PDC

Fulfill Force Automation (FFA)

DERCAS TÉCNICO

Versión 1.3

05/08/2025

Contenido

1 Historial de versiones

Autor	Versión	Fecha	Cambios
Luis Hernández	1.0	02/08/2025	Versión inicial.
Luis Hernández	1.1	03/08/2025	Se agrega Diagrama y CUT
Luis Hernández	1.2	04/08/2025	Se agrega diagrama de arquitectura y Listado de componentes
Luis Hernández	1.3	04/08/2025	Se agrega diagrama de flujo y servicios

2 Metodología y seguimiento

Marco de trabajo	**SCRUM**	X
	Cascada	
	Hibrido	

3 Control de versiones

Control de versiones	**DevOps**	
	Git	
	Hibrido	X

Se adjunta al presente documento el archivo https://github.com/HernandezGramajo/FFA.git

4 Requerimiento

Este documento corresponde a la versión 1.0 del DERCAS de usuario del proyecto FFA (Fulfill Force Automation)

FFA es una solución móvil y web orientada a vendedores, operadores y supervisores para mejorar la gestión de actividades comerciales en campo. El requerimiento principal es permitir a los usuarios acceder a funcionalidades como login offline, toma de pedidos, sincronización de datos, geolocalización y gestión avanzada de clientes, con el objetivo de reducir costos operativos y mejorar la eficiencia del proceso de distribución.

5 Alcances

El software a desarrollar contempla las siguientes funciones:

- Inicio de sesión con correo o teléfono, incluyendo autenticación offline mediante datos previamente sincronizados.
- Sincronización de datos entre el dispositivo y el backend para uso offline de funcionalidades como pedidos o gestión de clietnes.
- Gestión de clientes, permitiendo edición de información, geolocalización, ingreso de pedidos, contactos, direcciones y recordatorios.
- Dashboard de metas, que visualiza el cumplimiento de objetivos por parte del vendedor.
- Creación de clientes nuevos, incluyendo múltiples contactos, direcciones y fotografías del documento de identificación.
- Geolocalización automática con Google Maps para establecer la ubicación del cliente.
- Almacenamiento temporal local usando base de datos embebida (SQLite o equivalente)
- Sincronización segura y controlada hacia backend central mediante REST API
- Servicios a consumir
    - Google Maps API
    - SMS o Correo electrónico para autenticación
    - Servicio ERP para ingreso de clientes y pedidos
- Se contempla el uso de logs de las peticiones web realizadas por el backend.

6 Límites

Las siguientes funcionalidades o características quedan excluidas del software descrito en este documento:

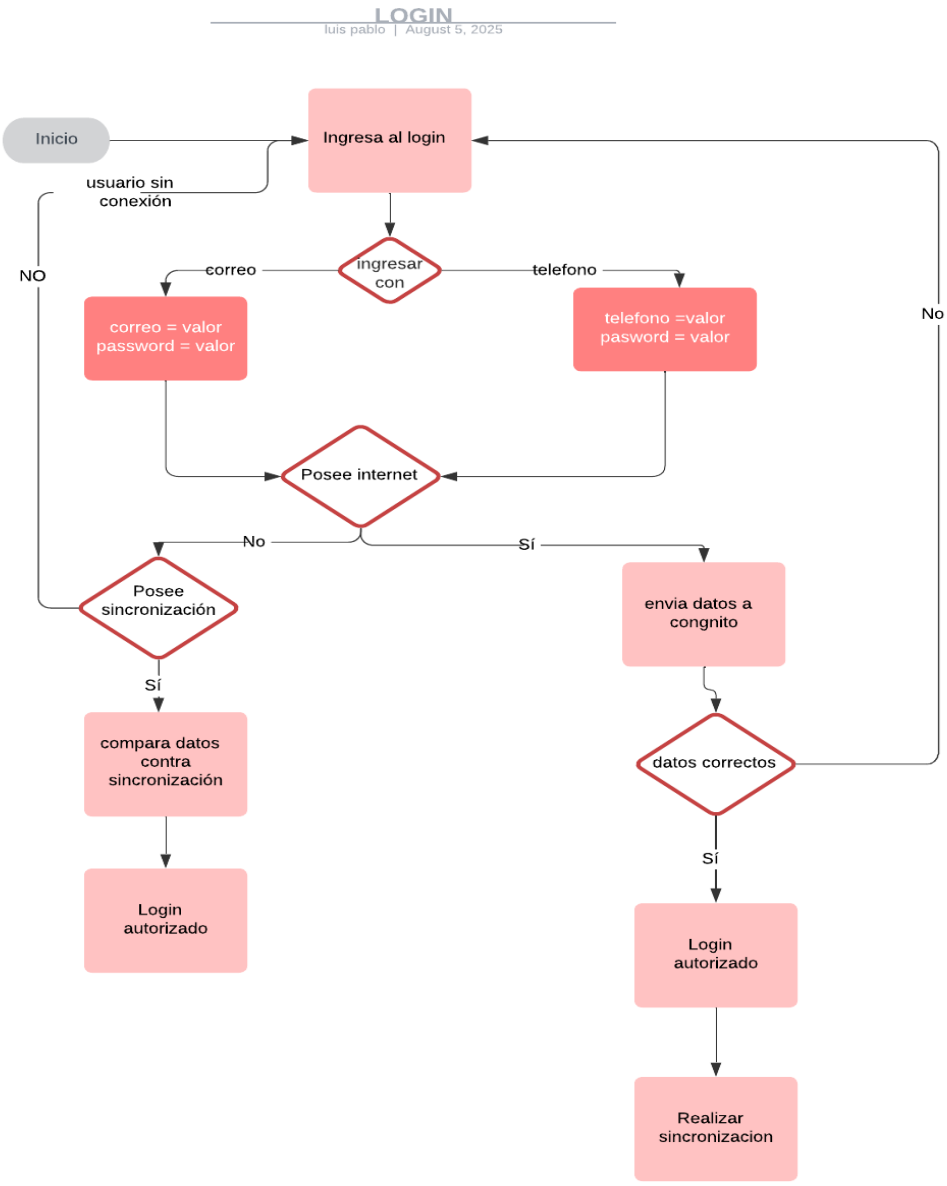
- Reportería administrativa o analítica avanzada (dashboard de negocios con filtros complejos)
- Integración con sistemas de facturación electrónica.
- Funcionalidades logísticas como control de rutas, inventarios, o tracking del vehículo entiempo real.
- Registro de edición de productos desde la app móvil.
- No se contempla la sincronización automática en segundo plano (background sync) en dispositivo sin conexión.
- No se implementa un sistema de mensajería o notificaciones push en esta fase.
- No se contempla multilenguaje.

7 Consideraciones

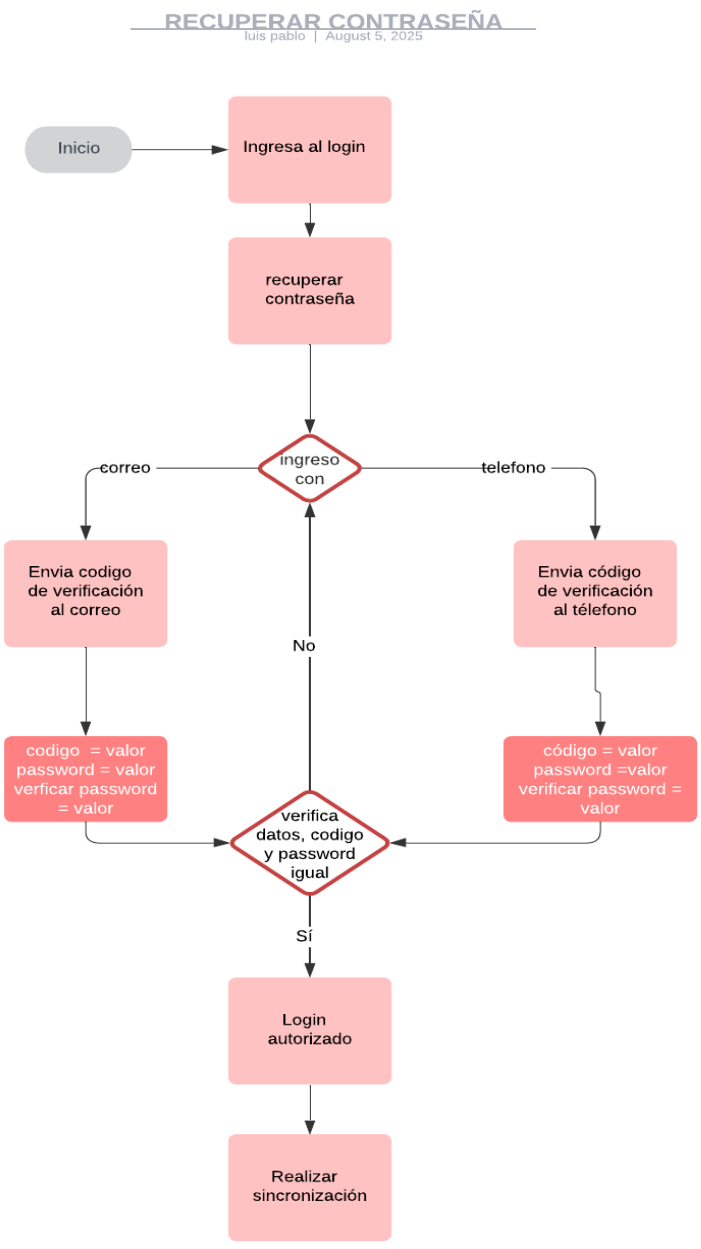
- La aplicación deberá funcionar en modo offline, siempre que el usuario haya iniciado sesión previamente y se haya realizado una sincronización exitosa.
- La sincronización de datos deberá manejar colisiones y fallos de rede de forma resiliente.
- Se sugiere almacenar los datos offline utilizando una base de datos local eficiente como SQLite, Hive o similar.
- La geolocalización deberá solicitar permisos adecuados y ofrecer retroalimentación al usuario en caso de error.
- La solución deberá permitir integración con sistemas existentes vía APIs.
- Se debe de contemplar compatibilidad con dispositivos Android e iOS mediante la compilación cruzada.
- El backend deberá ser diseñado con una arquitectura escalable, preferiblemente orientada a microservicios.
- Las credenciales o tokens utilizados para acceso offline deberán tener expiración configurable y estar cifrados localmente.

8 Diagrams de flujos

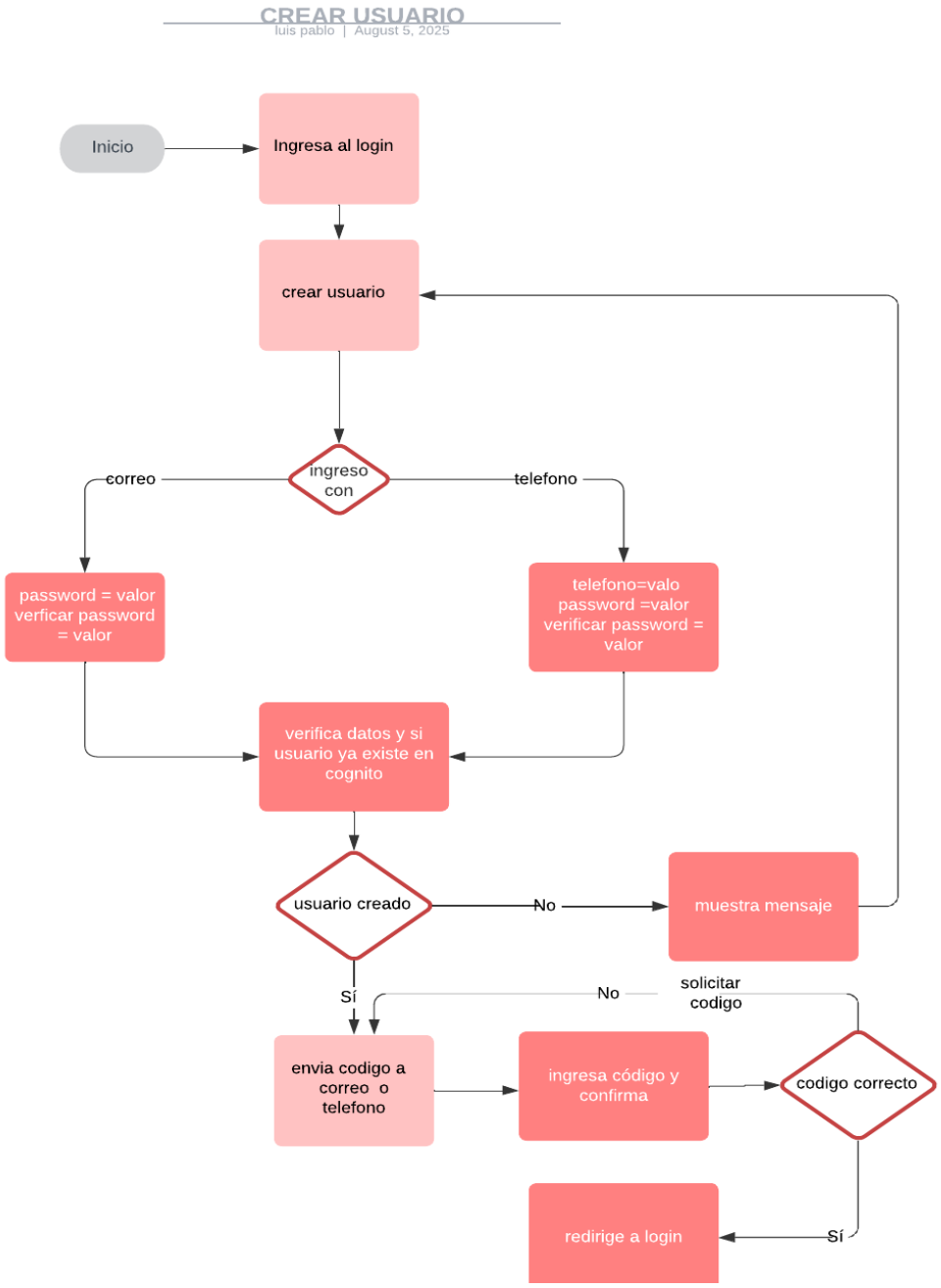
8.1 Login



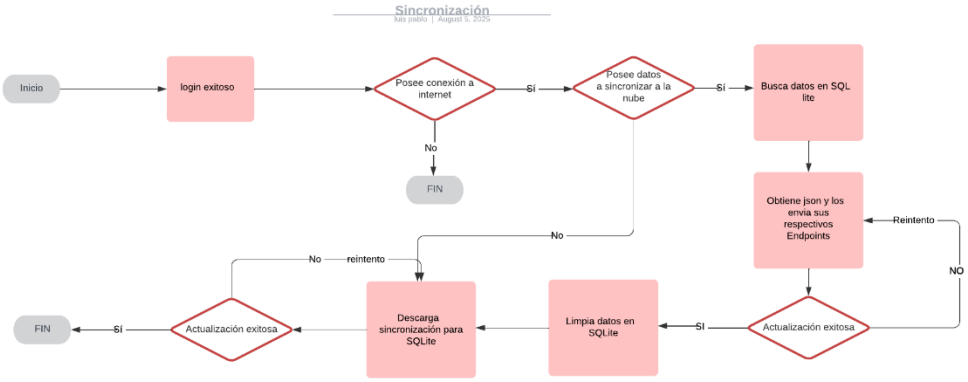
8.2 Recuperar contraseña



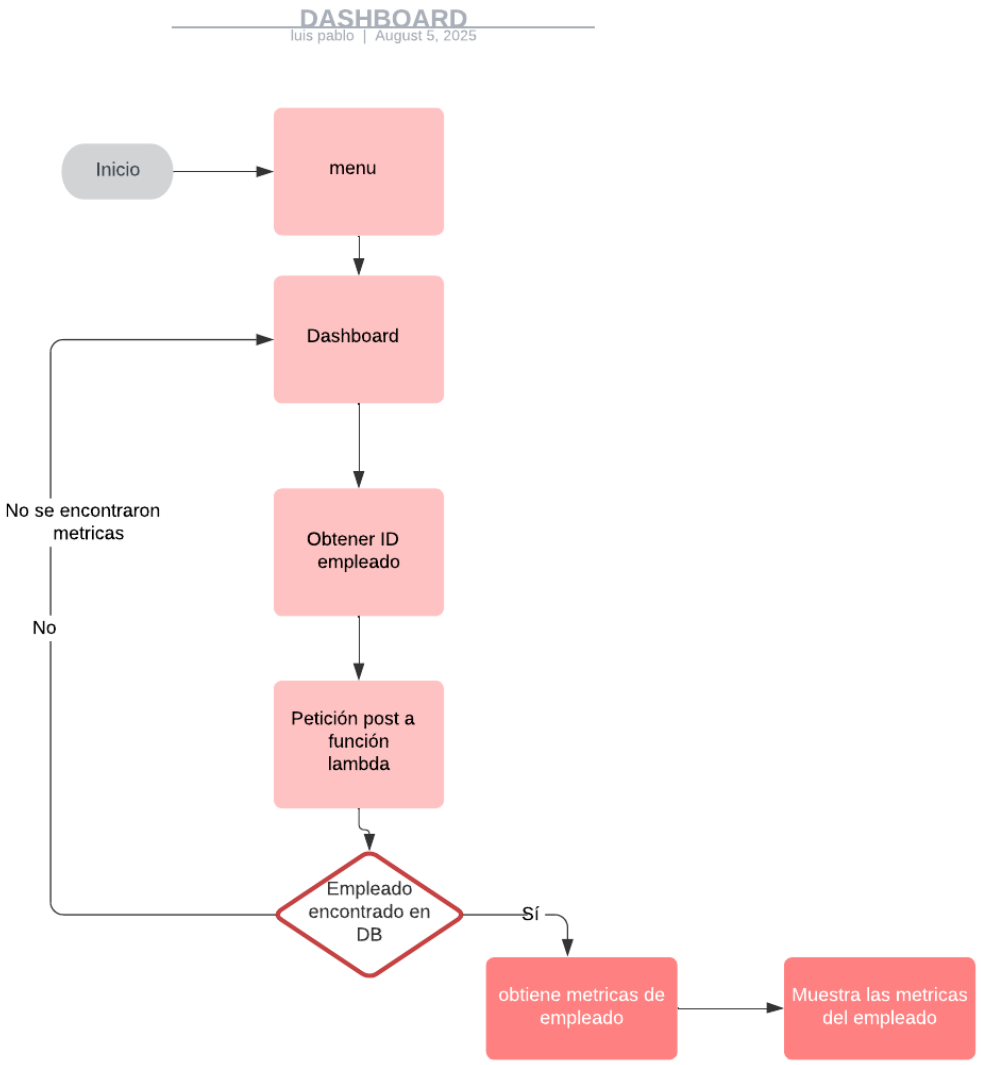
8.3 Crear usuario



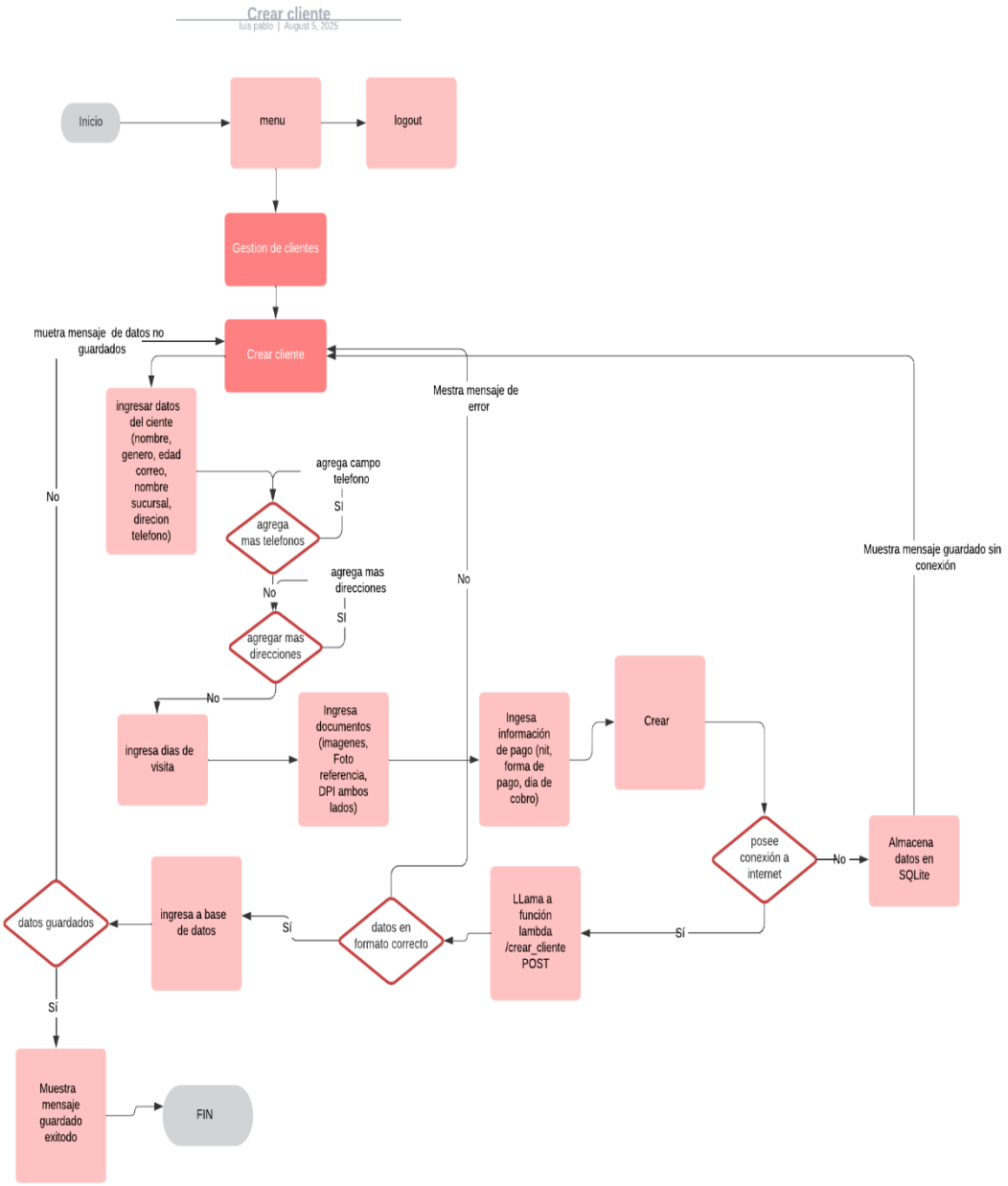
8.4 Sincronización



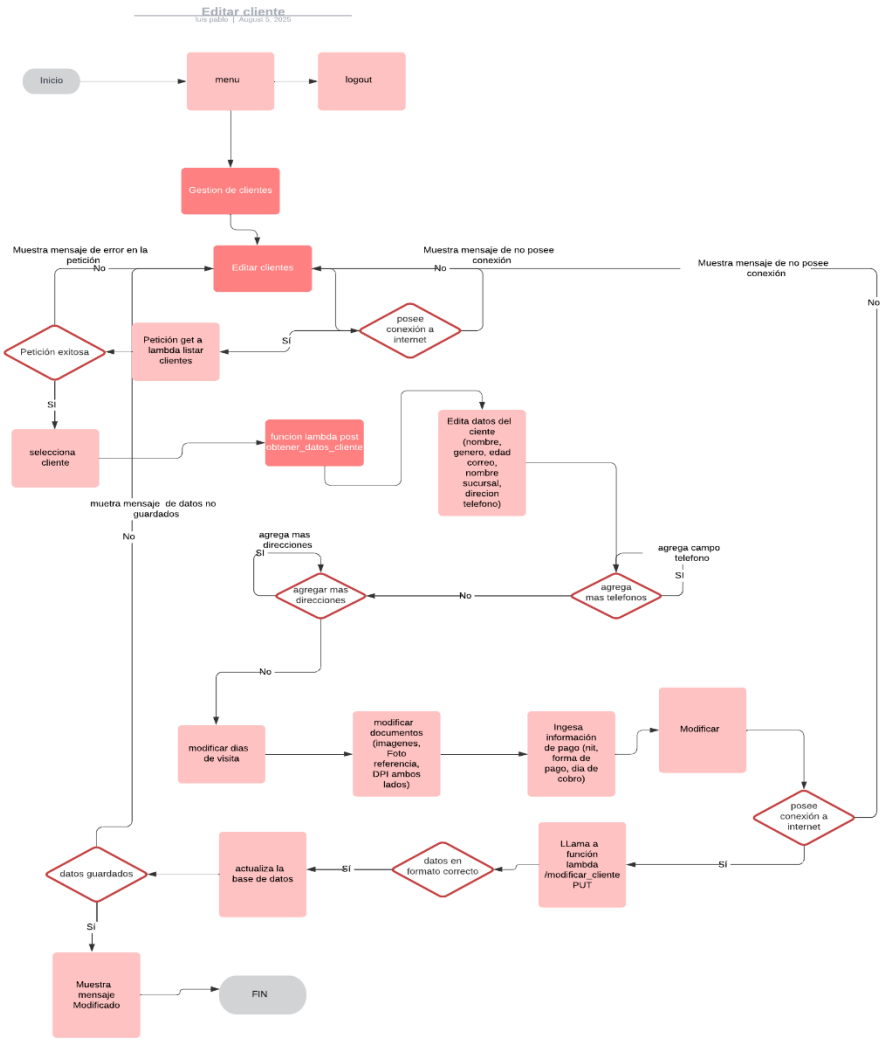
8.5 Dahsboard



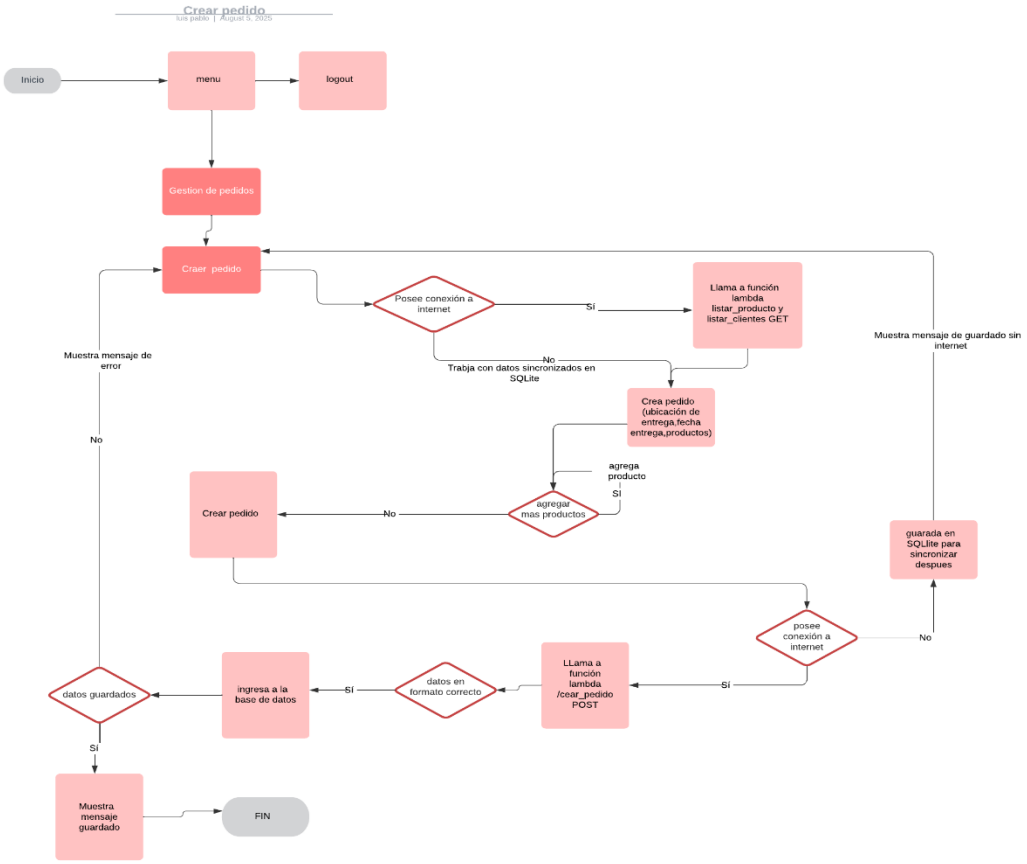
8.6 Crear cliente



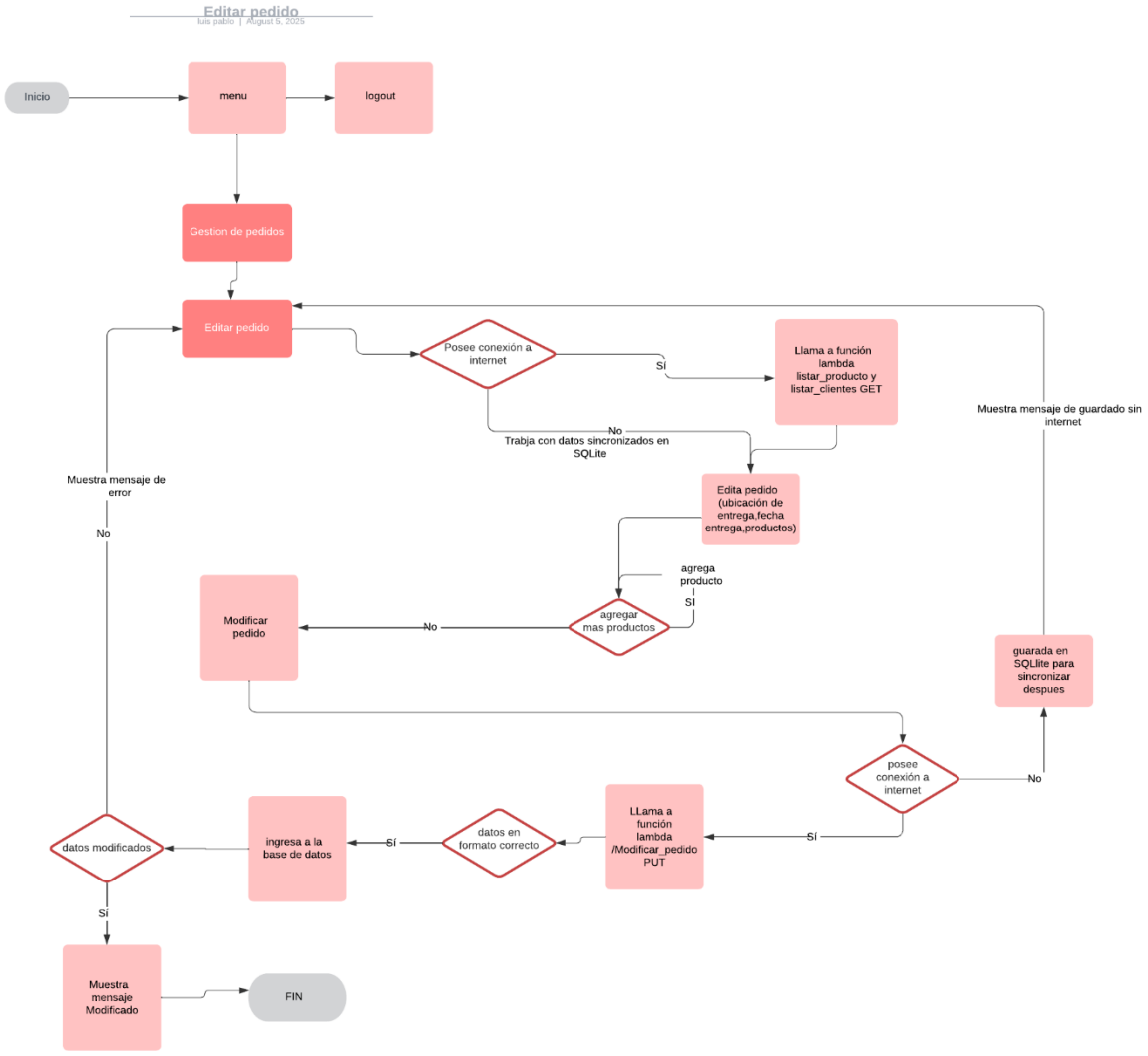
8.7 Editar cliente



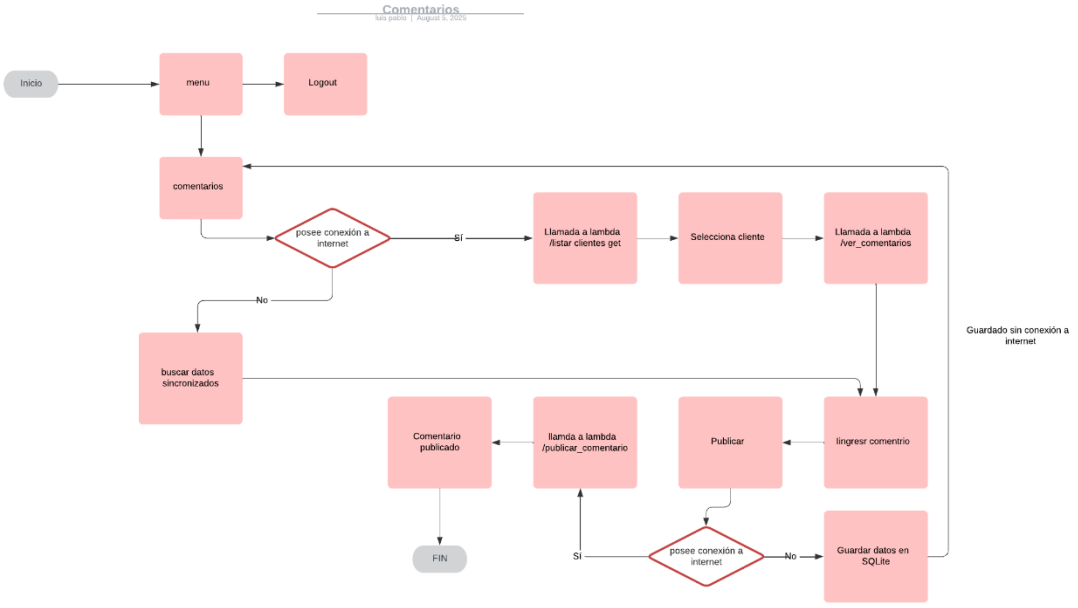
8.8 Crear pedido



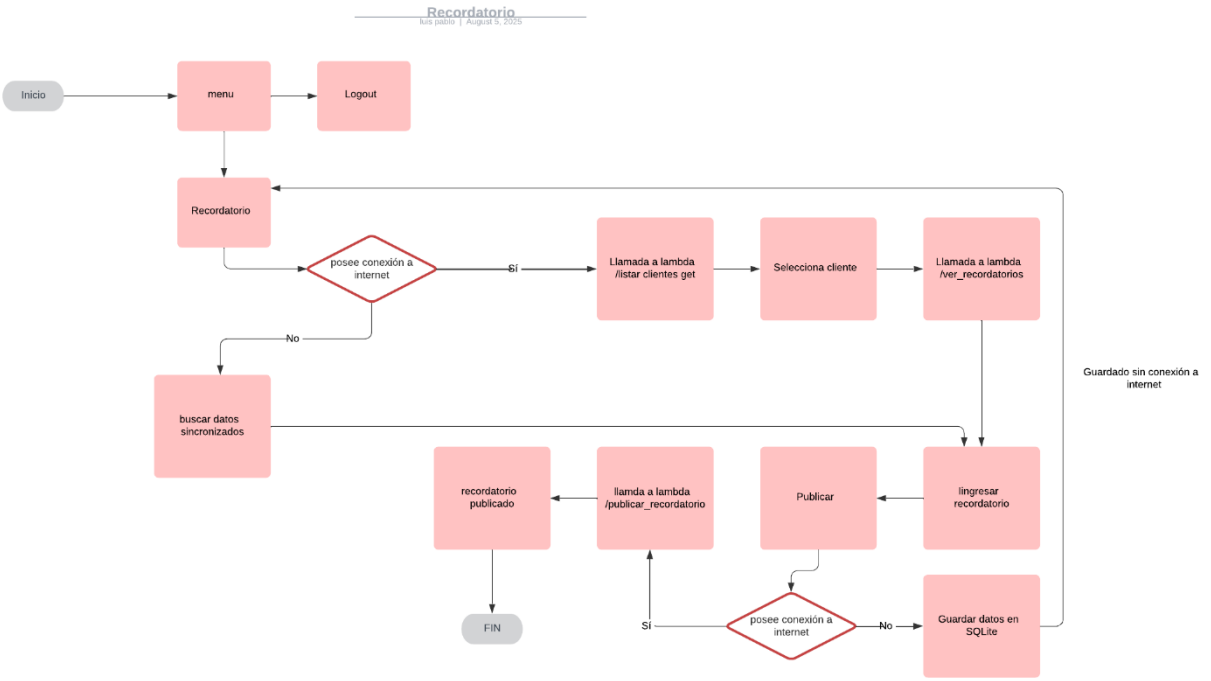
8.9 Editar pedido



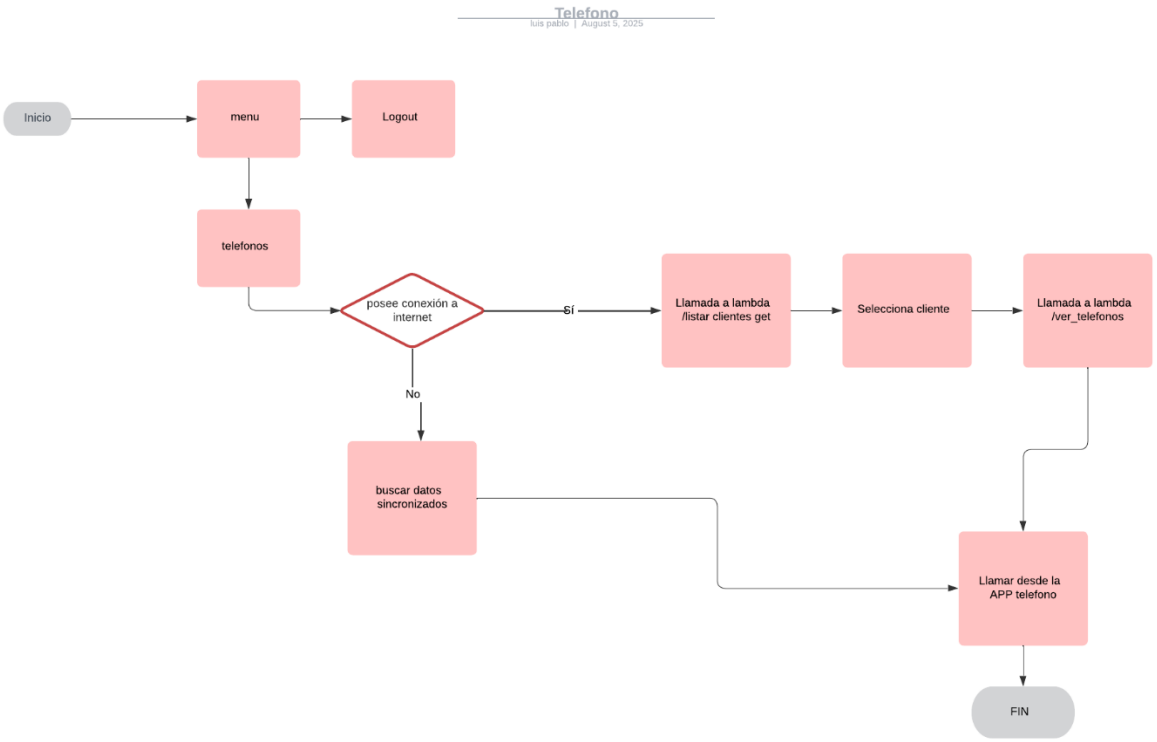
8.10 Comentario



8.11 Recordatorio



8.12 Teléfonos



9 Criterios técnicos de éxito

Los criterios técnicos de éxito son fundamentales para asegurar que el desarrollo del sistema cumpla con los estándares de calidad esperados. Estos criterios se enfocan en validar los siguientes aspectos técnicos durante la fase de pruebas.

- **Cumplimientos de requisitos funcionales**
  - Todas las funcionalidades descritas en los casos de uso deben estar implementadas y operativas.
  - Cada prueba debe demostrar que el sistema responde correctamente ante entradas válidas e invalidas
- **Pruebas unitarias y de integración superadas.**
  - Al menos el 90% del código debe estar cubierto por pruebas unitarias.
  - Las integraciones entre módulos deben ejecutarse correctamente sin errores o pérdidas de datos.
- **Estabilidad y desempeño.**
  - El sistema debe soportar una carga simultánea de usuarios.
  - No debe de existir cuellos de botella que afecten el rendimiento
- **Seguridad y manejo de errores.**
  - Se debe garantizar la protección de datos sensibles.
  - Todas las excepciones deben estar controladas adecuadamente y registrar los errores en los logs del sistema.
- **Validación por parte del usuario.**
  - Los resultados de las pruebas deben ser validados por los stakeholders principales o el product owner.
  - Se debe firmar un acta de validación de pruebas donde se evidencie la conformidad de los resultados obtenidos.

10 CUT - Casos de uso técnicos

Se debe documentar el comportamiento técnico del sistema desde una perspectiva detallada para uso de desarrolladores y arquitectos.

Los CUT representan una visión técnica de los casos de uso del sistema, orientados a clarificar cómo deben implementarse internamente.

CUT - 01	Autenticación de Usuario (Cognito + JWT)
Descripción	El sistema debe permitir la autenticación de usuarios utilizando AWS Cognito. Una vez autenticado, el usuario recibe un token JWT que se incluirá en cada petición a los endpoints protegidos.
Actores	• Usuario final • Servicio Cognito • Middleware de validación de token
Precondiciones	• Usuario registrado en Cognito • Configuración de pool de usuarios y clientes en Amplify
Flujo Principal	• El usuario ingresa sus credenciales. • Cognito valida y devuelve un JWT. • El frontend almacena el token localmente. • El token se envía en cada solicitud para acceder a microservicios.
Validación	• JWT debe ser válido y no estar expirado. • Acceso denegado si token no es válido o no se envía

CUT - 02	Creación de Pedido (Lambda + API Gateway + PostgreSQL)
Descripción	Permite crear un nuevo pedido desde el frontend. El pedido incluye: cliente, productos, ubicación y fecha de entrega. La lógica se procesa en AWS Lambda y los datos se almacenan en una base de datos PostgreSQL.
Actores	• Usuario autenticado • Función Lambda • API Gateway • RDS PostgreSQL
Precondiciones	• Usuario autenticado vía Cognito • Cliente y productos existentes en la base de datos
Flujo Principal	• El usuario selecciona cliente, productos y ubicación. • Se envía el JSON con los datos a un endpoint de API Gateway. • Lambda valida el cuerpo del JSON. • Lambda inserta el pedido en la base de datos. • Se devuelve una respuesta con el ID del pedido.
Validación	• Todos los campos deben tener formato correcto. • Si algún producto no existe, rechazar solicitud.

	• Confirmación de pedido guardado (código 200 + mensaje).

CUT - 03	Listado de Clientes
Descripción	Consulta a la base de datos para obtener una lista de clientes disponibles
Actores	• Usuario autenticado • Función Lambda • API Gateway • RDS PostgreSQL
Precondiciones	• Clientes registrados previamente
Flujo Principal	• El frontend llama al endpoint listar_clientes. • Lambda consulta la tabla de clientes. • Se devuelve un array de objetos con IDs y nombres.
Validación	• Verificar formato de respuesta (statusCode 200 + JSON de clientes). • Si no hay clientes, devolver lista vacía.

CUT - 04	Registro de Ubicaciones por Cliente
Descripción	Permite asociar múltiples ubicaciones (direcciones) a un cliente
Actores	• Usuario autenticado • Función Lambda • PostgreSQL
Precondiciones	N/A
Flujo Principal	• El usuario selecciona un cliente y agrega direcciones. • El JSON es enviado. • Lambda registra cada dirección en la base de datos.
Validación	• Cada objeto en el array debe tener la propiedad direccion. • Respuesta con confirmación de inserción.

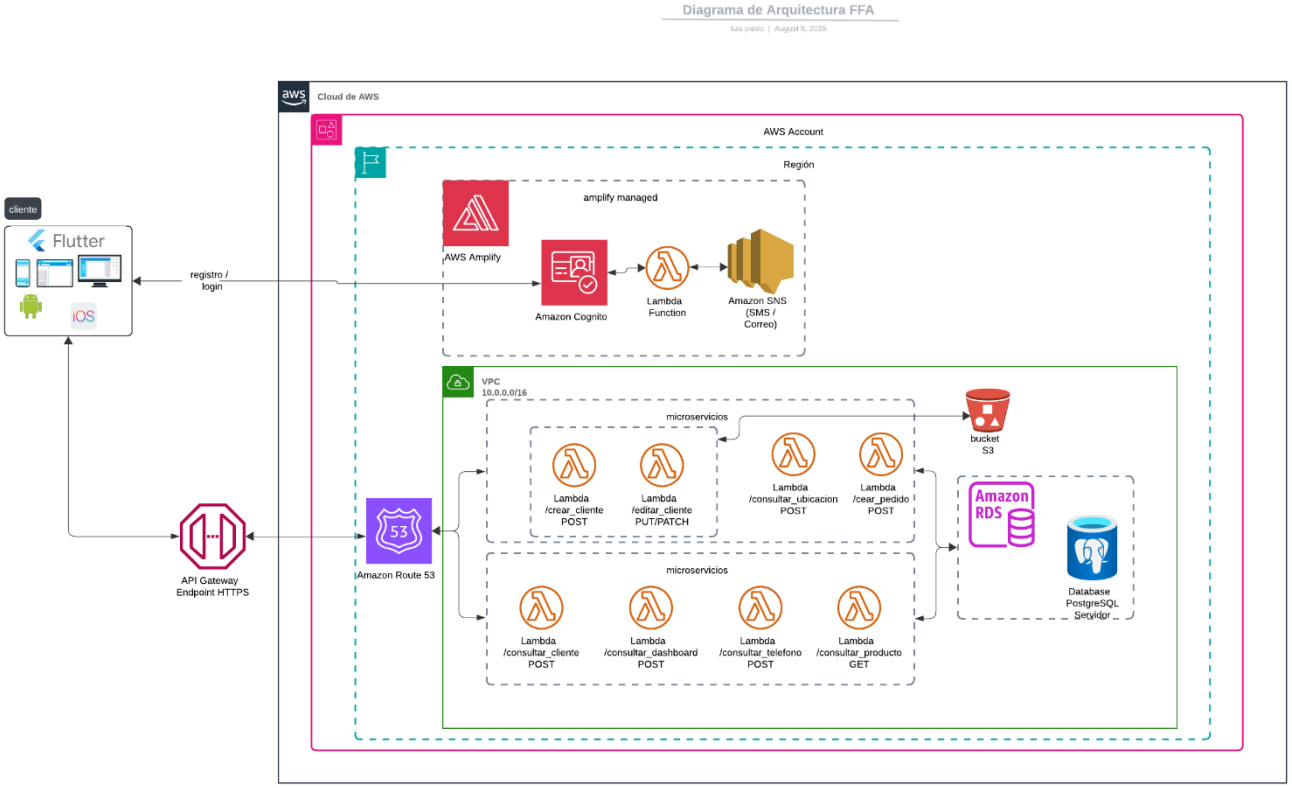
CUT - 05	Listado de Productos Disponibles
Descripción	Consulta los productos que pueden ser seleccionados para un pedido.
Actores	• Usuario autenticado • Lambda • PostgreSQL
Precondiciones	N/A
Flujo Principal	• El usuario consulta los productos. • Lambda responde con la lista de productos en formato:

Validación	• Código de respuesta 200 • Cada producto debe tener nombre y código

CUT - 06	Validación de Pedidos (Testing y QA)
Descripción	Verifica que un pedido creado cumpla con los criterios técnicos y funcionales del sistema.
Actores	• QA • API Gateway • Lambda • Base de Datos
Precondiciones	N/A
Flujo Principal	• QA realiza pruebas automáticas o manuales con casos válidos e inválidos. • Se verifica que la lógica de negocio se aplique correctamente. • Se valida respuesta HTTP, estructura del JSON, y persistencia en la base
Validación	• Respuestas esperadas para cada tipo de prueba • Verificación de registros en la base de datos • Verificación de errores y manejo de excepciones

11 Arquitectura

11.1 Diagrama de arquitectura



12 Listado de componentes

compomente	Descripción
Backend (lógica del servidor)	• Funciones lambda (AWS) : utilizadas como lógica de microservicios. • Lenguaje: Node.js v22 • API GateWay (AWS): Encargado de exponer los endpoints públicos de las funciones Lambda.

Componente	Descripción
Base de datos	• Sistema de gestión de base de datos: PostgreSQL • Motor en la nube: Amazon RDS • Modelo relacional (con uso de JSONB)

Componente	Descripción
Frontend (Aplicación cliente)	• Lenguaje: Dart • Framework: Flutter • Plataformas Android/iOS/Web • Consumo de api REST a través de librería http

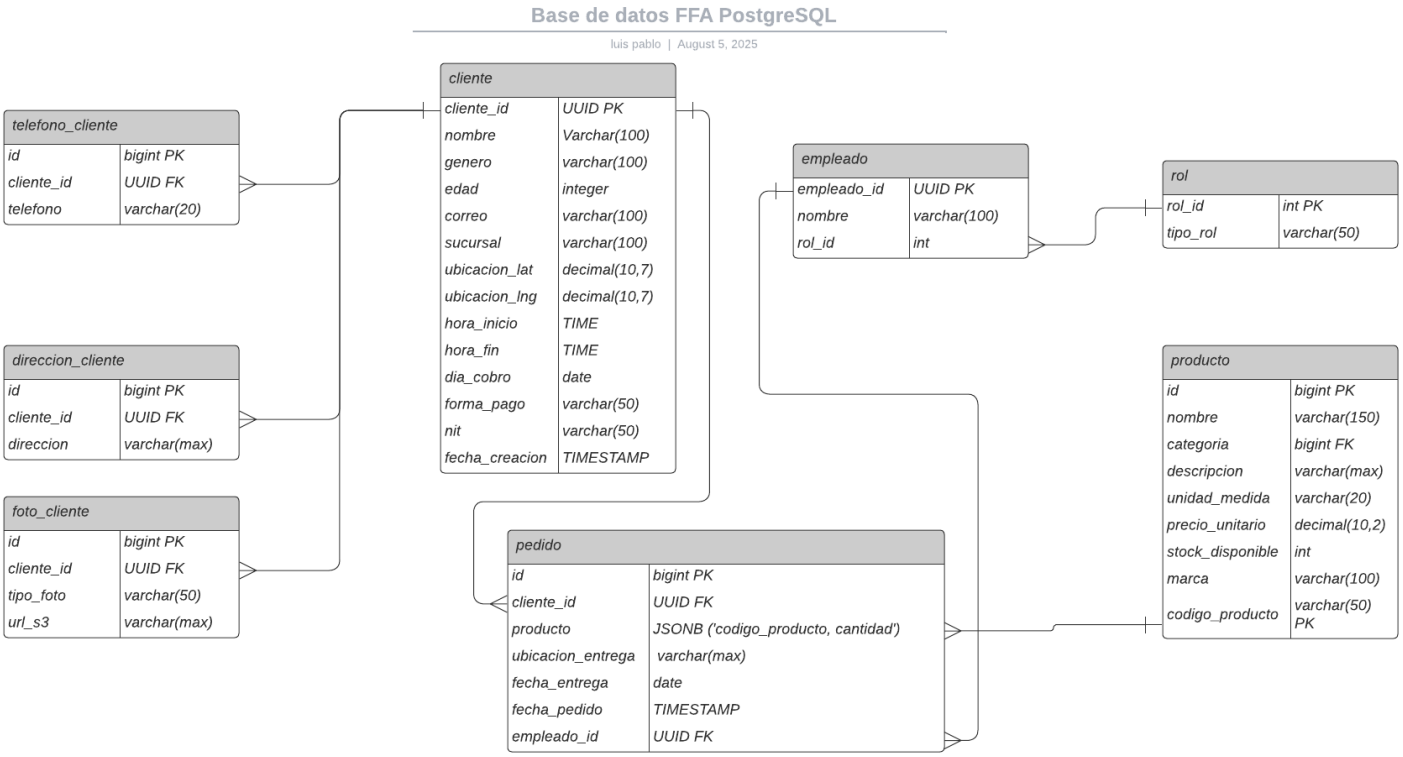
Componente	Descripción
Almacenamiento	• Amazon S3 : Para guardar imágenes o archivos adjuntos.

Componente	Descripción
Seguridad	• Autenticación y autorización: Amazon Cognito • Uso de vpc privada • Uso de Api GateWay HTTPS • Manejo de roles y grupos de usuario integrado con cognito y tokens JWT • AWS SNS para mensajes de recuperación de contraseña y creación de usuario.

Componente	Descripción
Desarrollo y control de versiones	• Repositorio Git (GitHub) • integración continua /despliegue (CI/CD) : AWS amplify CLI • Entorno de desarrollo Visual Studio Code

Componente	Descripción
Otros servicios de AWS	• CloudWatch: Para monitoreo de logs de errores o ejecución de funciones lambda. • IAM para control de acceso entre servidores

13 Diagrama de Base de datos



14 Servicios a usar

Endpoint a usar : https://3p0sm23xj5.execute-api.us-east-1.amazonaws.com

Ruta	/listar_clientes
Método HTTP	GET
Descripción	Lista los clients diponibles
Encabezados de la petición	
Nombre	**Descripción**
N/A	N/A
Códigos HTTP de respuesta	
Código	**Descripción**
200	Petición exitosa
Petición	
N/A	
Respuesta	

```
{
  "clientes": [
    {
      "id": "string (UUID)",
      "nombre": "string"
    },
    ...
  ]
}
```

Ruta	/crear_pedido
Método HTTP	POST
Descripción	Crea el pedido de un cliente
Encabezados de la petición	
Nombre	**Descripción**
headers	"Content-Type": "application/json"
Códigos HTTP de respuesta	
Código	**Descripción**
200	Petición exitosa
Petición	

```
{
  "body": {
    "cliente_id": "uuid",
    "productos": [
      {
        "codigo_producto": "string",
        "cantidad": number
      }
    ],
    "ubicacion": "string",
    "fecha_entrega": "YYYY-MM-DD"
  }
}
```

Respuesta

```
{
  "statusCode": number,
  "body": {
    "mensaje": "string",
    "pedido_id": number
  }
}
```

Ruta	/consultar_ubicacion
Método HTTP	POST
Descripción	Obtiene todas las ubicaciones del cliente
Encabezados de la petición	
Nombre	**Descripción**
headers	"Content-Type": "application/json"
Códigos HTTP de respuesta	
Código	**Descripción**
200	Petición exitosa
Petición	

{
"cliente_id": "uuid",
 	
}	

Respuesta	

{
"ubicacion": [
{
"direccion": "string"
}
]
}	

Ruta	/listar_productos
Método HTTP	GET
Descripción	Lista todos los productos existentes
Encabezados de la petición	
Nombre	Descripción
N/A	N/A
Códigos HTTP de respuesta	
Código	Descripción
200	Petición exitosa
Petición	
N/A	
Respuesta	

```
{
  "productos": [
    {
      "nombre": "string",
      "codigo_producto": "string"
    }
  ]
}
```

15 Sistemas externos

Nombre	ERP
Tipo	Servicio de manejo de clientes

16 Prerrequisitos

- Servidores disponibles de AWS
- Maquetado de vistas Android, IOS, WEB

17 Personal involucrado

Nombre	Rol
Pendiente	Producto
Pendiente	UX/UI
Luis Hernández	Arquitecto de Software
Pendiente	Desarrollador Backend
Pendiente	Desarrollador Frontend
Pendiente	DevSecOps
Pendiente	QA
Pendiente	Soporte

Apéndice

Área exclusiva para el programador

Script de creación de tablas

```sql
CREATE TABLE clientes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    nombre VARCHAR(100) NOT NULL,
    genero VARCHAR(20),
    edad INTEGER,
    correo VARCHAR(100),
    sucursal VARCHAR(100),
    ubicacion_lat DECIMAL(10, 7),
    ubicacion_lng DECIMAL(10, 7),
    hora_inicio TIME,
    hora_fin TIME,
    dia_cobro DATE,
    forma_pago VARCHAR(50),
    nit VARCHAR(20),
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


CREATE TABLE telefonos_cliente (
    id SERIAL PRIMARY KEY,
    cliente_id UUID REFERENCES clientes(id) ON DELETE CASCADE,
    telefono VARCHAR(20) NOT NULL
);


CREATE TABLE direcciones_cliente (
    id SERIAL PRIMARY KEY,
    cliente_id UUID REFERENCES clientes(id) ON DELETE CASCADE,
    direccion TEXT NOT NULL
);


CREATE TABLE fotos_cliente (
    id SERIAL PRIMARY KEY,
    cliente_id UUID REFERENCES clientes(id) ON DELETE CASCADE,
    tipo_foto VARCHAR(50), -- ejemplo: 'referencia', 'dpi_frente', 'dpi_atras'
    url_s3 TEXT NOT NULL
);
```

```sql
CREATE TABLE productos (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(150) NOT NULL,
    categoria VARCHAR(100) NOT NULL,
    descripcion TEXT,
    unidad_medida VARCHAR(20) NOT NULL,
    precio_unitario DECIMAL(10, 2) NOT NULL,
    stock_disponible INT DEFAULT 0,
    marca VARCHAR(100),
    codigo_producto VARCHAR(50) UNIQUE NOT NULL
);


CREATE TABLE pedidos (
    id SERIAL PRIMARY KEY,
    cliente_id UUID REFERENCES clientes(id) ON DELETE CASCADE,
    producto JSONB NOT NULL, -- Ejemplo: { "codigo": "PDCLIM001", "cantidad": 3
}
    ubicacion_entrega TEXT NOT NULL,
    fecha_entrega DATE NOT NULL,
    fecha_pedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


INSERT INTO productos (nombre, categoria, descripcion, unidad_medida,
precio_unitario, stock_disponible, marca, codigo_producto)
VALUES
('Desinfectante Amonio Cuaternario', 'Limpieza', 'Desinfectante de amplio
espectro para uso industrial', 'Galón', 45.00, 50, 'PDC', 'PDCLIM001'),

('Detergente Industrial Alta Espuma', 'Limpieza', 'Detergente líquido ideal para
superficies con grasa', 'Galón', 38.50, 80, 'PDC', 'PDCLIM002'),

('Cloro Industrial 5%', 'Desinfección', 'Cloro concentrado para uso
institucional', 'Litro', 8.75, 200, 'PDC', 'PDCDES001'),

('Desengrasante Multiusos', 'Cocina', 'Desengrasante potente para áreas de
cocina y superficies metálicas', 'Galón', 52.00, 65, 'PDC', 'PDCCOC001'),

('Jabón Líquido para Manos', 'Higiene', 'Jabón dermatológico para uso
frecuente', 'Litro', 12.00, 120, 'PDC', 'PDCHIG001'),

('Suavizante Textil', 'Lavandería', 'Suavizante industrial para lavandería
institucional', 'Galón', 27.50, 30, 'PDC', 'PDCLAV001'),

('Detergente en Polvo', 'Lavandería', 'Detergente en polvo para ropa blanca y de
color', 'Kg', 15.00, 70, 'PDC', 'PDCLAV002');
```