Informe de evaluación técnica y auditoría - IITA CRM

Repositorio: gviollaz/iita-system (rama main) + referencias externas

Alcance: documentación, modelo de datos (Supabase/PostgreSQL), Edge Functions, y evaluación de front-end/Make según repos disponibles

Fecha de auditoría: 2026-02-21 (America/Argentina/Salta)

Autor del informe: GPT-5.2 Pro (asistencia IA)

Nota de alcance: el repositorio de escenarios Make.com (gviollaz/iita-make-scenarios) no es accesible públicamente; el repositorio de front-end (IITA-Proyectos/iitacrm) sí es accesible y fue revisado como referencia del sistema.

Fecha	2026-02-21
Versión	v0.2 (auditoría técnica)
Preparado por	GPT-5.2 Pro (asistencia IA)
Fuentes analizadas	gviollaz/iita-system (main) + IITA-Proyectos/iitacrm (main). Repo Make (gviollaz/iita-make-scenarios) no accesible públicamente.

Índice

Este documento está pensado para generarse automáticamente a partir del repositorio. Si se abre en Word/LibreOffice, se puede insertar/actualizar un índice automático.

[Índice automático - completar al final]

1. Alcance y metodología

Objetivo: evaluar integralmente el repositorio iita-system, verificando alineación con la documentación de referencia (normas de uso y modelo de datos), y revisando su implementación en: (a) front-end, (b) estructura y modelo de datos, (c) backend de automatizaciones en Make (blueprints/escenarios).

Metodología sugerida:

- Inventario del repositorio (estructura, dependencias, entornos, scripts).
- Lectura de documentación de referencia y extracción de reglas verificables.
- Revisión estática del código (lint/typecheck, búsqueda de TODO/FIXME, patrones de error, manejo de estados).
- Verificación del modelo de datos (schemas/migraciones, validaciones, consistencia con el front-end y Make).
- Revisión de escenarios Make (blueprints JSON): entradas/salidas, webhooks, idempotencia, reintentos, manejo de errores y logging.
- Chequeo de seguridad (secretos, autenticación, autorización, OWASP básico, hardening de webhooks).
- Síntesis en hallazgos priorizados con recomendaciones y plan de remediación.

2. Resumen ejecutivo

Estado general (según el contenido actual del repo):

- El repositorio NO está aún "corregido" como repo unificado: faltan los directorios esperados de front-end (apps/crm-frontend) y automatizaciones (automations/make-scenarios) que la propia documentación menciona. Esto impide reproducir el sistema completo desde este repo.
- El esquema SQL versionado está parcialmente incompleto (varias funciones RPC clave aparecen como placeholders), y hay Edge Functions listadas sin código.
- Riesgo principal: exposición de datos por API Edge (service role + CORS abierto + ausencia de JWT según documentación), con potencial impacto de seguridad y cumplimiento.

Quick wins (1-3 días):

1. Incorporar (o subtreear) el código del CRM (IITA-Proyectos/iitacrm) y export de escenarios Make al repo iita-system o, alternativamente, corregir la documentación para reflejar el esquema multi-repo real.
2. Reemplazar placeholders SQL por implementaciones reales (o marcar explícitamente qué es "source of truth" y dónde está versionado).
3. Cerrar el agujero de seguridad del endpoint crm-api: activar autenticación, limitar endpoints/acciones y aplicar principio de menor privilegio.

Recomendaciones de alto impacto (1-4 semanas):

- Formalizar CI/CD mínimo (lint/build, pruebas, checks de seguridad) y un proceso de migraciones con snapshots verificables.
- Definir contrato API estable (OpenAPI/JSON Schema) entre CRM y backend, y versionarlo.
- Normalizar y alinear documentación (AGENTS.md / REPOSITORY-PHILOSOPHY.md / README) para eliminar contradicciones.

3. Inventario del repositorio

Estructura detectada (resumen):

Estructura de alto nivel observada:

- AGENTS.md / CLAUDE.md: contexto operativo, normas y advertencias del sistema.
- database/
- edge-functions/: código Deno de Edge Functions (crm-api; otras en estado incompleto).
- schema/: funciones SQL (RPC) y piezas de esquema (en este repo se ven principalmente funciones).
- docs/
- architecture/: overview y flujos (message-flow).
- bugs/: bugs conocidos (incluye hallazgos de seguridad).
- database/: diccionario de datos y ERD.
- features/: features y roadmap.
- make-pipeline/: descripción del pipeline de 117 escenarios (no incluye blueprints en este repo).
- operations/: log de migraciones y operaciones de datos.
- policies/: convenciones (naming, colaboración con IA, filosofía).
- proposals/: RFCs y propuestas de cambios.

Brecha vs lo esperado por la propia documentación:

- No existe en este repo: apps/crm-frontend/ (front-end).
- No existe en este repo: automations/make-scenarios/ (escenarios Make).

Tecnologías y toolchain:

Tecnologías principales (según docs y fuentes revisadas):

- Front-end: React + Vite (en repo externo IITA-Proyectos/iitacrm).
- Automatizaciones: Make.com (117 escenarios según documentación; blueprints no versionados aquí).
- Backend/DB: Supabase (PostgreSQL) + Edge Functions (Deno).
- Integración: webhooks Make → RPC SQL (process_incoming_message/process_outgoing_message) → triggers → Make + Edge Function crm-api como API HTTP para el CRM.

Toolchain y automatización:

- En iita-system no se observan workflows de CI/CD (GitHub Actions) ni scripts de verificación automáticos.
- Se documenta un enfoque "migraciones aplicadas en Supabase" con log manual (docs/operations/MIGRATIONS-LOG.md).

Convenciones y normas:

Normas y convenciones relevantes:

- docs/policies/NAMING-CONVENTIONS.md: define snake_case, prefijos, y documenta desviaciones históricas (campos con typos o enums heredados).
- docs/policies/AI-COLLABORATION.md: exige trailers y atribución para trabajo asistido por IA.
- docs/policies/REPOSITORY-PHILOSOPHY.md: declara intención de repositorio unificado y sincronización por git subtree, pero el estado actual del repo no refleja aún esa unificación.

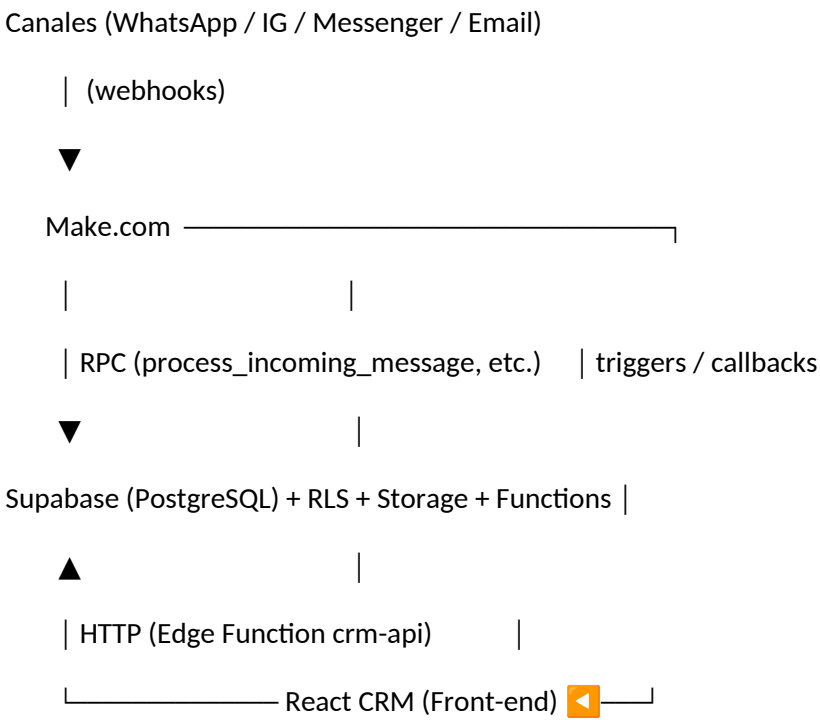
Observación clave:

- Hay contradicción entre "repo unificado" vs "sistema distribuido en 3 repos" y la estructura real publicada. Recomiendo elegir una fuente de verdad y alinear TODO (docs y estructura).

4. Arquitectura del sistema

Diagrama lógico (alto nivel):

Diagrama lógico (alto nivel) - texto:



Notas:

- El CRM consume endpoints HTTP que, en la implementación actual, delegan en RPC SQL y operaciones CRUD.
- El pipeline de IA ocurre principalmente en Make.com; la DB registra ai_interaction y estados de envío.

Flujos críticos:

- Autenticación / onboarding
- Creación/edición de entidades principales
- Sincronización con servicios externos
- Notificaciones y automatizaciones

5. Evaluación del front-end

5.1 Estructura y mantenibilidad

- Organización de carpetas (features vs layers), modularidad, duplicación.
- Gestión de estado (global/local), side-effects, caching, invalidación.

- Enrutado, protección de rutas, lazy-loading.

Hallazgos principales (front-end) basados en el repo IITA-Proyectos/iitacrm:

- La UI depende fuertemente de endpoints del backend (stats, msgs_per_day, volume_by_channel, channel_analysis, persons_enriched, person_full, etc.). Si las RPC están como placeholders en iita-system, la UI queda sin datos o con pantallas vacías.
- Se observan mitigaciones puntuales de bugs (comentarios tipo “BUG-002 FIX”) y guardas anti doble click (aiSavingRef), lo cual es bueno, pero sugiere deuda técnica previa.
- El front-end actual edita personas mediante un endpoint genérico de update (post({ action: 'update', table: 'persons', … })), lo que acopla la seguridad del sistema a que ese backend esté correctamente autenticado y con allowlist de tablas/campos.

5.2 Calidad (DX)

- Scripts de build/test/lint, configuración de TypeScript (si aplica).
- Tests unitarios/integración/e2e y cobertura en rutas críticas.
- Observabilidad: logs, manejo de errores en UI, trazabilidad.

Hallazgos principales (front-end) basados en el repo IITA-Proyectos/iitacrm:

- La UI depende fuertemente de endpoints del backend (stats, msgs_per_day, volume_by_channel, channel_analysis, persons_enriched, person_full, etc.). Si las RPC están como placeholders en iita-system, la UI queda sin datos o con pantallas vacías.
- Se observan mitigaciones puntuales de bugs (comentarios tipo “BUG-002 FIX”) y guardas anti doble click (aiSavingRef), lo cual es bueno, pero sugiere deuda técnica previa.
- El front-end actual edita personas mediante un endpoint genérico de update (post({ action: 'update', table: 'persons', … })), lo que acopla la seguridad del sistema a que ese backend esté correctamente autenticado y con allowlist de tablas/campos.

5.3 Seguridad y datos

- Uso de variables de entorno en el cliente.
- Validaciones de formularios y sanitización.
- Manejo de tokens/sesiones, expiración, refresh, storage.

Hallazgos principales (front-end) basados en el repo IITA-Proyectos/iitacrm:

- La UI depende fuertemente de endpoints del backend (stats, msgs_per_day, volume_by_channel, channel_analysis, persons_enriched, person_full, etc.). Si las RPC están como placeholders en iita-system, la UI queda sin datos o con pantallas vacías.
- Se observan mitigaciones puntuales de bugs (comentarios tipo “BUG-002 FIX”) y guardas anti doble click (aiSavingRef), lo cual es bueno, pero sugiere deuda técnica previa.
- El front-end actual edita personas mediante un endpoint genérico de update (post({ action: 'update', table: 'persons', … })), lo que acopla la seguridad del sistema a que ese backend esté correctamente autenticado y con allowlist de tablas/campos.

6. Modelo de datos y normas de uso

Fuentes de verdad del modelo (esperado):

- Documentación de referencia (ERD/diagrama, tablas/colecciones, campos, tipos).

- Schemas (JSON Schema/Zod/Yup), tipos TS, migraciones SQL, o definiciones equivalentes.
- Validaciones en front-end y en automatizaciones Make.

6.1 Entidades y relaciones

Entidades núcleo (según AGENTS.md + ERD + data dictionary):

- persons (persona/lead/alumno) 1..N person_conversation
- conversations (hilo) 1..1 person_conversation y 1..1 system_conversation (modela "lado persona" vs "lado sistema")
- interactions (mensajes): referencian id_person_conversation (entrante) o id_system_conversation (saliente); estados de procesamiento (new→...→send).
- ai_interaction: respuestas generadas por IA, evaluadas y aprobadas/rechazadas.
- channels / channel_providers: catálogo de canales y su proveedor (WhatsApp, Instagram, etc.).
- person_soft_data + person_enrichment_log: datos enriquecidos y su trazabilidad.
- cursos: courses / course_editions (catálogo y cohortes).

Relaciones relevantes:

- La "dirección" (entrante/saliente) se deduce por qué FK está presente (id_person_conversation vs id_system_conversation) y por constraint asociado (documentado).
- Triggers en interactions disparan webhooks a Make.com para preprocesar media y generar IA.

6.2 Reglas de integridad y validación

- Campos obligatorios/nullable, defaults, unicidad.
- Transiciones de estado permitidas.
- Reglas de negocio (derivaciones, cálculos, restricciones).

Comparación documentación vs implementación (muestra):

- Docs/Features y front-end esperan endpoints de "People Enriched" (persons_enriched, persons_filter_options, person_full). En este repo, varias funciones RPC están versionadas como placeholders (retornan {} o listas vacías), lo que sugiere drift: o la producción tiene otro SQL no versionado aquí, o la feature aún no está realmente implementada en el código fuente.
- Docs declaran Edge Functions adicionales (courses-crud, create-test-user) pero los archivos index.ts en este repo aparecen vacíos, lo que debilita la trazabilidad.
- Docs declaran repositorio unificado con subtree, pero la estructura publicada no contiene los subtrees. Esto rompe la promesa “cualquier IA puede ver todo” y dificulta auditoría y onboarding.

7. Backend en Make

Artefactos esperados en el repositorio:

- Blueprints/export JSON de escenarios.
- Documentación de cada escenario: propósito, trigger, módulos, contratos de datos.
- Variables/secretos (no en claro) y guía de configuración por entorno (dev/stage/prod).

7.1 Escenarios (inventario)

Inventario (limitado) de Make.com:

- Según docs/make-pipeline/pipeline-overview.md: existen 117 escenarios, con ~43 activos, organizados en 8 etapas (ingestión, preprocesamiento, IA, aprobación, envío, analíticas, mantenimiento, utilidades).
- En este repo NO están los blueprints JSON exportados ni el inventario por escenario (archivo scenarios-analysis.md se referencia pero no existe).

Recomendación de inventario mínimo (a incluir en el repo):

- Para cada escenario: nombre, ID, trigger (webhook/scheduler), entradas esperadas, salidas (tablas que escribe / endpoints que llama), errores/reintentos, y owner.

7.2 Robustez

- Idempotencia: evitar duplicados ante reintentos.
- Manejo de errores: rutas de error, alertas, dead-letter/cola manual.
- Rate limits y backoff, límites de Make y APIs externas.
- Versionado de escenarios y despliegue entre entornos.
- Logs y trazabilidad (IDs de correlación).

Robustez Make (evaluación por documentación; sin acceso a blueprints):

- Idempotencia: la documentación y el log de operaciones indican que existieron duplicados (mensajes eco, external_ref duplicados). Esto sugiere que los escenarios y/o la DB necesitan llaves únicas e idempotency keys consistentes en TODOS los webhooks.
- Manejo de errores y reintentos: no es verificable sin blueprints. Debe documentarse una política: backoff, dead-letter, alertas.
- Versionado: sin export JSON en repo, no hay manera de auditar cambios ni hacer rollback confiable.

Acción recomendada:

- Exportar todos los escenarios a JSON, versionarlos, y crear un script de validación (schema + lint) en CI.

8. Seguridad

- Escaneo de secretos: API keys, tokens, credenciales, URLs sensibles.
- Hardening de webhooks: autenticación (HMAC/secret), allowlist, validación de firma.
- Principio de menor privilegio en conexiones.
- Protección de datos personales: minimización, retención, registros de auditoría.

Hallazgos y acciones prioritarias (seguridad):

P0 - Cerrar exposición de API:

- La Edge Function crm-api usa service role y, según BUGS-CONOCIDOS, corre sin verify_jwt. Combinado con un endpoint genérico de CRUD, equivale a exponer la DB completa a Internet.
- Acción: activar verify_jwt, implementar autenticación y autorización real (roles), y eliminar o restringir "action/table" a un allowlist mínimo.

P0 - Alinear RLS:

- BUGS-CONOCIDOS reporta policies permisivas (USING true). Si es así en producción, incluso el anon key podría leer/escribir PII.
- Acción: diseñar RBAC y policies por rol; testear con supabase auth.

P1 - Seguridad de webhooks Make:

- Validar firma/secret en webhooks entrantes. Registrar correlation_id. Limitar CORS/orígenes si aplica.

P1 - Gestión de secretos:

- Garantizar que service_role y tokens de Make sólo existan en secretos de entorno, nunca en repos.

9. Calidad, pruebas y CI/CD

- Pipeline (GitHub Actions/GitLab/etc.): build, tests, lint, security checks.
- Estrategia de versionado y releases.
- Ambientes (dev/stage/prod): configuración y paridad.
- Monitorización: errores, performance, alertas.

Calidad, pruebas y CI/CD:

- En iita-system no se ve un pipeline de CI (lint/test/security). Recomendado: GitHub Actions con:
- checks de markdown (links), lint SQL (si aplica), gitleaks, y escaneo de dependencias.
- para el front-end (si se incorpora): npm ci + build + lint + tests.
- Añadir pruebas mínimas de RPC SQL (casos de idempotencia, constraints, estados) y contratos API.
- Añadir control de cambios de esquema: snapshots o migraciones versionadas (ideal: Supabase CLI + migrations).
- Observabilidad: estandarizar correlation_id (Make ⇄ DB ⇄ CRM) y logs estructurados.

10. Hallazgos

Convención sugerida: severidad P0 (crítico) a P3 (bajo).

Hallazgos priorizados:

- F-001 (P0 - Repo) - Repo incompleto vs docs (no apps/crm-frontend ni automations/make-scenarios).
  - Evidencia: README.md / AGENTS.md / árbol raíz
  - Acción: Incorporar subtrees/submodules o corregir docs; fijar versiones con tags.
- F-002 (P0 - DB) - RPCs clave versionadas como placeholder (People/Analytics).
  - Evidencia: get_channel_analysis.sql, get_person_full_profile.sql, get_persons_enriched.sql, get_persons_filter_options.sql
  - Acción: Subir implementación real + migraciones reproducibles + tests.
- F-003 (P0 - Seguridad) - crm-api con service role + acciones CRUD genéricas; docs reportan verify_jwt desactivado.
  - Evidencia: edge-functions/crm-api/index.ts + BUGS-CONOCIDOS
  - Acción: Activar JWT, allowlist estricto, RBAC, revisar exposición y rotar secretos.

- F-004 (P1 - Edge) - Edge functions adicionales sin implementación versionada.
    - Evidencia: courses-crud/index.ts, create-test-user/index.ts
    - Acción: Versionar código real o eliminar referencias; definir contrato de API.
- F-005 (P1 - DB) - Riesgo de colisión de identidad: resolución de persona por address sin discriminar proveedor/canal.
    - Evidencia: process_incoming_message.sql
    - Acción: Incluir channel_provider_id en llave; constraints/índices; tests de colisión.
- F-006 (P1 - Docs) - Contradicciones multi-repo vs repo-unificado + referencias a archivos inexistentes.
    - Evidencia: REPOSITORY-PHILOSOPHY.md + AGENTS.md + README.md
    - Acción: Alinear documentación; agregar link checker en CI.
- F-007 (P1 - Make) - Sin export JSON versionado no se auditan reintentos/errores; historial de duplicados sugiere falta de idempotencia.
    - Evidencia: pipeline-overview.md + DATA-OPERATIONS.md
    - Acción: Exportar y versionar escenarios; diseñar idempotency keys end-to-end.
- F-008 (P2 - Rendimiento) - Queries analíticas potencialmente costosas sin garantías de índices.
    - Evidencia: get_volume_by_provider.sql, get_conversations.sql
    - Acción: Revisar EXPLAIN; índices; materializaciones para dashboard.
- F-009 (P2 - Observabilidad) - Sin correlation_id estándar end-to-end (Make⇄DB⇄CRM).
    - Evidencia: docs/make-pipeline + crm-api
    - Acción: Propagar correlation_id y logs estructurados; alertas.
- F-010 (P2 - Calidad) - Ausencia de CI/CD y pruebas automatizadas en iita-system.
    - Evidencia: repo raíz
    - Acción: Agregar workflows (lint/test/gitleaks/links) y política de PRs.

Nota: la evidencia detallada por archivo y referencias cruzadas se desarrollan en las secciones 3 a 9.

11. Bugs y riesgos potenciales (checklist)

- Inconsistencia entre el modelo de datos documentado y lo que el front-end/envía/espera.
- Escenarios Make sin manejo de error (ruta de error vacía o silenciosa).
- No idempotencia en escenarios disparados por webhook (duplicados ante reintentos).
- Validaciones solo en UI (fácil de saltar) sin validación en backend/Make.
- Variables de entorno faltantes o incoherentes entre entornos.
- Secretos versionados en el repo o expuestos en logs.
- Dependencias con vulnerabilidades conocidas sin mitigación.
- Falta de tests en flujos críticos; regresiones difíciles de detectar.

Bugs documentados y observados (resumen):

- BUG-R001 (2026-02-20): get_chat_detail no devolvía media_url (migración fix_media_url_in_get_chat_detail).
- BUG-R002 (2026-02-19): mensajes eco / duplicados salientes (trigger prevent_echo_interaction + limpieza).
- BUG-R003 (2026-02-18/19): conversaciones duplicadas por condiciones de carrera (trigger prevent_duplicate_conversation).
- BUG-UI-002: envío desde CRM requería system_conversation_id (fix comentado en Conversations.jsx).
- BUG-SEC-010/011: RLS permisivo y Edge Functions sin verify_jwt (riesgo crítico).
- BUG-DRIFT-001: funciones de personas/analytics placeholder en repo, rompiendo features de People/Analysis.

12. Plan de remediación

Backlog priorizado (ejemplo):

- P0 (inmediato, 0-48h) - Bloquear crm-api: activar verify_jwt, requerir Authorization, eliminar CRUD genérico o limitar a allowlist, y revisar CORS.
- P0 (0-72h) - Rotar secretos: service_role, tokens Make y cualquier webhook secret; auditar logs por posible exposición.
- P0 (1 semana) - Reproducibilidad: incorporar front-end y blueprints Make al repo iita-system (subtree/submodule) o corregir documentación para reflejar el multi-repo real.
- P1 (1-2 semanas) - Completar esquema: reemplazar placeholders SQL por implementaciones reales; versionar migraciones con Supabase CLI; agregar rollback verificable.
- P1 (1-2 semanas) - Integridad/idempotencia: constraints de unicidad (external_ref por conversación), revisión de process_incoming_message para evitar colisiones por proveedor/canal.
- P2 (2-4 semanas) - Observabilidad: correlation_id end-to-end; dashboards de errores; alertas en Make.
- P2 (2-4 semanas) - CI/CD mínimo: lint/build/test + gitleaks + link checker; política de PRs y releases etiquetados.

Apéndice A - Comandos de verificación

Estos comandos se adaptan según el stack real detectado:

- Front-end (Node): npm ci / pnpm i; npm run lint; npm run test; npm run build
- Seguridad: npm audit; trivy fs .; gitleaks detect
- Formato: prettier --check; eslint
- Back-end (si aplica): pruebas unitarias, typecheck, y verificación de migraciones
- Make: validar blueprints JSON; documentar contratos de entrada/salida

Apéndice B - Artefactos recomendados para incluir en el repo

- docs/arquitectura.md (diagrama + decisiones)
- docs/modelo-datos.md (ERD + reglas + ejemplos)
- docs/normas-uso.md (convenciones, flujos, estilo de código, deploy)
- make/blueprints/ (export JSON por escenario)
- make/README.md (cómo importar, configurar, variables, conexiones)

- .env.example con variables mínimas (sin secretos)
- Scripts de verificación en package.json / pipeline CI