IMPLEMENTACIÓN DE LA APRICACIÓÓN WEB

La aplicación web del restaurante gestiona los pedidos recibidos, muestra la cola de cocina y permite al personal actualizar los estados de "recibido", "en_preparacion", "listo_para_entrega", "en_camino", "entregado". Se conecta con los microservicios orders-svc, delivery-svc, kitchen-svc y analytics-svc.

MODELO DE DATOS

El modelo de datos de la aplicación web está diseñado en Amazon DynamoDB, siguiendo un esquema serverless, multi-tenant y orientado a eventos. Cada tabla utiliza tenant_id para diferenciar las sucursales de Papas Queen’s, garantizando el aislamiento de datos entre locales.

TABLAS EN KITCHEN-SVC

Tabla: Kitchen. Guarda los pedidos asignados al área de cocina y el estado del proceso de preparación.

Campos:

- order_id (String): ID del pedido proveniente de orders-svc.
- tenant_id (String): Sucursal o franquicia.
- list_id_staff (List): IDs del personal que participa (cocinero, despachador).
- status (String): Estado del pedido dentro de la cocina ("recibido", "en_preparacion", "listo_para_entrega", "en_camino", "entregado")
- start_time (Timestamp): Inicio de la preparación.
- end_time (Timestamp): Fin de la preparación.
- updated_at (Timestamp): Última modificación del estado.

Tabla: MenuItems. Contiene los productos disponibles en la carta del restaurante.

Campos:

- id_producto (String): Identificador del producto.
- tenant_id (String): Sucursal o franquicia.
- nombre (String): Nombre del plato.
- categoria (String): Categoría (“papas”, “alitas”, “bebidas”).
- precio (Number): Precio del producto.
- available (Boolean): Disponibilidad del producto.
- image_url (String): URL del archivo en S3.
- created_at (Timestamp): Fecha de registro.
- updated_at (Timestamp): Última actualización.

Tabla: Staff. Registra la información del personal operativo del restaurante.

Campos:

- id_staff (String): Identificador único del empleado.
- tenant_id (String): Sucursal o franquicia.

- name (String): Nombre completo.
- dni (String): Documento o ID personal.
- role (String): Rol operativo (“cocinero”, “despachador”, “repartidor”, “admin”).
- email (String): Correo del empleado.
- password_hash (String): Hash de contraseña (para autenticación interna).
- status (String): Estado (“activo” o “inactivo”).
- phone (String): Número de contacto.
- hire_date (Timestamp): Fecha de contratación.
- last_login (Timestamp): Último inicio de sesión.
- id_sucursal (String): Sucursal asignada.

Tabla: Users. Registra usuarios externos o clientes.

Campos:

- id_user (String): Identificador único.
- type_user (String): “customer” o “staff”.
- role (String): Rol (solo si es staff).
- id_sucursal (String): Sucursal.
- name (String): Nombre.
- status (String): Activo o inactivo.

Tabla: Sucursal. Contiene la información de las sucursales multi-tenant.

Campos:

- id_sucursal (String): Identificador único.
- distrito (String): Nombre del distrito o zona.

TABLAS EN ANALYTICS-SVC

Tabla: Analytics. Centraliza las métricas de desempeño.

Campos:

- id_metric (String): Identificador único de la métrica.
- id_order (String): ID del pedido asociado.
- id_staff (String): ID del empleado o repartidor involucrado.
- status (String): Estado final del pedido.
- inicio (Timestamp): Hora de inicio del proceso.
- fin (Timestamp): Hora de finalización.
- tiempo_total (Number): Duración total del pedido en minutos.
- tenant_id (String): Sucursal o franquicia.

TABLAS EN ORDERS-SVC

Tabla: Orders. Registra los pedidos generados por los clientes.

Campos:

- id_order (String): Identificador único del pedido.
- tenant_id (String): Sucursal o franquicia.
- id_customer (String): ID del cliente (temporal, sin autenticación).
- list_id_products (List): Lista de IDs de productos seleccionados.
- status (String): Estado general del pedido ("recibido", "en_preparacion", "listo_para_entrega", "en_camino", "entregado").
- created_at (Timestamp): Fecha de creación.
- updated_at (Timestamp): Última actualización.

Tabla: Productos. Contiene el catálogo general de productos (carta).

Campos:

- id_producto (String): Identificador único.
- nombre (String): Nombre del producto.
- categoria (String): Categoría.
- precio (Number): Precio unitario.
- available (Boolean): Disponibilidad.

TABLAS EN DELIVERY-SVC

Tabla: Delivery. Registra la información del proceso de entrega de pedidos.

Campos:

- id_delivery (String): Identificador único de la entrega.
- id_order (String): Pedido asociado.
- id_staff (String): Repartidor asignado.
- direccion (String): Dirección de entrega.
- tiempo_salida (Timestamp): Hora de salida.
- tiempo_llegada (Timestamp): Hora de llegada.
- status (String): Estado ("asignado", "en_camino", "entregado").
- tenant_id (String): Sucursal o franquicia.

LAMBDA

MICROSERVICIO: ORDERS-SVC

Responsabilidad: gestionar pedidos desde la aplicación del cliente.

Funciones Lambda:

1. createOrder
    - Descripción: crea un nuevo pedido, lo guarda en la tabla Orders y emite el evento Order.Created.
    - Disparador: HTTP POST /orders (API Gateway).
    - Recursos usados: DynamoDB (Orders), EventBridge, Step Functions (inicio del flujo).

2. getOrder
    - Descripción: devuelve los datos completos de un pedido específico.
    - Disparador: HTTP GET /orders/{order_id}.
    - Recursos usados: DynamoDB (Orders).

3. getOrderStatus
    - Descripción: devuelve únicamente el estado actual del pedido (recien, inprogress, finished).
    - Disparador: HTTP GET /orders/{order_id}/status.
    - Recursos usados: DynamoDB (Orders).

4. getCustomerOrders
    - Descripción: lista todos los pedidos realizados por un cliente.
    - Disparador: HTTP GET /orders/customer/{id_customer}.
    - Recursos usados: DynamoDB (Orders).

5. updateOrderStatus
    - Descripción: permite al restaurante actualizar el estado del pedido (por ejemplo, de recien a inprogress).
    - Disparador: HTTP PATCH /orders/{order_id}/status.
    - Recursos usados: DynamoDB (Orders), EventBridge (Order.Updated).

6. cancelOrder
    - Descripción: cancela un pedido antes de ser preparado y emite Order.Cancelled.
    - Disparador: HTTP POST /orders/{order_id}/cancel.
    - Recursos usados: DynamoDB (Orders), EventBridge.

7. handleOrderDelivered
    - Descripción: escucha el evento Order.Delivered y actualiza el estado del pedido a finished.
    - Disparador: evento Order.Delivered (EventBridge).
    - Recursos usados: DynamoDB (Orders).

8. listProducts
    - Descripción: devuelve la lista de productos del menú para el cliente.
    - Disparador: HTTP GET /menu.
    - Recursos usados: DynamoDB (Productos), S3 (papasqueens-menu-images).

9. getProductsByCategory
    - Descripción: devuelve la lista de productos filtrados por categoría para el cliente
    - Disparador: HTTP GET /menu/category/{categoria}
    - Recursos usados: DynamoDB (Productos).

MICROSERVICIO: KITCHEN-SVC

Responsabilidad: manejar el flujo interno de la cocina y la administración del personal y del menú.

Funciones Lambda:

1. receiveOrder
    - Descripción: recibe el evento Order.Created y agrega el pedido a la tabla Kitchen.
    - Disparador: evento Order.Created (EventBridge).
    - Recursos usados: DynamoDB (Kitchen).

2. getKitchenQueue
    - Descripción: lista los pedidos pendientes o en preparación en la cocina.
    - Disparador: HTTP GET /kitchen/queue.
    - Recursos usados: DynamoDB (Kitchen).

3. acceptOrder
    - Descripción: asigna el pedido a un cocinero y cambia el estado a en_preparacion.
    - Disparador: HTTP POST /kitchen/orders/{order_id}/accept.
    - Recursos usados: DynamoDB (Kitchen), EventBridge (Order.Updated).

4. packOrder
    - Descripción: marca el pedido como listo y emite el evento Order.Prepared.
    - Disparador: HTTP POST /kitchen/orders/{order_id}/pack.
    - Recursos usados: DynamoDB (Kitchen), EventBridge.

5. listMenuItems
    - Descripción: devuelve los productos del menú del restaurante.
    - Disparador: HTTP GET /menu.
    - Recursos usados: DynamoDB (MenuItems), S3 (papasqueens-menu-images).

6. addMenuItem
    - Descripción: agrega un nuevo producto al menú y guarda su imagen en S3.
    - Disparador: HTTP POST /menu.
    - Recursos usados: DynamoDB (MenuItems), S3.

7. updateMenuItem
    - Descripción: modifica un producto existente (precio, disponibilidad, descripción).
    - Disparador: HTTP PATCH /menu/{id_producto}.
    - Recursos usados: DynamoDB (MenuItems).

8. deleteMenuItem
    - Descripción: elimina o desactiva un producto del menú.
    - Disparador: HTTP DELETE /menu/{id_producto}.
    - Recursos usados: DynamoDB (MenuItems).

9. manageStaff
    - Descripción: agrega o actualiza empleados en la tabla Staff.
    - Disparador: HTTP POST /staff (crear) o PATCH /staff/{id_staff} (actualizar).
    - Recursos usados: DynamoDB (Staff), EventBridge (Staff.Updated).

10. listStaff

- Descripción: lista el personal de la sucursal con sus roles y estados.
- Disparador: HTTP GET /staff.
- Recursos usados: DynamoDB (Staff).

11. syncKitchenMetrics
    - Descripción: calcula tiempos de preparación y envía métricas a analytics-svc.
    - Disparador: evento Order.Prepared o ejecución programada.
    - Recursos usados: DynamoDB (Kitchen), EventBridge (Kitchen.MetricsUpdated).

MICROSERVICIO: DELIVERY-SVC

Responsabilidad: manejar el despacho, seguimiento y confirmación de entregas.

Funciones Lambda:

1. receivePreparedOrder
    - Descripción: escucha el evento Order.Prepared y crea una entrada en Delivery.
    - Disparador: evento Order.Prepared (EventBridge).
    - Recursos usados: DynamoDB (Delivery).

2. assignDelivery
    - Descripción: asigna un repartidor disponible a un pedido listo.
    - Disparador: HTTP POST /delivery/assign.
    - Recursos usados: DynamoDB (Delivery), DynamoDB (Staff).

3. updateDeliveryStatus
    - Descripción: actualiza el estado de la entrega (“asignado”, “en_camino”, “entregado”).
    - Disparador: HTTP PATCH /delivery/{id_delivery}/status.
    - Recursos usados: DynamoDB (Delivery), EventBridge (Order.EnRoute o Order.Delivered).

4. handoffOrder
    - Descripción: registra la salida del pedido desde el restaurante.
    - Disparador: HTTP POST /delivery/orders/{id_order}/handoff.
    - Recursos usados: DynamoDB (Delivery), EventBridge (Order.EnRoute).

5. confirmDelivered
    - Descripción: confirma la entrega final, guarda prueba en S3 y emite Order.Delivered.
    - Disparador: HTTP POST /delivery/orders/{id_order}/delivered.
    - Recursos usados: DynamoDB (Delivery), S3 (papasqueens-delivery-proof), EventBridge.

6. getDeliveryStatus
    - Descripción: devuelve el estado actual de una entrega.
    - Disparador: HTTP GET /delivery/{id_order}.
    - Recursos usados: DynamoDB (Delivery).

7. trackRider
    - Descripción: devuelve la última ubicación GPS del repartidor.
    - Disparador: HTTP GET /delivery/{id_order}/track.
    - Recursos usados: DynamoDB (Delivery).

8. listRiders
    - Descripción: lista los repartidores activos de la sucursal.
    - Disparador: HTTP GET /riders.
    - Recursos usados: DynamoDB (Staff).

9. updateRiderStatus
    - Descripción: cambia el estado de disponibilidad del repartidor.
    - Disparador: HTTP PATCH /riders/{id_staff}/status.
    - Recursos usados: DynamoDB (Staff).

10. updateRiderLocation
    - Descripción: actualiza la última ubicación GPS del repartidor para un delivery en_camino.
    - Disparador: HTTP POST /delivery/location.
    - Recursos usados: DynamoDB (Delivery).

11. deliveryMetrics
    - Descripción: calcula métricas de entrega y las envía a analytics-svc.
    - Disparador: evento Order.Delivered o ejecución programada.
    - Recursos usados: DynamoDB (Delivery), EventBridge (Delivery.MetricsUpdated).

MICROSERVICIO: ANALYTICS-SVC

Responsabilidad: recopilar y mostrar métricas de desempeño de pedidos, empleados y entregas.

Funciones Lambda:

1. collectOrderMetrics
    - Descripción: escucha Order.Created y actualiza métricas de pedidos.
    - Disparador: evento Order.Created (EventBridge).
    - Recursos usados: DynamoDB (Analytics).

2. collectKitchenMetrics
    - Descripción: escucha Order.Prepared y calcula tiempos promedio de preparación.
    - Disparador: evento Order.Prepared (EventBridge).
    - Recursos usados: DynamoDB (Analytics).

3. collectDeliveryMetrics
    - Descripción: escucha Order.Delivered y calcula tiempos de entrega y eficiencia.
    - Disparador: evento Order.Delivered (EventBridge).
    - Recursos usados: DynamoDB (Analytics).

4. collectStaffMetrics

- Descripción: escucha Staff.Updated o Kitchen.MetricsUpdated para actualizar rendimiento de empleados.
- Disparador: evento Staff.Updated o Kitchen.MetricsUpdated (EventBridge).
- Recursos usados: DynamoDB (Analytics).

5. getAnalyticsOrders
   - Descripción: devuelve métricas agregadas de pedidos (cantidad, tiempo promedio).
   - Disparador: HTTP GET /analytics/orders.
   - Recursos usados: DynamoDB (Analytics).

6. getAnalyticsEmployees
   - Descripción: devuelve métricas de desempeño del personal.
   - Disparador: HTTP GET /analytics/employees.
   - Recursos usados: DynamoDB (Analytics).

7. getAnalyticsDelivery
   - Descripción: devuelve métricas de entregas y zonas de mayor demanda.
   - Disparador: HTTP GET /analytics/delivery.
   - Recursos usados: DynamoDB (Analytics).

8. getDashboard
   - Descripción: combina métricas generales en un solo dashboard.
   - Disparador: HTTP GET /analytics/dashboard.
   - Recursos usados: DynamoDB (Analytics), S3 (papasqueens-analytics-exports).

9. exportAnalyticsReport
   - Descripción: genera y guarda un reporte JSON o CSV en S3.
   - Disparador: ejecución programada (CloudWatch Event o Step Function).
   - Recursos usados: DynamoDB (Analytics), S3.

BUCKETS EN S3

1. **Bucket: papasqueens-menu-images**
   Contiene las imágenes de los productos del menú.
   Ruta ejemplo: /papasqueens-menu-images/{tenant_id}/{id_producto}.jpg
   Relacionado con: Tabla MenuItems (kitchen-svc)

2. **Bucket: papasqueens-delivery-proof**
   Contiene fotografías o firmas de las entregas.
   Ruta ejemplo: /papasqueens-delivery-proof/{tenant_id}/{id_order}/proof.jpg
   Relacionado con: Tabla Delivery (delivery-svc)

3. **Bucket: papasqueens-orders-receipts**
   Guarda comprobantes o recibos de pedidos completados.
   Ruta ejemplo: /papasqueens-orders-receipts/{tenant_id}/{id_order}/receipt.pdf
   Relacionado con: Tabla Orders (orders-svc)

4. **Bucket: papasqueens-staff-docs**
   Guarda documentos o imágenes de perfil del personal.
   Ruta ejemplo: /papasqueens-staff-docs/{tenant_id}/{id_staff}/profile.jpg
   Relacionado con: Tabla Staff (kitchen-svc)

5. **Bucket: papasqueens-analytics-exports**
   Guarda reportes o backups de métricas consolidadas.
   Ruta ejemplo: /papasqueens-analytics-exports/{tenant_id}/{fecha}/metrics.json
   Relacionado con: Tabla Analytics (analytics-svc)

EVENTBRIDGE

Bus de eventos: papasqueens-event-bus
Es el canal central de comunicación asíncrona entre microservicios.

Eventos principales y su relación con las tablas:

- Order.Created → Emisor: orders-svc / Receptor: kitchen-svc / Actualiza tabla Kitchen (inserta nuevo pedido).
- Order.Prepared → Emisor: kitchen-svc / Receptor: delivery-svc / Actualiza tabla Delivery (crea asignación).
- Order.EnRoute → Emisor: delivery-svc / Receptor: orders-svc / Actualiza tabla Orders (estado visible para cliente).
- Order.Delivered → Emisor: delivery-svc / Receptores: orders-svc y analytics-svc / Actualiza tablas Orders y Analytics.
- Staff.Updated → Emisor: kitchen-svc / Receptor: analytics-svc / Actualiza métricas de desempeño por empleado.

De esta manera, cada transición importante se refleja en los servicios que la necesitan sin acoplamiento directo.

STEP FUNCTIONS

State Machine: papasqueens-order-workflow. Flujo general del pedido en AWS Step Functions:

1. **ValidateOrder (orders-svc):**
   Valida la estructura del pedido, verificando que los productos existan, que el monto sea correcto y que el tenant sea válido.
   Lambda utilizada: createOrder.
   Servicios usados: API Gateway, DynamoDB (Orders).
   Resultado: pedido validado con status “recien”.

2. **SaveOrder (orders-svc):**
   Inserta el pedido en la tabla Orders y emite el evento Order.Created a EventBridge para notificar a kitchen-svc.
   Lambda utilizada: createOrder (segunda fase) o getOrder para confirmar guardado.
   Servicios usados: DynamoDB (Orders), EventBridge.

Resultado: pedido creado y almacenado, evento Order.Created publicado.

3. **EmitOrderCreated (orders-svc):**
   Publica el evento Order.Created en EventBridge para que kitchen-svc inicie el proceso interno de cocina.
   Lambda utilizada: EventBridge integrado en createOrder.
   Servicios usados: EventBridge.
   Resultado: kitchen-svc recibe el pedido y lo agrega a su cola interna.

4. **WaitForPrepared (kitchen-svc):**
   Espera el evento Order.Prepared emitido por kitchen-svc cuando el pedido está empacado y listo para entrega.
   Lambdas involucradas: receiveOrder (consume Order.Created) y packOrder (emite Order.Prepared).
   Servicios usados: DynamoDB (Kitchen), EventBridge.
   Resultado: el pedido pasa de "en_preparacion" a "listo_para_entrega".

5. **AssignDelivery (delivery-svc):**
   Asigna un repartidor disponible al pedido recién preparado y crea un registro en la tabla Delivery.
   Lambda utilizada: assignDelivery.
   Servicios usados: DynamoDB (Delivery), DynamoDB (Staff), EventBridge.
   Resultado: pedido asignado a un repartidor, estado "asignado".

6. **WaitForDelivered (delivery-svc):**
   Espera el evento Order.Delivered que se emite cuando el pedido fue entregado al cliente.
   Lambdas involucradas: confirmDelivered (emite Order.Delivered) y getDeliveryStatus (consulta de seguimiento).
   Servicios usados: DynamoDB (Delivery), S3 (papasqueens-delivery-proof), EventBridge.
   Resultado: entrega confirmada, prueba almacenada en S3, evento Order.Delivered enviado.

7. **UpdateAnalytics (analytics-svc):**
   Registra métricas de tiempos y estado final del pedido en la tabla Analytics.
   Lambdas utilizadas: collectDeliveryMetrics (consume Order.Delivered) y collectOrderMetrics (refresca totales).
   Servicios usados: DynamoDB (Analytics), EventBridge.
   Resultado: métricas actualizadas con tiempos de preparación y entrega.

8. **CloseOrder (orders-svc):**
   Cierra el pedido en la tabla Orders, cambiando su estado a "finished".
   Lambda utilizada: handleOrderDelivered (consume Order.Delivered).
   Servicios usados: DynamoDB (Orders).
   Resultado: pedido finalizado correctamente y visible como completado en la app del cliente.

CLOUDWATCH LOGS Y OBSERVABILIDAD

Configuración de logging y monitoreo para todas las funciones Lambda.

LOGGING ESTRUCTURADO

Helper: common/logger.py
Proporciona logging estructurado en formato JSON con información contextual útil para CloudWatch.
Funciones disponibles:

- log_info(message, event, context, extra): Log de información con contexto.
- log_error(message, error, event, context, extra): Log de error con stack trace completo.
- log_warning(message, event, context, extra): Log de advertencia con contexto.
- log_debug(message, event, context, extra): Log de debug con contexto.
- lambda_handler_wrapper(handler_func): Decorador que captura TODOS los errores no manejados automáticamente.

CONFIGURACIÓN EN SERVERLESS.YML

- Logs habilitados: accessLogging para HTTP API y REST API, frameworkLambda para logs del framework.
- Tracing habilitado: AWS X-Ray para Lambda y API Gateway.
- Permisos IAM: logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents, cloudwatch:PutMetricData.

INFORMACIÓN INCLUIDA EN LOGS

- timestamp: Fecha y hora UTC de la ejecución.
- request_id: ID único de la petición Lambda.
- function_name: Nombre de la función Lambda.
- user_type: Tipo de usuario (customer o staff)
- user_id: ID del usuario autenticado.
- path_params: Parámetros de ruta HTTP.
- query_params: Parámetros de consulta HTTP.
- error_type: Tipo de error (si aplica).
- error_message: Mensaje de error (si aplica).
- traceback: Stack trace completo (para errores).

LOG GROUPS EN CLOUDWATCH

Cada función Lambda crea automáticamente un log group:
/aws/lambda/papasqueens-platform-{stage}-{function-name}
Ejemplo: /aws/lambda/papasqueens-platform-dev-createOrder

MÉTRICAS AUTOMÁTICAS

CloudWatch captura automáticamente:

- Invocations: Número de invocaciones de la función.
- Errors: Número de errores.
- Duration: Tiempo de ejecución en milisegundos.
- Throttles: Número de throttles.
- ConcurrentExecutions: Ejecuciones simultáneas.

USO EN FUNCIONES LAMBDA

Opción 1: Decorador (recomendado para captura automática de errores).

```
@lambda_handler_wrapper
def handler(event, context):
    # cualquier error será capturado y logueado automáticamente
    return result
```

Opción 2: Logging manual (más control).

```
from common.logger import log_info, log_error
def handler(event, context):
    try:
        log_info("Iniciando", event, context)
        # código
    except Exception as e:
        log_error("Error", e, event, context)
```

BENEFICIOS

- Observabilidad completa: Todos los errores se capturan y loguean.
- Búsqueda rápida: Logs estructurados en JSON facilitan filtrado en CloudWatch Insights.
- Trazabilidad: Request ID conecta todos los logs de una petición.
- Debugging: Stack traces completos para todos los errores.
- Monitoreo: Fácil crear métricas y alertas basadas en logs.
- Auditoría: Logs completos de todas las acciones con contexto.