PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

SISTEMA GESTIÓN DE RESTAURANTES

FELIPE ANDRÉS GUAJARDO NÚÑEZ
BASTIÁN ANDRÉS RAMÍREZ URZÚA
JOSÉ ARTURO VERGARA TORRES

Programación Avanzada

Mayo, 2019

Índice

Lista de Figuras

1. Introducción.

Actualmente las grandes cadenas y empresas relacionadas al mercado de la comida rápida y restaurantes adquieren (o ya poseen) rápidamente *software* debido a las ventajas que ofrece en la automatización de procesos logísticos y estadísticos. Además de proveer información útil en la administración, también significa un gran avance en servicio al cliente, mejorando no solo la rapidez, sino que también la calidad de este. Es por esto, que se propone modelar una aplicación enfocada en aquellas medianas y pequeñas empresas del rubro de la comida rápida y Restaurantes, que se ven en desventaja al no tener acceso a un sistema que provea de información fundamental para el crecimiento de dichos negocios.

La aplicación proporcionará al restaurante diversas herramientas para el manejo del inventario, servicio al cliente, ingresos, entre otras que son fundamentales para un correcto manejo del negocio. Tanto la administración como los empleados, podrán hacer uso de esta, por ello, la aplicación debe ser eficiente y sencilla, legible y de gran utilidad para el cliente.

En el presente informe se abordarán las funciones del sistema en los diversos niveles que componen la aplicación, es decir: En un alto nivel de privilegio el administrador (dueño, jefe encargado) podrá gestionar inventario, productos, mesas, acceso a reportes, etc. Luego, con menores facultades, los empleados podrán acceder a ingresar pedidos a las mesas y concretar las ventas.

2. Dominio del Problema.

Desde la integración de *TICS* (Tecnologías de la información) en restaurantes y franquicias de comida rápida, se ha generado una gran brecha en las ganancias entre las grandes, medianas y pequeñas empresas del rubro. Dicha brecha aumenta particularmente en esta área, debido a la importancia que tiene ofrecer un servicio al cliente de manera eficaz, veloz y de alta calidad.

Empresas que actualmente no poseen algún mecanismo de almacenamiento de información, organización y funcionamiento logístico, deben enfrentar diversos problemas relacionados con la atención al cliente, como:

- Largas colas de espera.
- Colapso de espacios de atención (mesas).
- Problemas de concordancia de los pedidos a la cocina.

Por otro lado a nivel administrativo:

- Ausencia de historial de ventas.
- Ausencia de un sistema de inventario.
- Desconocimiento de estadísticas, tales como, balance de ventas, pedidos, productos más vendidos, etc.

Para una empresa en crecimiento es fundamental poseer herramientas que faciliten mencionadas tareas, es por esto que, la aplicación incluye diversas funcionalidades para la gestión, control y manejo de inventario y ventas, es decir, se podrá: agregar, eliminar y modificar *stock* de productos, historial de ventas con fecha y hora exacta, registro de pedidos por mesa, orden en los pedidos de productos mediante una cola, agilización de procesos de elaboración, entre otros. además habrá la posibilidad de generar reportes estadísticos que contribuyan al administrador del negocio a tomar decisiones óptimas.

3. Diseño.

Se visualizaran las soluciones del problema presentado.

3.1. Análisis.

Para poder dar solución a los problemas anteriormente planteados, se han abstraído ciertos elementos fundamentales para el funcionamiento del negocio. A continuación, se procederá a explicarlos:

- **Cajero:** Encargada de gestionar los movimientos del sistema. Esta contendrá un objeto de: ListaIngredientes, ListaAlimentos, ListaBebestibles, ListaMesas y ListaVentas, de esta forma se logra la comunicación entre los diferentes módulos que componen la aplicación.

- **ListaAlimentos:** Contiene una colección de objetos Alimento que puede ofrecer el restaurante (ejemplo: Churrasco, Completo Italiano, Completo, Chacarero, etc), los objetos están mapeados con una clave para una búsqueda/modificación/eliminación inmediata, debido al constante acceso a los elementos, además de la gestión de datos y generar las gráficas de Alimentos más vendidos.

- **ListaIngredientes:** Contiene una colección de objetos Ingrediente, los cuales se encuentran mapeados con una clave para facilitar el acceso inmediato sin necesidad de tenerlos ordenados. También generará las gráficas de los Ingredientes más vendidos.

- **ListaBebestibles:** Contiene una colección de objetos <Bebestible>que ofrece el restaurante. Al igual que clases anteriores, almacenará sus objetos en un *HashMap* debido a la alta concurrencia a los elementos. También se encarga de acceder a los datos y realizar las gráficas de Bebestibles más vendidos.

- **ListaMesas:** Contiene una colección de objetos Mesa, las cuales se encuentran mapeadas con una clave para un acceso inmediato. Se encarga de la comunicación entre los pedidos de los clientes y el sistema,

es decir, Alimentos y Bebestibles. También realiza gráficas con las mesas que más han vendido.

- **Mesa:** Esta clase es la representación de las mesas del restaurante, la cual, contiene dos colecciones (*LinkedList*), una de ellas con referencia a objetos Bebestible e Ingredientes, mediante el almacenamiento de los Identificadores de estos elementos y las veces que ha sido solicitado en la mesa, además del numero, estado y posición en el recinto. También contendrá un objeto de la clase "Lista Ventas" la cual almacena una colección de ventas y balances respectivos de la mesa.

- **ListaVentas:** Contiene una colección de objetos Venta y es la encargada de gestionar las ventas y gráficas asociadas.

- **Inventario:** Es una clase abstracta que cumple la función de generar identificadores específicos para cada producto que forma parte del Inventario del restaurante.

- **Producto:** Clase abstracta que extiende a Inventario agregándole un atributo precio y vendidos, de esta forma un elemento que solo forma del Inventario del restaurante se convierte en un producto vendible y poder gestionar sus valores.

- **Almacenamiento:** Interfaz que es implementada por las clases: ListaVentas, ListaIngredientes, ListaAlimentos, ListaBebestibles y ListaMesas. Dichas clases implementarán los métodos que guardan y cargan los datos en planillas Excel.

- **Ingrediente:** Esta clase es una abstracción de un ingrediente que forma parte de alimentos en el restaurante. Extiende Inventario y posee un objeto Unidad que le permite manejar una unidad de medida en el sistema (Gramo, Unidad o Litros), además almacena la cantidad del ingrediente que hay en el sistema.

- **Bebestible:** Es la abstracción de una bebida/trago que ofrece el restaurante. Esta clase extiende Producto, y forma parte de los elementos

que son vendidos en el restaurante, por lo cual posee un atributo *stock* que hace referencia a la cantidad restante de bebida que posee el restaurante.

- **Venta:** Contiene toda la información de una venta; el numero de la venta, el numero de la mesa en que se efectúo la compra, un *subTotal*, el monto de la propina, la fecha en que se realizo, el método de pago y por último un *ArrayList* de productos vendidos.

- **MetodoPago:** Es una clase *enum* define una colección de de métodos de pago como constantes.

- **Unidad:** Al igual que MetodoPago, Unidad, también es de tipo *enum*, y esta define las formas en que se almacenan los productos e ingredientes (Gramos, Kilogramos, Unidad, etc).

- **TipoInventario:** Una clase *eunm* que define el tipo de inventario de los elementos, ya sean alimentos, ingredientes, bebestibles, productos o stock.

- **Stock:** Esta clase contiene un atributo que almacena la cantidad requerida en un pedido de un producto, así como funcionalidades respectivas a reconocer el tipo de inventario que se está solicitando, es por esto que se extiende de Inventario.

  A continuación se visualizará un diagrama que sintetiza la aplicación y las relaciones que hay entre las clases.

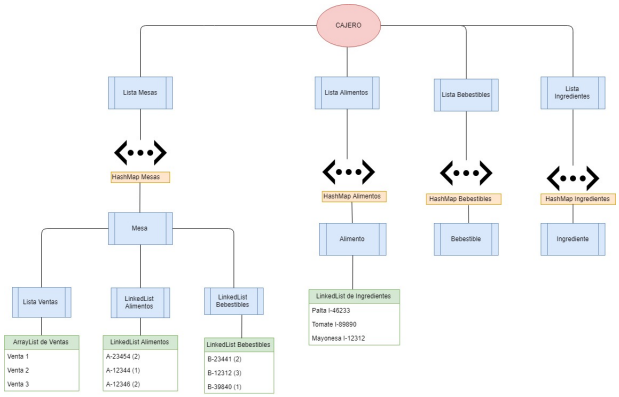


Figura 1: Diagrama de clases.

3.2. Interfaz grafica.

- **Ventanas:** Existe gran variedad de ventanas en la aplicación, para mejorar su uso, las cuales son: VentanaAdministrador, VentanaMenuVentas, VentanaAdministrarAlimentos, VentanaAdministrarBebestibles, VentanaAdministrarIngredientes, VentanaAdministrarMesas. Estas se mostraran más adelante.

3.3. Almacenamiento de Datos.

Los datos en general se guardarán en archivos de planilla, gracias a un método que genera un código que es determinado mediante el tipo de cada elemento y su fecha de ingreso al programa. Este código tiene como función principal ser una clave única que formará a ser el identificador para cada dato. Para poder generar el identificador de un elemento, se utiliza la librería *LocalDateTime*, para obtener la fecha y hora actual al momento del ingreso. Lo que realiza el algoritmo, es crear una cadena de caracteres e inicializarla con el primer carácter correspondiente al tipo de dato (Alimento, Bebestible o Ingrediente).

```
LocalDateTime fecha=LocalDateTime.now();
String codigo;

//Patron de diseno TemplateMethod
switch (identificarse())
```

```
{
     case ALIMENTO:
          codigo="A-";
          break;

     case BEBESTIBLE:
          codigo="B-";
          break;

     default:
          codigo="I-";
          break;
}
```

Luego se concatena el número del día correspondiente a lo que va de año y posteriormente se concatena la hora exacta de ingreso, es decir, hora, minutos y segundos.

```
codigo+=fecha.getDayOfYear();
codigo+=fecha.getHour();
codigo+=fecha.getMinute();
codigo+=fecha.getSecond();
```

3.4. Archivos.

Como se mencionó anteriormente, los datos van a ser almacenados en archivos de planilla, mas en concreto ficheros de formato *OpenDocument Spreadsheet* (.ods). Cada tipo de dato del programa va a poseer su propio formato de almacenamiento de acuerdo a sus características e información que manejan. Además también crea un fichero para manejar las diversas cuentas de usuario del programa, este contiene sus respectivas credenciales, como sus contraseñas encriptadas con un algoritmo de *MD5* por motivos de seguridad.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Nombre	ID	Precio	Vendidos	Cantidad	Ingredientes	Cantidades						
2	Completo	A-109146115	1500	15	4	I-109151125	1	I-109151307	1	I-109151212	1	I-109151609	30
3	Churrasco Italiano	A-109146221	1800	24	3	I-109151125	1	I-109151731	200	I-109151023	200		
4	Churrasco Palta	A-109146003	1150	15	3	I-109151125	1	I-109151458	2	I-109151023	200		
5	Ave Palta	A-125104545	1490	1	2	I-109151125	1	I-109152113	100				
6	Completo Italiano	A-109145847	1000	30	4	I-109151125	1	I-109151023	200	I-109151212	1	I-109151307	1
7	Bistec Pobre	A-109146524	5590	12	4	I-109151910	30	I-109152011	200	I-109151545	130	I-109151731	200
8	Chacarero	A-109146633	5590	15	4	I-109151731	100	I-109151833	220	I-109151212	1	I-109151125	1
9	Chorrillana Pollo	A-109146355	13000	6	4	I-109151910	300	I-109152011	500	I-1091515	200	I-109152113	300
10	Chorrillana Carne	A-109146402	10000	7	4	I-109151910	300	I-109152011	500	I-1091515	200	I-109151731	300
11													
12													

Figura 2: Formato de Archivo Alimentos

	A	B	C	D	E
1	Nombre	ID	Precio	Stock	Vendidos
2	Pepsi 1 Lt	B-109150410	1350	10	2
3	Fanta 250 ml	B-109150311	550	7	5
4	Sprite 250 ml	B-109150224	550	0	6
5	Coca Cola 250 ml	B-109150134	550	5	20
6	Tula 1 Lt	B-125103421	1700	4	5
7	Pepsi 250 ml	B-109150009	550	10	13
8	Pisco 1 Lt	B-125104014	5190	14	0
9	Fanta 1 Lt	B-109150705	1350	10	3
10	Sprite 1 Lt	B-109150629	1350	3	4
11	Coca Cola 1 Lt	B-109150530	1350	26	11
12					

Figura 3: Formato de Archivo Bebestibles

	A	B	C	D
1	Nombre	ID	Stock	Unidad
2	Cebolla	I-109152011	50	Gramos
3	Palta	I-109151023	13000	Gramos
4	Pollo	I-109152113	0	Gramos
5	Vienesa	I-109151307	4	Unidades
6	Lomo	I-109151458	100	Gramos
7	Pan	I-109151125	22	Unidades
8	Tomate	I-109151212	198	Gramos
9	Carne	I-109151731	300	Gramos
10	Poroto Verde	I-109151833	460	Gramos
11	Huevo	I-109151545	450	Unidades
12	Chucrut	I-109151609	40	Gramos
13	Papas	I-109151910	310	Gramos
14	Ketchup	I-125104705	1150	Gramos
15				

Figura 4: Formato de Archivo Ingredientes

	A	B	C
1	Numero	X	Y
2	1	200.0	109.0
3	2	440.0	109.0
4	3	685.0	109.0
5	4	200.0	250.0
6	5	440.0	250.0
7	6	692.0	249.0
8	7	201.0	380.0
9			

Figura 5: Formato de Archivo Mesas

	A	B	C	D
1	Nombre	Usuario	Password	Tipo
2	Administrador	admin	96bdf41419df9e882108bf0e34ba55c2	1
3	Felipe	advanze	926a6d81e4824984aefe76712bdf178a	2
4	Arturo	xtrem	b8df5557e519a03eef559c5d18432391	2
5	Bastian	zero	abee5e6ed7c532d600efad4118bc267e	2
6				

Figura 6: Formato de Archivo Usuarios

3.5. Patron de diseño modelo vista controlador.

El modelo-vista-controlador es uno de los patrones que mas se usan en la creación de programas a grandes y a menores escalas, siempre y cuando se utilice una interfaz.

El modelo es la parte principal de la aplicación es lo que contiene toda la lógica relacionada con la aplicación, el cerebro del programa, es el encargado de conseguir la información.

La vista corresponde a todo lo que el usuario percibe visualmente al interactuar con una aplicación, la principal función es ser una capa de todo lo que el usuario desea obtener, la idea principal es presentar de forma amigable la información al usuario.

Así es como el modelo no tiene que estar relacionado con la vista El controlador es el intermediario entre la vista y el modelo su responsabilidad es recibir solicitudes de la vista convertirla en algo que el modelo pueda procesar y viceversa.

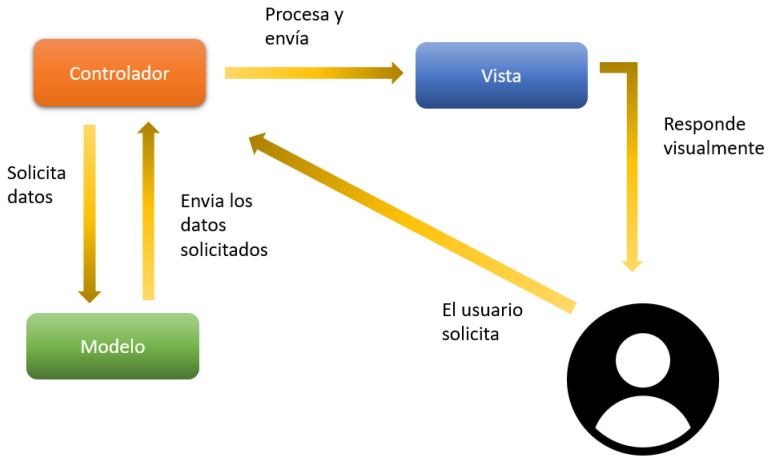


Figura 7: Modelo vista controlador

3.6. Patron de diseño Template Method.

El patrón de diseño *Template Method* es uno de patrones caracterizado por manejar el comportamiento entre la interacción de clases y objetos, lo que propone este patrón es aislar el comportamiento que comparten las clases, para que después de una clase abstracta extender ese comportamiento a otras.

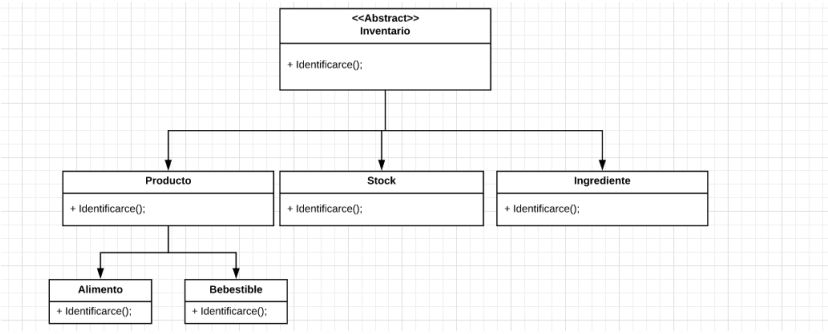


Figura 8: Patrón de diseño *Templathe Method*

3.7. Excepciones.

Las excepciones son errores en tiempo de ejecución de un programa, pueden ser datos específicos que se le piden al usuario y este por error entrego datos inesperados. Para esta aplicación se necesitara el uso de 3 excepciones, la primera se utilizará en caso de que se le entregue una id no valida de un producto del inventario , la segunda sera cuando se entregue un precio de un producto invalido, el cual se solo aceptara números no caracteres y por ultimo una excepción que se utilizara en el *stock*, este funcionara cuando el *stock* quede en 0.

4. Diseño Interfaz.

Se mostrará como está realizada la aplicación.

4.1. Diseño UML.

Es la visualización del proyecto en UML para facilitar su comprensión, en él se encuentra todas las clases, con sus atributos, métodos y sus respectivos niveles de visibilidad, para mayor apreciación ver anexo.

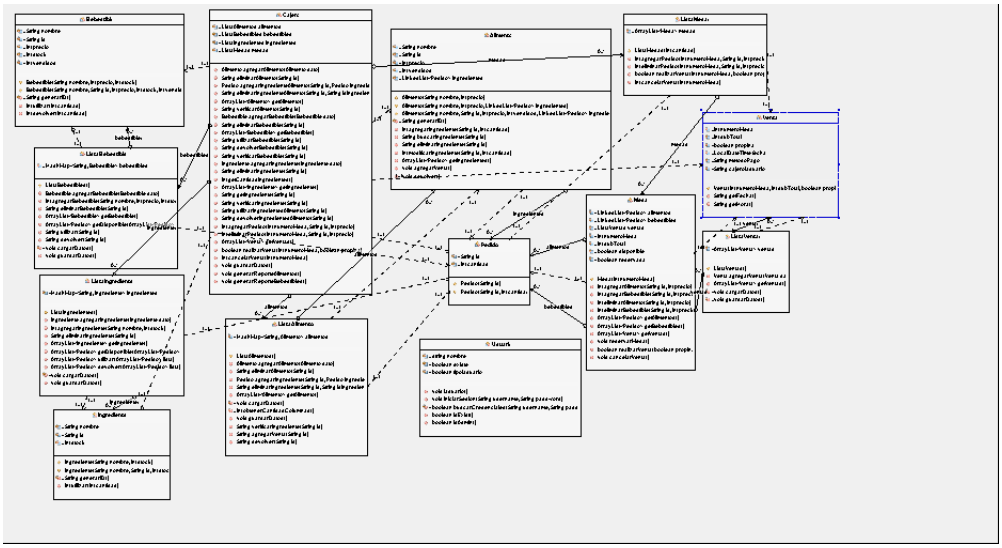


Figura 9: UML

4.2. Diagrama de interfaz.

La interfaz Almacenamiento se utilizará principalmente para el inventario del restaurante, en el cual se podrá generar reportes, guardar datos y cargar datos de las diferentes listas de alimentos, bebestibles e ingredientes.

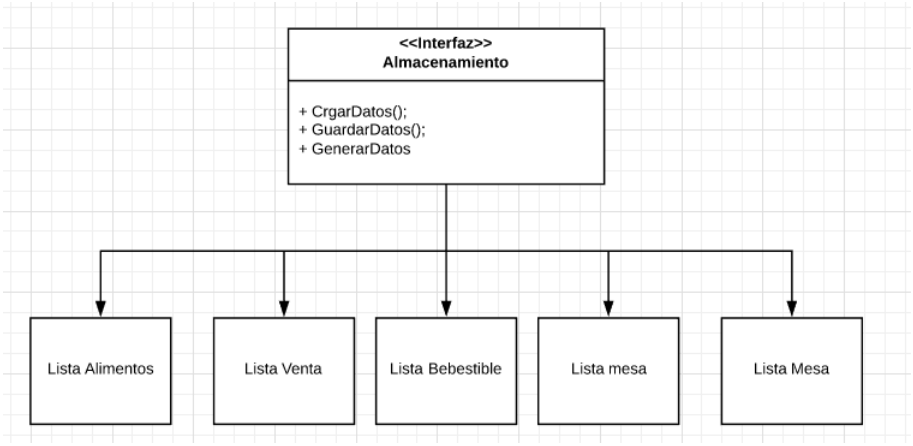


Figura 10: Diagrama Interfaz almacenamiento

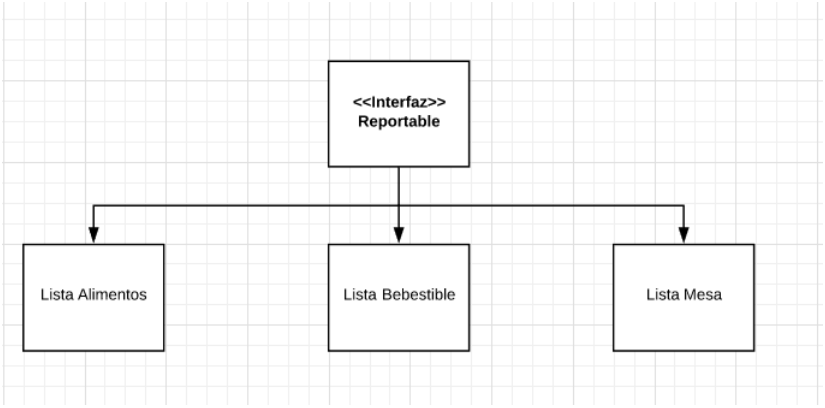


Figura 11: Diagrama Interfaz reportable

4.3. Diagrama de herencia.

La clase Inventario es de tipo abstracta y es el padre de todas las subclases que representan un tipo de dato para llevar a cabo el inventario del restaurante.

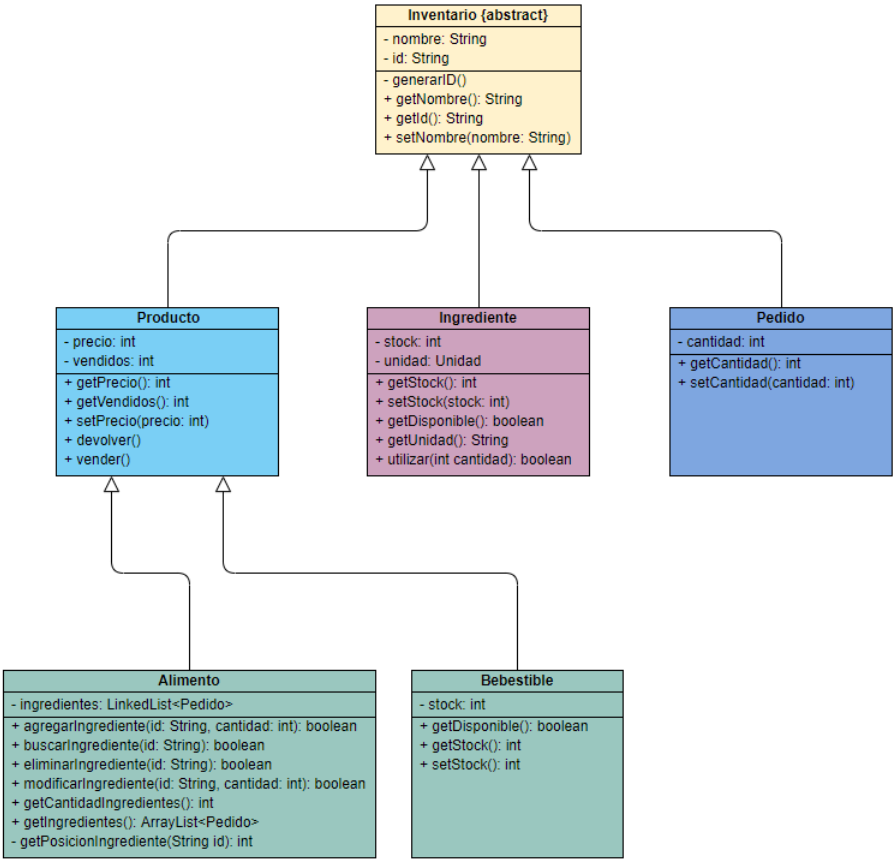


Figura 12: Diagrama de herencia

4.4. Ventana de Inicio.

En el menú de inicio, pedirá que se ingrese un usuario y contraseña. Dependiendo el usuario, se ingresara en modo administrador o modo usuario normal (cajero), estos usuarios van a estar guardados en los archivos de la aplicación, los cuales se comparan con los ingresados, para ver qué privilegios a que ventana pasarán.

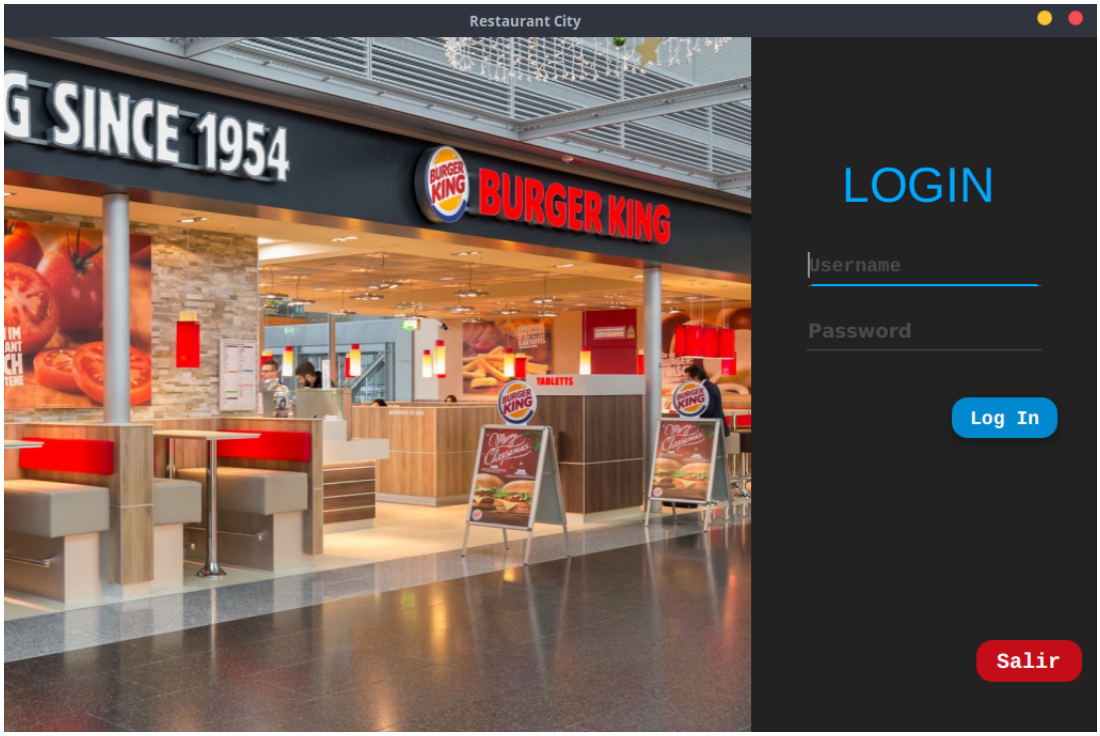


Figura 13: Ventana de Inicio

4.5. Ventana de Menú Administrador.

El usuario administrador va a poder gestionar los datos del restaurante, además de poder eliminar, modificar y agregar alimentos, ingredientes, bebestibles y mostrar reportes de alimentos y bebestibles más vendidos del restaurante.

Figura 14: Ventana de Administrador

4.6. Ventana de Administrar Alimentos.

En la ventana agregar alimento, se mostrará los alimentos que contiene el restaurante, en el cual se podrá agregar alimentos, ingresando el nombre del alimento y el precio, el código y la cantidad de ingredientes no son necesarios, ya que la aplicación los genera automáticamente. Al agregar un alimento, se tendrá que modificar, en el cual se abrirá la próxima ventana de modificar, en las cuales se tendrá que agregar los ingredientes que lleva el alimento deseado, si el ingrediente ya existe este no se agregará.

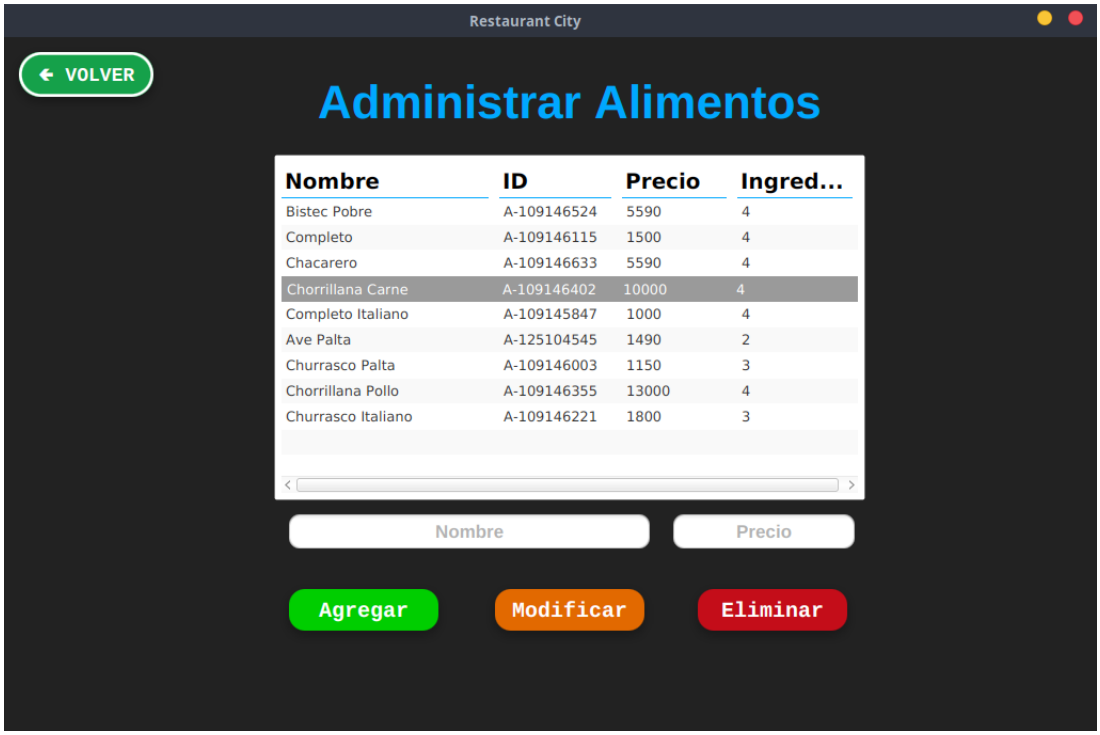


Figura 15: Ventana de Administrar Alimentos

4.7. Ventana de Administrar Bebestibles.

Al ingresar en los bebestibles, se mostrará visualmente una lista con todos los bebestibles que se han agregado o se encontraran en el restaurante, los cuales dentro de esta ventana se podrá agregar y eliminar un bebestible, esto

se hará rellenando los campos del nombre, precio y stock, además el id se genera automáticamente.

Figura 16: Ventana de Administrar Bebestibles

4.8. Ventana de Administrar Ingredientes.

Al administrar ingredientes, se abrirá una ventana en la cual se permite agregar o eliminar ingredientes, estos se agregaran ingresando el nombre y el *stock*, el *stock* será la cantidad en gramo, si es un producto no contable. Al tratar de eliminar un ingrediente, y si este se encuentra en algún alimento, la aplicación no deja eliminarlo y muestra los alimentos que contiene este ingrediente.

Figura 17: Ventana de Administrar Ingredientes

4.9. Ventana de Administrar Mesas.

El Administrador podrá agregar, modificar y eliminar mesas, las cuales son representadas por un cuadrado y su respectivo numero.

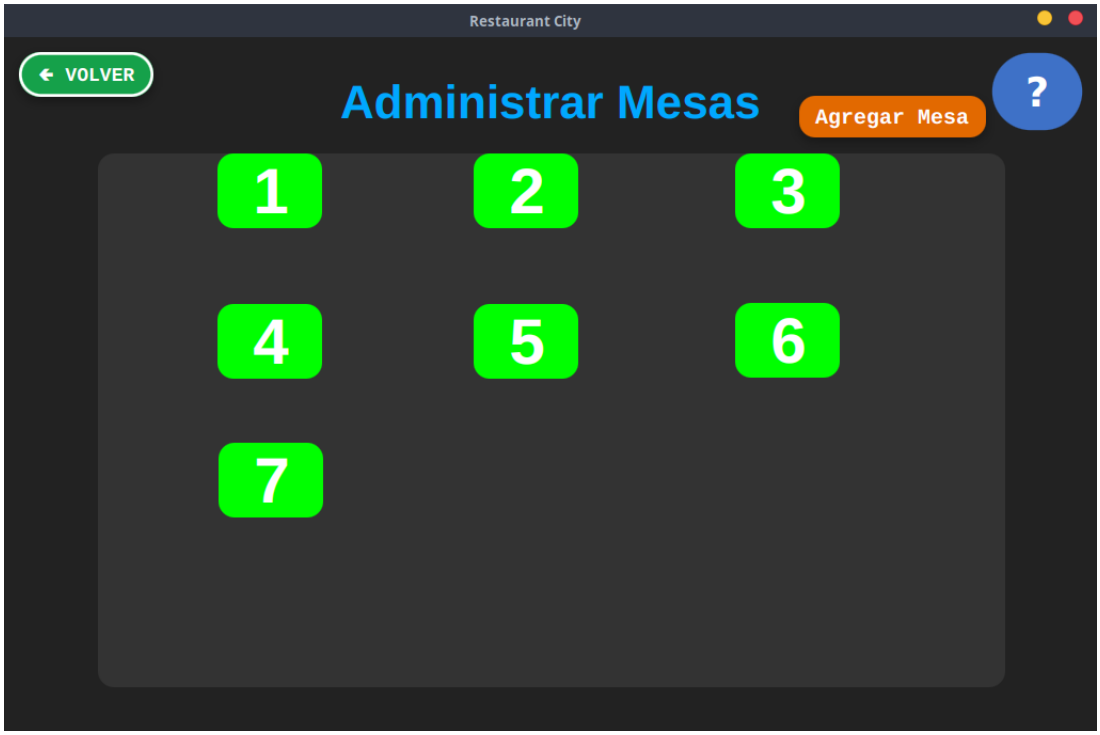


Figura 18: Ventana de Administrar Mesas

4.10. Ventana de Reportes

En la ventana reportes el usuario tendrá 4 opciones para genera un reporte de ventas o inventario, al seleccionar un reporte se mostrara un gráfico acerca de los productos que hay en el almacenamiento de l restaurantes y en del reporte ventas , se podrán aprecia cuales fueron los alimentos mas vendidos del mes.

Figura 19: Ventana reportes.

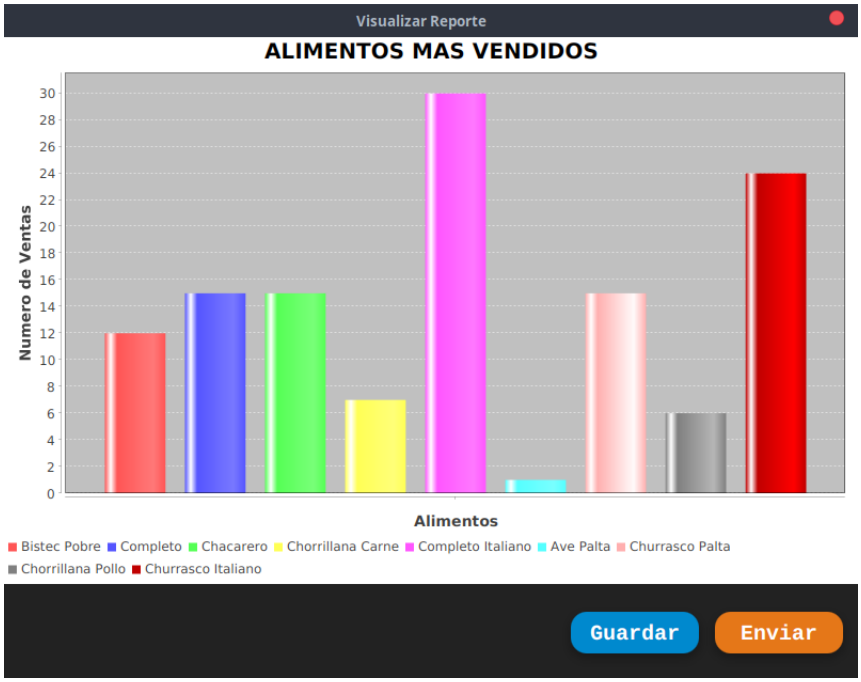


Figura 20: Reportes de alimentos.

4.11. Ventana de Menú Ventas.

El usuario cajero al ingresar su cuenta, se encontrará en una ventana con varios botones rectangulares, los que van a simbolizar las mesas del restaurante y su número id indicado en él. El cajero al hacer click en una mesa, se abrirá una ventana venta de la mesa especificada. En la cual se podrá agregar y eliminar, tanto alimentos como bebestibles, que formarán como pedido de la mesa. Al intentar agregar un bebestible o un alimento sin *stock*, es decir, que algunos de sus ingredientes no posea el *stock* suficiente para ser preparado, aparecerá una ventana emergente dando aviso al error e impidiendo la agregación de este. Como funcionalidad final nos dará la posibilidad de cancelar o realizar la venta, cabe destacar que además se puede trabajar con otras mesas, sin perder la información del pedido actual, para así finalizar la venta cuando el usuario estime. Al finalizar la venta, el programa solicitará en una ventana emergente si desea agregar propina sugerida del 10

Figura 21: Ventana de Menú Ventas

4.12. Ventana de Atender Mesa.

Al Atender mesa se mostrara el contenido de la mesa de la cual se quiera hacer una venta, en la cual mostrara los alimentos y bebestibles que se le podrá hacer a la mesa, además se podrá vender la venta.

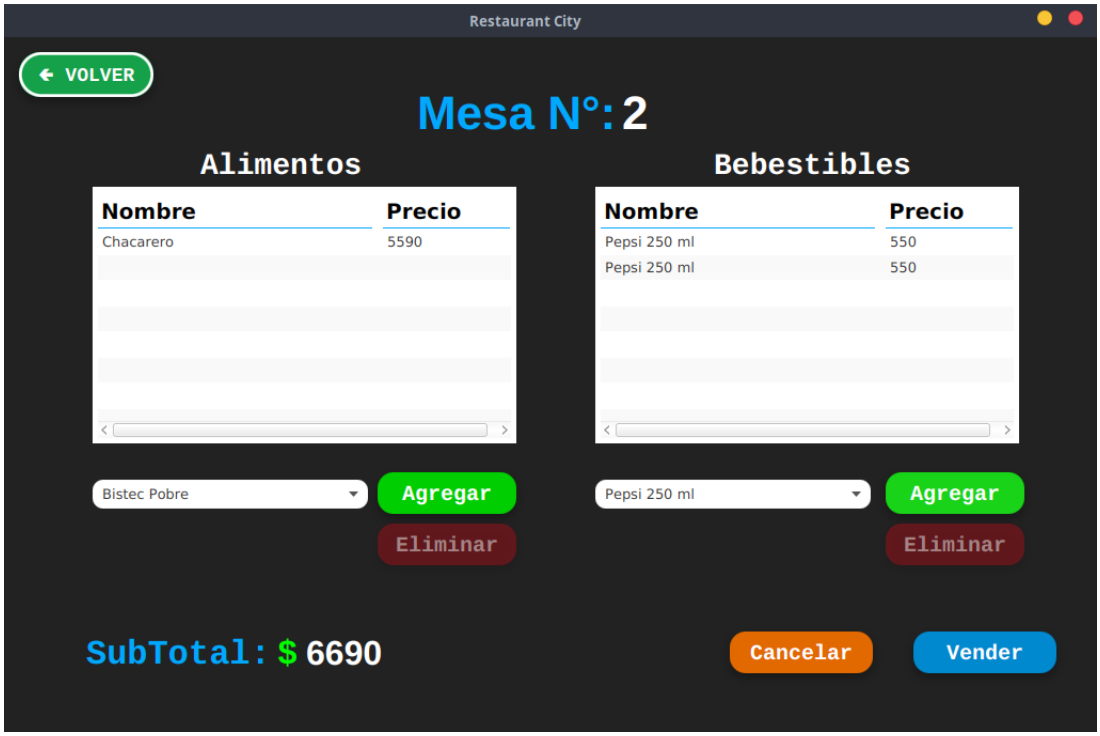


Figura 22: Ventana de Atender Mesa

5. Planificacion.

Se mostrara el trabajo estimado de los integrantes del grupo.

5.1. Carta Gantt Grupal.

Se mostrara el trabajo estimado del grupo.

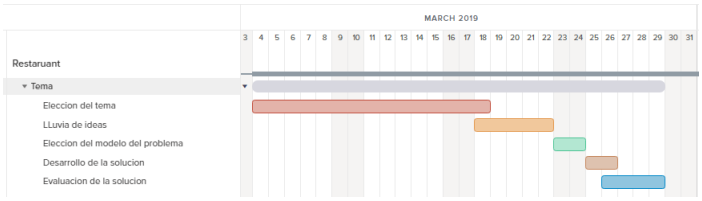


Figura 23: Carta Gantt Marzo.

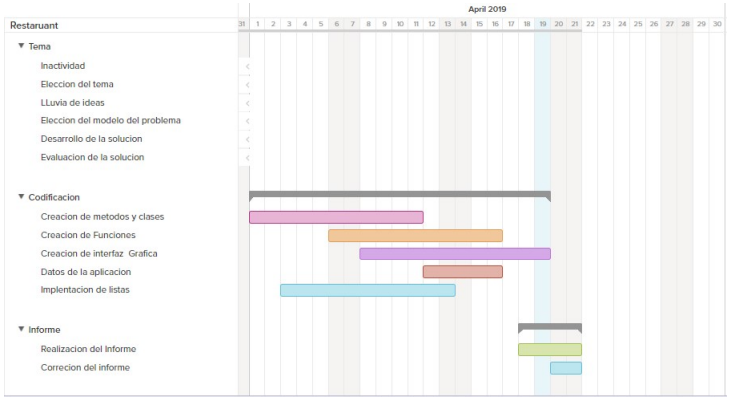


Figura 24: Carta Gantt Abril.

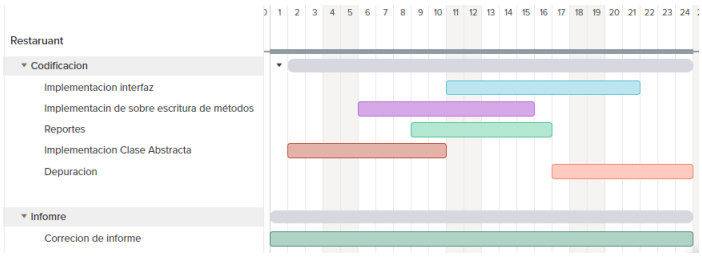


Figura 25: Carta Gantt Mayo.

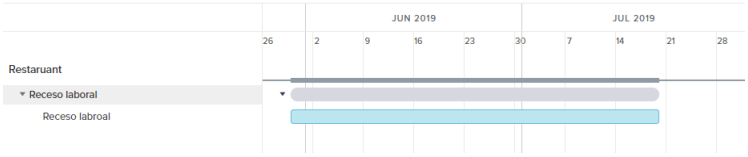


Figura 26: Carta Gantt junio-julio.

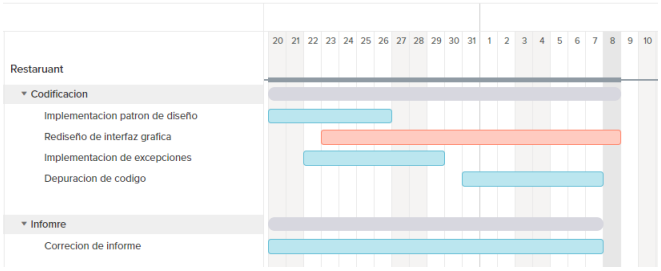


Figura 27: Carta Gantt Junio-Agosto.

5.2. Carta Gantt Felipe.

Carta de trabajo de felipe Guajardo

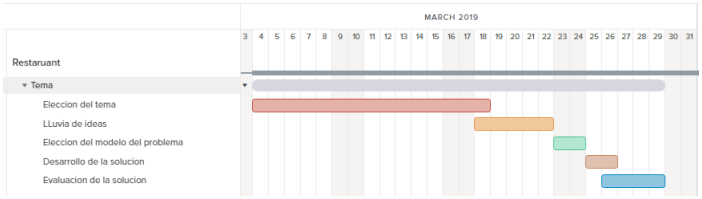


Figura 28: Carta Gantt Marzo Felipe Guajardo.

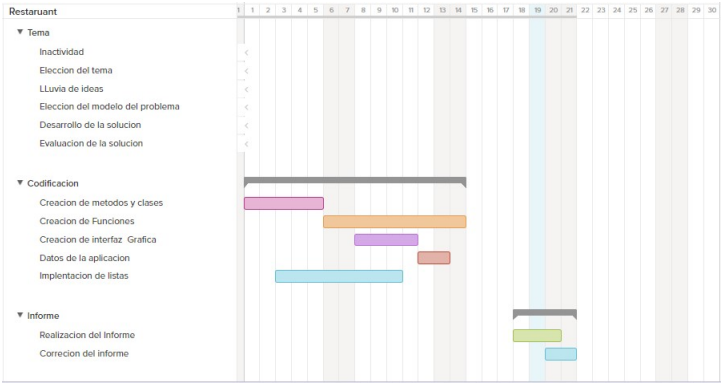


Figura 29: Carta Gantt Abril Felipe Guajardo.

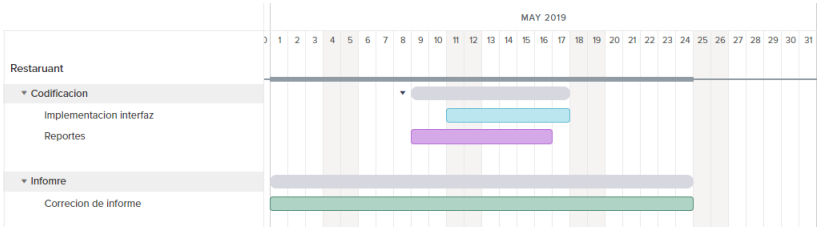


Figura 30: Carta Gantt Mayo Felipe Guajardo.

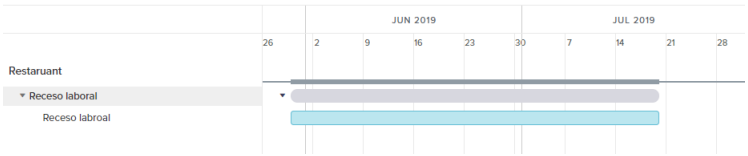


Figura 31: Carta Gantt junio-julio Felipe Guajardo.

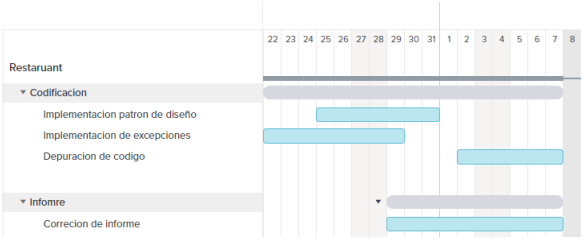


Figura 32: Carta Gantt Julio-Agosto Felipe Guajardo.

5.3. Carta Gantt Bastian.

Carta de trabajo de Bastian Ramirez

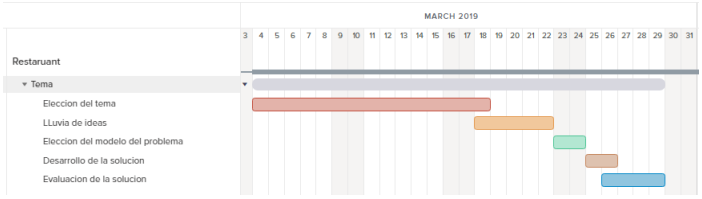


Figura 33: Carta Gantt Marzo Bastian Ramirez.

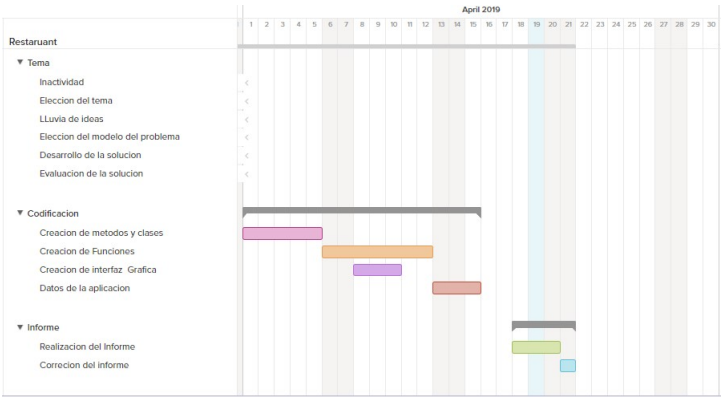


Figura 34: Carta Gantt Abril Bastian Ramirez.

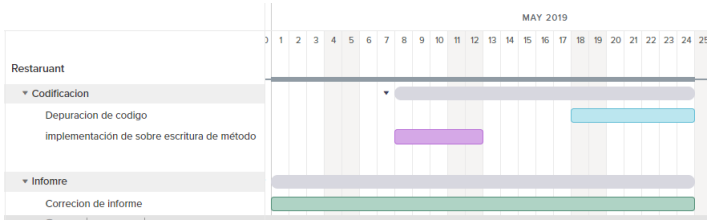


Figura 35: Carta Gantt Mayo Bastian Ramirez.

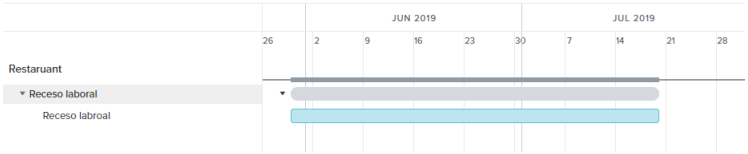


Figura 36: Carta Gantt junio-julio Bastian Ramirez.

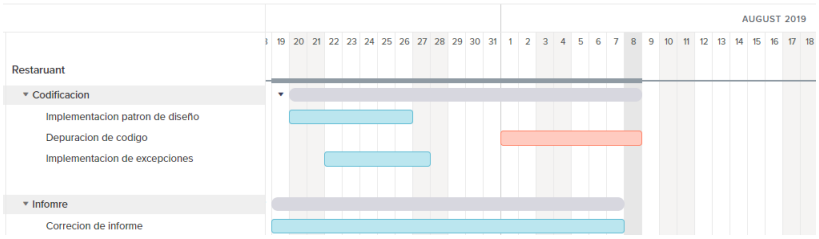


Figura 37: Carta Gantt Julio-Agosto Bastian Ramirez.

5.4. Carta Gantt Jose Arturo.

Carta de trabajo de Jose Arturo vergara

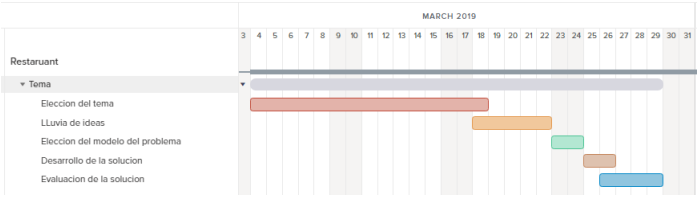


Figura 38: Carta Gantt Marzo Jose Arturo Vergara.

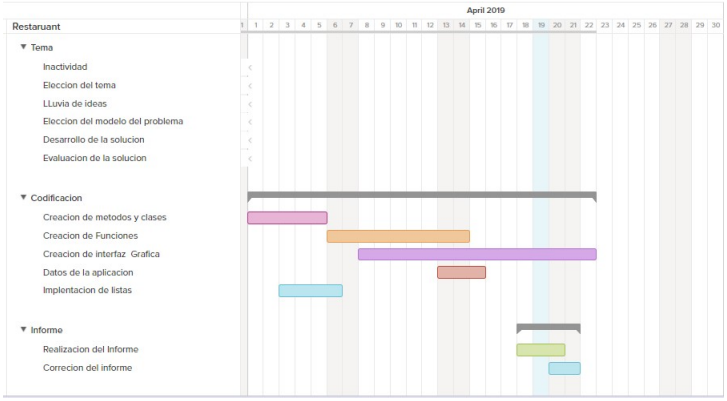


Figura 39: Carta Gantt Abril Jose Arturo Vergara.

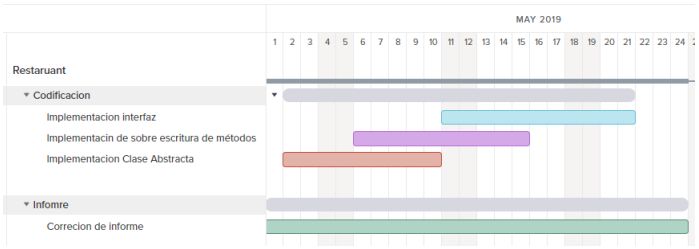


Figura 40: Carta Gantt Mayo Jose Arturo Vergara.

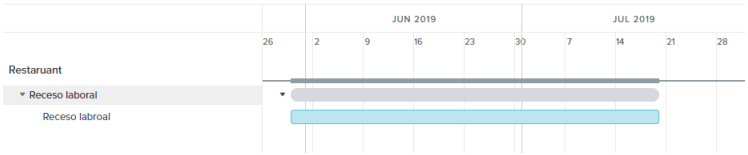


Figura 41: Carta Gantt junio-julio Jose Arturo Vergara.

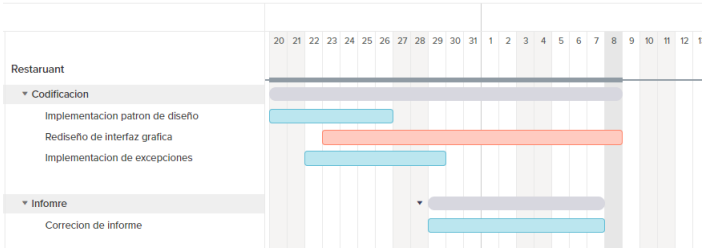


Figura 42: Carta Gantt Julio-Agosto Jose Arturo Vergara.

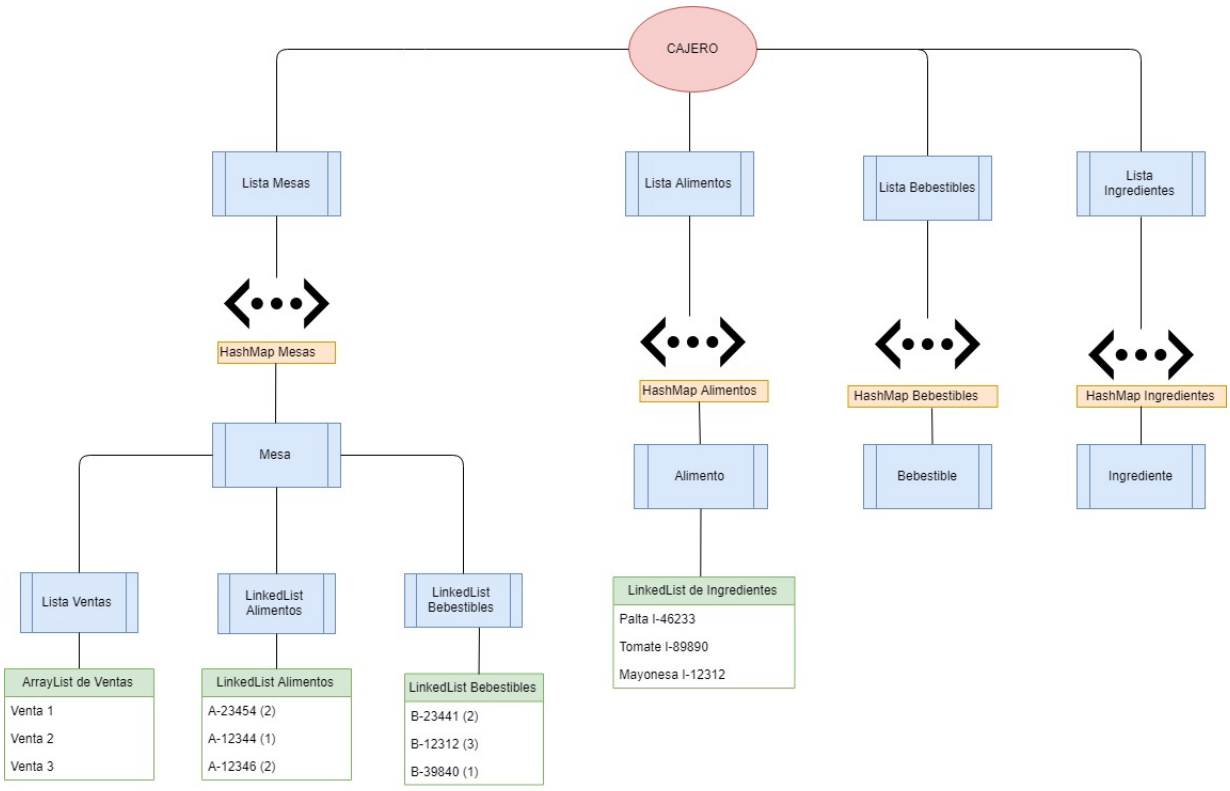
6. Conclusión.

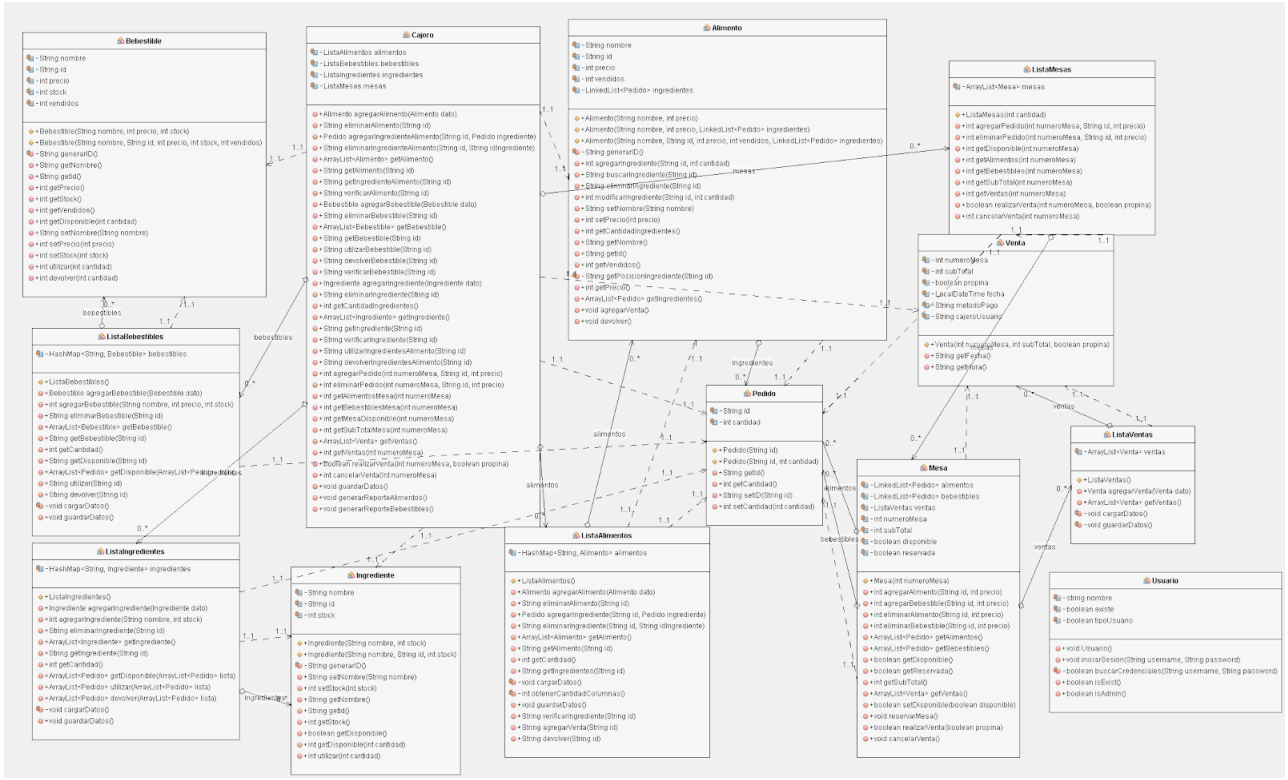
En relación a lo anteriormente expuesto se ha logrado identificar una gran diversidad de problemas que diariamente enfrenta el rubro relacionado a la comida rápida y restaurantes, principalmente en la atención al cliente. Es fundamental conocer el contexto en su totalidad para así visualizar los factores que generan problemas en el correcto funcionamiento del rubro cliente, para realizar un modelo que se acomode a las necesidades de este último. Tras un largo proceso de análisis y bosquejo de los factores que suscitaban los problemas, se logra desarrollar una aplicación que permite dar un gran paso a muchos medianos y pequeños empresarios que no pueden competir con el capital de grandes transnacionales y cadenas de comida.

Así como se puede profundizar en uno de los módulos de la aplicación, también se puede realizar un sistema íntegro, que cumpla diversas funciones, como la propuesta en el presente informe, en esta ocasión, la aplicación tiene un enfoque administrativo y logístico, por lo cual, se ha empleado mayor tiempo en el desarrollo de dichos módulos, llevando a cabo funcionalidades para dos tipos de usuario de la aplicación, en la cual, se logró generar una interfaz sencilla, fácil de gestionar y entender, eficiente y que fundamentalmente cumpla con los requerimientos del(los) cliente(s).

Restaurant City representa un gran desafío a la hora de definir los principales problemas que se desean solucionar en el nivel logístico de un restaurante. En esta ocasión se aprovechó el paradigma de orientación a objetos para realizar un sistema centrado en la administración del negocio. Además realizar una aplicación eficiente, fácil de aprender y utilizar en el día a día, y enfocada en medianas y pequeñas empresas, es el objetivo fundamental del presente proyecto.

7. Anexo.


CAJERO
Lista Mesas
Lista Alimentos
Lista Bebestibles
Lista Ingredientes
HashMap Mesas
HashMap Alimentos
HashMap Bebestibles
HashMap Ingredientes
Mesa
Alimento
Bebestible
Ingrediente
Lista Ventas
LinkedList Alimentos
LinkedList Bebestibles
LinkedList de Ingredientes
Palta I-46233
Tomate I-89890
Mayonesa I-12312
ArrayList de Ventas
Venta 1
Venta 2
Venta 3
LinkedList Alimentos
A-23454 (2)
A-12344 (1)
A-12346 (2)
LinkedList Bebestibles
B-23441 (2)
B-12312 (3)
B-39840 (1)


Bebestible
- String nombre
- String id
- int precio
- int stock
- int vendidos
+ Bebestible(String nombre, int precio, int stock)
+ Bebestible(String nombre, String id, int precio, int stock, int vendidos)
- String generarID()
+ String getNombre()
+ String getId()
+ int getPrecio()
+ int getStock()
+ int getVendidos()
+ int getDisponible(int cantidad)
+ String setNombre(String nombre)
+ int setPrecio(int precio)
+ int setStock(int stock)
+ int utilizar(int cantidad)
+ int devolver(int cantidad)
Cajero
- ListaAlimentos alimentos
- ListaBebestibles bebestibles
- ListaIngredientes ingredientes
- ListaMesas mesas
+ Alimento agregarAlimento(Alimento dato)
+ String eliminarAlimento(String id)
+ Pedido agregaringredienteAlimento(String id, Pedido ingrediente)
+ String eliminaringredienteAlimento(String id, String idingrediente)
+ ArrayList<Alimento> getAlimento()
+ String getAlimento(String id)
+ String getIngredienteAlimento(String id)
+ String verificarAlimento(String id)
+ Bebestible agregarBebestible(Bebestible dato)
+ String eliminarBebestible(String id)
+ ArrayList<Bebestible> getBebestible()
+ String getBebestible(String id)
+ String utilizarBebestible(String id)
+ String devolverBebestible(String id)
+ String verificarBebestible(String id)
+ Ingrediente agregaringrediente(Ingrediente dato)
+ String eliminaringrediente(String id)
+ int getCantidadingredientes()
+ ArrayList<Ingrediente> getIngrediente()
+ String getIngrediente(String id)
+ String verificarIngrediente(String id)
+ String utilizaringredientesAlimento(String id)
+ String devolveringredientesAlimento(String id)
+ int agregarPedido(int numeroMesa, String id, int precio)
+ int eliminarPedido(int numeroMesa, String id, int precio)
+ int getAlimentosMesa(int numeroMesa)
+ int getBebestiblesMesa(int numeroMesa)
+ int getMesaDisponible(int numeroMesa)
+ int getSubTotalMesa(int numeroMesa)
+ ArrayList<Venta> getVentas()
+ int getVentas(int numeroMesa)
+ boolean realizarVenta(int numeroMesa, boolean propina)
+ int cancelarVenta(int numeroMesa)
+ void guardarDatos()
+ void generarReporteAlimentos()
+ void generarReporteBebestibles()
Alimento
- String nombre
- String id
- int precio
- int vendidos
- LinkedList<Pedido> ingredientes
+ Alimento(String nombre, int precio)
+ Alimento(String nombre, int precio, LinkedList<Pedido> ingredientes)
+ Alimento(String nombre, String id, int precio, int vendidos, LinkedList<Pedido> ingredientes)
- String generarID()
+ int agregaringrediente(String id, int cantidad)
+ String buscaringrediente(String id)
+ String eliminaringrediente(String id)
+ int modificaringrediente(String id, int cantidad)
+ String setNombre(String nombre)
+ int setPrecio(int precio)
+ int getCantidadingredientes()
+ String getNombre()
+ String getId()
+ int getVendidos()
- String getPosicionIngrediente(String id)
+ int getPrecio()
+ ArrayList<Pedido> getIngredientes()
+ void agregarVenta()
+ void devolver()
ListaMesas
- ArrayList<Mesa> mesas
+ ListaMesas(int cantidad)
+ int agregarPedido(int numeroMesa, String id, int precio)
+ int eliminarPedido(int numeroMesa, String id, int precio)
+ int getDisponible(int numeroMesa)
+ int getAlimentos(int numeroMesa)
+ int getBebestibles(int numeroMesa)
+ int getSubTotal(int numeroMesa)
+ int getVentas(int numeroMesa)
+ boolean realizarVenta(int numeroMesa, boolean propina)
+ int cancelarVenta(int numeroMesa)
Venta
- int numeroMesa
- int subTotal
- boolean propina
- LocalDateTime fecha
- String metodoPago
- String cajeroUsuario
+ Venta(int numeroMesa, int subTotal, boolean propina)
+ String getPayPal()
+ String getEfectivo()
ListaBebestibles
- HashMap<String, Bebestible> bebestibles
+ ListaBebestibles()
+ Bebestible agregarBebestible(Bebestible dato)
+ int agregarBebestible(String nombre, int precio, int stock)
+ String eliminarBebestible(String id)
+ ArrayList<Bebestible> getBebestible()
+ String getBebestible(String id)
+ int getCantidad()
+ String getDisponible(String id)
+ ArrayList<Pedido> getDisponible(ArrayList<Pedido> lista)
+ String utilizar(String id)
+ String devolver(String id)
- void cargarDatos()
+ void guardarDatos()
Pedido
- String id
- int cantidad
+ Pedido(String id)
+ Pedido(String id, int cantidad)
+ String getId()
+ int getCantidad()
+ String setID(String id)
+ int setCantidad(int cantidad)
ListaVentas
- ArrayList<Venta> ventas
+ ListaVentas()
+ Venta agregarVenta(Venta dato)
+ ArrayList<Venta> getVentas()
- void cargarDatos()
+ void guardarDatos()
Mesa
- LinkedList<Pedido> alimentos
- LinkedList<Pedido> bebestibles
- ListaVentas ventas
- int numeroMesa
- int subTotal
- boolean disponible
- boolean reservada
+ Mesa(int numeroMesa)
+ int agregarAlimento(String id, int precio)
+ int agregarBebestible(String id, int precio)
+ int eliminarAlimento(String id, int precio)
+ int eliminarBebestible(String id, int precio)
+ ArrayList<Pedido> getAlimentos()
+ ArrayList<Pedido> getBebestibles()
+ boolean getDisponible()
+ boolean getReservada()
+ int getSubTotal()
+ ArrayList<Venta> getVentas()
+ boolean setDisponible(boolean disponible)
+ void reservarMesa()
+ boolean realizarVenta(boolean propina)
+ void cancelarVenta()
ListaIngredientes
- HashMap<String, Ingrediente> ingredientes
+ ListaIngredientes()
+ Ingrediente agregarIngrediente(Ingrediente dato)
+ int agregaringrediente(String nombre, int stock)
+ String eliminaringrediente(String id)
+ int getCantidad()
+ ArrayList<Ingrediente> getIngrediente()
+ String getIngrediente(String id)
+ ArrayList<Pedido> getDisponible(ArrayList<Pedido> lista)
+ ArrayList<Pedido> utilizar(ArrayList<Pedido> lista)
+ ArrayList<Pedido> devolver(ArrayList<Pedido> lista)
- void cargarDatos()
+ void guardarDatos()
Ingrediente
- String nombre
- String id
- int stock
+ Ingrediente(String nombre, int stock)
+ Ingrediente(String nombre, String id, int stock)
- String generarID()
+ String setNombre(String nombre)
+ int setStock(int stock)
+ String getNombre()
+ String getId()
+ int getStock()
+ boolean getDisponible()
+ int getDisponible(int cantidad)
+ int utilizar(int cantidad)
ListaAlimentos
- HashMap<String, Alimento> alimentos
+ ListaAlimentos()
+ Alimento agregarAlimento(Alimento dato)
+ String eliminarAlimento(String id)
+ Pedido agregaringrediente(String id, Pedido ingrediente)
+ String eliminaringrediente(String id, String idingrediente)
+ ArrayList<Alimento> getAlimento()
+ String getAlimento(String id)
+ int getCantidad()
+ String getIngredientes(String id)
- void cargarDatos()
- int obtenerCantidadColumnas()
+ void guardarDatos()
+ String verificaringrediente(String id)
+ String agregarVenta(String id)
+ String devolver(String id)
Usuario
- string nombre
- boolean existe
- boolean tipoUsuario
+ void Usuario()
+ void iniciarSesion(String username, String password)
- boolean buscarCredenciales(String username, String password)
+ boolean isExist()
+ boolean isAdmin()
bebestibles
alimentos
ingredientes
mesas
ventas

Bibliografía

[1] Michael K. Borregaard. *Different results for loops with decrement and increment*. Stackoverflow, 2017. Disponible vía web en `https://stackoverflow.com/a/46211563`