccionar la opción de mostrar todas las comandas.

Resultado esperado:

Mostrar los registros de la base de datos sin importar si el atributo completado es false o true

Escenario 2: Tener algún registro en la base de datos(no completadas)

- 1.-Abrir el programa.
- 2.-Seleccionar la opción de consulta.
- 3.-Seleccionar las comandas no completadas.

Resultado esperado:

Mostrar los registros que se tengan en la base de datos que tengan false en el campo completado.

Escenario 3: Tener algún registro en la base de datos(completadas)

- 1.-Abrir el programa.
- 2.-Seleccionar la opción de consulta.
- 3.-Seleccionar las comandas completadas.

Resultado esperado:

Mostrar los registros que se tengan en la base de datos que tengan true en el campo completado.

Escenario 4: No tener ningún registro de la base de datos

- 1.-Abrir el programa.
- 2.-Seleccionar la opción de consulta.
- 3.-Seleccionar alguna de las opciones de consulta (todas,no completadas o completadas).

Resultado esperado:

No mostrará nada y lo redireccionará directamente al menú

Id de caso de prueba	2
Nombre de caso de prueba	Consultar comandas registradas

Descripción	En cualquier momento tanto meseros como cocineros podrán entrar al apartado de pedidos donde tendrán una lista, en el orden en el que se irán preparando los pedidos.
Precondiciones	•Tener registros ya realizados de pedidos
Relaciones de casos de uso	NA
Pasos y condiciones para su ejecución	•Abrir el programa •Seleccionar la opción de pedidos
Resultado Esperado	En pantalla se muestren todos los pedidos que habían sido registrados anteriormente, en el orden que se prepararon y su tiempo estimado de preparación
Estado Caso de prueba	En construcción
Resultados obtenidos	-
Errores asociados	-
Responsable del diseño	Daniel Xithe Pintado
Responsable ejecución	-
Comentarios	-

Introducción al Caso de Prueba 3 - Orden de preparación de pedidos:

En el caso de prueba 3 se comprobará que el funcionamiento de nuestro patrón de concurrencia “round Robin” funcione correctamente al momento de preparar los platillos solicitados.

Escenario 1: Preparar pedidos

- 1.-Abrir el programa
- 2.-Tener al menos un registro de comanda que no haya sido preparada.
- 3.-Procesar los pedidos
- 4.-Registrar los pedidos ya realizados en un archivo txt

Resultado esperado:

Saldrá un mensaje en consola que el pedido ha sido registrado.

Escenario 2: No se tengan pedidos pendientes

- 1.-Abrir el programa
- 2.-No tener ningún registro de comanda pendiente
- 3.-Procesar los pedidos

Resultado esperado:

En este caso se redireccionará de nuevo al menú principal.

Id de caso de prueba	3
Nombre de caso de prueba	Orden de preparación de pedidos
Descripción	Comprobar que el orden de la preparación sea por partes (round robin) y no secuencialmente.
Precondiciones	•Tener comandas realizadas
Relaciones de casos de uso	NA
Pasos y condiciones para su ejecución	<ul style="list-style-type: none"> •Abrir el programa • Entrar a la sección de preparar pedidos
Resultado Esperado	Comprobar que los pedidos se hayan organizado conforme se hayan registrado, pero que sus tiempos de preparación se indique cuál es su tiempo total y cuanto va a durar la parte que se va a preparar.
Estado Caso de prueba	En construcción
Resultados obtenidos	-
Errores asociados	-
Responsable del diseño	Daniel Xithe Pintado
Responsable ejecución	-
Comentarios	-

Introducción al Caso de Prueba 4 - Registro de pedidos en archivo de texto:

En el caso de prueba 4 se comprobará que el registro de los datos sea exitoso y se mantengan fidedignos.

Id de caso de prueba	4
Nombre de caso de prueba	Registro de pedidos en el archivo txt
Descripción	Comprobar que cuando una comanda pasa del estado “completado”=false a true, se registre en nuestro archivo llamado registros_comandas.txt
Precondiciones	•Tener comandas realizadas
Relaciones de casos de uso	NA
Pasos y condiciones para su ejecución	•Abrir el programa • Entrar a la sección de pedidos •Esperar la preparación de pedidos
Resultado Esperado	Mensaje de confirmación Comanda x- Información guardada en registros_comandas.txt
Estado Caso de prueba	En construcción
Resultados obtenidos	-
Errores asociados	-
Responsable del diseño	Daniel Xitthe Pintado
Responsable ejecución	-
Comentarios	-

Pruebas de Software

Alcance del Proyecto:

- **Menú de Alimentos:**

Presentación correcta de la lista de alimentos en el menú.

Opciones claras para que el cliente seleccione los alimentos deseados.

- **Proceso de Pedido:**

Correcta asignación de tiempo de preparación para cada alimento seleccionado.

Verificación de que el proceso de planificación atiende a los pedidos en orden de llegada.

- **Manejo de Hilos:**

Verificación de la concurrencia mediante el uso de hilos para gestionar múltiples pedidos simultáneamente.

Garantía de que los hilos no generan conflictos o condiciones de carrera.

- **Base de Datos MongoDB:**

Conexión exitosa a la base de datos MongoDB.

Correcto almacenamiento de nuevos pedidos en la base de datos.

Recuperación precisa de pedidos almacenados en la base de datos.

- **Operaciones CRUD en la Base de Datos:**

Verificación de la capacidad para agregar nuevos pedidos.

Edición exitosa de pedidos existentes.

Eliminación adecuada de pedidos cancelados.

- **Interfaz de Usuario en Consola:**

Presentación clara y amigable del menú y opciones de pedido.

Actualización en tiempo real del estado del pedido para el cliente.

- **Manejo de Excepciones:**

Manejo adecuado de situaciones excepcionales, como pedidos inválidos o problemas de conexión con la base de datos.

Asegurarse de que el programa no se bloquee ante situaciones inesperadas.

- **Eficiencia y Tiempos de Respuesta:**

Verificación de que los tiempos de preparación se cumplen según lo establecido.

Evaluación de la eficiencia del proceso de planificación.

- **Seguridad:**

Protección contra posibles vulnerabilidades, especialmente en la interacción con la base de datos.

Características a Ser Probadas:

- **Menú:**

Despliegue correcto del menú con opciones legibles.

Funcionalidad de selección de alimentos.

- **Proceso de Pedido:**

Asignación adecuada de tiempos de preparación.

Atención a los pedidos en orden de llegada.

- **Hilos:**

Correcta ejecución concurrente de múltiples pedidos.

- **Base de Datos MongoDB:**

Conexión y almacenamiento exitosos.

- **Operaciones CRUD:**

Agregar, editar y eliminar pedidos con éxito.

- **Interfaz de Usuario:**

Presentación clara y actualización en tiempo real del estado del pedido.

- **Manejo de Excepciones:**

Respuestas adecuadas a situaciones inesperadas.

- **Eficiencia y Tiempos de Respuesta:**

Cumplimiento de los tiempos de preparación.

Evaluación de la eficiencia del proceso de planificación.

- **Seguridad:**

Garantía de protección contra vulnerabilidades

Requisitos Funcionales:

Historia de Usuario 1: Registrar Pedido

- RF1: El sistema debe permitir al mesero registrar un pedido, incluyendo los alimentos seleccionados y detalles relevantes.

Historia de Usuario 2: Seguir Proceso de Pedidos

- RF2: El sistema debe mostrar el estado actual de cada pedido, indicando si está en proceso, completado o detenido.

Historia de Usuario 3: Editar Pedido

- RF3: El mesero debe poder editar un pedido existente para corregir información o realizar cambios.

Historia de Usuario 4: Orden de Pedidos

- RF4: Los pedidos deben ser procesados en el orden en que son registrados para garantizar un control específico del tiempo y la secuencia.

Historia de Usuario 5: Detener (Eliminar) Pedido

- RF5: El sistema debe permitir a los cocineros detener (eliminar) un pedido en caso de contratiempos o falta de ingredientes.

Requisitos No Funcionales:

Historia de Usuario 1: Eficiencia en el Registro de Pedidos

- RNF1: El sistema debe registrar los pedidos de manera eficiente para no afectar la velocidad del servicio.

Historia de Usuario 2: Información Clara del Proceso de Pedidos

- RNF2: La interfaz debe proporcionar información clara sobre el estado actual de cada pedido para facilitar el seguimiento por parte del mesero.

Historia de Usuario 3: Edición Rápida de Pedidos

- RNF 3: El sistema debe permitir la edición rápida y eficiente de pedidos para minimizar el tiempo dedicado a correcciones.

Historia de Usuario 4: Gestión Eficiente de la Orden de Pedidos

- RNF4: El sistema debe procesar los pedidos de manera eficiente y en el orden correcto para cumplir con el control específico de tiempo.

Historia de Usuario 5: Disponibilidad de Detener (Eliminar) Pedidos

- RNF5: La funcionalidad de detener (eliminar) un pedido debe ser rápida y disponible para evitar contratiempos en la cocina.

Análisis de riesgos y Contingencias:

En el contexto de nuestro proyecto de programación concurrente para la gestión de pedidos en un restaurante, la identificación y mitigación de riesgos son elementos clave que no solo preservan la integridad del sistema, sino que también aseguran una experiencia eficiente y fiable para los usuarios. La anticipación de posibles contratiempos y la planificación de contingencias no sólo son prácticas preventivas, sino esenciales para garantizar que nuestro sistema opere sin contratiempos, cumpliendo con las expectativas de los usuarios y asegurando la continuidad del servicio.

Análisis de Riesgos:

- **Riesgo: Fallo en la Conexión a la Base de Datos**
Impacto: Pérdida de datos y falta de capacidad para gestionar pedidos.
Probabilidad: Moderada.
- **Riesgo: Conflictos en el Manejo de Hilos**
Impacto: Errores de sincronización, condiciones de carrera.
Probabilidad: Moderada.
- **Riesgo: Sobrecarga del Sistema**
Impacto: Ralentización del sistema, tiempos de respuesta más largos.
Probabilidad: Baja.
- **Riesgo: Errores en el Proceso de Planificación de Pedidos**
Impacto: Desorden en la preparación de alimentos.
Probabilidad: Moderada.
- **Riesgo: Problemas de Seguridad en la Interfaz de Usuario**
Impacto: Acceso no autorizado o manipulación de datos.
Probabilidad: Baja.

Contingencias:

- **Contingencia para Conflictos en el Manejo de Hilos:**
Plan: Implementar mecanismos de bloqueo y sincronización para evitar condiciones de carrera. Realizar pruebas exhaustivas de concurrencia.
- **Contingencia para Sobrecarga del Sistema:**
Plan: Monitorear el rendimiento del sistema y, en caso de sobrecarga, implementar optimizaciones de código, considerar escalabilidad o limitar temporalmente la aceptación de nuevos pedidos.
- **Contingencia para Errores en el Proceso de Planificación:**
Plan: Implementar un mecanismo de verificación de pedidos para garantizar que se procesen en el orden correcto. Realizar pruebas de estrés para evaluar el rendimiento del proceso de planificación.
- **Contingencia para Problemas de Seguridad en la Interfaz de Usuario:**
Plan: Implementar medidas de seguridad adicionales, como autenticación robusta y cifrado de datos. Realizar auditorías de seguridad periódicas.

Matriz de trazabilidad

Riesgo	Probabilidad	Impacto	Estrategias de mitigación	Plan de contingencia
Conflicto en el manejo de Hilos	Moderada	Alto	Implementar mecanismos de bloqueo y sincronización en los hilos.	Realizar pruebas de concurrencia y ajustar el código según sea necesario.
Sobrecarga del sistema	Baja	Moderado	Monitorear el rendimiento del sistema y realizar optimizaciones del código.	Implementar optimizaciones del código, considerar escalabilidad o limitar temporalmente la aceptación de nuevos pedidos.
Errores en el proceso de planificación	Moderada	Alto	Implementar verificación de pedidos para garantizar el orden correcto.	Evaluar el rendimiento del proceso de planificación.

La elaboración de un Plan de Pruebas se revela como un componente esencial en el desarrollo del proyecto de programación concurrente para la gestión de pedidos en un restaurante. Este plan proporciona una estructura organizada para validar la funcionalidad y eficiencia del sistema, anticipando posibles desafíos y asegurando la robustez del software implementado.

En nuestro contexto, el Plan de Pruebas adquiere una relevancia significativa al abordar aspectos clave como la concurrencia en el manejo de pedidos, la interacción con una base de datos no relacional, y la presentación en consola de información en tiempo real para los usuarios. Además, la identificación y mitigación proactiva de riesgos garantiza una respuesta eficaz ante situaciones imprevistas, manteniendo la integridad del sistema y la satisfacción del usuario.

Pruebas Funcionales:

Funcionalidad: Agregar una comanda

· Prueba funcional 1 - Agregar comanda válida:

Descripción: Se introduce una comanda con cantidades válidas y se verifica su inserción en la base de datos.

Herramienta: Consola de línea de comandos o interfaz de usuario.

Verificación: Consulta directa a la base de datos MongoDB para confirmar la inserción exitosa de la comanda.

Código:

```
def test_agregar_comanda_valida():

    # Simular la entrada de valores válidos para una comanda

    num_comanda = 10

    barbacoa = 2

    quesadilla = 3

    bebida = 1

    consome = 1

    # Crear un hilo para agregar la comanda a la base de datos

hilo = threading.Thread(target=escribir_base_de_datos, args=(threading.current_thread().name,
num_comanda, barbacoa, quesadilla, bebida, consome), name=f'HiloAgregarComanda')

    hilo.start()

    hilo.join()

    # Verificar si la comanda se ha agregado correctamente consultando la base de datos

client = MongoClient("mongodb+srv://danielxp:maspormasDF1@cluster0.glu8e4r.mongodb.net/?retryWrites=true&w=majority")

db = client["danielxp88"]

comandas_collection = db["comandas"]
```

```

comanda_agregada = comandas_collection.find_one({"num_comanda": num_comanda})

client.close()

assert comanda_agregada is not None, "La comanda no se ha agregado correctamente a la base de datos"

print("La comanda se ha agregado correctamente")

```

Salida en consola:

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\lpzlu\Desktop\Recuperados\MenuTech> & 'C:\Users\lpzlu\AppData\Local\Programs\Python\Python20.0\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '57474' '--' 'C:\Users\lpzlu\Desktop\Recuperados\MenuTech\MenuTech.py'
Menú de opciones:
1. Agregar Comanda
2. Borrar Comanda
3. Consultar Comandas
4. Actualizar Estado de Comanda
5. Preparar pedidos
6. Salir
Hilo MainThread: Comanda insertada con ID: 656e28353a993ea620867af2
El tiempo estimado de preparación para la Comanda 10 es de 20 segundos.
La comanda se ha agregado correctamente

```

En esta prueba se insertó una comanda por medio de una función de testeo (Mediante valores fijos desde código) para comprobar el funcionamiento del método de inserción y guardado en la base de datos.

Prueba funcional 2 - Verificación del Algoritmo de planificación Round-Robin:

Descripción: Se evalúa el comportamiento del algoritmo de planificación Round-Robin en el procesamiento de comandas con diferentes tiempos de preparación, confirmando su correcto funcionamiento.

Herramienta: Interfaz de usuario o consultas directas a la base de datos para observar y confirmar el estado y tiempos de las comandas.

Verificación: La correcta ejecución del algoritmo Round-Robin se confirma al verificar que las comandas se completen siguiendo el orden de llegada y dentro de

los tiempos de preparación definidos. Se comprueba a través de consultas a la base de datos o mediante la interfaz de usuario que las comandas procesadas coincidan con el orden esperado y los tiempos estimados.

Código:

```
def simular_round_robin():

    while True:

        # Realizar la conexión a la base de datos

        client = MongoClient("mongodb+srv://danielxp:maspormasDF1@cluster0.glu8e4r.mongodb.net/?retryWrites=true&w=majority")

        db = client["danielxp88"]

        comandas_collection = db["comandas"]

        # Obtener todas las comandas pendientes

        comandas_pendientes = comandas_collection.find({"$or": [{"completado": False}, {"completado": {"$exists": False}}]})

        # Iterar sobre cada comanda pendiente

        for comanda in comandas_pendientes:

            num_comanda = comanda["num_comanda"]

            completado = comanda.get("completado", False)

            tiempo_pedido = comanda.get("tiempo_pedido", 0)

            print(f"Comanda {num_comanda} - Completado: {completado}, Tiempo restante: {tiempo_pedido} segundos")

            # Si la comanda no está completada y el tiempo restante es mayor a cero, actualizar el tiempo de preparación

            if not completado and tiempo_pedido > 0:

                # Simular intervalo de ejecución (quantum de 2 segundos)
```

```

time.sleep(2)

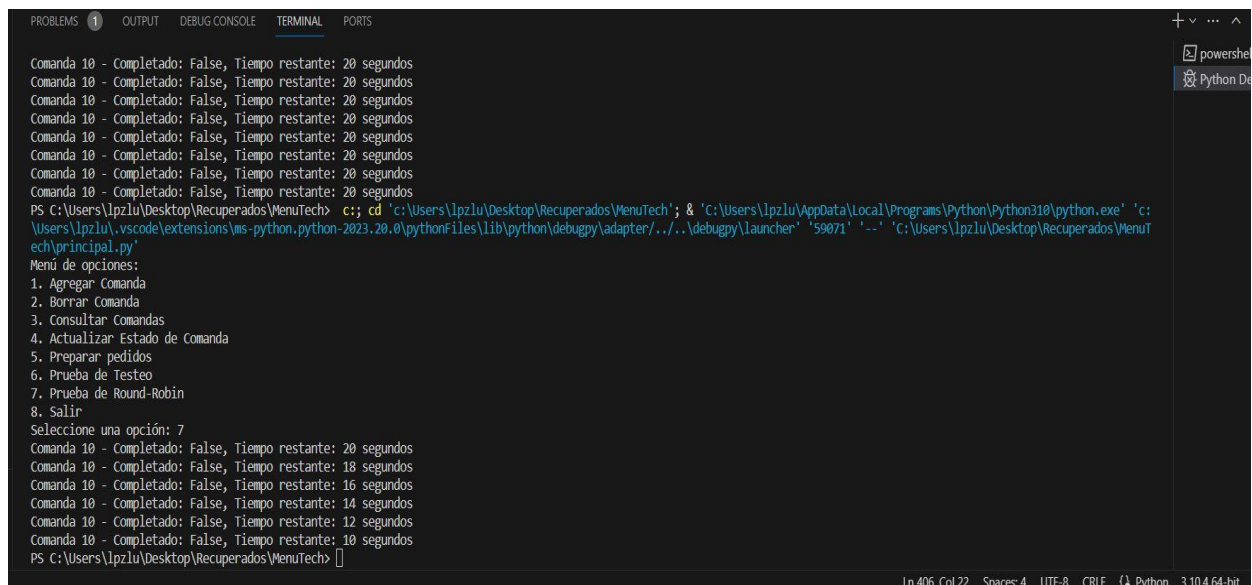
# Restar tiempo_pedido en la base de datos para la comanda actual
comandas_collection.update_one({"_id": comanda["_id"]}, {"$inc": {"tiempo_pedido": -2}})

# Cerrar la conexión
client.close()

# Simular el intervalo de tiempo entre iteraciones
time.sleep(1)

```

Salida en consola:



```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
PS C:\Users\lpzlu\Desktop\Recuperados\MenuTech> c; cd 'c:\Users\lpzlu\Desktop\Recuperados\MenuTech'; & 'c:\Users\lpzlu\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\lpzlu\vscode\extensions\ms-python.python-2023.20.0\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '59871' '---' 'C:\Users\lpzlu\Desktop\Recuperados\MenuTech\principal.py'
Menú de opciones:
1. Agregar Comanda
2. Borrar Comanda
3. Consultar Comandas
4. Actualizar Estado de Comanda
5. Preparar pedidos
6. Prueba de Testeo
7. Prueba de Round-Robin
8. Salir
Seleccione una opción: 7
Comanda 10 - Completado: False, Tiempo restante: 20 segundos
Comanda 10 - Completado: False, Tiempo restante: 18 segundos
Comanda 10 - Completado: False, Tiempo restante: 16 segundos
Comanda 10 - Completado: False, Tiempo restante: 14 segundos
Comanda 10 - Completado: False, Tiempo restante: 12 segundos
Comanda 10 - Completado: False, Tiempo restante: 10 segundos
PS C:\Users\lpzlu\Desktop\Recuperados\MenuTech>
Ln 406, Col 22 Spaces: 4 UTF-8 CRLF Python 3.10.4 64-bit

```

En esta prueba realiza la simulación del progreso de la comanda generada mediante la lógica del algoritmo de planificación Round-Robin.

Prueba funcional 3 - Mostrar comandas no completadas:

Descripción: Se consulta y muestra todas las comandas que no han sido completadas.

Herramienta: Interfaz de usuario o línea de comandos.

Verificación: Verificación visual en la consola o interfaz de usuario para confirmar la correcta visualización de las comandas no completadas.

Código:

```
def test_mostrar_comandas_no_completadas():

    # Simular mostrar comandas no completadas desde la base de datos

    print("-----")

    print("Test Mostrar Comandas no completadas:")

    # Llamar a la función que muestra comandas no completadas

    mostrar_datos_filtradas("3") # "3" corresponde a la opción de comandas no completadas

    # Verificar si se muestran correctamente las comandas no completadas

    client = MongoClient("mongodb+srv://danielxp:maspormasDF1@cluster0.glu8e4r.mongodb.net/?retryWrites=true&w=majority")

    db = client["danielxp88"]

    comandas_collection = db["comandas"]

    # Verificar si se muestran solo comandas no completadas

    comandas_no_completadas = comandas_collection.find({"$or": [{"completado": False}, {"completado": {"$exists": False}}]})

    for comanda in comandas_no_completadas:

        assert comanda.get("completado") in (False, None), "Se muestra una comanda completada en la lista de no completadas"

    print("Prueba de mostrar comandas no completadas finalizada")

    client.close()
```

Salida en consola:

```
Test Mostrar Comandas no completadas:
-----
Datos en la base de datos:
ID: 656e28353a993ea620867af2
Número de Comanda: 10
Barbacoa: 2
Quesadilla: 3
Bebida: 1
Consomé: 1
Tiempo estimado de preparación: 10 segundos
Completado: False

ID: 656e336ccf1ad0bb90a4a1aa
Número de Comanda: 41
Barbacoa: 2
Quesadilla: 3
Bebida: 1
Consomé: 1
Tiempo estimado de preparación: 20 segundos
Completado: False

Prueba de mostrar comandas no completadas finalizada
Menú de opciones:
1. Agregar Comanda
2. Borrar Comanda
3. Consultar Comandas
4. Actualizar Estado de Comanda
5. Preparar pedidos
6. Prueba de Testeo
7. Prueba de Comandas no completadas
8. Prueba de Round-Robin
9. Salir
Seleccione una opción: [ ]
```

Prueba funcional: Cálculo de tiempo de preparación según cantidad de un plato

Herramientas:

- Base de datos: MongoDB u otro sistema de almacenamiento.
- API de la base de datos: Para la consulta de datos.
- Código de cálculo del tiempo: Funciones que calculan el tiempo de preparación.

Descripción:

Esta prueba válida que el sistema calcule correctamente el tiempo total de preparación basado en la cantidad de un plato específico y su tiempo individual de preparación.

Esta prueba confirma que el sistema realiza correctamente el cálculo del tiempo total de preparación multiplicando el tiempo individual de un plato por la cantidad solicitada, asegurando así que se obtenga el tiempo esperado.

Código:

```
def test_calculo_tiempo_preparacion():

    # Definir valores de prueba

    cantidad_barbacoa = 4 # 4 barbacoas
```

```
    tiempo_individual_barbacoa = tiempos_comida["barbacoa"] # Obtener el tiempo individual de
preparación de barbacoa
```

```
# Calcular el tiempo total esperado
```

```
tiempo_esperado = cantidad_barbacoa * tiempo_individual_barbacoa
```

```
# Calcular el tiempo total según la función obtener_tiempo_total
```

```
orden = {
```

```
    "barbacoa": cantidad_barbacoa,
```

```
    "quesadilla": 0,
```

```
    "bebida": 0,
```

```
    "consome": 0
```

```
}
```

```
tiempo_calculado = obtener_tiempo_total(orden)
```

```
# Verificar si el tiempo calculado coincide con el tiempo esperado
```

```
assert tiempo_calculado == tiempo_esperado, f"Error en el cálculo del tiempo. Se esperaban
{tiempo_esperado} segundos, se obtuvieron {tiempo_calculado} segundos."
```

```
print(f"El cálculo del tiempo de preparación para {cantidad_barbacoa} barbacoas fue exitoso.")
```

Salida en consola:

```
4. Actualizar Estado de Comanda
5. Preparar pedidos
6. Prueba de Testeo
7. Prueba de Comandas no completadas
8. Prueba de tiempos de comandas
9. Prueba de Round-Robin
10. Salir
Seleccione una opción: 8
El cálculo del tiempo de preparación para 4 barbacoas fue exitoso.
Menú de opciones:
1. Agregar Comanda
2. Borrar Comanda
3. Consultar Comandas
4. Actualizar Estado de Comanda
5. Preparar pedidos
6. Prueba de Testeo
7. Prueba de Comandas no completadas
8. Prueba de tiempos de comandas
9. Prueba de Round-Robin
10. Salir
Seleccione una opción: 9
Comanda 41 - Completado: False, Tiempo restante: 20 segundos
Comanda 41 - Completado: False, Tiempo restante: 18 segundos
PS C:\Users\lpzlu\Desktop\Recuperados\MenuTech> █
```

Explicación Adicional:

Las pruebas funcionales garantizan que las diversas funcionalidades del sistema funcionen correctamente y proporcione los resultados esperados para los usuarios finales.

Herramientas

Para llevar a cabo la ejecución exitosa del proyecto, resulta importante definir las herramientas que se emplearán en la realización del proyecto. En este caso, se ha determinado que la selección de herramientas juega un papel importante en el logro de los objetivos propuestos. En áreas de garantizar eficiencia, coherencia y cohesión en todas las fases del proyecto, se ha optado por utilizar herramientas que nos permitan realizar un proyecto de mejor manera.

El proyecto será desarrollado en el entorno de **Visual Studio** con una versión de: **1.84.2**, haciendo uso del lenguaje de programación **Python 3.11.6**. Buscando e implementando mejores prácticas de colaboración, se ha elegido **Git Hub 3.10** como la plataforma para trabajar de manera colaborativa, facilitando así la gestión eficiente del código fuente y la amplia colaboración efectiva entre los miembros del equipo.

En términos de implementación, se pretende utilizar diversas **bibliotecas dentro de Visual Studio** para potenciar las capacidades del proyecto.

-**threading** Esta nos permite funcionalidades para trabajar con hilos.

-**time** import sleep permite el manejo de tiempo con el método sleep.

-**random** permite trabajar con valores aleatorios.

queue este se implementa en la utilización de colas.

asyncio permite la programación asíncrona.

pymongo este nos permite la conexión de nuestra app hacia la base de datos.

Así mismo, en el ámbito de la persistencia de datos, se ha determinado emplear una base de datos no relacional, específicamente **en MongoDB** en su versión 6.3.0. Esta elección se fundamenta en las características flexibles y escalables que ofrece esta tecnología, lo cual se alinea con las necesidades particulares del proyecto.

Entidades de los campos

```
"comanda":{
    "_id":objectId(""),
    "num_comanda": ,
    "barbacoa": ,
    "quesadilla": ,
    "bebida": ,
    "consome": ,
    "completado": ,
    "tiempo_pedido":
}
```

Patrón de concurrencia

Round-robin

1. Uso de Hilos (Threads): La función `mostrar_progreso_comanda` se ejecuta en un hilo separado (`hilo_progreso`) para permitir la ejecución concurrente de múltiples comandas al mismo tiempo. Se utiliza el módulo `threading` de Python para crear y gestionar estos hilos.
2. Bucle de Actualización del Progreso: La función utiliza un bucle `while True` para monitorear continuamente el progreso de la comanda. Este bucle se ejecuta hasta que la comanda se completa o hasta que el tiempo restante sea menor o igual a cero.
3. Intervalo de Espera (`time.sleep`): Después de cada iteración del bucle, se simula un intervalo de ejecución (quantum) mediante `time.sleep(2)`. Este intervalo representa un pequeño paso de tiempo durante el cual se actualiza el estado de la comanda.
4. Mientras el tiempo de preparación sea mayor a 0 se disminuye en 2 segundos (simulando el paso de tiempo). Esto se logra con la siguiente línea de código:

```
comandas_collection.update_one({"num_comanda": num_comanda}, {"$inc": {"tiempo_pedido": -2}})
```
5. Finalización del Hilo: Una vez que la comanda se ha completado o el tiempo restante es menor o igual a cero, se sale del bucle y se realiza la limpieza necesaria. Antes de salir, se actualiza el estado de "completado" en la base de datos y se imprime información relevante.
6. Sincronización y Locks: Se utiliza un mecanismo de bloqueo (`archivo_lock`) para garantizar que la escritura en el archivo de registros (`registros_comandas.txt`) sea segura y evite posibles conflictos entre hilos.

Permite manejar simultáneamente múltiples comandas en progreso, actualizando sus estados y mostrando el progreso en tiempo real de manera concurrente mediante el uso de hilos. Cada hilo ejecuta su tarea de forma equitativa, siguiendo el enfoque Round-robin para garantizar una distribución justa del tiempo de CPU entre los diferentes hilos.

```
150 def mostrar_progreso_comanda(num_comanda):
151     # Realizar la conexión a la base de datos
152     client = MongoClient("mongodb+srv://danielxp:maspormasDF1@cluster0.glu8e4r.mongodb.net/?retryWrites=true&w=majority")
153     db = client["danielxp88"]
154     comandas_collection = db["comandas"]
155
156     # Mostrar el progreso de la comanda en tiempo real
157     while True:
158         comanda = comandas_collection.find_one({"num_comanda": num_comanda})
159         if comanda:
160             completado = comanda.get("completado", False)
161             tiempo_pedido = comanda.get("tiempo_pedido", 0)
162             detalles = {
163                 "barbacoa": comanda.get("barbacoa", 0),
164                 "quesadilla": comanda.get("quesadilla", 0),
165                 "bebida": comanda.get("bebida", 0),
166                 "consome": comanda.get("consome", 0),
167             }
168
169             print(f"Comanda {num_comanda} - Completado: {completado}, Tiempo restante: {tiempo_pedido} segundos")
170
171             if completado or tiempo_pedido <= 0:
172                 break # Salir del bucle si la comanda ha sido completada o el tiempo restante es menor o igual a cero
173
174             # Simular intervalo de ejecución (quantum de 2 segundos)
175             time.sleep(2)
176
177             # Restar tiempo_pedido en la base de datos
178             comandas_collection.update_one({"num_comanda": num_comanda}, {"$inc": {"tiempo_pedido": -2}})
179
180             # Actualizar "completado" en la base de datos cuando la comanda ha terminado
181             comandas_collection.update_one({"num_comanda": num_comanda}, {"$set": {"completado": True}})
182
183             # Escribir información en archivo txt
184             escribir_archivo_txt(num_comanda, True, tiempo_pedido, detalles)
185
186             # Cerrar la conexión
187             client.close()
```

Uso de hilos

El programa en Python utiliza el paradigma de programación concurrente mediante el uso de hilos (threads). Este programa interactúa con una base de datos MongoDB para gestionar comandas en un sistema de pedidos de alimentos. La lógica del programa está estructurada en funciones que se ejecutan en hilos separados, permitiendo realizar operaciones simultáneas y mejorar la eficiencia del programa.

Función escribir_base_de_datos

- Se define la función “escribir_base_de_datos”, que toma varios parámetros, incluyendo cantidades de diferentes tipos de alimentos y bebidas.
- Se calcula el “tiempo_pedido” total basado en las cantidades y los tiempos predefinidos de preparación (“tiempos_comida”).
- Se crea un hilo (`threading.Thread`) llamado `hilo` que ejecutará la función “escribir_base_de_datos” con los parámetros dados.
- El hilo se inicia (`start()`) y se espera a que termine (`join()`), lo que significa que el programa principal esperará a que el hilo complete la ejecución antes de continuar.

```
17 def escribir_base_de_datos(hilo_id, num_comanda, barbacoa, quesadilla, bebida, consome):
18     tiempo_pedido = obtener_tiempo_total({
19         "barbacoa": barbacoa,
20         "quesadilla": quesadilla,
21         "bebida": bebida,
22         "consome": consome
23     })
24
25     client = MongoClient("mongodb+srv://danielxp:maspormasDF1@cluster0.glu8e4r.mongodb.net/?retryWrites=true&w=majority")
26     db = client["danielxp88"]
27     comandas_collection = db["comandas"]
28
29     comanda = {
30         "num_comanda": num_comanda,
31         "barbacoa": barbacoa,
32         "quesadilla": quesadilla,
33         "bebida": bebida,
34         "consome": consome,
35         "tiempo_pedido": tiempo_pedido
36     }
37     result = comandas_collection.insert_one(comanda)
38     print(f"Hilo {hilo_id}: Comanda insertada con ID: {result.inserted_id}")
39     print(f"El tiempo estimado de preparación para la Comanda {num_comanda} es de {tiempo_pedido} segundos.")
40     client.close()
```

Función menu

- Se define la función `menu`, que proporciona un menú de opciones al usuario.
- Dependiendo de la opción seleccionada, se crea un hilo correspondiente para ejecutar la operación asociada (agregar, borrar, mostrar).
- El hilo se inicia y se espera a que termine.

```
95 def menu():
96     while True:
97         print("Menú de opciones:")
98         print("1. Agregar Comanda")
99         print("2. Borrar Comanda")
100        print("3. Consultar Comandas")
101        print("4. Salir")
102
103        opcion = input("Seleccione una opción: ")
104
105        if opcion == "1":
106            cant_barbacoa = obtener_opcion("barbacoa")
107            cant_quesadilla = obtener_opcion("quesadilla")
108            cant_bebida = obtener_opcion("bebida")
109            cant_consomme = obtener_opcion("consomme")
110
111            hilo = threading.Thread(target=escribir_base_de_datos, args=(
112                threading.current_thread().name,
113                obtener_numero_comanda(),
114                cant_barbacoa,
115                cant_quesadilla,
116                cant_bebida,
117                cant_consomme
118            ), name='HiloAgregarComanda')
119
120            hilo.start()
121            hilo.join()
122
123        elif opcion == "2":
124            hilo = threading.Thread(target=borrar_datos, name='HiloBorrarComanda')
125            hilo.start()
```

```
126        hilo.join()
127
128        elif opcion == "3":
129            hilo = threading.Thread(target=mostrar_datos, name='HiloConsultarComandas')
130            hilo.start()
131            hilo.join()
132
133        elif opcion == "4":
134            print("Saliendo del programa. ¡Hasta luego!")
135            break
136
137        else:
138            print("Opción no válida. Por favor, seleccione una opción válida.")
139
```

Lock y sincronización

Se utiliza un objeto `"archivo_lock"` como un mecanismo de bloqueo para garantizar la escritura segura en el archivo `"registros_comandas.txt"`.



























1. Necesidad de Sincronización: Cuando varios hilos (threads) ejecutan concurrentemente, puede haber problemas si varios intentan escribir en el mismo archivo al mismo tiempo. Estos problemas pueden incluir la posibilidad de escritura intercalada, pérdida de datos o conflictos que generan un comportamiento inesperado.
2. Uso de Lock (``archivo_lock``): El ``archivo_lock`` es un objeto de bloqueo (``Lock``) proporcionado por el módulo ``threading`` de Python. Se utiliza para sincronizar el acceso a recursos compartidos, en este caso, el archivo de registros.
3. Entrando y Saliendo del Bloque Crítico: La declaración `"with archivo_lock:"` crea un bloque crítico del código. Al entrar en este bloque, el hilo adquiere el bloqueo (``Lock``), asegurándose de que solo un hilo a la vez pueda ejecutar el código dentro del bloque. Cuando el bloque se completa (ya sea de manera normal o debido a una excepción), el bloqueo se libera automáticamente, permitiendo que otros hilos entren en la sección crítica si es necesario.
4. Escritura Segura en el Archivo: Dentro del bloque crítico, se abre el archivo ``registros_comandas.txt`` en modo `"a"` (append), lo que permite agregar contenido al final del archivo. Todas las operaciones de escritura en el archivo se realizan dentro de este bloque crítico, asegurando que un hilo complete sus operaciones de escritura antes de que otro hilo pueda acceder al archivo.
5. Evitando Conflictos: El uso del ``archivo_lock`` evita posibles conflictos entre los hilos al garantizar que solo un hilo pueda escribir en el archivo a la vez. Esto asegura que la información de cada comanda se escriba de manera completa y sin intercalación con otras comandas escritas por otros hilos.

```
140 def escribir_archivo_txt(num_comanda, completado, tiempo_pedido, detalles):
141     # Nombre del archivo único para todos los registros
142     nombre_archivo = "registros_comandas.txt"
143
144     # Bloquear para garantizar escritura segura en el archivo
145     with archivo_lock:
146         with open(nombre_archivo, "a") as archivo: # Modo "a" para agregar contenido al archivo
147             archivo.write(f"Número de Comanda: {num_comanda}\n")
148             archivo.write(f"Completado: {completado}\n")
149             archivo.write(f"Tiempo estimado de preparación: {tiempo_pedido} segundos\n")
150             archivo.write("\nDetalles de la Comanda:\n")
151             for comida, cantidad in detalles.items():
152                 archivo.write(f"{comida.capitalize()}: {cantidad}\n")
153             archivo.write("\n" + "-"*40 + "\n") # Separador entre comandas
154
155     print(f"Comanda {num_comanda} - Información guardada en {nombre_archivo}")
```

Commits en Github

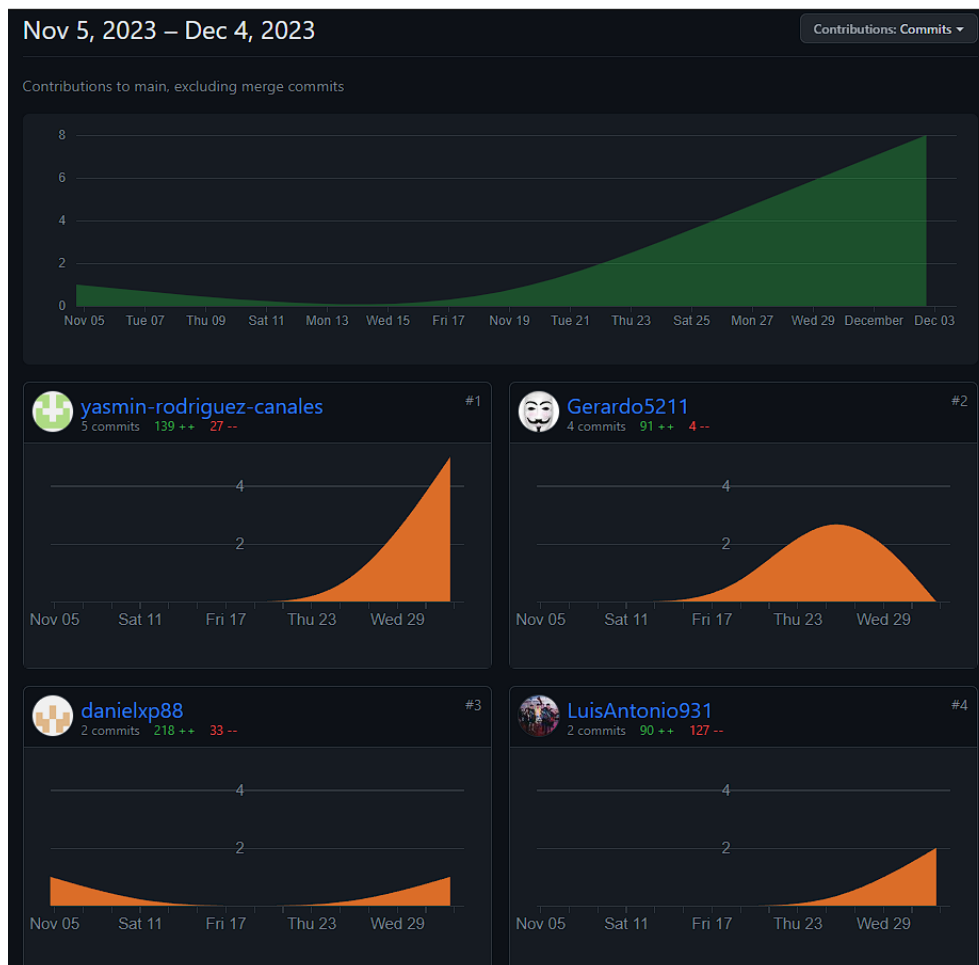
Se agrega captura de aquellos commit que se implementaron para el buen funcionamiento del proyecto, mediante la herramienta GitHub. En cada commit se hicieron actualizaciones importantes para la mejora y seguimiento del código, facilitando el trabajo colaborativo.

Los "commits" en GitHub son aquellos registros individuales de cambios que se realizan en un repositorio de código fuente. Cada "commit" representa una única unidad de cambio en el código y está asociado a un mensaje descriptivo que

programa terminado danielxp88 committed 3 hours ago	 4875165 
Commits on Dec 3, 2023	
Implementación de todas las funciones en un solo código LuisAntonio931 committed 11 hours ago	 fbd634a 
Inclusión de funciones para el tiempo de cada comanda generada LuisAntonio931 committed 14 hours ago	 86fa982 
Actualización para mostrar comandas yasmin-rodriguez-canales committed 15 hours ago	 9eb7f66 
Actualización de la función actualizar_completado yasmin-rodriguez-canales committed 15 hours ago	 0de79ab 
Se agrego la función de actualizar_completado yasmin-rodriguez-canales committed 16 hours ago	 e76441f 
Opción salir yasmin-rodriguez-canales committed 17 hours ago	 a102eb3 
Se agrego la función mostrar_datos yasmin-rodriguez-canales committed 17 hours ago	 f7dbe04 
Commits on Dec 1, 2023	
Se implemento la funcion Eliminar Gerardo5211 committed 3 days ago	 3437e50 
Se implemento la funcion Modificar Gerardo5211 committed 3 days ago	 5b35ecf 
Se implemento la funcion Crear Gerardo5211 committed 3 days ago	 4c30389 
En este fragmento se encuentra la conexion a la bd Gerardo5211 committed 3 days ago	 ecca96a 
Commits on Nov 9, 2023	
primer archivo danielxp88 committed last month	 d6e4eb6 

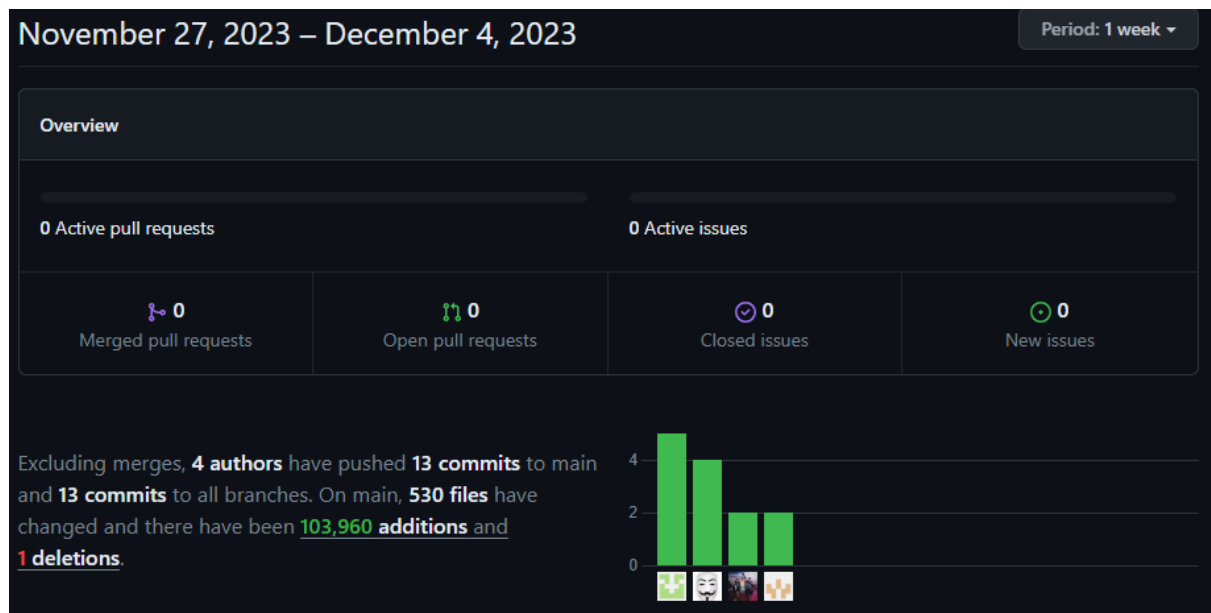
Gráfica de contribución

Se muestra una representación visual en forma de gráfica de toda la actividad que tuvo cada uno de los integrantes del equipo a lo largo del proyecto, desde la creación del repositorio, hasta la última actualización que realizó cada integrante. Otra cosa que se puede observar, son las líneas de código que aportó cada uno al código, junto con las líneas que fueron eliminadas entre todos los commit que se realizaron.



Insights

Se muestra una gráfica de la participación de cada integrante, donde se muestran el total de colaboradores, de commits, 530 archivos en los cuales se encuentran los archivos necesarios para la librería de MongoDB. Se cuentan con 103,960 de las cuales aproximadamente 340 son código.



Evaluación 😊

Nombre	Calificación
López Delgado Luis Antonio 🙌	10
Medina Licon Gerardo De Jesús ⚽	10
Rodríguez Canales Yasmin 🦆	10
Xithe Pintado Daniel 🎮	10

Información del proyecto:

GitHub:

<https://github.com/tiendaPelon/MenuTech>

Drive:

https://drive.google.com/drive/folders/1dwEm20rRE7hT_3weA2rneqWtWB0p89me?usp=sharing