Sistema de Administración

Gabriel Quinche, Karen Medina

No. grupo del curso: {*03*} y No. de Equipo de Trabajo: {*08*}

I. [1] INTRODUCCIÓN

El siguiente documento presenta un proyecto de software, con su propósito, requerimientos y posible alcance.Para este proyecto se espera utilizar diferentes estructuras de datos, tales como pilas, colas, árboles y demás estructuras que se desarrollen a lo largo de la materia Estructuras de datos.

I. DESCRIPCIÓN DEL PROBLEMA

En los restaurantes generalmente se generan conflictos y complicaciones al tener que transmitir “voz a voz” la información de las ventas y el inventario diario restante en la cocina. Se pretende lograr que todos los responsables de manejar esa información tengan acceso a una base de datos actualizada en tiempo real a tener en cuenta a la hora de despachar los pedidos.

II. USUARIOS DEL PRODUCTO DE SOFTWARE

Existirían dos tipos de usuarios clasificados de acuerdo a su interacción con el sistema: un administrador, con permisos para modificar cantidades de todos los elementos del sistema, y un usuario tipo “mesero” con accesibilidad restringida para modificar los valores del inventario mediante la generación de pedidos. Eventualmente podría incluirse un usuario tipo “cocina” que reciba pedidos y genere su cobro una vez despachado.

III. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

- **Llenar inventario**

Descripción:
El usuario administrador guardará los productos que se proyectan disponibles para el día.

Acciones iniciadoras y comportamiento esperado:
Se crean listas con los ingredientes disponibles que sólo el usuario administrador tiene la capacidad de modificar. Se espera ser este el paso inicial para las siguientes funcionalidades.

Requerimientos funcionales:
Por cada tipo de ingrediente existirá una lista. A esta lista se le agregan los ingredientes disponibles de su mismo tipo. (Tree)

- **Consultar disponibilidad**

Descripción:
Los usuarios pueden hacer un recorrido por las listas para saber qué productos hay en el inventario

Acciones iniciadoras y comportamiento esperado:
El usuario elige el tipo de producto del que desea saber. El sistema debe responder con la cantidad de productos en lista.

Requerimientos funcionales:
Cada lista cuenta con un contador que indica la cantidad de productos, este se modifica cada vez que se agregan productos o se recibe un pedido. Al hacer la consulta se muestra el respectivo nombre del ingrediente junto a su contador. (Tree)

- **Hacer pedido**

Descripción:
El usuario “mesero” selecciona una combinación de ingredientes que se encuentren disponibles en las listas. Luego el pedido pasa a una cola junto con los demás.

Acciones iniciadoras y comportamiento esperado:
Se eligen una serie de ingredientes que deben estar disponibles en tiempo real. El contador debe restar estos ingredientes del inventario apenas se ordene.

Requerimientos funcionales:
En caso de intentar pedir un ingrediente no disponible no se puede llevar a cabo el pedido. De modo que antes de enviar el pedido el sistema debe hacer una consulta al inventario. Si se lleva a cabo el pedido con éxito este pasará a una cola con los demás pedidos inicializados por el usuario ”mesero”.

- **Interfaz dinámica**

Descripción:
Para cualquier tipo de usuario la interfaz puede devolverse a la página anterior

Acciones iniciadoras y comportamiento esperado:

Requerimientos funcionales:

Los requerimientos funcionales de un sistema son aquellos que describen cualquier actividad que este deba realizar, en otras palabras, el comportamiento o función particular de un sistema o software cuando se cumplen ciertas condiciones.

[1]

Entre los posibles requerimientos funcionales de un sistema, se incluyen:

- Descripciones de los datos a ser ingresados en el sistema.
- Descripciones de las operaciones a ser realizadas en cada pantalla que se presenta.
- Descripción de los flujos de trabajo realizados por el sistema.
- Descripción de los reportes del sistema y otras salidas.
- Definición de quiénes pueden ingresar datos en el sistema.

De esta manera, se deben describir las interacciones que tendrán los usuarios con el software.

Cada funcionalidad se debe especificar así:

- *Nombre de la funcionalidad*

En el título de la funcionalidad, se recomienda utilizar nombres muy descriptivos para cada funcionalidad. No limitarse a nombrarlas "Funcionalidad 1", en cambio usar por ejemplo: "Autorización de pedido de compra".

- *Descripción:*

  - Descripción breve de la funcionalidad.

- *Acciones iniciadoras y comportamiento esperado*:
Secuencia de acciones del usuario y respuestas esperadas del programa para esta funcionalidad.

Requerimientos funcionales:

Lista detallada de los requerimientos funcionales asociados a esta funcionalidad.

Para cada requerimiento funcional se establece cómo debe mostrarse el software y cuáles comportamientos debe desempeñar para que el usuario pueda realizar la función que necesita.

Es recomendable prever y describir cómo debe responder el software ante condiciones de error y entradas de datos inválidas.

Las funcionalidades mínimas sobre los datos que se manejen deben prever operaciones de:

- Creación
- Actualización
- Eliminación
- Consulta total de los datos
- Búsqueda parcial de datos
- Ordenamiento
- Almacenamiento

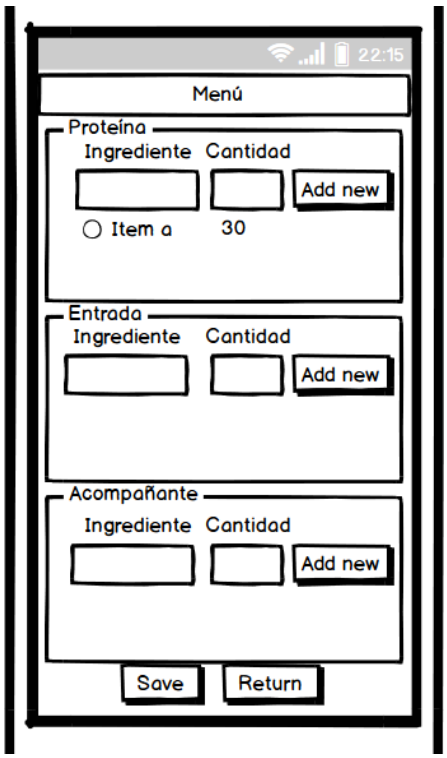
Aunque en otros cursos se estudian estrategias de organización y almacenamiento, en este curso el almacenamiento se requiere principalmente para facilitar las pruebas del prototipo de software. También, para facilitar su implementación, se deja abierta la opción a que se apoyen en el uso de sistemas manejadores de bases de datos, se haga almacenamiento por archivos, de objetos u otra estrategia que les convenga, siempre que se garantice la implementación y uso de las estructuras de datos, vistas en clase, en memoria dinámica.

IMPORTANTE: En cada una de las entregas para reportar el avance en el desarrollo del proyecto, se especificarán las funcionalidades mínimas y las estructuras de datos mínimas requeridas que se deben implementar. También, se debe presentar un análisis (especialmente, comparativo) breve de la eficiencia de las estructuras de datos usadas.

NOTA: En el siguiente enlace web (URL) puede encontrar una explicación de cómo diferenciar Requisitos Funcionales de los No Funcionales
https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

IV. AVANCE EN LA IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO

Definir menú:



V. Entornos de desarrollo y de operación

Se espera montar la lógica principal del programa mediante mockup sencillos usando Java y vaadin, en un repositorio compartido. Esto mediante la plataforma de desarrollo /framework de Vaadin, la cual contiene un conjunto de herramientas para el desarrollo de aplicaciones web.

Para el back end se espera utilizar una base de datos en MySQL, para esto se utiliza un modelo relacional con tres tablas básicas. Una para guardar los almuerzos que se venden al día, está contiene dos llaves foráneas que llevan a otras dos tablas. Existiría una tabla para los ingredientes y para los meseros.

VI. DESCRIPCIÓN DEL PROTOTIPO DE SOFTWARE

El proyecto se encuentra alojado en un repositorio de Github. Se puede consultar en el siguiente link:
https://github.com/gabrielqv/sares2.0

```
root
│   README.md
│
├── src                 // Carpeta donde se almacena el código fuente
│   └── my.packages     // Paquete principal del proyecto (kernel)
│       ├── Main.(*)  // Clase principal que ejecuta la aplicación. Ejemplo. Java: clase Main.java
│       ├── ...         // Otros archivos java con el código fuente de la app
│
├── docs
│   └── Entrega_(Plantilla).pdf     //Documento plantilla con la información de la entrega requerida
│   └── ......
│       OtrosDocumentos_X.pdf      //Otros documento según se requiera
├── data              //Carpeta con los datos de prueba del proyecto
├── dist              //Carpeta en donde se almacenan los archivos que ejecutan la aplicación
│   └── proyectoEntrega1.jar
│   └── proyectoEntrega2.jar
│   └── proyectoEntrega3.jar
│   └── ......
├── lib               // Carpeta con las librerías requeridas en la aplicación
├── Otros archivos...  // Otros archivos de ejecución y configuración necesarios.
                       // Ejemplo: project.xml, build.xml (ant), run.sh, run.bat (ejecución), etc.
```

Además, para mantener una versión gráfica de desarrollo del repositorio, se podrán apoyar en el uso de herramientas como Sourcetree, disponible en el siguiente URL:
https://www.sourcetreeapp.com/

Segundo, en este prototipo se deben implementar por lo menos una (1) instancia de las estructuras de datos tipo: **Lista encadenada (en alguna de sus diferentes versiones), Pila, Cola, Cola de prioridad, y Árbol (priorizando el árbol binario de búsqueda - BST)**.

Para el uso de la estructura de datos **Pila**, se requiere que el prototipo de software implemente la opción de hacer y deshacer algunas de las funcionalidades implementadas. También, a través del uso de la estructura de datos **Pila**, el prototipo implementado debe proveer una funcionalidad que permita a través de la interfaz de usuario establecida, "navegar" hacia adelante y hacia atrás en algunas de las funcionalidades. Por ejemplo, al estilo de la navegación que se hace en un navegador Web o en un explorador de archivos en un sistema operativo.

Tercero, para las listas encadenadas y árboles que se implementen se deben soportar por lo menos las siguientes operaciones funcionales:

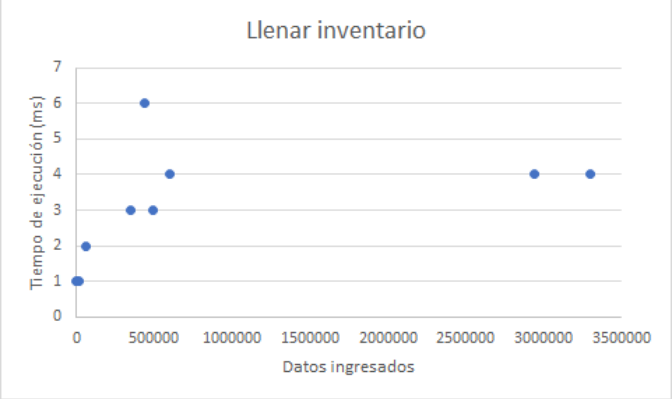
- Creación
- Inserción de un solo dato
- Actualización de un solo dato
- Eliminación de un solo dato
- Búsqueda de un solo dato
- Consulta de todos los datos
- Almacenamiento de los datos

VII. PRUEBAS DEL PROTOTIPO

Se deben realizar y documentar las pruebas del prototipo para algunos ejemplos (casos) de prueba para las funcionalidades que tomen más tiempo así:

- Escoger entre tres y cinco funcionalidades que sean las de mayor costo computacional en tiempo;
- Para cada funcionalidad se deben realizar pruebas para varios tamaños de datos de prueba (n), por lo menos para los siguientes tamaños:
    - 10 mil datos,
    - 100 mil datos,
    - 1 millón de datos,
    - 10 millones de datos, y
    - 100 millones de datos,
- Hacer una tabla y realizar gráficos comparativos de los tiempos que toma realizar las funcionalidades consideradas para los diferentes tamaños de los datos de prueba. Específicamente, se debe elaborar una tabla que permita observar la comparación de las funcionalidades del prototipo considerando las estructuras de datos implementadas y los tiempos que tomó realizar las operaciones para las diferentes cantidades de datos usados en las pruebas. Así mismo, realice un análisis de la complejidad con respecto a los resultados y tiempos obtenidos. Puede tomar la Tabla 1, como referencia para documentar este análisis. Podrá modificarla y, de ser necesario, usar alguna hoja de cálculo como Excel para facilitar su elaboración.

Gráfica 1. Comparación tiempo de ejecución consulta de inventario.



Gráfica 2. Comparación tiempo de ejecución llenar inventario.

Tabla 1. Tabla comparativa de las pruebas realizadas.

Nombre de la funcionalidad	Tipo(s) de estructura de datos	Cantidad de datos probados	Análisis realizado (Notación Big O)	Tiempos de ejcución

Se enfatiza que además de las tablas comparativas, se deben presentar gráficas que ilustren el análisis de complejidad realizado.

VIII. DIFICULTADES Y LECCIONES APRENDIDAS

Se encontraron dificultades respecto al almacenamiento de los datos. pues el framework maneja un paradigma de diseño, muy diferente al básico por objetos, con conceptos como inyeción de dependencias, o autowires, espera un manejo amplio de clases desvinculadas, y con un sistema de control, basado especialmente en una base de datos optimizada, por lo tanto los intentos de proponer otras estructuras óptimas, aunque se podian probar en el fondo no compaginas con el mantra del framework. La prueba con cien millones de datos no solo no concluía sino consumía excesivamente memoria, aunque en teoría nunca se llegara a tener un restaurante con esta cantidad de diferentes tipos de insumos, se ven las limitaciones del framework por su enfoque web, además de la falta del entorno chrome para comunicar este preocupante uso de memoria, sin embargo evitamos asumir una total descalificación de la eficiencia, pues realmente siendo que la implementación de árboles era recursiva y no se tienen aún muy bueno conceptos de probabilidad, era fácil llegar a casos extremos donde los árboles no eran utiles ni con una maquina especial, o un entorno diferente