Logotipo, nombre de la empresa

El contenido generado por IA puede ser incorrecto.

**Estructura de Datos**



**Maestra:**

**Lic. Blanca Aracely Aranda Machorro**

**Fecha:**

**18 de agosto del 2025**

Índice

[Sistema de manejo de pedidos de comida 3](#_Toc207502567)

[Objetivo 3](#_Toc207502568)

[Descripción 3](#_Toc207502569)

[Análisis de la importancia de la implementación 3](#_Toc207502570)

[Requisitos 4](#_Toc207502571)

[Diseño 7](#_Toc207502572)

[Diagrama de caso de uso empresarial 7](#_Toc207502573)

[Diagrama de clases UML 8](#_Toc207502574)

[Diseño de algoritmos y lógica 9](#_Toc207502575)

[Algoritmo de gestión de pedidos 9](#_Toc207502576)

[Algoritmo de cálculo de prioridad 10](#_Toc207502577)

[Algoritmo de procesamiento para la cocina 11](#_Toc207502578)

[Ciclo de un pedido 12](#_Toc207502579)

[Diagramas de flujo 12](#_Toc207502580)

[Flujo del sistema principal 12](#_Toc207502581)

[Flujo de cálculo de prioridad 13](#_Toc207502582)

[Flujo de atención en la cocina 14](#_Toc207502583)

[Desarrollo del sistema 15](#_Toc207502584)

[Colas 15](#_Toc207502585)

[Pilas 16](#_Toc207502586)

[Listas 17](#_Toc207502587)

[Main 18](#_Toc207502588)

[Pruebas del sistema 19](#_Toc207502589)

[Menú Principal 19](#_Toc207502590)

[Gestión de pedidos 19](#_Toc207502591)

[Consultar Cocina 22](#_Toc207502592)

[Clientes en espera 23](#_Toc207502593)

[Pedidos en Cocina 24](#_Toc207502594)

[Ver historial 25](#_Toc207502595)

[Extra: Página Web 27](#_Toc207502596)

[Glosario 28](#_Toc207502597)

[Bibliografía 29](#_Toc207502598)

[Autores 30](#_Toc207502599)

# Sistema de manejo de pedidos de comida

(Foxy Manager)

# Objetivo

Implementar y manipular diferentes estructuras de datos en Java, así como entender las aplicaciones prácticas de pilas, colas y listas en el manejo de información para desarrollar habilidades de programación y resolución de problemas en un contexto empresarial simulado.

# Descripción

Desarrollamos un programa en Java de gestión de un negocio de comida rápida. Utilizando las distintas estructuras de datos que hemos visto durante el curso (pilas, colas y lista) para atender las ordenes que se van realizando los clientes de una manera ordenada y con un sistema de prioridad.

# Análisis de la importancia de la implementación

Nuestro sistema, llamado Foxy Manager es un sistema de gestión para las cocinas de los locales de los restaurantes que compran nuestro servicio. Donde los clientes que llegan a estos y pueden hacer sus órdenes mediante un panel donde podrán ver los distintos menús disponibles, crear un pedido con varios artículos al cual se le asigna una prioridad en base a cómo se realizó la solicitud (autoservicio o en la caja), además de la dificultad del combo, significando que si el combo es fácil se asigna una prioridad mayor al igual que si se pide en autoservicio.

La importancia de contar con Foxy Manager radica en su capacidad para optimizar los dos factores clave en un negocio de comida rápida: tiempo y esfuerzo. Al dar prioridad a los combos más sencillos, el personal de cocina puede completar más pedidos en menos tiempo, reduciendo la espera de los clientes y mejorando su experiencia. Esto no solo aumenta la eficiencia del equipo, sino que también fomenta la satisfacción y fidelización de los clientes, convirtiendo al sistema en una herramienta estratégica para el crecimiento del negocio.

# 

# Requisitos

|  |  |  |  |
| --- | --- | --- | --- |
| ID\_requerimiento | Descripción | Atiende | ID\_evidencia |
| 1 | Implementar las tres estructuras de datos (pila, cola y lista) con todas sus operaciones básicas | Ivan Silverio & Demian Quiroga | A1, A2,A3 |
| 2 | Desarrollar interfaz de usuario en consola intuitiva con gestión completa de tareas (agregar, ver, eliminar) | Ivan Silverio & Demian Quiroga | B1,B2,B3,B4 |
| 3 | Implementar visualización de todas las tareas pendientes ordenadas por urgencia y departamento | Demian Quiroga & Diego Medellín | C1, C2 |
| 4 | Diseñar el programa con una estructura clara, organizada y bien documentada | Omar Guevara | D1,D2, D3 |
| 5 | Garantizar que el programa funcione de manera eficiente con lógica clara y estructurada | Omar Guevara | E1,E2 |
| 6 | Asegurar la fiabilidad mediante pruebas de todas las funcionalidades | Omar Guevara & Diego Medellín | F |

Tabla 1. Requisitos del sistema

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID Evidencia | ID Requisito | Nombre\_evidencia | Comentarios | Evidencia |
| A1 | 1 | Declaración de las estructuras |  | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/1/Declaracion%20de%20estructuras.jpeg> |
| A2 | 1 | Inicio de estructuras |  | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/1/Inicio%20de%20estructuras.jpeg> |
| A3 | 1 | Ejemplo de operaciones básicas |  | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/1/Ejemplo%20de%20operaciones%20basicas.jpeg> |
| B1 | 2 | menú de gestión de pedidos | Pensado para cajeros | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/2/menu%20de%20gestion%20de%20pedidos.jpeg> |
| B2 | 2 | Ventana de clientes en espera | Pensado para mover a los clientes a lista de pendientes en cocina. | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/2/ventana%20de%20clientes%20en%20espera.jpeg> |
| B3 | 2 | Ventana para cocineros | Pensado para que el personal de cocina pueda revisar los pedidos pendientes y marcarlos como listos. | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/2/ventana%20para%20cocineros.jpeg> |
| B4 | 2 | Mostrar combos | Sale a partir del menú de gestión de pedidos. | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/2/Mostrar%20combos.jpeg> |
| C1 | 3 | Pedidos ordenados por urgencia | Cuando no hay pedidos pendientes no muestra ningún texto. | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/3/Pedidos%20ordenados%20por%20urgencia.jpeg> |
| C2 | 3 | Lista sin pedidos pendientes |  | <https://github.com/Sunsetby2006/Foxy-Manager/blob/90793c9b1f1c7e0462202271d04a2e0fab5bedd1/Evidencias_requisitos/3/Lista%20sin%20pedidos%20pendientes.jpeg> |
| D1 | 4 | Foto de creación de diagramas | Se crearon en un whiteboard de canva . | <https://github.com/Sunsetby2006/Foxy-Manager/blob/308991a4c02e3a319388306528fb618192107c1b/Evidencias_requisitos/4/Creacion%20de%20diagramas.jpeg> |
| D2 | 4 | Whiteboard de Canva | Se adjunta enlace del canva original para poder hacer zoom a los diagramas. | <https://www.canva.com/design/DAGcYSFTWug/oogIERxGGY-7id__t5AAqA/view?utm_content=DAGcYSFTWug&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=hee2dd4f859> |
| D3 | 4 | Documentación y seudocódigos del diseño | Toda esta parte se realizó directamente en el word. | Sección "Diseño" dentro del documento TM\_ESTRUCTURA\_DE\_DATOS\_AVACE\_1EQUIPO\_FoxyManager\_18AGOSTO2025.docx |
| E1 | 5 | Comentarios 1 | Pusimos comentarios para tener una mayor claridad dentro de los códigos. | <https://github.com/Sunsetby2006/Foxy-Manager/blob/308991a4c02e3a319388306528fb618192107c1b/Evidencias_requisitos/5/Comentarios%201.jpeg> |
| E2 | 5 | Comentarios 2 |  | <https://github.com/Sunsetby2006/Foxy-Manager/blob/308991a4c02e3a319388306528fb618192107c1b/Evidencias_requisitos/5/Comentarios%202.jpeg> |
| F | 6 | Pruebas | Toda esta parte se realizó directamente en el word. | Sección "Pruebas" dentro del documento TM\_ESTRUCTURA\_DE\_DATOS\_AVACE\_1EQUIPO\_FoxyManager\_18AGOSTO2025.docx |

Tabla 2. Evidencia de requisitos

# Diseño

Para diseñar el proyecto, debemos definir las bases del sistema, sus componentes y flujo de operaciones, a continuación, se presenta el diseño detallado de Foxy Manager.

## Diagrama de caso de uso empresarial

Primero debemos de empezar analizando cómo todos en el entorno del sistema van a coexistir, definimos al cliente, cajero, cocinero y nuestro sistema como loa actores clave, donde cada uno tiene roles específicos las cuales debe realizar para el funcionamiento óptimo de este.

Diagrama

El contenido generado por IA puede ser incorrecto.

Diagrama 1. Diagrama de caso de uso de la empresa.

Aquí tenemos claras las funciones que realiza cada elemento que toma parte del sistema. Notamos que algunas actividades se extienden de otras, lo que significa que son opcionales y solo ocurren si la acción base ya fue realizada. Otras se incluyen, que significa que se espera que al hacer una se haga la otra.

## Diagrama de clases UML

Lo siguiente es definir qué va a contener nuestro sistema, creamos clases con sus atributos, que las características que tiene, y sus métodos, que son las acciones que hacen.

Escala de tiempo

El contenido generado por IA puede ser incorrecto.

Diagrama 2. Clases de Foxy Manager.

Viendo claro el diagrama de clases, vemos de una mejor manera que el cliente accede al menú, y en base al menú crea una orden o pedido, el cual obviamente contiene elementos del menú, por otro lado, la orden se clasifica en base a la urgencia y en dónde se generó. Por lo que se puede almacenar en la pila o en la cola de espera. Finalmente la orden se almacena en la lista del historial, que sirve como un registro de todos los pedidos que se generaron y ya fueron entregados.

## Diseño de algoritmos y lógica

### Algoritmo de gestión de pedidos

   INICIO

       Crear colaPrioridad (PriorityQueue)

       Crear pilaAutoservicio (Stack)

       Crear listaHistorial (ArrayList)

       MIENTRAS (sistemaActivo) HACER:

           pedido <- ObtenerSiguienteEvento()

           SEGUN (pedido.tipoEvento):

               CASO "NUEVO\_PEDIDO":

                   prioridad <- CalcularPrioridad(pedido)

                   pedido.setPrioridad(prioridad)

                   SI (pedido.esAutoservicio) ENTONCES

                       SI (pilaAutoservicio.estáVacía()) ENTONCES

                           pilaAutoservicio.push(pedido)

                           pedido.estado = "EN\_PROCESO"

                       SINO

                           colaPrioridad.add(pedido)

                       FIN SI

                   SINO

                       colaPrioridad.add(pedido)

                   FIN SI

               CASO "PEDIDO\_COMPLETADO":

                   SI (pedido.esAutoservicio) ENTONCES

                       pilaAutoservicio.pop()

                       SI (NO pilaAutoservicio.estáVacía()) ENTONCES

                           siguiente <- pilaAutoservicio.peek()

                           siguiente.estado = "EN\_PROCESO"

                       FIN SI

                   FIN SI

                   listaHistorial.add(pedido)

                   pedido.estado = "ENTREGADO"

               CASO "CONSULTA\_COCINA":

                   SI (NO pilaAutoservicio.estáVacía()) ENTONCES

                       pedidoActual <- pilaAutoservicio.peek()

                       DEVOLVER pedidoActual

                   SINO SI (NO colaPrioridad.estáVacía()) ENTONCES

                       pedidoActual <- colaPrioridad.poll()

                       pedidoActual.estado = "EN\_PROCESO"

                       DEVOLVER pedidoActual

                   SINO

                       DEVOLVER null

                   FIN SI

           FIN SEGUN

       FIN MIENTRAS

   FIN

En este pseudocódigo, se observa que cuando llega un nuevo pedido, primero se calcula su prioridad y se determina si es de autoservicio o no. Los pedidos de autoservicio se almacenan en la pila para ser procesados uno a la vez, mientras que los demás se agregan a la cola de prioridad según su importancia. A medida que los pedidos se completan, su estado se actualiza a entregado y se registran en la lista de historial, que funciona como un registro de todos los pedidos ya procesados.

Además, el sistema permite que la cocina consulte en todo momento cuál es el pedido que está en proceso. Primero se verifica la pila de autoservicio y si está vacía, se toma el siguiente pedido de la cola de prioridad. De esta manera, se mantiene un flujo organizado, asegurando que los pedidos urgentes se atiendan primero y que los pedidos de autoservicio se gestionen de forma secuencial y así evitar las grandes filas dentro del autoservicio.

### Algoritmo de cálculo de prioridad

   INICIO

       prioridad = 0

       // Valor por tipo de pedido

       SEGUN (tipo):

           CASO "AUTOSERVICIO":

               prioridad = prioridad + 3

           CASO "CAJA":

               prioridad = prioridad + 1

       FIN SEGUN

       // Valor por dificultad del combo

       SEGUN (dificultad):

           CASO "FÁCIL":

               prioridad = prioridad + 2

           CASO "MEDIO":

               prioridad = prioridad + 1

           CASO "DIFÍCIL":

               prioridad = prioridad + 0

       FIN SEGUN

       // Prioridad inversa (mayor valor = mayor prioridad)

       DEVOLVER prioridad

   FIN

En este, primero se asigna un valor según el tipo de pedido: los pedidos de autoservicio reciben más puntos que los de caja, ya que se consideran más urgentes. Después, se suma un valor adicional según la dificultad del combo; los pedidos fáciles obtienen más puntos que los difíciles, lo que significa que se deben de atender primero los que se preparan más rápido.

Al final, la función devuelve un número que representa la prioridad total del pedido, donde un valor más alto indica que el pedido debe ser procesado antes que otros. De esta manera, el sistema puede organizar los pedidos de forma eficiente, combinando urgencia y facilidad de preparación para optimizar el flujo de trabajo en la cocina.

### Algoritmo de procesamiento para la cocina

    INICIO

        // Primera prioridad: pedidos de autoservicio en pila

        SI (NO pilaAutoservicio.estáVacía()) ENTONCES

            pedidoActual <- pilaAutoservicio.peek()

            SI (pedidoActual.estado != "EN\_PROCESO") ENTONCES

                pedidoActual.estado = "EN\_PROCESO"

            FIN SI

            DEVOLVER pedidoActual

        FIN SI

        // Segunda prioridad: pedidos en cola de prioridad

        SI (NO colaPrioridad.estáVacía()) ENTONCES

            pedidoActual <- colaPrioridad.poll()

            pedidoActual.estado = "EN\_PROCESO"

            DEVOLVER pedidoActual

        FIN SI

        // No hay pedidos pendientes

        DEVOLVER null

    FIN

Le damos prioridad a los pedidos de autoservicio que están en la pila. Si hay alguno, se toma el de arriba y se asegura que su estado sea “EN\_PROCESO”. Solo si la pila está vacía, se revisa la cola de prioridad para tomar el siguiente pedido más urgente y también actualizar su estado.

Si no hay pedidos en ninguno de los dos lugares, la función devuelve null, indicando que no hay pedidos pendientes por procesar. De esta manera, se mantiene un flujo ordenado, atendiendo primero los pedidos de autoservicio de manera secuencial y luego los demás según su prioridad.

### Ciclo de un pedido

Imagen que contiene Escala de tiempo

El contenido generado por IA puede ser incorrecto.

Diagrama 3. Proceso de un pedido.

Pendiente significa que se encuentra en espera a ser atendido, ya sea por que está en la pila o en la cola, en proceso es cuando se esta elaborando o atendiendo una orden, y al final entregado es cuando se le da al cliente la orden, después de esto, se elimina de la espera y se almacena en el historial.

## Diagramas de flujo

### Flujo del sistema principal

Diagrama

El contenido generado por IA puede ser incorrecto.

Diagrama 4. Flujo del sistema principal de Foxy Manager.

El diagrama de flujo muestra de manera clara el funcionamiento del sistema de gestión de pedidos. Primero se inicializan las estructuras principales: la cola de prioridad, la pila de autoservicio y la lista de historial. A partir de ahí, cada vez que ocurre un evento, el sistema evalúa su tipo para decidir la acción a realizar. Si se trata de un nuevo pedido, se calcula su prioridad y se determina si va a la pila o a la cola; si un pedido se completa, se retira de su estructura y se envía al historial marcándolo como entregado; finalmente, cuando la cocina realiza una consulta, el sistema devuelve el pedido en proceso desde la pila de autoservicio o desde la cola de prioridad. En caso de no existir pedidos pendientes, se devuelve un valor nulo. De esta manerarepresentamos el ciclo continuo de recepción, procesamiento y entrega de pedidos.

### Flujo de cálculo de prioridad

Diagrama, Forma

El contenido generado por IA puede ser incorrecto.

Diagrama 5. Flujo de cálculo de prioridades.

Esto nos representa el proceso para calcular la prioridad de un pedido. El procedimiento inicia con un valor de prioridad igual a cero y luego se evalúa el tipo de pedido: si es de autoservicio se suman 3 puntos y si es en la caja, se suma 1 punto. Posteriormente, se analiza la dificultad del combo; en este caso, los pedidos fáciles aportan 2 puntos adicionales, los de dificultad media suman 1 y los difíciles no suman nada. Finalmente, el sistema devuelve el valor total calculado, que servirá para organizar los pedidos según su urgencia y facilidad de preparación.

### Flujo de atención en la cocina

Diagrama

El contenido generado por IA puede ser incorrecto.

Diagrama 6. Flujo de como debe atender personal de cocina.

Finalmente, en la cocina se sigue un proceso ordenado para determinar qué pedido atender. Primero se revisa la pila de autoservicio y, si contiene un pedido, este se obtiene y se valida su estado; en caso de no estar marcado como EN\_PROCESO, se actualiza antes de entregarlo. Si la pila está vacía, el sistema pasa a la cola de prioridad, de donde toma el siguiente pedido y también lo coloca en estado EN\_PROCESO. Solo si ambas estructuras están vacías se concluye que no existen pedidos pendientes. Con este flujo, se garantiza que los pedidos de autoservicio tengan prioridad sobre los demás y que los pedidos en cola se atiendan de manera organizada.

# Desarrollo del sistema

## Colas



Imagen 1. Captura del código de colas

En esta imagen podemos observar un poco del código que es para la clase de Colas (**ClassColas**), colas lo usamos para poder realizar y registrar los pedidos; guardamos lo que son los combos junto al valor de las prioridades de orden y sus precios. Las funciones que mandamos a llamar son:

* cargarPedidosCSV() “Basicamente carga los pedidos pendientes.”
* cargarHistorial() “Carga todo el historial de los pedidos.”
* crearBoton() “Aquí creamos el botón para el panel.”
* agregarPedido() “Acá permite registrar nuevos pedidos.”
* consultaCocina() “Por aquí es donde se ven los pedidos pendientes.”
* mostrarHistorial() “Muestra el historial de todos los pedidos ya hechos.”
* guardarPedidosCSV() “Aquí guardamos todos los pedidos activos.”

## Pilas



Imagen 2. Captura del código de pilas

Continuamos con la clase de Pilas (**ClassPilas**), en esta clase es donde cargamos los pedidos, donde los finalizamos y donde se van al historial de pedidos; Las funciones que llamamos aqui son estas:

* cargarPedidos() “Aquí se cargan y muestran los pedidos.”
* completarPedidos() “Aquí se marca un pedido como completado; se mueve del archivo de pedidos al historial en un csv dentro de \data\logs y se refresca la tabla.”

## Listas



Imagen 3. Captura del código de listas

Seguimos con la clase de Listas (**ClassListas**), acá es donde se pueden ver los pedidos en las tablas, donde se actualiza la información de los pedidos, las funciones a llamar son:

* cargarPedidosDesdeCSV() “Carga los pedidos desde el archivo CSV.”
* actualizarTabla() “Actualiza los datos de la tabla según a lo que se realizó.”
* refrescar() “Refresca la tabla para mostrar la información más reciente.”

## Main



Imagen 4. Captura del código main ejecutable.

Y, por último, continuamos con el Main, aquí básicamente es donde se despliega todo el panel del proyecto. Aquí se mandan a llamar todas las clases (Listas, Colas y Pilas) para que todo que el sistema funcione correctamente. Las funciones que se mandan a llamar aqui son:

* mostrarPantallaDeCarga(); “Se despliega la pantalla de carga de inicio del sistema.”
* mostrarMenuInicio(); “Nos muestra el panel en donde podemos interactuar con todo el sistema.”

# Pruebas del sistema

Una vez implementados los códigos y explicadas sus funciones, es necesario demostrar el funcionamiento completo del sistema. En esta sección se presentan las pruebas realizadas para validar su operación.

## Menú Principal

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Imagen 5. Menú principal con las 4 opciones principales.

## Gestión de pedidos

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Imagen 6. Menú de todos los combos para agregar un pedido.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Imagen 6.1. Ventana para agregar el nombre del cliente.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Imagen 6.2. Ventana para seleccionar por dónde se realiza el pedido (Esto define en parte la prioridad).

Interfaz de usuario gráfica, PowerPoint

El contenido generado por IA puede ser incorrecto.

Imagen 6.3. Ventana para confirmar el pedido.

Diagrama

El contenido generado por IA puede ser incorrecto.

Imagen 6.4. Ventana de confirmación.

## Consultar Cocina

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Imagen 7. Mensaje con los pedidos pendientes (Únicamente existe el creado anteriormente)

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Imagen 7.1. Pedidos pendientes mostrando urgencia (Se agregaron más pedidos).

## Clientes en espera

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Imagen 8. Ventana de clientes pendientes antes de seleccionar “Refrescar”.

Tabla

El contenido generado por IA puede ser incorrecto.

Imagen 8.1. Ventana con todas los clientes a los que aún no se les entrega su pedido.

## Pedidos en Cocina

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen 9. Ventana de pedidos en cocina pendientes antes de seleccionar “Refrescar”.

Tabla

El contenido generado por IA puede ser incorrecto.

Imagen 9.1 Ventana de pedidos en cocina pendientes después de seleccionar “Refrescar”.

Interfaz de usuario gráfica, Aplicación, PowerPoint

El contenido generado por IA puede ser incorrecto.

Imagen 9.2 Ventana de pedidos en cocina después de completar el pedido de Omar.

## Ver historial

Después de que se terminan de entregar todas las órdenes, podemos consultar el historial. El cuál de encuentra en la carpeta /data/logs en forma de un archivo .csv, previamente hemos hecho varias pruebas, por lo que ya tiene varios registros, incluyendo a “Omar Gurvara” de la imagen 9.2.

Texto

El contenido generado por IA puede ser incorrecto.

Imagen 10. Archivo historial.cvs lleno con los datos hasta la orden de Omar Guevara

Ahora marcaremos como completado el resto de las órdenes que teníamos pendientes.

Texto

El contenido generado por IA puede ser incorrecto.

Imagen 10.1. Archivo historial.cvs lleno con los datos completos.

Nota: Vemos que en Visual Studio los caracteres especiales no se muestran correctamente, por lo que recomendamos registrar los usuarios sin acentos ni “Ñ”.

Ya que contamos con todos los pedidos, desde nuestro sistema podemos revisar también el historial, sin tener que acceder al .csv.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

imagen 11. Ventana con el historial completo.

Nota: Para que se pudiera actualizar, tenemos que reiniciar el sistema por completo, sin embargo, el archivo .csv si se actualiza a tiempo real.

# Extra: Página Web

Creamos una página web con ayuda del repositorio de Github, donde hablamos de Foxy Manager como empresa, lo que nos sirve para cuando podamos emprender, tener ya la herramienta para darnos a conocer de una manera más formal y tecnológica, en esta página hablamos de nuestro logo, el cual pusimos en los diagramas que adjuntamos en la sección del diseño. Resumimos la metodología del sistema y mencionamos algunas ventajas de contratar nuestro software, al igual de mostrar al equipo completo.

Si desea visitar nuestra página puede acceder mediante el siguiente enlace: <https://sunsetby2006.github.io/Foxy-Manager/>

Texto

El contenido generado por IA puede ser incorrecto.

Imagen 12. Vista previa de la página web.

# Glosario

**Algoritmo**: Conjunto de pasos ordenados que permiten resolver un problema o realizar una tarea de manera lógica.

**Clase**: Plantilla en programación que define las características (atributos) y acciones (métodos) de los objetos que se crean a partir de ella.

**Código**: Instrucciones escritas en un lenguaje de programación que forman un programa.

**Cola (Queue)**: Estructura de datos en la que el primer elemento en entrar es el primero en salir (como una fila para las tortillas).

**Consola:** Ventana de texto donde se muestran mensajes y se escriben comandos para interactuar con un programa.

**Estructura de datos**: Forma organizada de guardar información en la memoria de la computadora para usarla de manera eficiente.

**GitHub**: Plataforma en línea que permite guardar proyectos de programación en repositorios, compartirlos y trabajar en equipo.

**Historial:** Registro de pedidos o acciones que ya se realizaron en el sistema.

**Interfaz de usuario**: Parte del sistema que ven y usan las personas para interactuar con el programa.

**Java:** Lenguaje de programación utilizado en el proyecto para desarrollar el sistema.

**Lista (List)**: Estructura de datos que almacena elementos de forma ordenada, la cual funciona como una lista de compras.

**Main:** Punto principal de inicio de un programa en Java, donde comienza su ejecución.

**Página web**: Documento accesible desde internet a través de un navegador.

**Pila (Stack**): Estructura de datos en la que el último elemento en entrar es el primero en salir (como una pila de platos).

**Prioridad**: Valor que define qué pedido o tarea debe atenderse primero, según su importancia o urgencia.

**Pseudocódigo**: Forma de escribir la lógica de un programa usando instrucciones, sin necesidad de seguir las reglas estrictas de un lenguaje de programación.

**Repositorio de GitHub**: Espacio en GitHub donde se guarda un proyecto con su código y archivos, incluyendo historial de versiones.

**Usuario**: Persona que utiliza el sistema, como cliente, cajero o cocinero.

# Bibliografía

*Acerca de los repositorios - Documentación de GitHub*. (s. f.). GitHub Docs. https://docs.github.com/es/repositories/creating-and-managing-repositories/about-repositories

Alarcón, J. M. (2020, 11 diciembre). *Cómo leer y escribir archivos CSV con Java - campusMVP.es*. campusMVP.es. https://www.campusmvp.es/recursos/post/como-leer-y-escribir-archivos-csv-con-java.aspx

Crespo, A. (2024, 25 noviembre). Curso de Java: colas de prioridad. *RedesZone*. https://www.redeszone.net/2012/04/02/curso-de-java-colas-de-prioridad/

*Estructura de datos: ¿Para qué sirve y qué tipos existen?* (s. f.). UCMA. https://www.universitatcarlemany.com/actualidad/blog/estructura-datos/

Kinsta. (2025, 26 febrero). *¿Qué es el Pseudocódigo y Cómo Puede Mejorar tu Programación?* Kinsta®. https://kinsta.com/es/base-de-conocimiento/que-es-pseudocodigo/

M, G. P. (2023, 17 mayo). *Las principales estructuras de datos que deberías saber para tu próxima entrevista de programación*. freeCodeCamp.org. https://www.freecodecamp.org/espanol/news/las-principales-estructuras-de-datos-que-deberias-saber-para-tu-proxima-entrevista-de-programacion/

Nivardo. (2024, 10 septiembre). *Prioridades de Hilo en Java*. Oregoom.com. https://oregoom.com/java/prioridades-de-hilo/

*Prioridad*. (2022, 1 noviembre). Portal Académico del CCH. https://portalacademico.cch.unam.mx/cibernetica2/programacion-java/prioridad

Ramos, C. (2025, 13 mayo). GitHub: Qué es, cómo funciona y sus ventajas para desarrolladores. *Blog de LucusHost*. https://www.lucushost.com/blog/github/

Robledano, A. (2024a, septiembre 23). Qué es pseudocódigo y por qué es esencial en programación. *OpenWebinars.net*. https://openwebinars.net/blog/que-es-pseudocodigo/

TodoCode. (2022, 21 septiembre). *🔴 ¿Cómo crear una APP Java DESDE CERO? 👨🏻‍💻👩🏻‍💻 ¡Con CRUD con JPA! - Clase 100% PRÁCTICA (19-09)* [Vídeo]. YouTube. https://www.youtube.com/watch?v=Gd4QQtQz5DE

Unionn\_Estudio\_Creativo. (2023, 14 julio). 161. Psicología del color Naranja: innovación, entusiasmo y creatividad - Unionn Estudio Creativo. *Unionn Estudio Creativo*. https://unionn.es/161-psicologia-del-color-naranja-innovacion-entusiasmo-y-creatividad/

# Autores

|  |  |
| --- | --- |
| Un joven sentado en el pasto  El contenido generado por IA puede ser incorrecto. | Omar Fernando Guevara Cavazos (Arquitecto y creador de página web) |
|  | Ivan Gerardo Tenorio Silverio (Desarrollador y programador) |
|  | Diego Alejandro Medellín Méndez (Analista y escritor técnico) |
|  | Demian Kalil Quiroga Suarez (Desarrollador y programador) |