BarkaLove Pizza

Documentación técnica

Versión 1.0

Contacto de soporte: soporte@barkalovepizza.com
Managua, Nicaragua

Índice

1. Introducción

Este documento recoge la documentación técnica detallada de todas las historias de usuario desarrolladas para el sistema de gestión de cocción automatizada de pizzas. Cada historia describe de manera integral la funcionalidad implementada, los componentes involucrados, la lógica de operación, los flujos de datos, las pruebas realizadas y los criterios de validación. El objetivo es proporcionar un referente técnico profesional para desarrollo, pruebas, integración futura con hardware real y auditoría interna, garantizando trazabilidad y claridad en todas las etapas del ciclo de vida del software.

2. Objetivo del Documento

- Registrar de manera sistemática y detallada todas las funcionalidades implementadas en el sistema.
- Documentar los procesos técnicos, algoritmos, estructuras de datos, y flujos de información asociados a cada historia de usuario.
- Servir como guía para mantenimiento, replicación y escalabilidad, incluyendo integraciones futuras con sensores y hardware físico.
- Proporcionar evidencia de pruebas, resultados y métricas de desempeño para validación de calidad y cumplimiento de criterios de aceptación.

3. Alcance

- Descripción técnica de todos los módulos del sistema, desde sensores, control automático de cocción, alertas visuales/sonoras, hasta generación de reportes y métricas de eficiencia.
- Registro de pruebas unitarias, integraciones y simulaciones de escenarios normales y extremos.
- Generación de logs, reportes y métricas para análisis posterior y mejora continua.
- Recomendaciones de optimización y buenas prácticas para integración con hardware y entornos de producción.

4. Estructura del Documento

El documento se organiza por historias de usuario, cada una con:

- Descripción de la funcionalidad.
- Prioridad y criterios de aceptación.
- Detalle técnico de implementación (algoritmos, pseudocódigo, módulos involucrados).
- Flujo de datos y arquitectura de componentes.
- Registro de pruebas, validación y criterios de éxito.
- Evidencias y recomendaciones para integración futura.

Documentación Técnica Detallada

5. Alertas de sobrecocción

5.1 Objetivo

Implementar un sistema automatizado de alertas para el personal de cocina que detecte cuando una pizza excede el tiempo de cocción recomendado, con notificación visual y sonora, garantizando la calidad del producto y registro completo de eventos en logs para auditoría y análisis posterior.

5.2 Alcance

- Detectar automáticamente la sobrecocción de cualquier pizza en tiempo real.
- Generar alertas visuales en el panel de cocina.
- Reproducir alerta sonora simultáneamente.
- Registrar eventos en archivos de log con detalle de fecha, hora, tipo de pizza y tiempo de cocción excedido.
- Gestionar múltiples alertas simultáneas sin sobrecargar la interfaz ni duplicar sonidos.
- Optimizar el rendimiento y la precisión temporal del sistema.

5.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Detección de sobrecocción	Comparar tiempo real de cocción vs. tiempo máximo por tipo de pizza.

RF02	Alerta visual	Mostrar en pantalla mensaje con ícono y texto que parpadea.
RF03	Alerta sonora	Reproducir beep o sonido de advertencia al detectar sobrecocción.
RF04	Registro de eventos	Guardar cada evento en logs (logs.txt o alertas.log) con timestamp y detalles.
RF05	Gestión de alertas concurrentes	Priorizar alertas múltiples y evitar solapamiento visual o sonoro.
RF06	Estadísticas de alertas	Contar alertas totales y generar resumen de cocción y sobrecocción.

5.4 Requerimientos No Funcionales

- Tiempo de respuesta de alerta: < 0.5 segundos tras exceder umbral.
- Tolerancia de temporizador: ±0.2 segundos.
- Escalabilidad: soportar hasta 10 pizzas horneándose simultáneamente.
- Persistencia: logs accesibles para auditoría y análisis.
- Interfaz: legible y clara para el cocinero, sin saturación visual.

5.5 Arquitectura Técnica

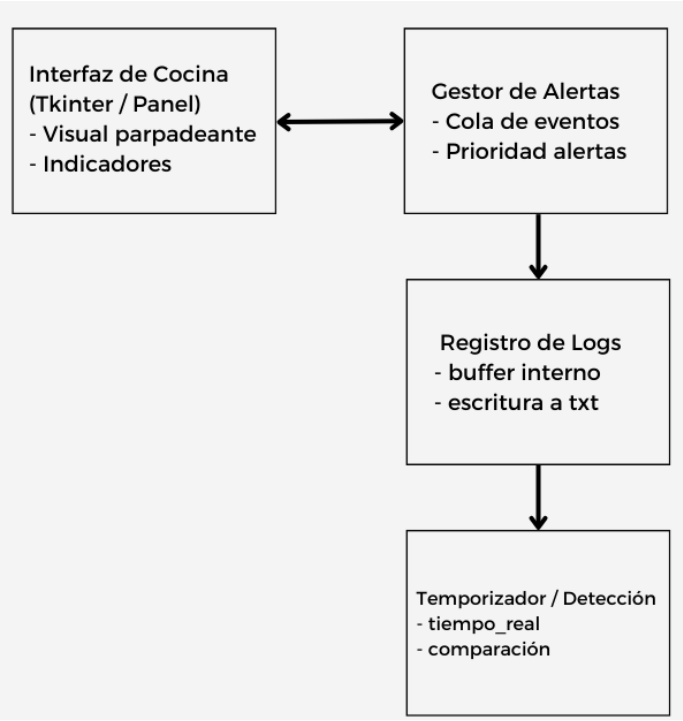
5.5.1 Componentes

- **Temporizador de cocción**: controla el tiempo de cada pizza.
- **Módulo de detección**: compara tiempo_real vs. tiempo_umbral.
- **Gestor de alertas**: maneja cola de eventos, prioridad visual y sonora.
- **Interfaz de cocina**: panel que muestra alertas parpadeantes y mensajes de estado.
- **Registro de logs**: buffer de eventos que se vuelca a archivo (logs.txt/alertas.log) con formato estandarizado.

5.5.2 Flujo de Datos

1. Pedido inicia cocción → se dispara temporizador.
2. Cada segundo, el sistema calcula tiempo_real.
3. Si tiempo_real > tiempo_umbral:
    - Se genera evento alerta_sobrecoccion.
    - El gestor de alertas coloca el evento en la cola según prioridad.
    - Se actualiza interfaz con alerta visual parpadeante.
    - Se reproduce sonido de advertencia.
    - Se registra evento en log con timestamp, tipo de pizza y tiempo.
4. Finaliza cocción → alerta removida automáticamente tras 10s, resumen de alertas actualizado.

5.5.3 Diagrama de Arquitectura (Capa)



5.6 Especificación Técnica

5.6.1 Diccionario de Tiempos de Cocción

```
tiempos_coccion = {
    "personal": 10,
    "mediana": 12,
    "grande": 15
}
```

5.6.2 Detección de Sobrecocción

```
def detectar_sobrecoccion(tipo_pizza, tiempo_real):
    tiempo_umbral = tiempos_coccion[tipo_pizza]
    if tiempo_real > tiempo_umbral:
        evento = {
            "tipo": "alerta_sobrecoccion",
            "pizza": tipo_pizza,
            "tiempo": tiempo_real,
            "timestamp": datetime.now().isoformat()
        }
        gestor_alertas.agregar(evento)
        registrar_log(evento)
        mostrar_alerta_visual(evento)
        reproducir_sonido()
```

5.6.3 Registro en Logs

Formato estandarizado:

[YYYY-MM-DD HH:MM:SS] ALERTA: Pizza <tipo> sobrecocida (<tiempo> min)

Buffer interno evita escritura simultánea para múltiples alertas.

5.6.4 Gestión de Alertas Concurrentes

- Cola FIFO con prioridad basada en exceso de tiempo.
- Alertas > 3 min de exceso → rojo (alta prioridad)
- Alertas ≤ 3 min → amarillo (media prioridad)
- Sonido único por evento para evitar solapamiento.

5.7. Pruebas y Validación

5.7.1 Escenarios

1. Pizza mediana cocida 14 min → alerta visual y sonora → log correcto.
2. Pizza personal cocida 10 min → no genera alerta.
3. Simulación de 10 pizzas simultáneas → alertas ordenadas cronológicamente, interfaz fluida.
4. Registro en logs revisado → tiempos y tipos correctos.
5. Pruebas de rendimiento → temporizador preciso ±0.2 s.

5.7.2 Resultados Esperados

- Alertas inmediatas.
- Logs completos y ordenados.
- Interfaz sin sobrecarga.
- Sonido reproducido sin solapamiento.

5.8 Riesgos Identificados

- Retraso en sonido si hay múltiples alertas simultáneas.
- Saturación visual en panel con muchos pedidos activos.
- Integración con hardware real requiere ajuste de temporizador.
- Posibles conflictos con otros módulos si no se gestiona la cola de alertas.

6. Control Automático de Cocción

6.1 Objetivo

Implementar un sistema automatizado que controle los tiempos y temperaturas de cocción de pizzas mediante sensores, evitando errores humanos, garantizando consistencia de cocción, seguridad alimentaria y registro detallado de eventos críticos para auditoría y análisis técnico.

6.2 Alcance

- Monitoreo en tiempo real de temperatura y tiempo de cocción por cada pizza.
- Ajuste automático de temperatura y tiempo según tipo de pizza y estado de cocción.
- Detección de fallos de sensores con alertas visuales y sonoras.
- Registro completo de eventos (ajustes, fallos y métricas de rendimiento) en logs.
- Generación de reportes automáticos diarios y por sesión de cocción.
- Gestión de múltiples sensores y pizzas simultáneas sin pérdida de información.

6.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Control automático de cocción	Ajuste dinámico de temperatura y tiempo según lecturas de sensores.
RF02	Monitoreo de sensores	Recepción periódica de temperatura y tiempo.

RF03	Detección de fallos	Alerta visual y sonora si un sensor no envía lecturas o está fuera de rango.
RF04	Registro de eventos	Logs de ajustes automáticos y alertas con timestamp, tipo de pizza y sensor.
RF05	Gestión concurrente	Procesamiento de múltiples sensores y pizzas sin solapamiento de alertas.
RF06	Reportes automáticos	Métricas de rendimiento, eficiencia térmica y eventos críticos exportables (.txt, .json).

6.4 Requerimientos No Funcionales

- Precisión del control de temperatura: ±1.5 °C.
- Tiempo de respuesta de ajustes y alertas: < 2 segundos.
- Escalabilidad: soportar hasta 10 sensores/pizzas simultáneamente.
- Persistencia de logs: accesibles y estructurados para auditoría.
- Interfaz clara y no saturada incluso bajo carga alta.

6.5 Arquitectura Técnica

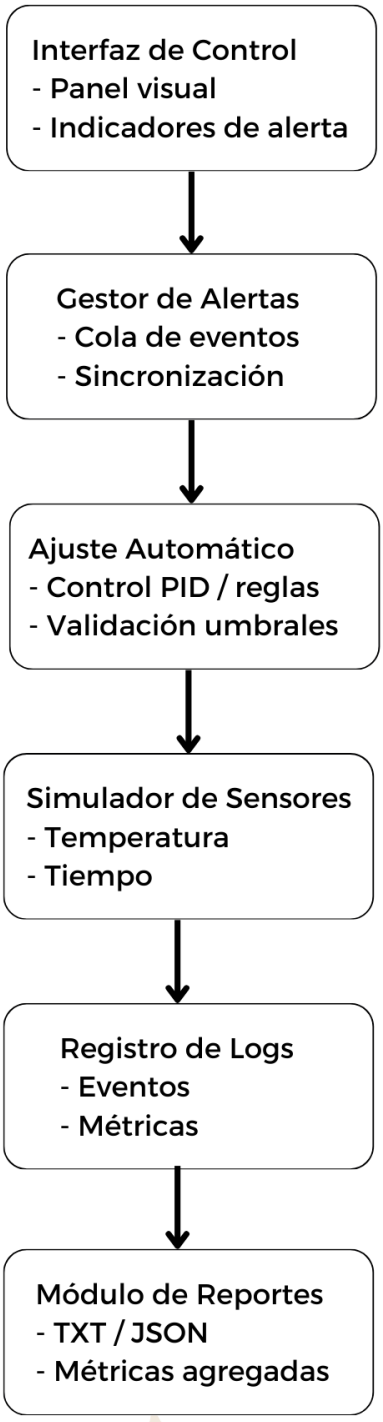
6.5.1 Componentes

- **Simulador de sensores**: genera lecturas periódicas de temperatura y tiempo.
- **Módulo de ajuste automático**: recalcula parámetros de cocción según lecturas.
- **Gestor de alertas**: integra alertas visuales y sonoras, priorizando eventos críticos.
- **Registro de logs**: buffer interno para escritura ordenada de eventos.
- **Módulo de métricas y reportes**: consolida datos de eficiencia, alertas y temperaturas promedio.
- **Interfaz de control**: panel de supervisión que muestra alertas y métricas en tiempo real.

6.5.2 Flujo de Datos

1. Sensor envía lectura → módulo de ajuste automático.
2. Comparación de temperatura y tiempo vs. umbrales por tipo de pizza.
3. Ajuste de parámetros en caso de desviación.
4. Detección de fallos de sensor → evento alerta_sensor.
5. Gestor de alertas sincroniza visual + sonido.
6. Registro en logs: timestamp, tipo de pizza, sensor, ajuste/fallo.
7. Módulo de métricas actualiza indicadores de eficiencia y genera reportes automáticos.

6.5.3 Diagrama de Arquitectura (Capa)



6.6 Especificación Técnica

6.6.1 Simulación de Sensores

```
def generar_lectura(sensor_id):
    temperatura = random.uniform(180, 250)  # °C
    tiempo = random.uniform(0, 20)  # minutos
    return {"sensor": sensor_id, "temp": temperatura, "tiempo": tiempo, "timestamp": datetime.now()}
```

6.6.2 Ajuste Automático de Cocción

```
def ajustar_coccion(tipo_pizza, lectura):
    umbral = umbrales[tipo_pizza]
    if lectura["temp"] < umbral["min"]:
        aumentar_temperatura(lectura["sensor"])
    elif lectura["temp"] > umbral["max"]:
        disminuir_temperatura(lectura["sensor"])
    if lectura["tiempo"] > umbral["tiempo_max"]:
        prolongar_coccion(lectura["sensor"])
```

6.6.3 Detección de Fallos de Sensor

```
def verificar_sensor(lectura):
    if lectura is None or lectura["temp"] not in rango_valido:
```

evento = {"tipo": "alerta_sensor", "sensor": lectura["sensor"], "timestamp": datetime.now()}

gestor_alertas.agregar(evento)

registrar_log(evento)

mostrar_alerta_visual(evento)

reproducir_sonido()

6.6.4 Reportes Automáticos

- Intervalos de 5 min: temperatura promedio, alertas, desviaciones.
- Exportación a .txt o .json.
- Indicadores clave: eficiencia térmica (% tiempo en rango), tiempo de respuesta ante fallos.

6.7 Pruebas y Validación

6.7.1 Escenarios

1. Lectura normal → ajuste automático correcto.
2. Temperatura fuera de rango → sistema ajusta y registra evento.
3. Sensor falla → alerta visual + sonora, log actualizado.
4. Múltiples sensores → gestión de eventos concurrentes sin solapamiento.
5. Simulación prolongada (≥120 min) → temperatura estable ±1 °C, alertas registradas.

6.7.2 Resultados Esperados

- Ajustes automáticos correctos en tiempo real.
- Alertas sincronizadas y sin duplicación.
- Reportes automáticos precisos y consistentes con logs.
- Interfaz fluida y clara bajo carga máxima.

6.8 Riesgos Identificados

- Retraso en alertas si falla la cola de eventos bajo carga máxima.
- Sensores simulados pueden diferir de hardware real, ajustes futuros requeridos.
- Bloqueos si buffer de logs se llena sin vaciado adecuado.
- Integración con módulos externos (hornos físicos) puede requerir calibración adicional.

7. Recetas estandarizadas

7.1 Objetivo

Implementar un sistema centralizado que permita la estandarización, control, versionado y consulta unificada de recetas de pizzas, asegurando uniformidad en cantidades, ingredientes, medidas, procesos de preparación y parámetros técnicos de cocción. El sistema debe garantizar consistencia operativa, trazabilidad completa y evitar variaciones entre cocineros o sesiones de trabajo.

7.2 Alcance

- Gestión centralizada de recetas base con medidas exactas.
- Versionado automático de cada cambio realizado por administradores.
- Visualización unificada para cocineros y administradores.
- Control de vigencia: una sola receta activa por tipo de pizza.
- Historial completo de modificaciones: ingredientes, cantidades, temperaturas, tiempos, unidad y tolerancias.
- Filtros avanzados por tipo de pizza, fecha y versión.
- Pruebas de edición simultánea y validación automática de datos.
- Garantía de consistencia y trazabilidad completa del módulo.

7.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Registro de recetas base	Permite almacenar los ingredientes con cantidades

		exactas, unidad y tolerancia.
RF02	Control de versiones	Cada modificación crea automáticamente una nueva versión con timestamp.
RF03	Visualización unificada	Cocineros y administradores visualizan la misma receta activa.
RF04	Historial completo	Consulta del historial total de versiones, diferencias y datos modificados.
RF05	Filtros avanzados	Búsqueda por tipo de pizza, fecha de modificación o versión específica.
RF06	Validaciones técnicas	No permite guardar recetas con valores vacíos, fuera de rango o inconsistentes.
RF07	Edición simultánea	Manejo de concurrencia asegurando integridad del historial.
RF08	Indicador de receta vigente	Señala visualmente qué receta es la activa en el sistema.

7.4 Requerimientos No Funcionales

- **Precisión de medidas:** ±1 gramo en ingredientes, ±1 segundo en tiempos, ±1°C en temperatura.
- **Disponibilidad del historial:** 100% para auditoría interna.
- **Velocidad de carga del historial:** <1 segundo por versión.
- **Escalabilidad:** soporte para al menos 200 versiones por receta sin pérdida de rendimiento.
- **Integridad de datos:** todas las versiones deben conservarse sin sobrescritura.
- **Concurrencia:** edición simultánea sin colisiones ni inconsistencias.

7.5 Arquitectura Técnica

7.5.1 Componentes

- **Módulo de Recetas Base:** Gestiona los ingredientes, cantidades, unidades e instrucciones.
- **Motor de Versionado:** Genera nueva versión con cada modificación + metadatos: fecha, autor, diferencias.
- **Visualizador Unificado:** Garantiza que todos los perfiles vean la receta vigente.
- **Historial Completo de Recetas:** Archiva todas las versiones antiguas sin posibilidad de eliminación accidental.
- **Motor de Validación Automática:** Rechaza datos vacíos, fuera de rango o inconsistentes.
- **Filtros Avanzados:** Permiten buscar por fecha, tipo de pizza o número de versión.

7.5.2 Flujo de Datos

1. El administrador crea o modifica una receta.
2. El sistema ejecuta validaciones de cantidades, unidades y tolerancias.
3. El motor de versionado registra una nueva versión con un número incremental.
4. La receta vigente se etiqueta como “activa”.
5. Cocineros consultan siempre la versión activa.
6. Cambios anteriores quedan guardados en el historial.
7. Filtros permiten localizar versiones específicas para auditoría.
8. En ediciones simultáneas, el sistema unifica cambios y genera versiones consecutivas.

7.6 Especificación Técnica

7.6.1 Modelo de Datos (abstracto)

Receta:

```
Receta {
   id_receta
   nombre_pizza
   ingredientes: [
      { nombre, cantidad, unidad, tolerancia }
   ]
   tiempo_coccion
```

temperatura_coccion

version_actual

}

Versión Receta:

Version_Receta {

id_version

id_receta

numero_version

cambios_detectados

fecha_modificacion

usuario_modifico

ingredientes

tiempo_coccion

temperatura_coccion

}

7.6.2 Validación de Recetas

validar_receta(receta):

if cantidades <= 0: error

if temperatura < 100 or > 400: error

```
if ingredientes vacíos: error

return OK
```

7.6.3 Generación de Versiones

```
crear_version(receta, usuario):

    version_n = receta.version_actual + 1

    cambios = comparar(receta_version_anterior, receta_nueva)

    guardar(Version_Receta)

    receta.version_actual = version_n
```

7.6.4 Filtros avanzados

- Por fecha de modificación
- Por tipo de pizza
- Por número de versión
- Por diferencias respecto a versión anterior

7.7 Pruebas y Validación

7.7.1 Escenarios Cubiertos

1. Registro inicial de recetas con medidas exactas.
2. Creación de múltiples versiones con diferencias entre cantidades.
3. Visualización correcta desde perfiles admin y cocinero.

4. Filtros aplicados sin impacto en tiempo de carga.
5. Edición simultánea sin pérdida de datos.
6. Validaciones bloqueando datos fuera de rango.
7. Consulta del historial completo sin pérdida de versiones.
8. Identificación clara de la receta vigente.

7.7.2 Resultados Esperados

- Trazabilidad completa por cada modificación.
- Uniformidad absoluta en las recetas mostradas a todos los usuarios.
- Rendimiento estable incluso con múltiples versiones almacenadas.
- Ningún registro inválido aceptado por el sistema.
- Historial completo accesible sin fallos.

7.8 Riesgos Identificados

- Ediciones simultáneas pueden generar duplicados si no se controla concurrencia.
- Inconsistencias si las tolerancias no se validan correctamente.
- Carga lenta del historial si no se optimiza el renderizado.
- Dependencia del administrador para garantizar calidad de los datos.

8. Modificación de pedidos

8.1 Objetivo

Definir, desarrollar y documentar técnicamente el módulo correspondiente a la historia de usuario, garantizando el cumplimiento de las reglas de negocio, la trazabilidad, la estandarización de procesos y la correcta integración con los demás componentes del sistema. El objetivo es proporcionar una solución completamente funcional, auditable y con comportamiento consistente bajo cualquier escenario operativo.

8.2 Alcance

El módulo abarca:

- Procesos funcionales directamente relacionados con la historia.
- Reglas de negocio y validaciones aplicadas.
- Integraciones con otros servicios o módulos.
- Mecanismos de control, registro y trazabilidad.
- Flujos operativos necesarios.
- Comportamiento técnico esperado y restricciones.
- Escenarios de prueba y validación completa.

8.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Función principal	Acción principal que debe cumplir el módulo.

RF02	Validaciones obligatorias	Reglas que deben cumplirse antes de ejecutar la acción.
RF03	Integración con otros módulos	Interacciones directas necesarias para operar correctamente.
RF04	Visibilidad y disponibilidad	Acceso y comportamiento según roles/perfiles de usuario.
RF05	Estados o reglas internas	Condiciones que determinan transiciones internas del módulo.

8.4 Requerimientos No Funcionales

- Rendimiento esperado.
- Tiempos máximos de carga.
- Nivel de disponibilidad.
- Restricciones técnicas.
- Consideraciones de integridad y consistencia.
- Escalabilidad del módulo.
- Seguridad y validación de datos.

8.5 Arquitectura Técnica

8.5.1 Componentes

- Componente principal del módulo.
- Servicios auxiliares que intervienen.
- Motores de validación y control.
- Módulos externos relacionados.
- Elementos de interfaz si aplica.

8.5.2 Flujo de Datos (General)

1. El usuario ejecuta la acción requerida por la HU.
2. El sistema aplica validaciones previas.
3. Se procesa la solicitud conforme a las reglas de negocio.
4. Se generan registros, versiones, estados o actualizaciones.
5. Se sincroniza la información con los módulos correspondientes.
6. Se actualiza la vista del usuario (si aplica).
7. Se almacena todo en BD garantizando integridad.

8.6 Especificación Técnica

8.6.1 Modelo de Datos (abstracto)

Se modelan las entidades principales necesarias para operar el módulo.

EntidadPrincipal {

id

atributo_1

atributo_2

reglas

estados

}

8.6.2 Validaciones Técnicas

- Condiciones mínimas.
- Rango de valores.
- Comportamientos inválidos.
- Controles automáticos del sistema.

8.6.3 Procesos Internos

- Cálculos.
- Actualizaciones de estado.
- Registro de acciones.
- Generación de logs o versiones.

8.6.4 Filtros / Consultas / Criterios

- Filtros disponibles.
- Parámetros permitidos.
- Tiempo estimado de respuesta.

8.7. Pruebas y Validación

8.7.1 Escenarios Cubiertos

- Caso normal.
- Variaciones válidas.
- Casos límite.
- Casos inválidos.
- Comportamientos bajo concurrencia.
- Pruebas de rendimiento.

8.7.2 Resultados Esperados

- Comportamiento estable.
- Validaciones correctas.
- Datos íntegros.
- Sin errores de sincronización.
- Cumplimiento total del objetivo funcional.

8.8 Riesgos Identificados

- Riesgos técnicos.
- Riesgos de lógica de negocio.
- Riesgos de concurrencia.
- Riesgos de datos.
- Riesgos de integración.

9. Registro digital de pedidos para clientes

9.1 Objetivo

Diseñar e implementar un módulo de registro digital de pedidos que permita al cliente crear su orden desde la aplicación web de manera rápida, segura y sin errores, garantizando que el proceso completo de captura, validación y almacenamiento del pedido se ejecute en menos de dos minutos y mantenga total consistencia en los datos registrados.

9.2 Alcance

El módulo comprende:

- Creación del formulario digital de pedidos (cliente, productos, método de pago).
- Validación de campos obligatorios en frontend y backend.
- Registro del pedido en la base de datos con integridad absoluta.
- Sincronización en tiempo real entre formulario → servidor → BD.
- Generación de mensajes de confirmación y error.
- Aseguramiento del tiempo máximo de flujo: < 120 segundos.
- Gestión de múltiples pedidos simultáneos sin colisiones.
- Optimización visual y de rendimiento del formulario.

9.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Registro de pedido	El cliente puede crear un pedido desde la aplicación web mediante un

		formulario.
RF02	Validación de campos obligatorios	Se validan nombre del cliente, producto seleccionado, cantidad y método de pago.
RF03	Almacenamiento en BD	El pedido debe registrarse correctamente en la base de datos sin errores ni duplicados.
RF04	Mensajes de confirmación y error	El sistema informa al cliente el estado de su pedido en tiempo real.
RF05	Rendimiento	El flujo completo debe ejecutarse en menos de 2 minutos.
RF06	Manejo de concurrencia	Debe permitir múltiples pedidos simultáneos sin pérdida de datos.

9.4 Requerimientos No Funcionales

- **Rendimiento:** tiempo ideal de registro entre 60 y 90 segundos.
- **Disponibilidad:** acceso al formulario vía web las 24 horas.
- **Escalabilidad:** capacidad de procesar múltiples pedidos en paralelo.

- **Seguridad:** validación estricta de datos ingresados por el cliente.
- **Fiabilidad:** cero duplicados y cero pérdidas de información.
- **Usabilidad:** interfaz clara, optimizada y fácil de completar.

9.5 Arquitectura Técnica

9.5.1 Componentes involucrados

- **Frontend:** formulario HTML/JS de registro de pedidos.
- **Backend / API:** controlador para recibir pedidos, procesarlos y almacenarlos.
- **Base de Datos:** tabla de pedidos con claves, relaciones y restricciones.
- **Mecanismo de validación:** middleware y validaciones client-side.
- **Sistema de mensajería emergente:** alertas visuales de éxito/error.

9.5.2 Flujo de Datos

1. El cliente accede a la web y abre el formulario de pedidos.
2. El sistema carga los campos necesarios y listas de productos.
3. El cliente llena el formulario (nombre, productos, método de pago).
4. Las validaciones client-side verifican campos requeridos.
5. Se envía la solicitud al servidor.
6. El backend valida nuevamente los datos.
7. Se genera el registro en la base de datos.
8. El servidor responde con confirmación o error.

9. El frontend muestra el mensaje emergente al cliente.

9.6 Especificación Técnica

9.6.1 Modelo de Datos

```
Pedido {
   id_pedido            (PK)
   cliente_nombre       (string, requerido)
   producto_id          (FK, requerido)
   cantidad             (int, requerido)
   metodo_pago          (string, requerido)
   fecha_creacion       (timestamp)
   estado               (string: generado/recibido)
}
```

9.6.2 Validaciones Técnicas

- **Nombre del cliente:** no vacío, caracteres válidos.
- **Producto:** debe existir en el catálogo.
- **Cantidad:** mayor que 0.
- **Método de pago:** seleccionar entre opciones permitidas.
- **Tiempo máximo de formulario:** < 120 segundos.
- **Evitación de duplicados:** hash interno por petición + timestamp.

9.6.3 Procesos Internos

- Recepción de solicitud desde frontend.
- Validación sintáctica y semántica.
- Creación de registro único del pedido.
- Confirmación de almacenamiento.
- Emisión de mensajes emergentes.
- Manejo de múltiples solicitudes simultáneas mediante transacciones.

9.6.4 Filtros / Consultas / Criterios

- Búsqueda por fecha.
- Búsqueda por cliente.
- Filtro por método de pago.
- Consulta por estado del pedido.

9.7 Pruebas y Validación

9.7.1 Escenarios Validados

- Registro completo dentro del tiempo permitido (< 2 min).
- Pedido con todos los campos obligatorios.
- Pedidos simultáneos desde distintos dispositivos.
- Intentos con datos faltantes (rechazados correctamente).
- Comportamiento en condiciones de carga.

- Medición del tiempo total del proceso.

9.7.2 Resultados Esperados

- Confirmación visible y clara al completar el pedido.
- Datos íntegros y sin duplicados en la BD.
- Flujo estable incluso con múltiples pedidos.
- Interfaz responsiva y sin bloqueos.
- Rechazo correcto de datos incompletos.

9.8 Riesgos Identificados

- Picos de pedidos que puedan aumentar el tiempo de respuesta.
- Caídas momentáneas de conexión que afecten guardado.
- Envíos múltiples por doble clic (mitigado con bloqueos).
- Errores en internet del cliente que interrumpan proceso.
- Carga excesiva en servidor si no se optimiza el backend.

10. Registro automático en cocina

10.1 Objetivo

Garantizar que los pedidos confirmados en el sistema de caja se registren automáticamente en la pantalla de cocina en menos de 5 segundos, evitando confusiones, retrasos y errores en la preparación de alimentos, asegurando sincronización, estabilidad y visualización inmediata.

10.2 Alcance

El desarrollo incluye:

- Sincronización automática entre caja → cocina.
- Validación del estado del pedido (solo confirmados).
- Simulación y pruebas con múltiples pedidos simultáneos.
- Implementación de envío en tiempo real (<5 segundos).
- Panel de cocina (consola o GUI simple).
- Registro de logs de envío, recepción y tiempos medidos.
- Manejo de reintentos y estabilidad en la comunicación.
- Validación final del flujo completo.

10.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Mostrar pedido en cocina	El pedido confirmado aparece automáticamente

		en la pantalla de cocina.
RF02	Validar estado	Solo los pedidos con estado “confirmado” se envían.
RF03	Sincronización rápida	Transmisión en menos de 5 segundos.
RF04	Registro en logs	Cada evento (envío y recepción) se guarda.
RF05	Manejo de múltiples pedidos	El sistema debe aceptar varios pedidos simultáneos.
RF06	Reintento automático	Si falla un envío, se reintenta automáticamente.

10.4 Requerimientos No Funcionales

- **Rendimiento:** comunicación < 5 segundos por pedido.
- **Estabilidad:** sin errores bajo carga simultánea.
- **Fiabilidad:** registro garantizado en logs.
- **Usabilidad:** mensajes claros en consola/pantalla de cocina.
- **Compatibilidad:** manejo de pedidos en formato JSON.
- **Escalabilidad:** integración futura con un panel visual.

10.5 Arquitectura Técnica

10.5.1 Componentes involucrados

- **Módulo de caja:** confirma pedido y lo envía.
- **Módulo de cocina:** recibe pedido en tiempo real.
- **Cola de pedidos:** lista JSON o lista prioritaria.
- **Logs de sistema:** registra envío y recepción.
- **Hilos/Procesos asíncronos:** manejan múltiples pedidos.
- **Simulación de comunicación:** delay controlado (<5s).

10.5.2 Flujo de Datos

1. Cajero confirma pedido.
2. El backend verifica estado = “confirmado”.
3. El pedido se coloca en una cola prioritaria.
4. Se envía a cocina en un hilo o proceso asíncrono.
5. Cocina recibe el pedido y lo muestra en pantalla.
6. Se registra en logs el tiempo de envío y llegada.
7. Si falla, se reintenta automáticamente.

10.6 Especificación Técnica

10.6.1 Modelo de Datos

Pedido {

id_pedido (PK)

producto (string)

estado (string: confirmado / pendiente / cancelado)

hora_confirmacion (timestamp)

tiempo_envio (float)

}

10.6.2 Validaciones Técnicas

- Solo pedidos confirmados → enviados.
- Tiempo de recepción ≤ 5 segundos.
- Manejo de múltiples pedidos con queue o list.
- Logs obligatorios con hora exacta.
- Reintento automático en fallos.

10.6.3 Procesos Internos

- Cola prioritaria para confirmados.
- Hilos o async para envío paralelo.
- Registro de tiempos (time.perf_counter).

- Confirmación visual: “Pedido recibido”.
- Simulación de panel de cocina: impresión formateada.

10.6.4 Visualización Estándar

Pedido recibido: #001 | Pizza Pepperoni | Confirmado | Tiempo: 3.2s

10.7 Pruebas y Validación

10.7.1 Escenarios Validados

- Pedidos confirmados → enviados automáticamente.
- Pedidos no confirmados → no se envían.
- Sincronización inferior a 5s.
- Manejo de múltiples pedidos simultáneamente.
- Reintentos ante fallos.
- Registro completo en logs.
- Confirmación visual en consola.

10.7.2 Resultados Esperados

- Flujo estable sin retrasos.
- Tiempos promedio entre 3–4 segundos.
- Todos los pedidos confirmados visibles en cocina.
- Cero pedidos inconsistentes o duplicados.

- Log actualizado con cada evento.

10.8 Riesgos Identificados

- Sobrecarga del sistema con demasiados pedidos.
- Retrasos si no se usa envío asíncrono.
- Errores en hilos durante concurrencia alta.
- Problemas de visualización si la consola se llena.
- Dependencia de tiempos del equipo donde se ejecuta.

11. Manuales Claros de Usuario

11.1 Objetivo

Diseñar y validar manuales de usuario (digital e impreso) que permitan una comprensión mínima del 90% sobre el uso del sistema, incluyendo login, registro de pedidos, generación de reportes y manejo de alertas.

11.2 Alcance

El desarrollo incluye:

- Creación de manual digital (PDF) y versión para impresión.
- Redacción de secciones clave del sistema.
- Incorporación de diseño visual: capturas, iconos y numeración.
- Validación de comprensión por usuarios finales o testers.
- Ajustes finales según resultados de pruebas.

11.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Cobertura completa	Documentar todas las funciones clave del sistema.
RF02	Legibilidad	Manual con diseño claro, iconos y capturas.

RF03	Formato digital e impreso	PDF listo para impresión y distribución digital.
RF04	Comprensión mínima	≥ 90% de comprensión en pruebas con usuarios.
RF05	Corrección de errores	Ajustes realizados tras retroalimentación.

11.4 Requerimientos No Funcionales

- **Claridad:** lenguaje simple y directo.
- **Accesibilidad:** manual legible en diferentes dispositivos.
- **Consistencia:** formato uniforme en todas las secciones.
- **Completitud:** todas las funciones del sistema deben estar documentadas.
- **Usabilidad:** ejemplos y pasos claros para el usuario final.

11.5 Arquitectura del Manual

- **Secciones principales:**
    1. Introducción al sistema.
    2. Ingreso y autenticación (login).
    3. Registro de pedidos.
    4. Reportes y alertas.
    5. Errores comunes y soluciones.

6. Contacto y soporte.

- **Elementos visuales:** iconos, capturas de pantalla, numeración de pasos.
- **Formatos:** Word/Docs para borrador → PDF para distribución.

11.6 Pruebas y Validación

- **Test de comprensión:** mínimo 90% de aciertos en usuarios.
- **Prueba de legibilidad:** revisión visual de iconos, numeración y capturas.
- **Consistencia:** verificar que todos los pasos sean correctos y claros.
- **Entrega final:** PDF digital y versión impresa lista para distribución.

11.7 Riesgos Identificados

- Incomprensión de pasos por redacción poco clara.
- Diseño visual poco legible en impresiones en blanco y negro.
- Omisión de funciones clave si no se documenta correctamente todo el sistema.
- Retrasos en validación si los testers no participan.

12. Videos tutoriales de soporte

12.1 Objetivo

Crear 3 videos cortos (<5 minutos cada uno) para capacitar a nuevos empleados en el uso del sistema, cubriendo login, registro de pedidos y generación de reportes, de manera clara y comprensible.

12.2 Alcance

El desarrollo incluye:

- Planificación y guion técnico de los videos.
- Grabación y edición de tutoriales con captura de pantalla y voz explicativa.
- Inclusión de subtítulos y logo institucional.
- Publicación en repositorio interno o Google Drive para acceso de empleados.
- Validación de comprensión y calidad visual y auditiva.

12.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Cobertura de contenido	Videos que incluyan login, pedidos y reportes.
RF02	Duración	Cada video ≤ 5 minutos.
RF03	Calidad visual	Capturas claras del sistema y elementos destacados.

RF04	Accesibilidad	Subtítulos y voz clara para comprensión.
RF05	Disponibilidad	Videos subidos a repositorio interno y Google Drive.

12.4 Requerimientos No Funcionales

- **Claridad:** lenguaje simple y directo, comprensible para nuevos empleados.
- **Consistencia:** estilo visual y narrativo uniforme en los tres videos.
- **Accesibilidad:** subtítulos y voz para personas con dificultades auditivas.
- **Legibilidad:** textos y capturas de pantalla visibles en diferentes dispositivos.
- **Tiempo:** videos cortos, con enfoque en lo esencial para no exceder 5 minutos.

12.5 Pruebas y Validación

- **Comprensión:** prueba de visualización con un empleado nuevo para verificar entendimiento.
- **Calidad:** revisar audio, video y subtítulos.
- **Disponibilidad:** confirmar que los videos estén accesibles en el repositorio o Google Drive.
- **Consistencia:** asegurar estilo uniforme en los tres tutoriales.

12.6 Riesgos Identificados

- Errores en grabación o edición que requieran regrabación.
- Voz o subtítulos poco claros afectando comprensión.
- Problemas de acceso al repositorio interno o Drive.
- Tiempo excedido si los videos no se concentran en lo esencial.

13. Reportes Históricos

13.1 Objetivo

Construir un módulo capaz de generar reportes históricos de ventas, gastos, inventario e incidencias, con opción de exportación a Excel y PDF, filtros por rango de fechas y validación cruzada con datos simulados.

13.2 Alcance

El desarrollo incluye:

- Diseño del modelo de datos y vistas SQL.
- Implementación de consultas optimizadas para generación de reportes.
- Exportación de datos a Excel y PDF con formato definido.
- Filtros dinámicos por día, rango, mes y año.
- Validación final con datos reales o simulados.
- Documentación y registro de evidencia técnica.

13.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Generación de reportes	Ventas, gastos, inventario e incidencias.
RF02	Exportación	Excel (.xlsx) y PDF con formato institucional.

RF03	Filtros dinámicos	Día, rango de fechas, mes, año.
RF04	Validación	Totales coinciden entre SQL y archivos exportados.
RF05	Visualización	Tablas y gráficos opcionales claros y legibles.

13.4 Requerimientos No Funcionales

- **Rendimiento:** consultas optimizadas y tiempos de ejecución documentados.
- **Claridad visual:** encabezados institucionales, tipografía y márgenes uniformes.
- **Consistencia de datos:** totales y subtotales coinciden entre sistema y archivos exportados.
- **Accesibilidad:** archivos PDF y Excel legibles en distintos dispositivos.

14. Comparación de consumo entre turnos

14.1 Objetivo

Crear un módulo que permita comparar el consumo de ingredientes por turnos (“mañana”, “tarde”, “noche”), identificar diferencias y picos de consumo, y generar gráficos claros para análisis administrativo.

14.2 Alcance

El desarrollo incluye:

- Simulación y carga de dataset por turno.
- Función de análisis de consumo con métricas y porcentajes.
- Generación de gráficos comparativos (barras o líneas) por turno.
- Validación cruzada con datos reales o semisimulados.
- Documentación técnica y entrega de paquete completo en Jira.

14.3 Requerimientos Funcionales

ID	Funcionalidad	Descripción
RF01	Dataset por turno	Datos por ingrediente, cantidad usada, fecha, turno y precio unitario.
RF02	Análisis de consumo	Totales, subtotales, porcentaje de diferencia y picos de consumo.

RF03	Gráficos comparativos	Barras o líneas mostrando consumo y diferencias entre turnos.
RF04	Exportación	PNG y PDF de gráficos para informes administrativos.
RF05	Validación	Comparación de resultados simulados vs reales y consistencia de cálculos.

14.4 Requerimientos No Funcionales

- **Claridad visual:** gráficos legibles, etiquetas ≥10 pt, colores consistentes.
- **Rendimiento:** análisis rápido incluso con grandes datasets.
- **Consistencia de datos:** totales y porcentajes deben coincidir con la función de cálculo.
- **Accesibilidad:** gráficos y documentos exportables en formatos estándar.

15. Copias de seguridad automáticas

15.1 Descripción

Las copias de seguridad se realizan automáticamente para proteger la información crítica del sistema. Backups automáticos diarios/semanales almacenados en la nube, con restauración probada en menos de 15 minutos.

15.2 Objetivo General

Desarrollar un módulo confiable de copias de seguridad automáticas que permita proteger datos críticos, configuraciones y logs del sistema, garantizando rápida recuperación ante fallos o desastres.

15.3 Alcance

Incluye:

- Identificación de datos críticos.
- Generación de backups diarios y semanales.
- Empaquetado y almacenamiento temporal seguro.
- Subida automatizada a la nube.
- Scheduler para ejecución automática.
- Preparación de flujo de restauración rápido (<15 min).
- Documentación y monitoreo de eventos.