Universidad Del Valle De Guatemala

Programación orientada a objetos

Sección 11

PROYECTO: Final

Esteban Cabrera Arévalo, 17781

Miguel Valle Quinto, 17102

Rodrigo Morales Castellanos, 17027

* Retrospectiva
  + La situación problemática seleccionada fue:

Proponer una técnica que permita hacer las crepas y/o panes más rápido en RED

Ya que se pretende mejorar la atención y eficiencia en los procesos para que exista una mejor atención hacia todos los estudiantes.

* + El brief elaborado:

Proporcionar una mejor atención y un servicio más rápido en la Cafetería RED a los alumnos y profesores de la Universidad del Valle de Guatemala

* Necesidades:

-Ahorrar tiempo

-Espacio para sentarse y comer a la hora de los recesos

-Velocidad en fabricación de la comida

-Precios bajos

-Relación precio-sabor

-Servicio amigable

-Variedad de comida

-Evitar hacer cola

* Oportunidades:

-Mejorar el servicio y tomar en cuenta la opinión de los clientes.

-Variar un poco el menú para que los clientes prueben nuevas cosas

-Preguntar cuáles son las comidas que desean y mejorar la calidad y el precio de los productos.

-Incrementar el espacio y cantidad de mesas

-Tratar de incrementar el personal

-Autoservicio en que la persona haga su propia crepa

-Ampliar la cocina para más maquinaria para hacer las crepas

-Buscar manera de preparar la comida con anticipación sin que se arruine

-Poner promociones semanales

-Un carrito especializado en crepas para que no se haga cola

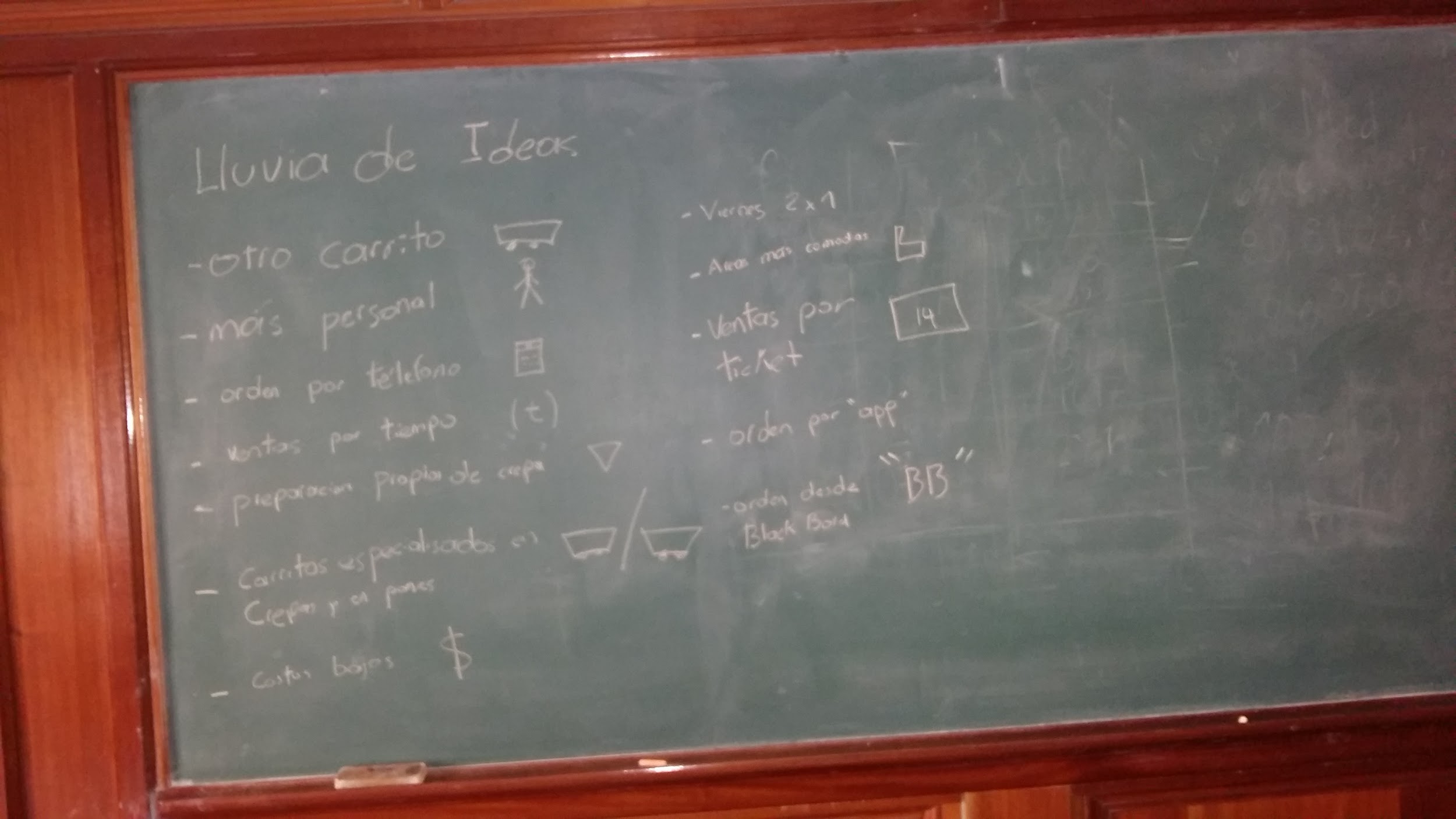
-Pedidos anticipados

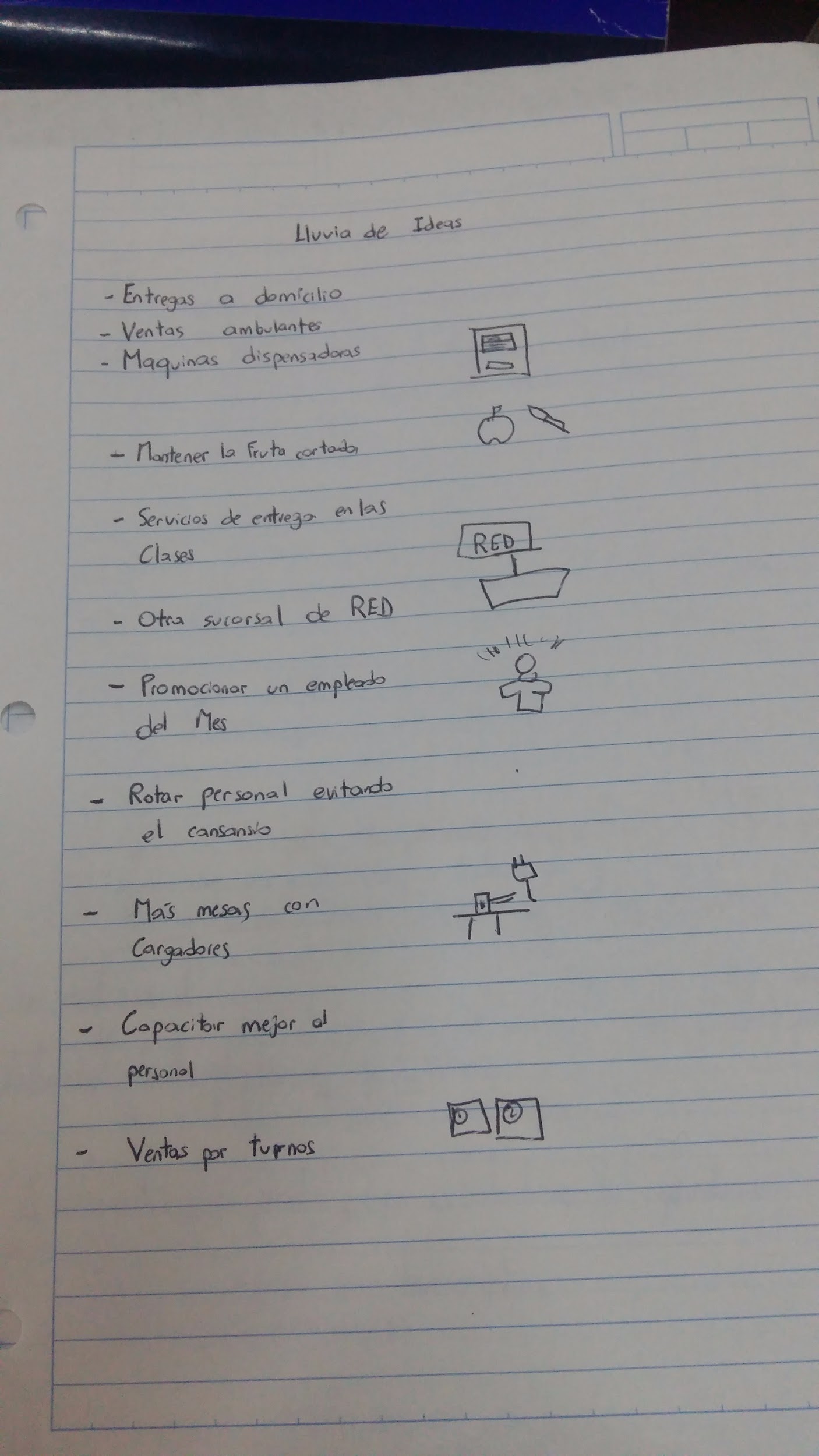
**Listado de preguntas**

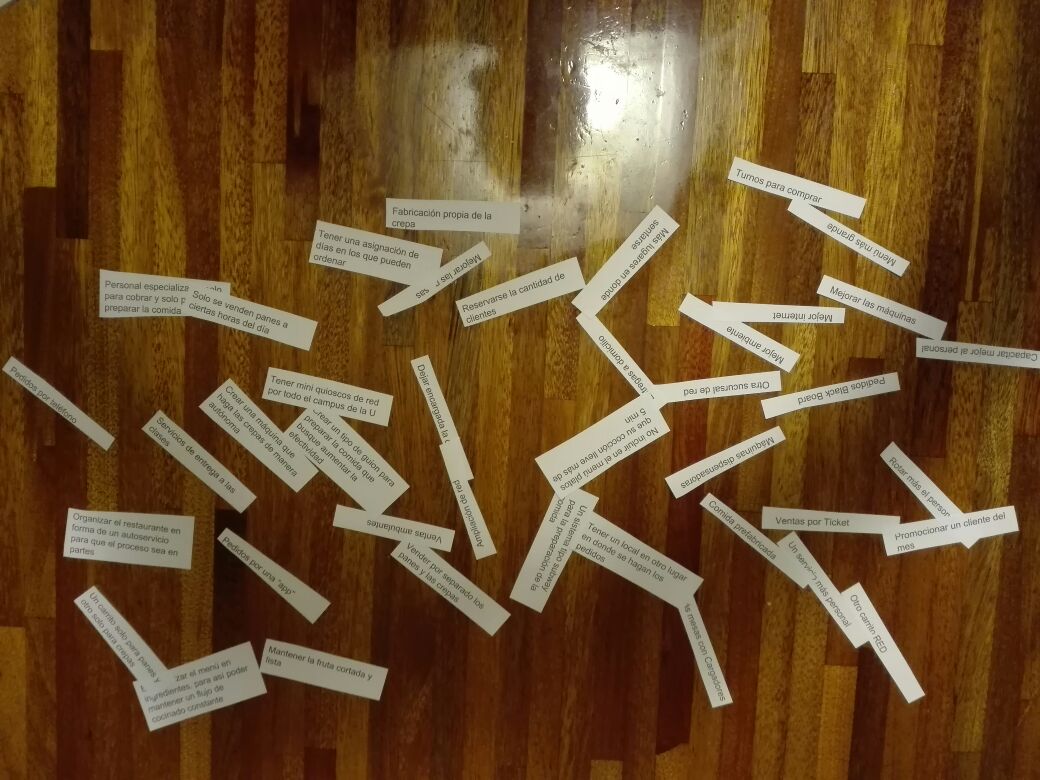
1. ¿Cómo podríamos volver RED más eficiente ?
2. ¿Como tener una mejor estadía en RED al esperar ?
3. ¿Cómo podríamos reducir el tiempo de espera en RED ?
4. ¿Como hacer para que RED aplique estos cambios?
5. ¿Cómo generar cambios que están dentro del presupuesto de RED?

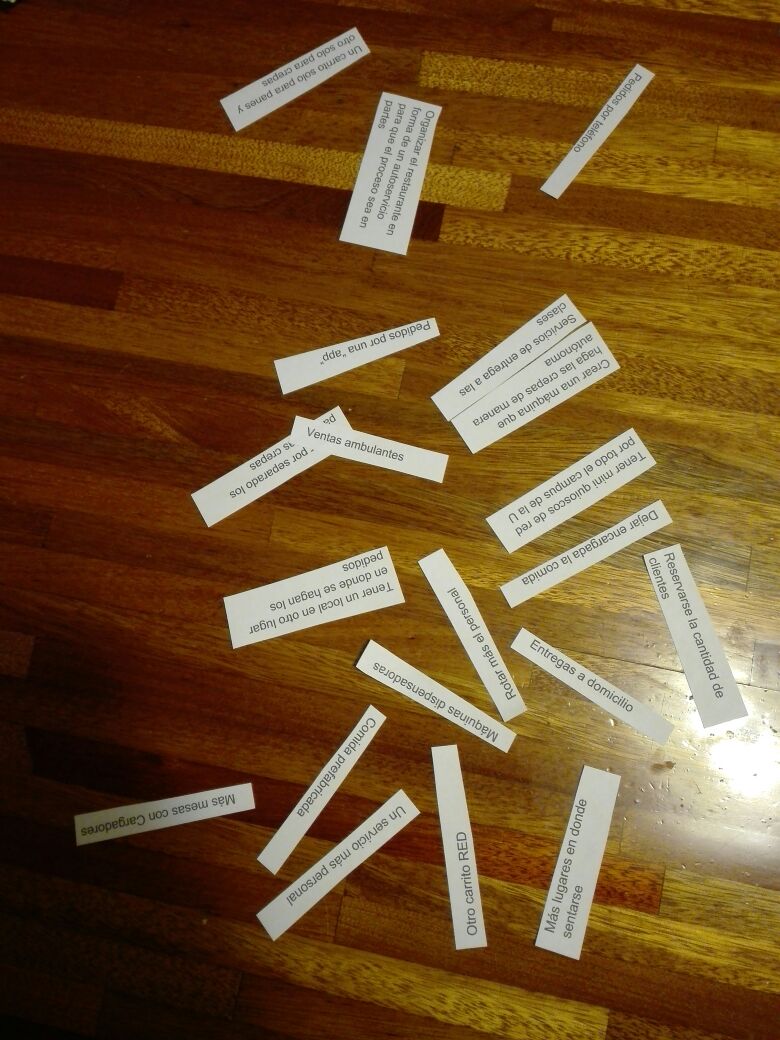
**Lluvia de Ideas**

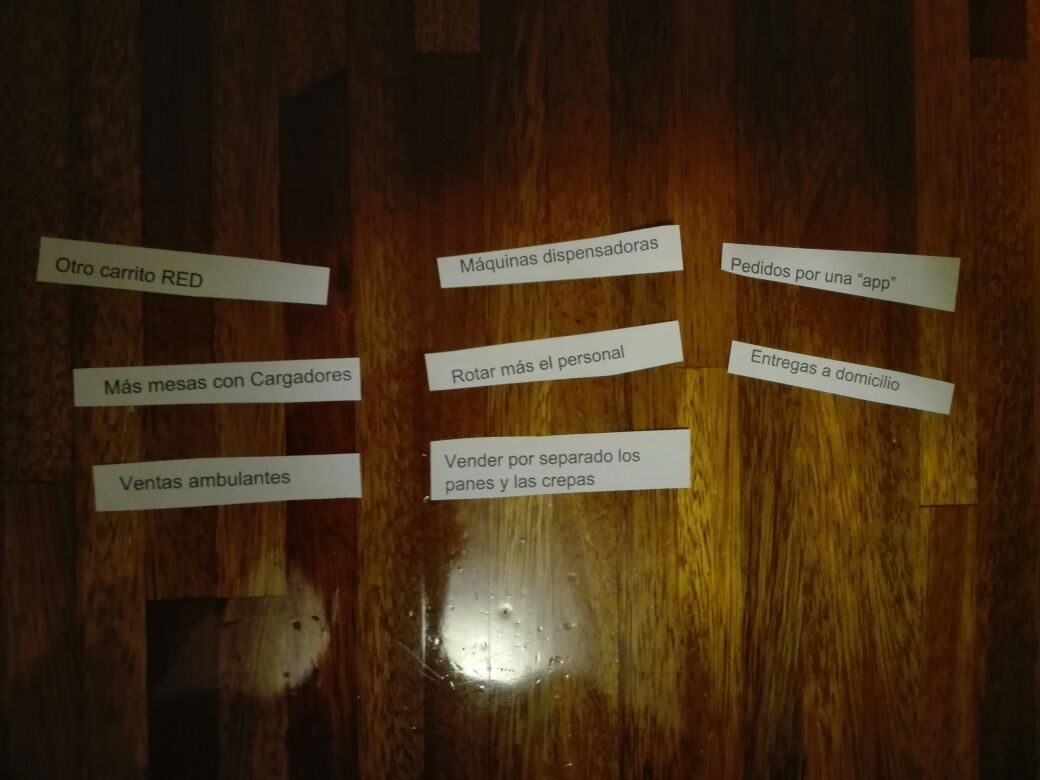
* Otro carrito RED
* Un servicio más personal
* Comida prefabricada
* Vender por separado los panes y las crepas
* Dejar encargada la comida
* Pedidos por teléfono
* Fabricación propia de la crepa
* Turnos para comprar
* Solo se venden panes a ciertas horas del dia
* Servicios de entrega a las clases
* Pedidos por una “app”
* Personal especializado, solo para cobrar y solo para preparar la comida
* Un carrito solo para panes y otro solo para crepas
* Mantener la fruta cortada y lista
* Ventas ambulantes
* Entregas a domicilio
* Otra sucursal de red
* Máquinas dispensadoras
* Ampliación de red
* Mejorar las mesas
* Más lugares en donde sentarse
* Mejor ambiente
* Mejor internet
* Menú más grande
* Mejorar las máquinas
* Capacitar mejor al personal
* Promocionar un cliente del mes
* Rotar más el personal
* Pedidos Black Board
* Ventas por Ticket
* Más mesas con Cargadores
* Un sistema tipo subway para la preparación de la comida
* Tener un local en otro lugar en donde se hagan los pedidos
* Crear un tipo de guion para preparar la comida que busque aumentar la efectividad
* Reservarse la cantidad de clientes
* Tener una asignación de días en los que pueden ordenar
* Tener mini quioscos de red por todo el campus de la U
* Crear una máquina que haga las crepas de manera autónoma
* Organizar el restaurante en forma de un autoservicio para que el proceso sea en partes
* Estandarizar el menú en ingredientes, para así poder mantener un flujo de cocinado constante
* No incluir en el menú platos que su cocción lleve más de 5 min

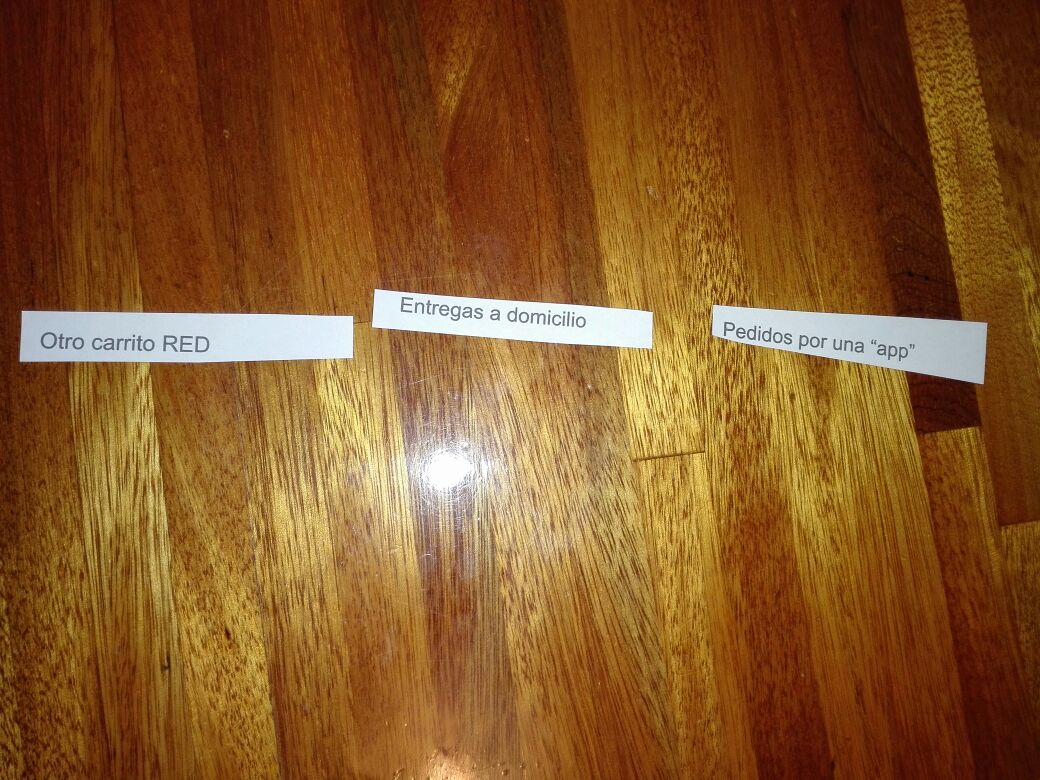




****

****

****

****

**Proceso de selección de Ideas :**

Para la selección de las ideas se trabajo con un sistema de : **“3 de 4 votos”.** Cada una de las ideas debía aprobar con 3 votos de los integrantes del grupo, para ser tomada en cuenta, de forma que tres de los integrantes del grupo aportan características positivas de esta y un solo un integrante del grupo aporta algo negativo a la idea Para esta selección se consideraron aspectos como el costo de aplicarla, la fabricación si llevaba un proceso y si era aplicable por la empresa RED, tratando de no afectar negativamente sus ingresos.

**Ideas Elegidas :**

1. **Aplicación**: desarrollar una aplicación en la que se puedan hacer pedidos y pagar de forma electrónica.
2. **Un carro de crepas**: tener un carrito solo para hacer crepas y panes y así hacer el servicio más rápido.
3. **Entregas a clases**: como parte del servicio de RED que lleven los pedidos a las clases y evitar las filas.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Agosto | | | | | |
| Dia | Lunes 14 | Martes 15 | Miércoles 16 | Jueves 17 | Viernes 18 |
| Actividad | Definir Proyecto | Definir las de Clases del programa | Crear el UML y comenzar a programar en java | Continuar con programación en java | Entregar |
| Septiembre | | | | | |
| Dia | Jueves 22 | Lunes 25 | Martes 26 | Miércoles 27 | Jueves 28 |
| Actividad | Hacer GUI | Re hacer clases para implementarlas | Implementar clases en GUI | Modificar Análisis e UML | Retroalimentación |

**Descripción de clases**

* Red

En esta clase se llevan a cabo operaciones en las cuales se interactúa directamente con las clases “Comida” y “Pedido”. Entre

las operaciones se encuentran desde agregar nuevos elementos

al inventario e incluso gestionar los pedidos realizados por los

usuarios.

* Comida

Esta clase es la que se encarga de asignar los respectivos valores información necesaria para el registro de un objeto de la misma.

* Pedido

En esta clase se construyen los pedidos realizados por la clase usuario. En el proceso mencionado, se asignan valores a los atributos que requieren de la información necesaria para procesarlos y almacenarlos en la clase “Red”.

* Usuario

En esta clase únicamente permite al usuario registrarse como un cliente para poder realizar pedidos a la cafetería.

Clase Pedidos

**Objetivo:** Clase donde se guarda la información básica del pedido

**Atributos:**

-NumeroPedido: Int

-Carnet: int

-Total: Double

-Entrega: String

**Métodos:**

+Constructor(int pedido; int Carnet; double Total; String LugarEntrega): void agrega un nuevo pedido.

Clase PedidosExtendido

**Objetivo:** Clase donde se guarda la información extendida del pedido (los productos y las cantidades del mismo)

**Atributos:**

-NumeroPedido: Int

-Producto: int

-Cantidad: int

-PrecioUnitario: Double

**Metodos:**

+Constructor(int pedido; int cant; int Producto; double PrecioUnitario): void

agrega un nuevo producto a un pedido.

+getNumeroPedido(): int

+getCantidad(): int

+getProducto():int

+getPrecio(): double

Clase Producto

**Objetivo:** Clase donde se encuentra la información de una comida disponible

**Atributos:**

-Name: String

-Price: Double

-Code: int

**Métodos:**

+Constructor (int Codigo; String Nombre; double Precio): void

agrega un nuevo producto.

Clase RED

**Objetivo:** Clase donde se encuentran los métodos para realizar los pedidos y proveer un servicio al usuario.

**Atributos:**

-PedidoActual:int

-Productos: Producto[]

-Ordenes: Pedidos[]

-OrdenesDetalle: PedidosExtendido[]

**Métodos:**

+NuevoPedido (int refPedido, int refCarnet, String refLugarEntrega):void

Crea un nuevo pedido y lo hace el actual para usarlo.

+MostrarProductos( int nPedidoExt): PedidosExtendido

Muestra una lista de todos los productos con sus precios y códigos.

+AgregarProducto(int codigoPedido, int cantidad, Producto p): void

Agrega un producto al pedido actual.

+getCode(String name): int

+getPrecio(String name): double

Producto

-Name: String

-Price: Double

-Code: int

+Constructor (int Codigo; String Nombre; double Precio): void

MAINRED

+total double

+main(String args[])

RED

-PedidoActual:int

-Productos: Producto[]

-Ordenes: Pedidos[]

-OrdenesDetalle:

PedidosExtendido[]

+NuevoPedido (int refPedido, int refCarnet, String refLugarEntrega):void

+MostrarProductos( int nPedidoExt): PedidosExtendido

+AgregarProducto(int codigoPedido, int cantidad, Producto p): void

+getCode(String name): int

+getPrecio(String name): double





Pedidos

NumeroPedido: Int

-Carnet: int

-Total: Double

-Entrega: String

+Constructor(int pedido; int Carnet; double Total; String LugarEntrega): void

PedidosExtendido

-NumeroPedido: Int

-Producto: int

-Cantidad: int

-PrecioUnitario: Double

+Constructor(int pedido; int cant; int Producto; double PrecioUnitario): void

+getNumeroPedido(): int

+getCantidad(): int

+getProducto():int

+getPrecio(): double



**Códigos de error**

|  |  |  |
| --- | --- | --- |
| **Código de error** | **Situación concreta en la que podría ocurrir** | **Mensaje a mostrar al usuario** |
| **1001** | **No exista ningún elemento en el inventario.** | **“El inventario de productos esta vacío.”** |
| **1002** | **Cuando se quiere eliminar un producto del inventario y este esté vacío.** | **“La operación no puede ser posible debido a que no existe ningún producto en el inventario.”** |
| **1003** | **Cuando no es posible agregar un producto al inventario.** | **“No fue posible registrar el producto.”** |
| **1004** | **Debido a que no es posible la carga de información en la base datos.** | **“No se encontró una respuesta del servidor de la base de datos.”** |

**Retroalimentación**

Henry Rodríguez

Cree que la interfaz debería ser más identificable como que fuera de RED.

Heather Castillo

Piensa que sería mejor que los productos tuvieran imágenes.

Guillermo Sandoval

Pregunta por qué tiene que ser solo de RED.

Alberto Urizar

Podrían hacer un menú aparte para elegir los productos

Raúl Monzón

La interfaz debería ser más bonita.

Antonio Reyes

Hay falta de productos

**Resumen**

* Mejorar la estética de la interfaz gráfica.
* Mostrar imágenes de productos aparte
* Agregar productos

**Planteamiento de Problema:** El tiempo de atención en RED

**Nuevo Enfoque:** Modificar el programa para asegurarnos que se encargue de resolver el problema planteado utilizando persistencia, para poder comunicar 2 interfaces gráficas por medio de una base de datos, y así simular la interacción entre usuario y RED. Se busca por medio de esto reducir el tiempo de servicio en RED, y así lograr un servicio más agradable.

**Requisitos Funcionales:**

* Realizar pedido
* Completar pedido

**Clases Nuevo Enfoque**

**Clases de Administrador**

Clase Pedido

**Objetivo:** Clase donde se guarda la información básica del pedido

**Atributos:**

-id: Long

-Codigo: int

-Carne: int

-Nombre: String

-Complete: boolean

**Métodos:**

+ Pedido(int Codigo, int Carne, String Nombre, boolean Status): void

+ getId(): Long

+ setId(Long id): void

Clase PedidoExtendido

**Objetivo:** Clase donde se guarda la información extendida de pedido

**Atributos:**

-id: Long

-Codigo: int

-Cantidad: int

-Producto: int

**Métodos:**

+ PedidoExtendido(int Codigo, int Cantidad, int Producto): void

+ getId(): Long

+ setId(Long id): void

Clase Producto

**Objetivo:** Clase donde se guarda la información del producto

**Atributos:**

-id: Long

-Codigo: int

-Nombre: String

-Precio: double

**Métodos:**

+ Producto(int Codigo, String Nombre, double Precio): void

+ getId(): Long

+ setId(Long id): void

Clase Red

**Objetivo:** Clase donde se manejan las interacciones de pedidos y el usuario

**Atributos:**

-emf: EntityManagerFactory

-em: EntityManager

-AllPedidos: ArrayList<Pedido>

-AllPedidosExtendido: ArrayList<PedidoExtendido>

-AllProducto: ArrayList<Producto>

**Metodos:**

+GetPedidos(): void

+GetPedidosExtendidos(): void

+GetProductos(): void

+GetSubtotal(int cantidad, String ProductoD):double

+GetPrecio(String ProductoD): double

+GetPrecio2(String ProductoD): double

+GetCode(String ProductoD): int

+nuevoPedido(Pedido x): void

+nuevoPedidoextendido(PedidoExtendido x): void

+GetPedidoStatus(int ID):boolean

+GetPedido(int ID):Pedido

+CompletarPedido (Pedido p): void

**UML**

Producto

-id: Long

-Codigo: int

-Nombre: String

-Precio: double

+ Producto(int Codigo, String Nombre, double Precio): void

+ getId(): Long

+ setId(Long id): void

PedidoExtendido

-id: Long

-Codigo: int

-Cantidad: int

-Producto: int

+ PedidoExtendido(int Codigo, int Cantidad, int Producto): void

+ getId(): Long

+ setId(Long id): void

Pedido

-id: Long

-Codigo: int

-Carne: int

-Nombre: String

-Complete: boolean

+ Pedido(int Codigo, int Carne, String Nombre, boolean Status): void

+ getId(): Long

+ setId(Long id): void

Red

-emf: EntityManagerFactory

-em: EntityManager

-AllPedidos: ArrayList<Pedido>

-AllPedidosExtendido: ArrayList<PedidoExtendido>

-AllProducto: ArrayList<Producto>

+GetPedidos(): void

+GetPedidosExtendidos(): void

+GetProductos(): void

+GetSubtotal(int cantidad, String ProductoD):double

+GetPrecio(String ProductoD): double

+GetPrecio2(String ProductoD): double

+GetCode(String ProductoD): int

+nuevoPedido(Pedido x): void

+nuevoPedidoextendido(PedidoExtendido x): void

+GetPedidoStatus(int ID):boolean

+GetPedido(int ID):Pedido

+CompletarPedido (Pedido p): void

RedApp

+main(String args[]):void

**Clases de Usuario**

Clase Pedido

**Objetivo:** Clase donde se guarda la información básica del pedido

**Atributos:**

-id: Long

-Codigo: int

-Carne: int

-Nombre: String

-Complete: boolean

**Métodos:**

+ Pedido(int Codigo, int Carne, String Nombre, boolean Status): void

+ getId(): Long

+ setId(Long id): void

Clase Red

**Objetivo:** Clase donde se manejan las interacciones de pedidos y el usuario

**Atributos:**

-emf: EntityManagerFactory

-em: EntityManager

-AllPedidos: ArrayList<Pedido>

**Metodos:**

+GetPedidos(): void

+GetPedidoStatus(int ID): boolean

+GetTurnos(int id): String

**UML**

Pedido

-id: Long

-Codigo: int

-Carne: int

-Nombre: String

-Complete: boolean

+ Pedido(int Codigo, int Carne, String Nombre, boolean Status): void

+ getId(): Long

+ setId(Long id): void

Red

-emf: EntityManagerFactory

-em: EntityManager

-AllPedidos: ArrayList<Pedido>

+GetPedidos(): void

+GetPedidoStatus(int ID): boolean

+GetTurnos(int id): String

RedApp

+main(String args[]):void