PROJECT_ARCHITECTURE

Project Architecture: RivanIT - Agentic Delivery Platform

1. Project Overview

What the project does

The Agentic Delivery Framework (branded as RivanIT) is a multi-agent enterprise platform for project management. It provides end-to-end project lifecycle management: clients, projects, contracts, financials, resources, risks, staffing/recruitment (Dream Keepers), and integrations with BambooHR, Jira, Pod Journey, and GitHub.

Core features and responsibilities

- **Client and project management**: CRUD for clients and projects; project types (fixed-price, staff-augmentation, time-material, agentic-acceleration); phase and deliverable tracking; PM assignment.
- **Contracts and financials**: Contract creation, approval workflow, milestones, payment terms; P&L, budget, cost tracking, invoices.
- **Resources and allocations**: Resource management, allocation to projects, timelines, Gantt views.
- **Risk and change management**: Risk matrix, mitigation plans, issues/dependencies, change request tracking and notifications.
- **PODs (Project-Oriented Delivery)**: POD creation, constitution, member assignment, sync with Pod Journey.
- **Staffing and recruitment (Dream Keepers)**: Candidate pipeline (BambooHR), questionnaires, English tests (EF Set), speaking tests (SpeechAce), TestGorilla; AI classification and interview workflows; Dream Keeper interview invitations.

- **User and permissions**: Role-based access (director, pm, pod-leader, member); domain-based auth (Google OAuth, Microsoft / Azure AD, magic links); permission matrix; user management; API keys.
- **Notifications**: In-app notifications (risk, change request, PM/POD assignment); email via Brevo.
- **Integrations**: BambooHR (webhooks, full sync, interviews), Jira (proxy, sync), Pod Journey (webhook, sync), GitHub metrics.

Tech stack

- **Frontend**: React 18, TypeScript, Vite, React Router, TanStack Query, Framer Motion, React Hook Form, Zod.
- **UI**: shadcn/ui (Radix primitives), Tailwind CSS, Lucide icons, Recharts, Sonner/Toaster.
- **Backend / data**: Supabase (PostgreSQL, Auth, Realtime, Storage, Edge Functions).
- **External**: BambooHR API, Jira API, Pod Journey API, Brevo (email), Mistral AI (classification/analysis), SpeechAce, TestGorilla, EF Set.

How frontend, backend, and edge functions interact

- **Frontend** talks to Supabase via `@supabase/supabase-js` (client) for Auth, PostgREST (tables/views), and RPC. It also calls Edge Functions over HTTP (Bearer JWT or X-API-Key) for emails, webhooks, syncs, and AI.
- **Supabase** provides: PostgreSQL (schema, RLS, migrations), Auth (email/password, OAuth, magic link), and Edge Functions (Deno).
- **Edge Functions** are triggered by HTTP (frontend or external webhooks), use the service role for privileged DB access, call Brevo/Mistral/BambooHR/Jira/Pod Journey, and often respond with JSON.

Overall data flow

1. User authenticates (Supabase Auth); AuthContext loads user profile and permissions from `users` and `user_permissions`.

2. Pages and components use hooks (e.g. useProjects, useFinancials) that call lib services; services use Supabase client (and sometimes Edge Functions).
3. External events (BambooHR webhook, Pod Journey webhook) hit Edge Functions; functions update DB and may send emails or call other APIs.
4. Notifications are written to DB and read by the frontend via hooks (e.g. useNotifications); email is sent via Edge Functions using Brevo.

---

2. High-Level Architecture

Application layers

- **UI**: Pages and components in `src/pages`, `src/components` (layout, feature components, `src/components/ui` for primitives).
- **State and data**: React Query (server state), AuthContext (user/session), local component state.
- **Business logic**: Hooks in `src/hooks` (e.g. useProjects, useFinancials) and services in `src/lib` (e.g. ProjectService, ContractService).
- **Data access**: Supabase client in `src/lib/supabase.ts`; services call `supabase.from()`, `supabase.rpc()`, and fetch Edge Functions.
- **Integrations**: Edge Functions for BambooHR, Jira, Pod Journey, email, AI; frontend uses `ApiService.callEdgeFunction` or direct fetch.

Key architectural patterns

- **Service layer**: Domain logic in static classes (e.g. ProjectService, ClientService) that encapsulate Supabase and RPC calls.
- **Hooks over services**: Feature hooks (useProjects, useContracts) wrap services and React Query for loading/error/cache.
- **RLS and RPC**: Row Level Security on tables; RPCs like `get_clients`, `get_project_contracts`, `create_contract` used where RLS bypass or complex logic is needed.
- **Domain-driven types**: `src/types` holds interfaces for entities (project, client, contract, user, etc.); services and components import these.

Coordination of state, API calls, and edge functions

- **Auth**: AuthContext initializes session, fetches/creates user profile and permissions; local dev can skip auth and use a mock user.
- **Server state**: TanStack Query with stable query keys (e.g. `['projects', user?.id, includeDetails]`); stale/gc times set in QueryClient and per-hook.
- **Edge Functions**: Called via `ApiService.callEdgeFunction()` or direct `fetch` to `SUPABASE_URL/functions/v1/<name>` with Bearer or X-API-Key; used for email, sync, webhooks, AI.

---

3. Folder-by-Folder Breakdown

Folder: /

Purpose: Repository root; config and entry files for app, build, and tooling.

Contains: package.json, vite.config.ts, tsconfig*.json, index.html, tailwind.config.ts, postcss.config.js, components.json, eslint.config.js, Dockerfile, docker-compose.yml, nginx.conf, .env.example, .gitignore, .dockerignore, .lighthouserc.json, README.md, schema.sql (if present).

Used by: Developers and CI; Vite/Node resolve and build.

Folder: src

Purpose: Application source: entry, app shell, routing, and all feature code.

Contains: main.tsx, App.tsx, App.css, index.css, vite-env.d.ts; subfolders components, contexts, data, hooks, lib, pages, types.

Used by: Vite build (tsconfig.app.json include).

Folder: src/components

Purpose: Reusable React components: layout, feature widgets, and shadcn-based UI primitives.

Contains: Layout, AppSidebar, LoginForm, SignUpForm, page-specific components (e.g. ProjectCard, ContractManagement), and `ui/` (button, card, dialog, form, etc.).

Used by: App.tsx, pages, and other components.

Folder: src/components/ui

Purpose: Low-level UI primitives (shadcn/ui style): buttons, inputs, dialogs, tabs, etc.

Contains: TSX components and use-toast.ts; built with Radix UI and Tailwind.

Used by: Feature components and pages across the app.

Folder: src/contexts

Purpose: React context providers for global state.

Contains: AuthContext.tsx (user, auth methods, permissions).

Used by: App.tsx and any component that uses useAuth().

Folder: src/data

Purpose: Static or mock data and templates for development and defaults.

Contains: mockData.ts, mockChangeRequests.ts, mockGitHubData.ts, projectTemplates.ts, setupTasks.ts.

Used by: Components and services when real data is unavailable or for demos.

Folder: src/hooks

Purpose: Custom React hooks that encapsulate data fetching, mutations, and side effects.

Contains: useProjects, useFinancials, useAuth (re-export from context), use-toast, use-mobile, usePODs, useNotifications, etc.

Used by: Pages and components.

Folder: src/lib

Purpose: Services, utilities, and shared infrastructure (Supabase client, API helpers).

Contains: supabase.ts, api.ts, utils.ts, *Service.ts (project, client, contract, billing, etc.), bambooProxy, costCalculation, pdfGenerator, etc.

Used by: Hooks, components, and pages.

Folder: src/pages

Purpose: Route-level page components that compose layout and feature components.

Contains: Index, Clients, Projects, Contracts, Financials, RiskManagement, Login, UserManagement, Staffing, DreamKeepers, etc.

Used by: App.tsx via React Router routes.

Folder: src/types

Purpose: TypeScript interfaces and types for domain entities and API shapes.

Contains: user.ts, project.ts, client.ts, contract.ts, permissions.ts, and other entity types. The file `supabase.ts` (Database type for Supabase client) is imported by src/lib/supabase.ts but may not exist in the repo; it is typically generated with `supabase gen types typescript` and can be committed or kept in .gitignore.

Folder: public

Purpose: Static assets served as-is.

Contains: favicon.ico, logo.png, form.html, placeholder.svg, dream_campus_logo.png, dream_star_logo.png, pod_journey_logo.png, robots.txt.

Used by: index.html and app assets.

Folder: supabase

Purpose: Supabase project config, migrations, and Edge Functions.

Contains: config.toml, .branches, .temp, functions/, migrations/, templates/.

Used by: Supabase CLI and deployment.

Folder: supabase/functions

Purpose: Deno-based Edge Functions (HTTP handlers).

Contains: One folder per function (e.g. bamboo-webhook, send-email-notification, podjourney-webhook, api, pods-api, generate-api-key).

Used by: Frontend (fetch) and external systems (webhooks).

Folder: supabase/migrations

Purpose: PostgreSQL migrations for schema and RLS.

Contains: Numbered SQL files (e.g. 20240101000000_*.sql).

Used by: Supabase CLI (supabase db push / migrate).

Folder: supabase/templates

Purpose: Email templates for Auth (e.g. confirmation, recovery).

Contains: confirmation.html, email_change.html, recovery.html.

Used by: Supabase Auth when sending emails.

Folder: scripts

Purpose: Operational and one-off scripts (migrations, sync, backup, E2E setup).

Contains: Shell and .mjs scripts for Bamboo sync, Pod Journey setup, API keys, backups, E2E auth; some SQL and markdown.

Used by: Developers and CI/cron; not part of the runtime app.

Folder: tests

Purpose: E2E (Playwright) and unit (Vitest) tests.

Contains: tests/e2e (auth, clients, contracts, projects, utils), tests/unit (components, hooks, services, mocks, fixtures).

Used by: npm scripts (e.g. playwright, vitest).

Folder: archive_backup_baseline_migrations

Purpose: Archived backups and baseline migration SQL.

Contains: backups/ (SQL dumps, migration backups), Prod_Local Data Migration/baseline (schema and data SQL).

Used by: Manual recovery or reference; not used by app or CLI.

Folder: coverage, playwright-report, test-results

Purpose: Test and coverage output.

Contains: HTML/JSON/XML reports and artifacts.

Used by: Developers; typically gitignored or CI artifacts.

---

4. File-by-File Documentation

File: src/App.tsx

Type: Component

Responsibility: Root app component: providers (Theme, QueryClient, Tooltip, Router, Auth), routing for unauthenticated vs authenticated users, onboarding gate, and lazy-loaded routes. In local development it skips auth and renders AuthenticatedApp.

Key logic: AppContent checks loading and user; if localhost, always renders AuthenticatedApp. Otherwise unauthenticated routes (signup, auth/confirm, auth/callback, questionnaire, login) vs Layout-wrapped routes. AuthenticatedApp uses useLoginRedirect, shows OnboardingWizard when !hasCompletedOnboarding, and questionnaire routes without Layout. QueryClient default options set staleTime/gcTime/retry.

Dependencies: react, react-router-dom, framer-motion, @tanstack/react-query, contexts/AuthContext, theme-provider, page-transition, LoginForm, SignUpForm, OnboardingWizard, EmailConfirmationHandler, GoogleAuthCallback, Layout, pages (Index, Clients, Projects, etc.), useLoginRedirect.

Used by: main.tsx

Notes: Lazy-loaded pages: Reports, RiskManagement, InterviewsPage, Staffing, DreamKeepers.

File: src/main.tsx

Type: Other (entry)

Responsibility: React entry point: creates root and renders App; imports index.css.

Key logic: createRoot(document.getElementById("root")).render().

Dependencies: react-dom/client, App.tsx, index.css.

Used by: index.html (script src).

Notes: None.

File: src/App.css

Type: Other (styles)

Responsibility: Optional global app-level CSS (if any).

Key logic: N/A.

Dependencies: None.

Used by: Potentially imported by App or elsewhere; may be minimal or unused if Tailwind covers styles.

Notes: Check for actual rules; may be empty or legacy.

File: src/index.css

Type: Other (styles)

Responsibility: Global design system: Tailwind layers, CSS variables (HSL colors, radii, sidebar, gradients, shadows), base layer for body, and dark mode variables.

Key logic: @tailwind base/components/utilities; :root and .dark define –background, –primary, –sidebar-*, etc.; @layer base applies to body.

Dependencies: Tailwind.

Used by: main.tsx (import).

Notes: All colors in HSL; single source for theme tokens.

File: src/vite-env.d.ts

Type: Config

Responsibility: Vite client type reference for import.meta.env and other Vite globals.

Key logic: /// .

Dependencies: vite/client.

Used by: TypeScript when compiling src.

Notes: None.

File: src/contexts/AuthContext.tsx

Type: Component (context provider)

Responsibility: Provides authentication state and methods (user, login, logout, signup, signInWithGoogle, signInWithMicrosoft, updateUser, reconnect); loads user profile and permissions; enforces allowed domains and hardcoded directors; supports local dev with mock user.

Key logic: AuthProvider state: user, isLoading, connectionError, isReconnecting. On init, local dev uses mock user (optional DB fetch for admin@company.example.com); production uses getSession and onAuthStateChange. fetchUserProfile looks up users by email, applies ensureCorrectRole (HARDCODED_DIRECTORS, DOMAIN_ROLE_MAPPING), loads permissions via PermissionService or getFallbackPermissions. login/signup/sendMagicLink validate isDomainAllowed; signInWithGoogle and signInWithMicrosoft use Supabase OAuth (Google / Azure AD). reconnect retries getSession and fetchUserProfile.

Dependencies: react, types/user, types/permissions, supabase, permissionService.

Used by: App.tsx and any component using useAuth().

Notes: ALLOWED_DOMAINS, GOOGLE_OAUTH_DOMAINS, MAGIC_LINK domains; fallback permissions when DB unavailable.

File: src/lib/supabase.ts

Type: Config / Infrastructure

Responsibility: Creates and exports the Supabase client for browser use (Auth, PostgREST, Realtime).

Key logic: createClient(supabaseUrl, supabaseAnonKey, { auth: autoRefreshToken, persistSession, detectSessionInUrl, flowType: 'pkce' }). Imports Database from @/types/supabase for typing (that file may be generated).

Dependencies: @supabase/supabase-js, @/types/supabase.

Used by: All services and AuthContext that need DB or Auth.

Notes: VITE_SUPABASE_URL and VITE_SUPABASE_ANON_KEY from env. If @/types/supabase is missing, create it with supabase gen types typescript or stub Database type.

File: src/lib/api.ts

Type: API / Utility

Responsibility: ApiService: call Edge Functions with Bearer token and sendEmailNotification helper.

Key logic: callEdgeFunction(name, data) gets session, POSTs to supabase.supabaseUrl/functions/v1/{name} with JSON body and Authorization; returns { success, data?, error? }. sendEmailNotification(type, data) calls send-email-notification function.

Dependencies: supabase.

Used by: Components or services that trigger emails or other Edge Functions.

Notes: Requires active session; no API key path here (pods-api/api use X-API-Key in Edge Functions).

File: src/lib/utils.ts

Type: Utility

Responsibility: cn() helper to merge class names (clsx + tailwind-merge).

Key logic: export function cn(...inputs: ClassValue[]) { return twMerge(clsx(inputs)); }.

Dependencies: clsx, tailwind-merge.

Used by: UI and feature components for conditional Tailwind classes.

Notes: Standard shadcn pattern.

File: src/lib/projectService.ts

Type: API / Service

Responsibility: Project CRUD and project-phase-deliverable logic; uses Supabase and default phases from projectTemplates.

Key logic: createProject inserts project then phases/deliverables; getProjectsForUser filters by role (director sees all, else by assignment); getProjectById with phases; updateProject, deleteProject; PM assignment and handover-related helpers.

Dependencies: supabase, types/project, types/user, data/projectTemplates.

Used by: useProjects, useProject, pages/Projects, components (ProjectCard, ProjectCreationDialog, etc.).

Notes: RLS and RPC usage where needed; phases/deliverables structure in DB.

File: src/lib/clientService.ts

Type: API / Service

Responsibility: Client CRUD; uses RPC get_clients for list (RLS bypass) and direct table access for getById/create/update/delete.

Key logic: getClients/getActiveClients via supabase.rpc('get_clients'); getClientById from clients table; createClient, updateClient, deleteClient with Supabase client.

Dependencies: supabase, types/client.

Used by: Clients page, ClientRegistrationDialog, and any client dropdowns.

Notes: get_clients RPC returns normalized client shape.

File: src/lib/contractService.ts

Type: API / Service

Responsibility: Contract CRUD and listing by project; uses RPC create_contract and get_project_contracts.

Key logic: createContract via create_contract RPC then fetch by get_project_contracts; getProjectContracts; update contract status/fields; transform DB snake_case to Contract type (milestones, payment terms).

Dependencies: supabase, types/contract.

Used by: Contract hooks, Contracts page, ContractManagement, ContractCreationDialog, ContractApprovalDialog.

Notes: Status flow: draft → under-review → approved/signed/executed/terminated/rejected.

File: src/lib/contractApprovalService.ts

Type: API / Service

Responsibility: Contract approval workflow (submit for approval, approve, reject) and related notifications.

Key logic: Submit contract for approval; approve/reject with user context; may call Edge Function or update status and create notifications.

Dependencies: supabase, types/user, possibly api (Edge Function).

Used by: Contract approval UI and hooks.

Notes: Integrates with permission canApproveContracts.

File: src/lib/permissionService.ts

Type: API / Service

Responsibility: Fetch and create user permissions; map between DB snake_case and app camelCase.

Key logic: getUserPermissions(userId), createDefaultPermissions(userId, role); mapDBToApp / mapAppToDB for user_permissions table.

Dependencies: supabase, types/permissions, types/user.

Used by: AuthContext when loading or creating user permissions.

Notes: Fallback permissions live in AuthContext when DB fails.

File: src/lib/billingService.ts

Type: API / Service

Responsibility: Billing-related operations (invoices, usage, or billing entities).

Key logic: Likely Supabase calls for billing/invoice data; see unit test for behavior.

Dependencies: supabase, possibly types/invoice or financials.

Used by: Financials or billing UI and hooks.

Notes: Check tests/unit/services/billingService.test.ts for contract.

File: src/lib/changeRequestService.ts

Type: API / Service

Responsibility: Change request CRUD and workflow; may trigger change-request-notifications Edge Function.

Key logic: Create, list, update change requests; possibly notify via Edge Function.

Dependencies: supabase, types/changeRequest, possibly api.

Used by: Change request hooks and ChangeRequestTracker component.

Notes: None.

File: src/lib/complianceService.ts

Type: API / Service

Responsibility: Compliance-related data or checks (e.g. compliance dashboard).

Key logic: Likely reads compliance or audit data from Supabase.

Dependencies: supabase, possibly types/audit or compliance.

Used by: ComplianceDashboard or compliance views.

Notes: None.

File: src/lib/financialService.ts

Type: API / Service

Responsibility: P&L, budget, cost tracking, and financial aggregates for projects.

Key logic: Fetch/update budget, costs, P&L statement; possibly invoice summaries.

Dependencies: supabase, types/financials.

Used by: useFinancials, Financials page, FinancialManagement components.

Notes: See tests/unit/services/financialService.test.ts.

File: src/lib/notificationService.ts

Type: API / Service

Responsibility: In-app notifications: list, mark read, create (if allowed from client).

Key logic: Supabase from('notifications') or similar; filter by user; markAsRead update.

Dependencies: supabase, types/notification.

Used by: useNotifications, Layout (notification center).

Notes: None.

File: src/lib/userService.ts

Type: API / Service

Responsibility: User list and profile updates (non-auth); may integrate with users and user_permissions.

Key logic: getUsers, getUserById, updateUser (profile/role/department).

Dependencies: supabase, types/user.

Used by: UserManagement page, user dropdowns.

Notes: Auth profile creation is in AuthContext.

File: src/lib/riskService.ts

Type: API / Service

Responsibility: Risk CRUD and listing by project; may trigger risk-notifications Edge Function.

Key logic: Create, list, update risks; map to Risk type; possibly call Edge Function for email.

Dependencies: supabase, types/risk, possibly api.

Used by: useRisks, RiskManagement, RiskMatrix pages.

Notes: None.

File: src/lib/mitigationService.ts

Type: API / Service

Responsibility: Mitigation plans for risks (CRUD or link to risks).

Key logic: Supabase queries for mitigation data.

Dependencies: supabase, types/risk.

Used by: MitigationPlans page, risk components.

Notes: None.

File: src/lib/resourceService.ts

Type: API / Service

Responsibility: Resources and allocations: list resources, allocate to projects, timeline data.

Key logic: CRUD resources; allocation tables; possibly RPC for complex queries.

Dependencies: supabase, types/resource.

Used by: useResources, Resources page, ResourceManagement, allocation dialogs.

Notes: None.

File: src/lib/podService.ts

Type: API / Service

Responsibility: POD CRUD, member assignment, and Pod Journey ID linking.

Key logic: Create/update PODs; assign members; update podjourney_id; uses RPC where needed (e.g. update_pod_podjourney_id).

Dependencies: supabase, types/pod, types/user.

Used by: usePODs, PODManagement, PODCreationDialog, podjourney-webhook (Edge).

Notes: Pod Journey sync in Edge Functions.

File: src/lib/phaseService.ts

Type: API / Service

Responsibility: Project phases and deliverables (status, progress, dates).

Key logic: Update phase status/progress; deliverable completion; dependencies.

Dependencies: supabase, types/project or phases.

Used by: usePhases, ProjectDashboard, PhaseCard, PhaseDetail.

Notes: None.

File: src/lib/phaseSkipService.ts

Type: API / Service

Responsibility: Phase skip requests and approval (canSkipPhases).

Key logic: Create skip record; approve/reject; link to phase.

Dependencies: supabase, types/phases.

Used by: usePhaseSkip, PhaseSkipDialog.

Notes: None.

File: src/lib/invoiceService.ts

Type: API / Service

Responsibility: Invoice CRUD, status (draft/submitted/approved), and link to contracts/projects.

Key logic: Create, list, update invoices; submit for approval; approve (canApproveInvoices).

Dependencies: supabase, types/invoice.

Used by: Financials, CreateInvoiceDialog, InvoiceApprovalDialog.

Notes: None.

File: src/lib/paymentService.ts

Type: API / Service

Responsibility: Payment records and status (e.g. linked to invoices or milestones).

Key logic: Supabase payment-related tables.

Dependencies: supabase, types/payment.

Used by: Financials or payment UI.

Notes: None.

File: src/lib/handoverReportService.ts

Type: API / Service

Responsibility: Project handover reports and checklist data.

Key logic: Create/fetch handover reports; checklist items.

Dependencies: supabase, types/handover.

Used by: useHandoverReport, HandoverChecklist, ProjectHandoverDashboard.

Notes: None.

File: src/lib/dreamKeeperService.ts

Type: API / Service

Responsibility: Dream Keepers (staffing) data: candidates, interviews, pipeline stages.

Key logic: Fetch candidates, interviews, evaluations; possibly call Edge Functions (bamboo-interviews, send-dreamkeeper-interview-invitation).

Dependencies: supabase, types/dreamKeeper, possibly api.

Used by: DreamKeepers page, interview components.

Notes: BambooHR and Edge Functions drive pipeline.

File: src/lib/staffingService.ts

Type: API / Service

Responsibility: Staffing pipeline: new candidates, English test, TestGorilla, questionnaire, speaking test, talent pool, rejected.

Key logic: Integrates with BambooHR and internal tables; may invoke Edge Functions for sync or email.

Dependencies: supabase, types/staffing, types/newCandidate, types/rejectedCandidate, etc.

Used by: Staffing page, Interviews page, hooks.

Notes: Heavy use of Edge Functions for email and BambooHR moves.

File: src/lib/interviewsService.ts

Type: API / Service

Responsibility: Interview scheduling and evaluations; BambooHR and internal evaluations.

Key logic: Fetch evaluations, save evaluations; move candidates (talent pool, questions/cvs, rejected); call bamboo-interviews Edge Function.

Dependencies: supabase, types/interviews, types/dreamKeeper.

Used by: Interviews page, evaluation UI.

Notes: None.

File: src/lib/notificationService.ts

Type: API / Service

Responsibility: (See above under notificationService.)

File: src/lib/jiraService.ts

Type: API / Service

Responsibility: Jira API calls (issues, projects) from frontend or proxy.

Key logic: Fetch issues/projects; may use jira-proxy or jira-proxy-project Edge Function or bambooProxy.

Dependencies: env (VITE_JIRA, VITE_JIRA_EMAIL), fetch or proxy.

Used by: JiraKanban, Projects (Jira tab).

Notes: Token stored in env; proxy can avoid CORS.

File: src/lib/jiraIntegrationService.ts

Type: API / Service

Responsibility: Jira integration (sync project, link project to Jira, sync issues).

Key logic: May call jira-sync-project or similar Edge Function; link project to Jira project.

Dependencies: supabase, api or fetch.

Used by: Projects page, Jira integration UI.

Notes: None.

File: src/lib/githubService.ts

Type: API / Service

Responsibility: GitHub metrics or repo data for projects.

Key logic: Fetch repo stats, PRs, or metrics (possibly via Edge Function or direct API).

Dependencies: Possibly env, fetch.

Used by: GitHubMetrics component, Projects.

Notes: mockGitHubData may be used for demos.

File: src/lib/emailService.ts

Type: API / Service

Responsibility: Send email via Edge Function (e.g. send-email-notification) or Brevo from client (if any).

Key logic: Likely wraps ApiService.sendEmailNotification or similar.

Dependencies: api.

Used by: Components that send PM/POD/resource emails.

Notes: Most email is sent from Edge Functions, not client.

File: src/lib/storageService.ts

Type: API / Service

Responsibility: Supabase Storage: upload/download/delete files (e.g. contract documents, avatars).

Key logic: supabase.storage.from(bucket).upload/list/remove.

Dependencies: supabase.

Used by: Contract uploads, profile photos, etc.

Notes: Bucket policies and RLS on storage.

File: src/lib/pdfGenerator.ts

Type: Utility

Responsibility: Generate PDFs (e.g. reports, contracts) in the browser (e.g. jspdf, html2canvas).

Key logic: Build PDF from HTML or data; download or blob.

Dependencies: jspdf, possibly html2canvas.

Used by: ReportExporter, contract print/export.

Notes: None.

File: src/lib/costCalculation.ts

Type: Utility

Responsibility: Cost and margin calculations for projects or financials.

Key logic: Pure functions for cost, margin, P&L line items.

Dependencies: types/financials or project.

Used by: financialService, FinancialManagement components.

Notes: None.

File: src/lib/estimationValidation.ts

Type: Utility

Responsibility: Validate estimates (e.g. effort, budget) before save.

Key logic: Zod or custom validation rules.

Dependencies: Possibly zod, types.

Used by: Project or financial forms.

Notes: None.

File: src/lib/apiKeyService.ts

Type: API / Service

Responsibility: List/create/revoke API keys for the current user via Edge Function generate-api-key or direct Supabase.

Key logic: Call generate-api-key Edge Function with name; list from api_keys table (with RLS); revoke = update is_active.

Dependencies: supabase, api (Edge Function).

Used by: APITokens page.

Notes: generate-api-key returns key once; store securely.

File: src/lib/bambooProxy.ts

Type: API / Utility

Responsibility: Proxy or wrapper for BambooHR API calls (e.g. from frontend to avoid CORS or to add auth).

Key logic: Forward requests to BambooHR or to an Edge Function that calls BambooHR.

Dependencies: env (Bamboo API key), fetch.

Used by: Staffing/Interview flows that need BambooHR from browser.

Notes: Many BambooHR operations are in Edge Functions (webhook, full-sync, interviews).

File: src/lib/podJourneyService.ts

Type: API / Service

Responsibility: Pod Journey API: list PODs, create/update POD in Pod Journey, sync members.

Key logic: HTTP calls to Pod Journey API with PODJOURNEY_API_TOKEN; used for push from RivanIT to Pod Journey.

Dependencies: env, fetch.

Used by: POD creation/update UI, podjourney-sync-frontend Edge Function.

Notes: Webhook receives events from Pod Journey.

File: src/lib/podJourneySyncService.ts

Type: API / Service

Responsibility: Sync state between RivanIT and Pod Journey (pull or push).

Key logic: May call podjourney-sync-frontend or podjourney-sync-periodic Edge Function; or orchestrate sync from UI.

Dependencies: api or fetch, podJourneyService.

Used by: usePodJourneySync, PodJourneyConfig page.

Notes: usePodJourneySync can run periodic sync from Layout.

File: src/lib/skillService.ts

Type: API / Service

Responsibility: Skills and skill matrix; current user enrichment (e.g. employee_photo).

Key logic: Fetch skills, skill matrix data; getCurrentUser(userId) for profile/photo.

Dependencies: supabase.

Used by: SkillMatrix page, Layout (user photo), DirectorDashboard.

Notes: employee_photo may come from BambooHR or storage.

File: src/lib/employeeFullService.ts

Type: API / Service

Responsibility: Enriched employee data (e.g. photo URL validation, full profile).

Key logic: isValidPhoto(url); fetch full employee from Bamboo or users.

Dependencies: supabase or bamboo proxy.

Used by: Layout (avatar photo), profile components.

Notes: None.

File: src/lib/riskNotificationService.ts

Type: API / Service

Responsibility: Trigger risk notification emails (e.g. call risk-notifications Edge Function).

Key logic: Call Edge Function with risk data and recipient info.

Dependencies: api.

Used by: Risk creation/update flows.

Notes: None.

File: src/lib/reportsService.ts

Type: API / Service

Responsibility: Report definitions and data for Reports page (exports, aggregates).

Key logic: Fetch report configs and data from Supabase.

Dependencies: supabase.

Used by: Reports page, ReportExporter.

Notes: None.

File: src/lib/projectMetricsService.ts

Type: API / Service

Responsibility: Project metrics (health, progress, KPIs).

Key logic: Aggregate from projects/phases/deliverables or dedicated metrics table.

Dependencies: supabase.

Used by: ProjectMetrics component, dashboards.

Notes: None.

File: src/lib/projectTimelineService.ts

Type: API / Service

Responsibility: Timeline and Gantt data for projects (phases, dates, dependencies).

Key logic: Query phases and deliverables with dates; return structure for Gantt/timeline.

Dependencies: supabase, types/project.

Used by: useProjectTimeline, ProjectTimeline, ResourceGanttChart.

Notes: None.

File: src/lib/newCandidatesService.ts

Type: API / Service

Responsibility: New candidates (BambooHR application stage) CRUD or list.

Key logic: Supabase and/or BambooHR; may trigger sync Edge Functions.

Dependencies: supabase, types/newCandidate.

Used by: Staffing pipeline (new candidates stage).

Notes: None.

File: src/lib/rejectedCandidatesService.ts

Type: API / Service

Responsibility: Rejected candidates and rejection reasons.

Key logic: List/update rejected candidates; store reason.

Dependencies: supabase, types/rejectedCandidate.

Used by: Staffing/Interviews (rejected list).

Notes: None.

File: src/lib/talentPoolService.ts

Type: API / Service

Responsibility: Talent pool candidates (post-interview or post-questionnaire).

Key logic: List/add/update talent pool entries; link to BambooHR.

Dependencies: supabase, types.

Used by: Staffing, Interviews, process-questionnaire-response / bamboo-interviews Edge Functions.

Notes: None.

File: src/lib/timeTrackingService.ts

Type: API / Service

Responsibility: Time entries for resources or projects.

Key logic: CRUD time entries; aggregate by project/resource.

Dependencies: supabase, types/timeTracking.

Used by: TimeTracking component, Financials.

Notes: None.

File: src/lib/phaseTest.ts, src/lib/projectTest.ts, src/lib/testDependencies.ts, src/lib/testPhaseManagement.ts

Type: Other (test helpers or dev)

Responsibility: Test utilities or stubs for phase/project logic; may be used by Vitest or dev only.

Key logic: Varies; exports for tests.

Dependencies: Other lib or types.

Used by: Unit tests or dev scripts.

Notes: Not part of production bundle if tree-shaken or test-only imports.

File: src/types/user.ts

Type: Other (types)

Responsibility: User and AuthContextType interfaces; UserRole.

Key logic: User (id, name, email, role, department, created_at, updated_at, permissions, preferences); AuthContextType (user, login, logout, signup, signInWithGoogle, updateUser, isLoading, connectionError, reconnect, isReconnecting).

Dependencies: types/permissions.

Used by: AuthContext, permissionService, userService, components.

Notes: None.

File: src/types/permissions.ts

Type: Other (types)

Responsibility: UserPermissions and UserPermissionsDB; PERMISSION_CATEGORIES for UI.

Key logic: CamelCase vs snake_case permission flags; categories for permission editor.

Dependencies: None.

Used by: AuthContext, permissionService, UserManagement (permission editing).

Notes: canManageClients deprecated in favor of granular client permissions.

File: src/types/project.ts

Type: Other (types)

Responsibility: PhaseStatus, Template, Deliverable, Phase, ProjectType, Project, Risk.

Key logic: Project has phases, currentPhaseId, totalProgress, riskLevel, currentPM, handoverId; Phase has deliverables, dependencies, canSkip, skipRecord.

Dependencies: None.

Used by: projectService, phaseService, components (ProjectCard, ProjectDashboard, etc.).

Notes: None.

File: src/types/client.ts

Type: Other (types)

Responsibility: ClientStatus, Client, CreateClientData, UpdateClientData; CLIENT_REGIONS, CLIENT_VERTICALS.

Key logic: Client has prospect_id, tag, vertical, region, portfolio, tier; arrays for regions/verticals.

Dependencies: None.

Used by: clientService, ClientRegistrationDialog, Clients page.

Notes: None.

File: src/types/contract.ts

Type: Other (types)

Responsibility: Contract, ContractTerm, ContractMilestone, PaymentTerm.

Key logic: Contract status flow; milestones and payment terms linked.

Dependencies: None.

Used by: contractService, contractApprovalService, Contracts page, contract components.

Notes: None.

File: src/types/financials.ts

Type: Other (types)

Responsibility: PnLStatement, Budget, CostTracking, and related line-item types.

Key logic: Structures for P&L, budget, cost entries.

Dependencies: None.

Used by: financialService, useFinancials, Financials page.

Notes: None.

File: src/types/invoice.ts

Type: Other (types)

Responsibility: Invoice and related status/types.

Key logic: Invoice shape for list/detail/approval.

Dependencies: None.

Used by: invoiceService, Financials, invoice dialogs.

Notes: None.

File: src/types/notification.ts

Type: Other (types)

Responsibility: AppNotification and related (type, priority, read, link).

Key logic: Shape for notification center and mark-as-read.

Dependencies: None.

Used by: notificationService, useNotifications, Layout.

Notes: None.

File: src/types/risk.ts

Type: Other (types)

Responsibility: Risk and mitigation types (probability, impact, status).

Key logic: Risk shape for matrix and mitigation.

Dependencies: None.

Used by: riskService, RiskManagement, RiskMatrix.

Notes: None.

File: src/types/resource.ts

Type: Other (types)

Responsibility: Resource and allocation types.

Key logic: Resource profile; allocation (project, role, dates).

Dependencies: None.

Used by: resourceService, Resources page, allocation dialogs.

Notes: None.

File: src/types/pod.ts

Type: Other (types)

Responsibility: POD and POD member types; Pod Journey ID.

Key logic: POD shape; member role.

Dependencies: None.

Used by: podService, POD components, podjourney-webhook.

Notes: None.

File: src/types/phases.ts

Type: Other (types)

Responsibility: Phase and phase-skip types (if separate from project.ts).

Key logic: Phase status, skip record.

Dependencies: None.

Used by: phaseService, phaseSkipService, phase components.

Notes: None.

File: src/types/handover.ts

Type: Other (types)

Responsibility: Handover report and checklist types.

Key logic: Handover structure for PM handover flow.

Dependencies: None.

Used by: handoverReportService, handover components.

Notes: None.

File: src/types/dreamKeeper.ts

Type: Other (types)

Responsibility: Dream Keeper / interview candidate types.

Key logic: Candidate and evaluation shapes.

Dependencies: None.

Used by: dreamKeeperService, Interviews, DreamKeepers page.

Notes: None.

File: src/types/staffing.ts

Type: Other (types)

Responsibility: Staffing pipeline types: EnglishTestCandidate, TestGorillaCandidate, Questionnaire, etc.

Key logic: Pipeline stages and form types.

Dependencies: None.

Used by: staffingService, Staffing page, QuestionnaireForm, Edge Functions.

Notes: None.

File: src/types/interviews.ts

Type: Other (types)

Responsibility: Interview and evaluation types.

Key logic: Evaluation, feedback, stage.

Dependencies: None.

Used by: interviewsService, Interviews page.

Notes: None.

File: src/types/newCandidate.ts

Type: Other (types)

Responsibility: New candidate (BambooHR) type.

Key logic: Candidate fields from BambooHR.

Dependencies: None.

Used by: newCandidatesService, Staffing.

Notes: None.

File: src/types/rejectedCandidate.ts

Type: Other (types)

Responsibility: Rejected candidate and rejection reason.

Key logic: RejectedCandidate, RejectionReason.

Dependencies: None.

Used by: rejectedCandidatesService, Interviews.

Notes: None.

File: src/types/apiKey.ts

Type: Other (types)

Responsibility: APIKey and APIKeyWithVisibility (for list with masked key).

Key logic: id, name, key (optional in list), is_active, last_used_at, user_id.

Dependencies: None.

Used by: apiKeyService, APITokens page.

Notes: None.

File: src/types/changeRequest.ts

Type: Other (types)

Responsibility: Change request and status.

Key logic: ChangeRequest shape for CRUD and notifications.

Dependencies: None.

Used by: changeRequestService, ChangeRequestTracker.

Notes: None.

File: src/types/audit.ts

Type: Other (types)

Responsibility: Audit log or audit event types.

Key logic: Audit entry shape.

Dependencies: None.

Used by: Compliance or audit views, complianceService.

Notes: None.

File: src/types/email.ts

Type: Other (types)

Responsibility: Email payload or template types (for send-email-notification, etc.).

Key logic: Params for email sending.

Dependencies: None.

Used by: emailService or Edge Function types (in functions).

Notes: None.

File: src/types/github.ts

Type: Other (types)

Responsibility: GitHub repo or metric types.

Key logic: Repo, PR, metric shapes.

Dependencies: None.

Used by: githubService, GitHubMetrics.

Notes: None.

File: src/types/payment.ts

Type: Other (types)

Responsibility: Payment record types.

Key logic: Payment status, amount, date.

Dependencies: None.

Used by: paymentService, Financials.

Notes: None.

File: src/types/timeTracking.ts

Type: Other (types)

Responsibility: Time entry types.

Key logic: Entry (resource, project, hours, date).

Dependencies: None.

Used by: timeTrackingService, TimeTracking.

Notes: None.

File: src/types/supabase.ts (optional / generated)

Type: Other (types)

Responsibility: Supabase generated Database type (schema types for tables and RPC). Used to type the Supabase client in src/lib/supabase.ts.

Key logic: export type Database = { ... }; tables and RPC definitions.

Dependencies: None (generated from live DB or migrations).

Used by: src/lib/supabase.ts (createClient()).

Notes: May be missing from repo; generate with `supabase gen types typescript --project-id <ref> > src/types/supabase.ts` or equivalent. If missing, TypeScript may error on import; stub with `export type Database = Record<string, unknown>;` or add to .gitignore and generate in CI.

File: src/data/mockData.ts

Type: Other (data)

Responsibility: Mock risks, issues, dependencies, contracts for development/demo.

Key logic: Exported arrays of mock entities.

Dependencies: None.

Used by: Projects page (mockRisks, mockIssues, etc.) when real data not used.

Notes: Replace with real data in production paths.

File: src/data/mockChangeRequests.ts

Type: Other (data)

Responsibility: Mock change requests for UI development.

Key logic: Exported mock change request list.

Dependencies: None.

Used by: Change request components when needed.

Notes: None.

File: src/data/mockGitHubData.ts

Type: Other (data)

Responsibility: Mock GitHub metrics for demos.

Key logic: Exported mock repo/PR data.

Dependencies: None.

Used by: GitHubMetrics when API not available.

Notes: None.

File: src/data/projectTemplates.ts

Type: Other (data)

Responsibility: Default phases and deliverables per project type (defaultPhases).

Key logic: Template structure used by projectService.createProject.

Dependencies: None.

Used by: projectService.

Notes: None.

File: src/data/setupTasks.ts

Type: Other (data)

Responsibility: Onboarding or setup checklist tasks (e.g. for new project or tenant).

Key logic: List of task definitions.

Dependencies: None.

Used by: OnboardingWizard or setup flows.

Notes: None.

File: src/hooks/useProjects.ts

Type: Hook

Responsibility: React Query hooks for projects list and single project; uses ProjectService and current user/role.

Key logic: useProjects(includeDetails) queryKey [‘projects’, user?.id, user?.role, includeDetails]; queryFn ProjectService.getProjectsForUser(user.id, user.role, includeDetails); enabled when user exists. useProject(projectId) fetches single project.

Dependencies: @tanstack/react-query, projectService, useAuth.

Used by: Projects page, project components.

Notes: staleTime 5 min, gcTime 10 min.

File: src/hooks/useProject.ts

Type: Hook

Responsibility: (Often exported from useProjects.ts as useProject.) Single project by ID.

Key logic: useQuery([‘project’, projectId], () ⇒ ProjectService.getProjectById(projectId)).

Dependencies: react-query, projectService.

Used by: Project detail views.

Notes: None.

File: src/hooks/useFinancials.ts

Type: Hook

Responsibility: Project financials: P&L, budget, cost tracking, invoices; mutations for create cost/invoice, submit, update.

Key logic: useQuery for financial data; useMutation for createCost, createInvoice, updateInvoiceDraft, submitInvoice; keyed by projectId.

Dependencies: react-query, financialService, invoiceService.

Used by: Financials page, FinancialManagementWithData.

Notes: None.

File: src/hooks/useNotifications.ts

Type: Hook

Responsibility: In-app notifications: list, unread count, mark as read; real-time subscription via NotificationService.

Key logic: useState for notifications; fetchNotifications by user.email; subscribeToNotifications (Supabase realtime) for INSERT/UPDATE; markAsRead; unreadCount and unreadNotifications derived.

Dependencies: notificationService, useAuth, types/notification.

Used by: Layout (notification center, badge).

Notes: Realtime channel per user email.

File: src/hooks/useLoginRedirect.ts

Type: Hook

Responsibility: Post-login welcome toast (once per session).

Key logic: useEffect when user is set; useRef(hasShownWelcome); toast.success("Welcome back, ...") once.

Dependencies: react-router-dom, sonner, useAuth.

Used by: AuthenticatedApp in App.tsx.

Notes: None.

File: src/hooks/use-toast.ts

Type: Hook

Responsibility: Toast state and API (add, dismiss, update, remove); used by Toaster and components that show toasts.

Key logic: useReducer for toast list; genId; add, dismiss, update, remove actions; toast() helper returns id; TOAST_LIMIT and TOAST_REMOVE_DELAY.

Dependencies: react, components/ui/toast types.

Used by: Toaster, components that call toast().

Notes: Lives in src/hooks and possibly re-exported from components/ui/use-toast.ts.

File: src/hooks/use-mobile.tsx

Type: Hook

Responsibility: Detect mobile viewport (e.g. matchMedia (max-width: 768px)).

Key logic: useState + useEffect for window matchMedia; return boolean.

Dependencies: react.

Used by: Sidebar or responsive components.

Notes: shadcn pattern for responsive UI.

File: src/hooks/usePODs.ts

Type: Hook

Responsibility: POD list and mutations (create, update, delete, assign members).

Key logic: useQuery for PODs; useMutation for create/update/delete/assign; uses podService.

Dependencies: react-query, podService, useAuth.

Used by: PODManagement, PODCreationDialog, PODDetails.

Notes: None.

File: src/hooks/usePermissions.ts

Type: Hook

Responsibility: Current user permissions from AuthContext (user.permissions).

Key logic: useAuth().user?.permissions; may expose canX helpers.

Dependencies: useAuth.

Used by: Components that gate by permission (e.g. canCreateProjects).

Notes: None.

File: src/hooks/useChangeRequests.ts

Type: Hook

Responsibility: Change requests list and mutations; uses changeRequestService.

Key logic: useQuery for list; useMutation for create/update/approve/reject.

Dependencies: react-query, changeRequestService.

Used by: ChangeRequestTracker, IssuesAndDependencies.

Notes: None.

File: src/hooks/useCompliance.ts

Type: Hook

Responsibility: Compliance or audit data; uses complianceService.

Key logic: useQuery for compliance/audit data.

Dependencies: react-query, complianceService.

Used by: ComplianceDashboard.

Notes: None.

File: src/hooks/useFinancials.ts

Type: Hook

Responsibility: (Documented above under useFinancials.)

File: src/hooks/useHandoverReport.ts

Type: Hook

Responsibility: Handover report for a project; uses handoverReportService.

Key logic: useQuery by projectId; possibly useMutation for update.

Dependencies: react-query, handoverReportService.

Used by: HandoverChecklist, ProjectHandoverDashboard.

Notes: None.

File: src/hooks/useMitigation.ts

Type: Hook

Responsibility: Mitigation plans for risks; uses mitigationService.

Key logic: useQuery for mitigations; useMutation for update.

Dependencies: react-query, mitigationService.

Used by: MitigationPlans page.

Notes: None.

File: src/hooks/usePhases.ts

Type: Hook

Responsibility: Phases for a project; uses phaseService.

Key logic: useQuery by projectId; useMutation for phase update.

Dependencies: react-query, phaseService.

Used by: ProjectDashboard, PhaseCard, PhaseDetail.

Notes: None.

File: src/hooks/usePhaseSkip.ts

Type: Hook

Responsibility: Phase skip requests and approval; uses phaseSkipService.

Key logic: useMutation for request skip / approve / reject.

Dependencies: phaseSkipService, useAuth.

Used by: PhaseSkipDialog.

Notes: None.

File: src/hooks/usePodJourneySync.ts

Type: Hook

Responsibility: Trigger Pod Journey sync (e.g. periodic every 30 min from Layout).

Key logic: useEffect with interval; call podjourney-sync-periodic or podjourney-sync-frontend Edge Function or podJourneySyncService.

Dependencies: api or podJourneySyncService.

Used by: Layout (usePodJourneySync(true)).

Notes: None.

File: src/hooks/useProjectMetrics.ts

Type: Hook

Responsibility: Project metrics (health, KPIs); uses projectMetricsService.

Key logic: useQuery by projectId.

Dependencies: react-query, projectMetricsService.

Used by: ProjectMetrics component.

Notes: None.

File: src/hooks/useProjectTimeline.ts

Type: Hook

Responsibility: Timeline/Gantt data for project; uses projectTimelineService.

Key logic: useQuery by projectId.

Dependencies: react-query, projectTimelineService.

Used by: ProjectTimeline, ResourceGanttChart.

Notes: None.

File: src/hooks/useResources.ts

Type: Hook

Responsibility: Resources and allocations; uses resourceService.

Key logic: useQuery for resources; useMutation for allocate/update/remove.

Dependencies: react-query, resourceService.

Used by: Resources page, ResourceManagement, allocation dialogs.

Notes: None.

File: src/hooks/useRisks.ts

Type: Hook

Responsibility: Risks for project; uses riskService.

Key logic: useQuery by projectId; useMutation for create/update.

Dependencies: react-query, riskService.

Used by: RiskManagement component, RiskMatrix.

Notes: None.

File: src/hooks/useSkills.ts

Type: Hook

Responsibility: Skills or skill matrix data; uses skillService.

Key logic: useQuery for skills or matrix.

Dependencies: react-query, skillService.

Used by: SkillMatrix page.

Notes: None.

File: src/components/ui/use-toast.ts

Type: Hook

Responsibility: Same as src/hooks/use-toast.ts (toast state and API). May be the canonical location; src/hooks/use-toast may re-export.

Key logic: (See use-toast above.)

Dependencies: react, toast types.

Used by: Toaster, components.

Notes: Check for single source (hooks vs components/ui).

Pages (summary)

Each page under src/pages is a route-level component that composes layout (via App routing) and feature components, uses hooks for data, and calls services or Edge Functions where needed.

- **Index.tsx**: Dashboard (director/PM view); uses projects, risks, notifications.
- **Clients.tsx**: Client list and CRUD; ClientRegistrationDialog; clientService, useAuth.
- **Projects.tsx**: Project list and detail; tabs (Overview, Risk, Contract, Financials, GitHub, Jira); ProjectCard, ProjectDashboard, RiskManagement, ContractManagement, FinancialManagement, GitHubMetrics, JiraKanban; useProjects, useFinancials, projectService, JiraIntegrationService.
- **Contracts.tsx**: Contract list and detail; ContractManagement, ContractCreationDialog, ContractApprovalDialog; contractService, useAuth.
- **Financials.tsx**: Financials list and per-project financials; P&L, budget, costs, invoices; useFinancials, financialService, invoiceService.
- **RiskManagement.tsx**: Risk overview and mitigation (lazy-loaded).
- **RiskMatrix.tsx**: Risk matrix view.

- **MitigationPlans.tsx**: Mitigation plans list and detail.
- **IssuesAndDependencies.tsx**: Issues and dependencies; ChangeRequestTracker.
- **Resources.tsx**: Resource list and allocations; ResourceManagement, allocation dialogs.
- **UserManagement.tsx**: User list and permission editing; userService, permissionService.
- **Notifications.tsx**: Full notifications page (list and mark read).
- **Reports.tsx**: Reports and exports (lazy-loaded).
- **ProjectTraceabilityPage.tsx**: Traceability dashboard.
- **ProjectTraceability.tsx**: (If different from above, traceability view.)
- **Staffing.tsx**: Staffing pipeline (lazy-loaded); candidate stages, questionnaires, tests.
- **DreamKeepers.tsx**: Dream Keepers / interviews (lazy-loaded).
- **Interviews.tsx**: Interviews and evaluations (lazy-loaded).
- **Login.tsx**: (Unused if App always renders LoginForm for unauthenticated.)
- **NotFound.tsx**: 404 view.
- **APITokens.tsx**: API key list, create (via Edge Function), revoke; apiKeyService, types/apiKey.
- **PodJourneyConfig.tsx**: Pod Journey sync and config; usePodJourneySync, podJourneySyncService.
- **QuestionnaireForm.tsx**: Public questionnaire for candidates (route /questionnaire/:application_id); types/staffing.
- **EnglishTestForm.tsx**: (If present) English test form for candidates.

Components (feature components)

- **Layout.tsx**: App shell: SidebarProvider, AppSidebar, AppsLauncher, theme toggle, user avatar and dropdown, notification popover,

ConnectionErrorBanner; useAuth, useNotifications, usePodJourneySync; fetches user photo via SkillService/EmployeeFullService.

- **AppSidebar.tsx**: Navigation menu (Home, Dashboard, Clients, Projects, Risk Management submenu, Contracts, Financials, Resources, etc.); Sidebar components from ui/sidebar.
- **AppsLauncher.tsx**: Grid or list of app shortcuts (dashboard, clients, projects, etc.).
- **LoginForm.tsx**: Email/password and Google OAuth; domain-based auth method (Google vs magic link vs email-password); local dev URL params for auto-login; useAuth, useToast.
- **SignUpForm.tsx**: Sign-up form; domain validation; useAuth.
- **OnboardingWizard.tsx**: First-time onboarding steps; updateUser preferences.hasCompletedOnboarding.
- **EmailConfirmationHandler.tsx**: Handles /auth/confirm (email confirmation link).
- **GoogleAuthCallback.tsx**: Handles /auth/callback (OAuth redirect); completes sign-in.
- **ConnectionErrorBanner.tsx**: Shows connection error and reconnect button; useAuth.connectionError, reconnect.
- **ProjectCard.tsx**: Card for one project (name, client, status, progress); links to project detail.
- **ProjectDashboard.tsx**: Project overview: phases, deliverables, progress; PhaseCard, PhaseDetail.
- **ProjectCreationDialog.tsx**: Create project form; ProjectService.createProject; client dropdown; default phases from projectTemplates.
- **ProjectEditDialog.tsx**: Edit project form; ProjectService.updateProject.
- **DeleteProjectDialog.tsx**: Confirm and delete project; ProjectService.deleteProject.
- **ProjectTimeline.tsx**: Timeline or Gantt for project phases; useProjectTimeline.
- **ProjectMetrics.tsx**: Metrics cards or charts; useProjectMetrics.

- **ProjectHandoverDashboard.tsx**: Handover checklist and report; useHandoverReport.
- **ProjectManagerAssignmentDialog.tsx**: Assign PM to project; projectService; may trigger send-email-notification.
- **ClientRegistrationDialog.tsx**: Create/edit client; Client, CreateClientData, CLIENT_REGIONS, CLIENT_VERTICALS; clientService.
- **ContractManagement.tsx**: Contract list and actions for a project; ContractCreationDialog, ContractApprovalDialog.
- **ContractCreationDialog.tsx**: Create contract form; contractService.createContract.
- **ContractApprovalDialog.tsx**: Approve/reject contract; contractApprovalService.
- **ContractRatesManagement.tsx**: Contract rates and payment terms.
- **ContractApprovalDialog.tsx**: (Duplicate name in list; same as above.)
- **FinancialManagement.tsx**: P&L, budget, costs, invoices for a project; AddCostDialog, CreateInvoiceDialog, InvoiceApprovalDialog; useFinancials.
- **AddCostDialog.tsx**: Add cost entry form.
- **CreateInvoiceDialog.tsx**: Create invoice form.
- **InvoicePreviewDialog.tsx**: Preview invoice before submit.
- **InvoiceApprovalDialog.tsx**: Approve/reject invoice.
- **PaymentRegistrationDialog.tsx**: Register payment for invoice/milestone.
- **PaymentEvidenceDialog.tsx**: Upload or link payment evidence.
- **RiskManagement.tsx**: Risk list and mitigation for project; useRisks; risk creation/update.
- **RiskMatrix.tsx**: (Page or component) Risk matrix grid.
- **ResourceManagement.tsx**: Resource list and allocation; ResourceAllocationDialog, EditResourceDialog, AddResourceDialog.
- **ResourceAllocationDialog.tsx**: Allocate resource to project (role, dates).

- **EditResourceDialog.tsx**: Edit resource or allocation.
- **AddResourceDialog.tsx**: Add new resource.
- **ResourceTimeline.tsx**: Timeline of resource allocations.
- **ResourceGanttChart.tsx**: Gantt view for resources; gantt-task-react or similar.
- **AllocationDetailsDialog.tsx**: View allocation details.
- **PODManagement.tsx**: POD list and CRUD; PODCreationDialog, PODDetails.
- **PODCreation.tsx**: Create POD flow (may be same as PODCreationDialog).
- **PODCreationDialog.tsx**: Create POD form; podService.
- **PODDetails.tsx**: POD detail and members; PODMemberAssignmentDialog.
- **PODMemberAssignmentDialog.tsx**: Assign member to POD; may trigger email.
- **PODConstitution.tsx**: POD constitution document or form.
- **PhaseCard.tsx**: Phase summary card.
- **PhaseDetail.tsx**: Phase detail and deliverables.
- **PhaseSkipDialog.tsx**: Request or approve phase skip; usePhaseSkip.
- **SetupPhaseChecklist.tsx**: Checklist for phase setup; uses Database icon from lucide (SetupPhaseChecklist).
- **HandoverChecklist.tsx**: Handover checklist items; handoverReportService.
- **ChangeRequestTracker.tsx**: Change request list and actions; useChangeRequests.
- **ComplianceDashboard.tsx**: Compliance or audit view; useCompliance.
- **AuditTrail.tsx**: Audit log list.
- **NotificationCenter.tsx**: (May be inline in Layout) Notification list and mark read.
- **EmailNotificationDashboard.tsx**: Email notification history or config.
- **TraceabilityDashboard.tsx**: Traceability matrix or view.
- **StakeholderMap.tsx**: Stakeholder map view.

- **DecisionLog.tsx**: Decision log list.
- **KnowledgeBase.tsx**: Knowledge base content.
- **KnowledgeManagement.tsx**: Knowledge management view.
- **ReportExporter.tsx**: Export report (PDF/CSV); pdfGenerator or similar.
- **SkillMatrix.tsx**: Skill matrix view (default home); useSkills, SkillService.
- **SkillMatrixDemo.tsx**: Demo variant of skill matrix.
- **DirectorDashboard.tsx**: Director-specific dashboard; Project, Risk types; project and risk data.
- **JiraKanban.tsx**: Jira issues in Kanban; jiraService or Jira proxy.
- **GitHubMetrics.tsx**: GitHub repo metrics; githubService; mockGitHubData fallback.
- **GitHubConnectionDialog.tsx**: Connect project to GitHub repo.
- **TimeTracking.tsx**: Time entry list and form; timeTrackingService.
- **PresalesModule.tsx**: Presales workflow or view.
- **EstimationWizard.tsx**: Estimation wizard for project.
- **CandidateDetailsModal.tsx**: Candidate detail (staffing/interviews); BambooHR data.
- **EmployeeCard.tsx**: Employee card (avatar, name, role).
- **DreamKeeperCard.tsx**: Dream Keeper candidate card.
- **EndorsementModal.tsx**: Endorsement modal.
- **ReceivedEndorsements.tsx**: List of received endorsements.
- **CamundaIntegration.tsx**: Camunda BPM integration (if used).
- **ArchiveProjectDialog.tsx**: Archive project confirm.
- **UserEditDialog.tsx**: Edit user (name, role, department); userService.
- **UserAssignmentDialog.tsx**: Assign user to project or role.
- **theme-provider.tsx**: Theme context (light/dark/system); localStorage; applies class to document root.

- **theme-toggle.tsx**: Toggle button for theme; useTheme.
- **page-transition.tsx**: Wraps children with AnimatePresence or transition for route changes.
- **ConnectionErrorBanner.tsx**: (Documented above.)

UI components (src/components/ui)

Primitives from shadcn/ui (Radix + Tailwind). Each provides a single primitive or pattern; used across the app.

- **accordion.tsx**: Collapsible accordion (Radix Accordion).
- **alert.tsx**: Alert message box.
- **alert-dialog.tsx**: Confirmation dialog (Radix AlertDialog).
- **aspect-ratio.tsx**: Aspect ratio wrapper.
- **avatar.tsx**: Avatar image with fallback.
- **badge.tsx**: Badge/chip.
- **breadcrumb.tsx**: Breadcrumb navigation.
- **button.tsx**: Button with variants (default, destructive, outline, etc.).
- **calendar.tsx**: Date picker calendar (react-day-picker).
- **card.tsx**: Card, CardHeader, CardTitle, CardDescription, CardContent, CardFooter.
- **carousel.tsx**: Carousel (embla-carousel-react).
- **chart.tsx**: Recharts wrapper for consistent chart config.
- **checkbox.tsx**: Checkbox (Radix).
- **collapsible.tsx**: Collapsible section (Radix Collapsible).
- **command.tsx**: Command palette (cmdk).
- **context-menu.tsx**: Context menu (Radix).
- **dialog.tsx**: Modal dialog (Radix Dialog).
- **dropdown-menu.tsx**: Dropdown menu (Radix DropdownMenu).

- **drawer.tsx**: Drawer (vaul).
- **form.tsx**: Form wrapper with react-hook-form and Zod; FormField, FormItem, FormLabel, FormControl, FormMessage.
- **hover-card.tsx**: Hover card (Radix HoverCard).
- **input.tsx**: Text input.
- **input-otp.tsx**: OTP input (input-otp).
- **label.tsx**: Label (Radix Label).
- **menubar.tsx**: Menubar (Radix Menubar).
- **navigation-menu.tsx**: Navigation menu (Radix NavigationMenu).
- **pagination.tsx**: Pagination controls.
- **popover.tsx**: Popover (Radix Popover).
- **progress.tsx**: Progress bar (Radix Progress).
- **radio-group.tsx**: Radio group (Radix RadioGroup).
- **resizable.tsx**: Resizable panels (react-resizable-panels).
- **scroll-area.tsx**: Scroll area (Radix ScrollArea).
- **select.tsx**: Select (Radix Select).
- **separator.tsx**: Separator (Radix Separator).
- **sheet.tsx**: Sheet (slide-out panel) (Radix Dialog variant).
- **sidebar.tsx**: Sidebar layout (collapsible, SidebarProvider, SidebarContent, SidebarMenu, etc.).
- **skeleton.tsx**: Loading skeleton.
- **slider.tsx**: Slider (Radix Slider).
- **sonner.tsx**: Sonner toaster (toast container).
- **switch.tsx**: Switch (Radix Switch).
- **table.tsx**: Table, TableHeader, TableBody, TableRow, TableCell, etc.
- **tabs.tsx**: Tabs (Radix Tabs).

- **textarea.tsx**: Textarea.
- **toast.tsx**: Toast component and Toaster (Radix Toast or custom).
- **toaster.tsx**: Toaster container (uses use-toast state).
- **toggle.tsx**: Toggle button (Radix Toggle).
- **toggle-group.tsx**: Toggle group (Radix ToggleGroup).
- **tooltip.tsx**: Tooltip (Radix Tooltip).

---

5. Supabase Edge Functions

Each Edge Function is in `supabase/functions/<name>/index.ts`, served at `SUPABASE_URL/functions/v1/<name>`. They use Deno, Supabase service role for DB, and env vars for secrets (Brevo, BambooHR, Mistral, Pod Journey, etc.).

Edge Function: api

Location: supabase/functions/api/index.ts

Purpose: General API gateway for authenticated requests; supports Bearer JWT or X-API-Key; routes to projects, clients, users, etc.

Trigger: HTTP (GET/POST/PUT/DELETE).

Input: Authorization header (Bearer or X-API-Key); path and body per route.

Processing: getUserFromRequest (API key or JWT); getSupabaseClient(authorization); route handlers for projects, clients, users, contracts, etc.; RLS bypass via service role when using API key.

Output: JSON response; side effects: DB reads/writes.

Used by: Frontend or external API consumers (e.g. Pod Journey, scripts).

Edge Function: pods-api

Location: supabase/functions/pods-api/index.ts

Purpose: POD CRUD and member operations via API; supports Bearer or X-API-Key.

Trigger: HTTP.

Input: Authorization; path (e.g. /pods, /pods/:id); body for create/update.

Processing: getUserFromRequest; getSupabaseClient; CRUD on pods and pod_members; RPC or direct table access.

Output: JSON; DB writes.

Used by: Frontend (POD management), Pod Journey integration, scripts.

Edge Function: bamboo-webhook

Location: supabase/functions/bamboo-webhook/index.ts

Purpose: Receives BambooHR webhooks (employee.created, employee.updated, employee.deleted); syncs employees to RivanIT users table and optionally to Supabase Auth.

Trigger: HTTP POST from BambooHR (webhook).

Input: Signature header for validation; body: event type and employee payload.

Processing: validateSignature(BAMBOO_WEBHOOK_SECRET); mapBambooToUser; upsert users table; optionally sync to Auth (signUp/signIn); HARDCODED_DIRECTORS preserved.

Output: 200 OK or error; side effects: users table and Auth.

Used by: BambooHR webhook configuration.

Edge Function: bamboo-full-sync

Location: supabase/functions/bamboo-full-sync/index.ts

Purpose: Full sync of all BambooHR employees to RivanIT users; used for one-off or scheduled sync.

Trigger: HTTP (manual or cron).

Input: Optional auth; BAMBOOHR_API_KEY, BAMBOOHR_SUBDOMAIN in env.

Processing: fetchAllEmployeesFromBamboo(); mapBambooToUser; determineUserRole(jobTitle, department); syncEmployeeToAuth; upsert users.

Output: JSON (count, errors); side effects: users table and Auth.

Used by: Scripts (run-bamboo-sync.mjs), cron, or manual invoke.

Edge Function: bamboo-cleanup

Location: supabase/functions/bamboo-cleanup/index.ts

Purpose: Remove or deactivate users no longer in BambooHR.

Trigger: HTTP (manual or cron).

Processing: Compare users table to BambooHR; deactivate or delete stale users.

Output: JSON; side effects: users table.

Used by: Scripts (run-bamboo-cleanup.mjs), cron.

Edge Function: bamboo-interviews

Location: supabase/functions/bamboo-interviews/index.ts

Purpose: Fetch candidates from BambooHR for interviews; save evaluations; move candidates (talent pool, questions_cvs, rejected); post comments to BambooHR.

Trigger: HTTP (frontend or cron).

Input: Authorization (JWT or API key); body: action (fetch, save_evaluation, move, etc.) and payload.

Processing: getBambooHeaders(); fetchCandidatesFromBamboo(); fetchEvaluations(); saveEvaluation(); moveToTalentPool/moveToQuestionsCvsFromInterview/moveToRejected; updateBambooStatus; postBambooComment.

Output: JSON; side effects: DB (evaluations, candidate tables), BambooHR API.

Used by: Interviews page, dreamKeeperService, interviewsService.

Edge Function: bamboo-migrate

Location: supabase/functions/bamboo-migrate/index.ts

Purpose: One-off migration of existing users table rows to Supabase Auth (create auth users for existing profiles).

Trigger: HTTP (manual).

Processing: migrateExistingUsersToAuth(); syncEmployeeToAuth per user.

Output: JSON; side effects: Auth users.

Used by: One-time migration script.

Edge Function: bamboohr-update

Location: supabase/functions/bamboohr-update/index.ts

Purpose: Update BambooHR record (e.g. status, custom field, comment) from RivanIT.

Trigger: HTTP.

Input: application_id or employee_id; field updates.

Processing: BambooHR API PATCH or POST comment.

Output: JSON; side effects: BambooHR.

Used by: Staffing/Interview flows when moving candidates.

Edge Function: podjourney-webhook

Location: supabase/functions/podjourney-webhook/index.ts

Purpose: Receives Pod Journey webhooks (pod.created, pod.updated, etc.); links RivanIT POD to Pod Journey POD (update_pod_podjourney_id); syncs members or status.

Trigger: HTTP POST from Pod Journey.

Input: X-Webhook-Secret; body: event, pod (id, dreamit_pod_id, etc.).

Processing: Validate PODJOURNEY_WEBHOOK_SECRET; switch(event): pod.created → update_pod_podjourney_id if dreamit_pod_id; pod.updated → sync; call Pod Journey API if needed.

Output: 200 OK or error; side effects: pods table.

Used by: Pod Journey webhook configuration.

Edge Function: podjourney-sync-frontend

Location: supabase/functions/podjourney-sync-frontend/index.ts

Purpose: Sync PODs from Pod Journey to RivanIT (pull); create or update pods and pod_members in DB.

Trigger: HTTP (frontend or manual).

Input: Optional auth.

Processing: fetchPODsFromPodJourney(); createOrUpdatePODInRivanIT(); syncPODMembers(); uses PODJOURNEY_API_TOKEN.

Output: JSON; side effects: pods, pod_members.

Used by: PodJourneyConfig page, usePodJourneySync (periodic from Layout).

Edge Function: podjourney-sync-periodic

Location: supabase/functions/podjourney-sync-periodic/index.ts

Purpose: Same as podjourney-sync-frontend but intended for cron/scheduled invocation.

Trigger: HTTP (cron).

Processing: Same sync logic as podjourney-sync-frontend.

Output: JSON; side effects: DB.

Used by: Cron job; frontend may call this for periodic sync.

Edge Function: send-email-notification

Location: supabase/functions/send-email-notification/index.ts

Purpose: Send transactional emails via Brevo: PM assignment, PM removal, POD member assignment/removal, resource invitation/assignment/end reminder.

Trigger: HTTP POST from frontend (ApiService.sendEmailNotification) or other Edge Functions.

Input: type (pm_assignment, pod_member_assignment, etc.); recipient and context data (project, POD, resource, etc.).

Processing: Build EmailParams; generate*Email HTML (generateProjectManagerAssignmentEmail, etc.); sendEmailViaBrevo(BREVO_API_KEY).

Output: 200 OK or error; side effects: Brevo send.

Used by: Frontend (PM/POD/resource assignment flows), contractApprovalService or similar.

Edge Function: send-dreamkeeper-interview-invitation

Location: supabase/functions/send-dreamkeeper-interview-invitation/index.ts

Purpose: Send Dream Keeper interview invitation: create Google Calendar event with Meet link; send email via Brevo with calendar attachment.

Trigger: HTTP (frontend or bamboo-interviews).

Input: candidate, interview date/time, interviewer, etc.

Processing: getNextBusinessDay(); generateGoogleCalendarLink; generateICSContent; createGoogleCalendarEventWithMeet (if Google API configured); sendEmailViaBrevo.

Output: JSON; side effects: Calendar event, email.

Used by: Dream Keepers / Interviews flow when scheduling interview.

Edge Function: send-questionnaire-email

Location: supabase/functions/send-questionnaire-email/index.ts

Purpose: Send questionnaire link to candidate (Brevo); used when candidate moves to questionnaire stage.

Trigger: HTTP (sync-questions-cvs-candidates or manual).

Input: application_id, candidate, questionnaire_id, form URL.

Processing: findQuestionnaireByJob(); generateQuestionnaireUrl(); generateQuestionnaireEmail(); sendEmailViaBrevo.

Output: JSON; side effects: email.

Used by: Staffing pipeline (questions/cvs stage).

Edge Function: send-questionnaire-reminders

Location: supabase/functions/send-questionnaire-reminders/index.ts

Purpose: Send reminder emails to candidates who have not completed questionnaire (scheduled).

Trigger: HTTP (cron).

Processing: Query incomplete questionnaire candidates; generate reminder email; sendEmailViaBrevo.

Output: JSON; side effects: email.

Used by: Cron.

Edge Function: send-speaking-test-invitation

Location: supabase/functions/send-speaking-test-invitation/index.ts

Purpose: Generate SpeechAce test link; send invitation email; update BambooHR status and post comment.

Trigger: HTTP (classify-candidates-ai or manual).

Input: application_id, candidate, job title (for course selection).

Processing: classifyJobTitle(); generateTestLink (SpeechAce API); sendEmailInvitation (Brevo); updateBambooHRStatus; addBambooHRComment.

Output: JSON; side effects: BambooHR, email.

Used by: Staffing pipeline (speaking test stage).

Edge Function: send-speaking-test-reminders

Location: supabase/functions/send-speaking-test-reminders/index.ts

Purpose: Send reminders for incomplete speaking tests; optionally regenerate test link.

Trigger: HTTP (cron).

Processing: Query candidates with pending speaking test; regenerateTestLink if needed; sendReminderEmail.

Output: JSON; side effects: email, BambooHR comment.

Used by: Cron.

Edge Function: send-staffing-email

Location: supabase/functions/send-staffing-email/index.ts

Purpose: Send staffing pipeline emails (English test, TestGorilla, etc.) to candidates.

Trigger: HTTP (sync-staffing-candidates, sync-speaking-test-candidates, etc.).

Input: application_id, emailType, candidate.

Processing: generateFormUrl(); generateEnglishTestEmail or generateTestGorillaEmail; sendEmailViaBrevo.

Output: JSON; side effects: email.

Used by: Staffing sync functions.

Edge Function: send-staffing-reminders

Location: supabase/functions/send-staffing-reminders/index.ts

Purpose: Send reminders for incomplete English or TestGorilla tests.

Trigger: HTTP (cron).

Processing: Query pending candidates; generate reminder email; sendEmailViaBrevo.

Output: JSON; side effects: email.

Used by: Cron.

Edge Function: process-questionnaire-response

Location: supabase/functions/process-questionnaire-response/index.ts

Purpose: Handle questionnaire form submission: validate application_id; store responses in questionnaire_responses; analyze with Mistral AI; move candidate to talent pool or rejected; update BambooHR status and post comment.

Trigger: HTTP POST from questionnaire form (public).

Input: application_id, responses (question_id, answer).

Processing: Find candidate in questions_cvs_candidates; analyzeQuestionnaireWithMistralAI; getOrCreateCandidateRecord;

moveToTalentPoolFromQuestionnaire or moveToRejected; updateBambooStatus; post comment to BambooHR.

Output: JSON; side effects: questionnaire_responses, talent_pool or rejected_candidates, BambooHR.

Used by: QuestionnaireForm page (submit).

Edge Function: process-speaking-test-results

Location: supabase/functions/process-speaking-test-results/index.ts

Purpose: Process SpeechAce webhook or manual result: fetch report; extract scores; validate with Mistral AI; approve or reject; move to TestGorilla or rejected; update BambooHR.

Trigger: HTTP (speechace-webhook or manual).

Input: application_id, report URL or payload.

Processing: fetchReportData; extractScores; validateWithMistralAI; determineApproval; moveToTestGorilla or moveToRejected; updateBambooHRStatus; addBambooHRComment.

Output: JSON; side effects: DB, BambooHR, possibly send-staffing-email (TestGorilla).

Used by: speechace-webhook, Staffing pipeline.

Edge Function: process-english-results

Location: supabase/functions/process-english-results/index.ts

Purpose: Process EF Set English test result: scrape certificate URL; analyze with Mistral AI; update BambooHR status and comment; move to talent pool or rejected.

Trigger: HTTP (form submission or cron).

Input: application_id, certificate_url, candidate.

Processing: scrapeEfsetCertificate; analyzeWithMistralAI; generateBambooComment; updateBambooStatus; getOrCreateCandidateRecord; moveToRejected or talent pool.

Output: JSON; side effects: DB, BambooHR.

Used by: English test form or Staffing pipeline.

Edge Function: process-testgorilla-results

Location: supabase/functions/process-testgorilla-results/index.ts

Purpose: (If present) Process TestGorilla result; update BambooHR; move to next stage or rejected.

Trigger: HTTP (webhook or manual).

Processing: Parse TestGorilla payload; analyze; update status; move candidate.

Output: JSON; side effects: DB, BambooHR.

Used by: TestGorilla webhook or check-testgorilla-results.

Edge Function: check-speaking-test-completions

Location: supabase/functions/check-speaking-test-completions/index.ts

Purpose: Poll or check SpeechAce for completed tests; trigger process-speaking-test-results or move candidate.

Trigger: HTTP (cron).

Processing: Query candidates in speaking test stage; check SpeechAce API for completion; process result.

Output: JSON; side effects: DB, BambooHR.

Used by: Cron.

Edge Function: check-testgorilla-results

Location: supabase/functions/check-testgorilla-results/index.ts

Purpose: Check BambooHR comments for TestGorilla scores; analyze with Mistral AI; move to questions_cvs or rejected; optionally send questionnaire email.

Trigger: HTTP (cron).

Processing: Fetch candidates in TestGorilla stage; fetchBambooComments; findTestGorillaComment; parseTestGorillaScores; analyzeWithMistralAI;

moveToQuestionsCvs or moveToRejected; autoSendInitialQuestionnaireEmail.

Output: JSON; side effects: DB, BambooHR, email.

Used by: Cron.

Edge Function: speechace-webhook

Location: supabase/functions/speechace-webhook/index.ts

Purpose: Receive SpeechAce webhook when speaking test is completed; fetch report; extract scores; validate with Mistral AI; update BambooHR; move to TestGorilla or rejected.

Trigger: HTTP POST from SpeechAce.

Input: Webhook payload (application_id, report URL, scores).

Processing: Same as process-speaking-test-results core logic; updateBambooHRStatus; moveToTestGorilla or moveToRejected.

Output: 200 OK or error; side effects: DB, BambooHR.

Used by: SpeechAce webhook configuration.

Edge Function: classify-candidates-ai

Location: supabase/functions/classify-candidates-ai/index.ts

Purpose: Classify new candidates with Mistral AI (CV analysis, job fit); move to speaking test or rejected; update BambooHR status and post comment; optionally send speaking test invitation.

Trigger: HTTP (cron or manual); can target single application_id in development.

Input: Optional body: application_id, process_single.

Processing: Fetch from new_candidates (FIFO or by application_id); fetchCandidateDetails (BambooHR); buildCandidateProfile; buildJobRequirements; classifyWithMistralAI; moveToSpeakingTest or moveToRejected; updateBambooHRStatus; postCommentToBambooHR; optionally call send-speaking-test-invitation.

Output: JSON; side effects: new_candidates (classification_status, cv_analyzed), BambooHR, email.

Used by: Cron (auto-sync-and-classify), Staffing pipeline.

Edge Function: auto-sync-and-classify

Location: supabase/functions/auto-sync-and-classify/index.ts

Purpose: Sync new candidates from BambooHR to new_candidates table; then invoke classify-candidates-ai (or similar) for AI classification.

Trigger: HTTP (cron).

Processing: sync-new-candidates logic; then classify-candidates-ai.

Output: JSON; side effects: DB, BambooHR, Mistral.

Used by: Cron.

Edge Function: sync-new-candidates

Location: supabase/functions/sync-new-candidates/index.ts

Purpose: Fetch new applications from BambooHR; insert into new_candidates table.

Trigger: HTTP (cron or manual).

Processing: BambooHR API for applications; map to new_candidates schema; insert.

Output: JSON; side effects: new_candidates.

Used by: auto-sync-and-classify, scripts.

Edge Function: sync-speaking-test-candidates

Location: supabase/functions/sync-speaking-test-candidates/index.ts

Purpose: Sync candidates in speaking test stage from BambooHR; ensure DB is in sync; optionally send invitations.

Trigger: HTTP (cron).

Processing: Fetch from BambooHR; upsert speaking_test_candidates or similar; send invitation if needed.

Output: JSON; side effects: DB, email.

Used by: Cron.

Edge Function: sync-questions-cvs-candidates

Location: supabase/functions/sync-questions-cvs-candidates/index.ts

Purpose: Sync candidates in questions/cvs stage from BambooHR; optionally send questionnaire email.

Trigger: HTTP (cron).

Processing: fetchCandidatesFromBamboo (status for questions/cvs); syncCandidatesToTable; autoSendInitialQuestionnaireEmail.

Output: JSON; side effects: questions_cvs_candidates, email.

Used by: Cron.

Edge Function: sync-staffing-candidates

Location: supabase/functions/sync-staffing-candidates/index.ts

Purpose: Sync staffing pipeline stages from BambooHR (e.g. English test, TestGorilla); optionally send emails.

Trigger: HTTP (cron).

Processing: fetchCandidatesFromBamboo; syncCandidatesToTable; autoSendInitialTestGorillaEmail or similar.

Output: JSON; side effects: DB, email.

Used by: Cron.

Edge Function: risk-notifications

Location: supabase/functions/risk-notifications/index.ts

Purpose: Send risk-related emails via Brevo: risk_created, default_risks_created, risk_mitigated, risk_report.

Trigger: HTTP (frontend or other Edge Function when risk is created/updated).

Input: RiskNotificationData (type, projectId, projectName, riskData, mitigationData, reportData, recipients).

Processing: sendRiskNotification; generate*Email (generateRiskCreatedEmail, etc.); sendEmailViaBrevo.

Output: JSON; side effects: email.

Used by: riskService, RiskManagement (when creating/updating risks).

Edge Function: change-request-notifications

Location: supabase/functions/change-request-notifications/index.ts

Purpose: Send change request emails: created, approved, rejected, assigned.

Trigger: HTTP (frontend or changeRequestService).

Input: ChangeRequestNotificationData; type (created, approved, rejected, assigned).

Processing: generateChangeRequest*Email; sendChangeRequestNotification (Brevo).

Output: JSON; side effects: email.

Used by: changeRequestService, ChangeRequestTracker.

Edge Function: resource-email-notifications

Location: supabase/functions/resource-email-notifications/index.ts

Purpose: Send resource emails: invitation, assignment, end reminder.

Trigger: HTTP (frontend when assigning/inviting resources).

Input: type, ResourceInvitationData | ResourceAssignmentData | ResourceEndReminderData.

Processing: generateResource*Email; sendEmailViaBrevo.

Output: JSON; side effects: email.

Used by: Resource allocation flows.

Edge Function: weekly-risk-reports

Location: supabase/functions/weekly-risk-reports/index.ts

Purpose: Generate and send weekly risk summary report to directors/PMs (Brevo).

Trigger: HTTP (cron, weekly).

Processing: Query risks; build report; sendRiskNotification type risk_report.

Output: JSON; side effects: email.

Used by: Cron.

Edge Function: generate-api-key

Location: supabase/functions/generate-api-key/index.ts

Purpose: Create a new API key for the authenticated user; insert into api_keys with service role; return key once (client must store).

Trigger: HTTP POST; Bearer JWT required.

Input: body: { name }.

Processing: Verify JWT (supabaseAuth.auth.getUser(token)); generate random key (crypto.getRandomValues); insert api_keys (supabaseService); return { apiKey, name, id }.

Output: JSON with apiKey (only time it is returned); side effects: api_keys table.

Used by: APITokens page (apiKeyService or direct fetch).

Edge Function: jira-proxy

Location: supabase/functions/jira-proxy/index.ts

Purpose: Proxy Jira API requests to avoid CORS and hide token; frontend sends request, function forwards to Jira with JIRA token from env.

Trigger: HTTP (frontend).

Input: path, method, body.

Processing: Forward to Jira API with Authorization; return response.

Output: Jira API response.

Used by: jiraService, JiraKanban.

Edge Function: jira-proxy-project

Location: supabase/functions/jira-proxy-project/index.ts

Purpose: Jira proxy for project-specific operations (e.g. project key, issues for project).

Trigger: HTTP.

Processing: Same as jira-proxy with project context.

Output: Jira response.

Used by: Projects page, Jira integration.

Edge Function: jira-sync-project

Location: supabase/functions/jira-sync-project/index.ts

Purpose: Sync Jira project and issues to RivanIT (store in DB or link project to Jira).

Trigger: HTTP (frontend or manual).

Processing: Fetch Jira project and issues; upsert into projects_jira or similar.

Output: JSON; side effects: DB.

Used by: jiraIntegrationService, Projects.

Edge Function: analyze-candidate-roles

Location: supabase/functions/analyze-candidate-roles/index.ts

Purpose: Analyze candidate fit for roles using Mistral AI (position requirements vs candidate profile).

Trigger: HTTP (frontend or bamboo-interviews).

Input: candidate data, position/role data.

Processing: analyzeWithMistralAI; return fit score and reasoning.

Output: JSON; side effects: none or save analysis.

Used by: Interviews, staffing pipeline.

Edge Function: extract-skills-ai

Location: supabase/functions/extract-skills-ai/index.ts

Purpose: Extract skills from text (CV, job description) using Mistral AI; fallback to simple extraction if no key.

Trigger: HTTP (frontend or other function).

Input: text.

Processing: extractSkillsWithMistral; fallbackExtraction.

Output: JSON { skills: ExtractedSkill[] }; side effects: none.

Used by: Skill matrix, staffing, project setup.

Edge Function: generate-skill-embedding

Location: supabase/functions/generate-skill-embedding/index.ts

Purpose: Generate embedding for a skill (for similarity search or matching).

Trigger: HTTP.

Input: skill name or id.

Processing: Call embedding API (e.g. Mistral); store or return vector.

Output: JSON; side effects: optional DB.

Used by: Skill matrix, matching.

Edge Function: generate-skills-suggestions

Location: supabase/functions/generate-skills-suggestions/index.ts

Purpose: Suggest skills for a role or person based on context (AI or rules).

Trigger: HTTP.

Input: role, context.

Processing: Query or AI; return suggestions.

Output: JSON; side effects: none.

Used by: Skill matrix, onboarding.

Edge Function: validate-skill

Location: supabase/functions/validate-skill/index.ts

Purpose: Validate skill name or id (exists, normalized).

Trigger: HTTP.

Input: skill.

Processing: Lookup in skills table; return valid/invalid.

Output: JSON; side effects: none.

Used by: Frontend (skill input).

Edge Function: execute-migration

Location: supabase/functions/execute-migration/index.ts

Purpose: Run a SQL migration via Edge Function (admin only; service role). Used when CLI cannot be used.

Trigger: HTTP (manual).

Input: migration name or SQL (secured).

Processing: Execute SQL with service role.

Output: JSON; side effects: DB.

Used by: Scripts (apply-migration-via-api.mjs, etc.).

Edge Function: diagnose-allocations, diagnose-more-employees, diagnose-problem-employees

Location: supabase/functions/diagnose-allocations/index.ts, etc.

Purpose: Diagnostic helpers: check resource allocations, duplicate employees, problem employees; return report for debugging.

Trigger: HTTP (manual).

Processing: Query allocations/employees; detect duplicates or issues; return list.

Output: JSON; side effects: none.

Used by: Admins, scripts.

Edge Function: fix-duplicate-allocations, fix-all-duplicates, fix-allocation-function, fix-viviana

Location: supabase/functions/fix-duplicate-allocations/index.ts, etc.

Purpose: One-off fixes: merge duplicate allocations, fix specific user (e.g. Viviana), repair allocation function.

Trigger: HTTP (manual).

Processing: fixDuplicateAllocations; fixEmployeeAllocations; fixVivianaAllocation; etc.

Output: JSON; side effects: DB.

Used by: Admins (one-time).

Edge Function: bamboo-interviews (see above)

Edge Function: projects-test, projects-api-test

Location: supabase/functions/projects-test/index.ts, projects-api-test/index.ts

Purpose: Test endpoints for projects API (auth, RLS, or integration tests); may mirror api/projects behavior.

Trigger: HTTP (tests or manual).

Processing: getSupabaseClient; getUserFromRequest; project CRUD or read.

Output: JSON.

Used by: E2E or integration tests.

Edge Functions: send-dreamkeeper-interview-invitation, send-questionnaire-email, send-speaking-test-invitation, send-staffing-email, resource-email-notifications

(See individual entries above.)

Edge Function: campus-progress-proxy

Location: supabase/functions/campus-progress-proxy/index.ts

Purpose: Proxy requests to the Campus Progress (learning) API to avoid CORS and centralize service credentials.

Trigger: HTTP (frontend)

Processing: Forwards authenticated requests to Campus Progress using service credentials; validates and normalizes responses.

Output: API response; side effects: none.

Used by: Campus progress UI and `sync-campus-progress`.

Edge Function: podjourney-sync

Location: supabase/functions/podjourney-sync/index.ts

Purpose: Orchestrator/wrapper for Pod Journey sync operations (delegates to frontend/periodic sync routes).

Trigger: HTTP (manual/cron)

Processing: Coordinates Pod Journey pull/push workflows; calls Pod Journey API and updates `pods` / `pod_members`.

Output: JSON; side effects: pods, pod_members.

Used by: Manual syncs, cron jobs, Pod Journey onboarding flows.

Edge Function: dora-metrics

Location: supabase/functions/dora-metrics/index.ts

Purpose: Calculate DORA engineering metrics (deployment frequency, lead time for changes, MTTR, change-fail-rate) for projects.

Trigger: HTTP (manual or scheduled)

Processing: Aggregate commit/PR/deploy data (DB or GitHub) and compute metrics; store or return aggregates.

Output: JSON (metrics) and optional DB writes.

Used by: Reports, DirectorDashboard, engineering metrics views.

Edge Function: employee-full

Location: supabase/functions/employee-full/index.ts

Purpose: Return enriched employee profiles (photo URL resolution, BambooHR enrichment, computed fields).

Trigger: HTTP

Processing: Read `users` and external sources (BambooHR), validate photo URLs, enrich profile payload.

Output: JSON; side effects: none.

Used by: Layout avatar, EmployeeCard, skill enrichment.

Edge Function: improve-text-ai

Location: supabase/functions/improve-text-ai/index.ts

Purpose: Improve or normalize free-text (job descriptions, candidate summaries) using AI (Mistral) for consistency and readability.

Trigger: HTTP

Processing: Call AI model to rewrite/normalize input text; return suggestions or update payload.

Output: JSON { suggested_text }; side effects: none.

Used by: Form helpers, staffing templates, content editors.

Edge Function: sync-bamboo-jobs

Location: supabase/functions/sync-bamboo-jobs/index.ts

Purpose: Sync job postings / position data from BambooHR into local `jobs` / `job_applications` tables.

Trigger: HTTP (cron/manual)

Processing: Fetch jobs from BambooHR and upsert to DB.

Output: JSON; side effects: jobs table.

Used by: Staffing pipeline, job listings.

Edge Function: sync-campus-progress

Location: supabase/functions/sync-campus-progress/index.ts

Purpose: Periodically sync learning/progress data from Campus Progress into RivanIT.

Trigger: HTTP (cron)

Processing: Call `campus-progress-proxy`, fetch progress records, upsert to learning tables.

Output: JSON; side effects: learning/progress tables.

Used by: Campus progress UI and reporting.

Edge Function: sync-employee-photos

Location: supabase/functions/sync-employee-photos/index.ts

Purpose: Fetch and cache employee photos from HR systems (BambooHR) into Supabase Storage and update `users.photo_url`.

Trigger: HTTP (cron/manual)

Processing: Download photos, resize/optimize, upload to storage, update DB rows.

Output: JSON; side effects: Storage uploads, `users` updates.

Used by: Layout avatar, employee profile pages.

---

6. UI Components (Summary)

Reusable UI components live in `src/components` (feature) and `src/components/ui` (primitives). Feature components (Layout, AppSidebar, ProjectCard, ContractManagement, etc.) use hooks and services and compose UI primitives. UI primitives are shadcn-style: Radix UI + Tailwind, with consistent variants (e.g. button: default, destructive, outline). Styling is via Tailwind and CSS variables in index.css (–primary, –sidebar-*, etc.). Reusability: UI primitives are used app-wide; feature components are used by pages and other feature components. Theme is provided by ThemeProvider (theme-provider.tsx) and theme-toggle (theme-toggle.tsx); dark mode uses .dark class and CSS variables.

---

7. Hooks (Summary)

Custom hooks encapsulate server state (React Query) and side effects. Key hooks:

- **useAuth**: From AuthContext; provides user, login, logout, signup, signInWithGoogle, updateUser, isLoading, connectionError, reconnect.
- **useProjects / useProject**: Project list and single project; ProjectService; keyed by user and role.
- **useFinancials**: Project financials (P&L, budget, costs, invoices) and mutations (create cost/invoice, submit, approve).
- **useNotifications**: Notification list, unread count, mark as read; realtime subscription via NotificationService.
- **useLoginRedirect**: One-time welcome toast after login.
- **use-toast**: Toast state and API (add, dismiss, update, remove); used by Toaster and components.
- **usePODs**: POD list and mutations; podService.
- **usePermissions**: user.permissions from useAuth.
- **useRisks, usePhases, usePhaseSkip, useResources, useChangeRequests, useCompliance, useHandoverReport, useMitigation, useProjectMetrics, useProjectTimeline, useSkills**: Domain-specific data and mutations; each uses corresponding service and React Query or local state.
- **usePodJourneySync**: Triggers periodic Pod Journey sync (e.g. from Layout every 30 min).
- **use-mobile**: Viewport mobile detection for responsive UI.

Components that rely on hooks: all pages and feature components that need data (Projects, Financials, Layout, etc.).

---

8. Core Application Flows

Authentication and authorization

- **Trigger**: User opens app; production requires login unless on localhost (AppContent skips auth for localhost).
- **Steps**: (1) AuthProvider mounts; local dev: mock user (optional DB fetch for admin@company.example.com) and set user. Production: getSession(); if session, fetchUserProfile(authUser). (2) fetchUserProfile: validate isDomainAllowed(email); lookup users by email; ensureCorrectRole (HARDCODED_DIRECTORS, DOMAIN_ROLE_MAPPING); load permissions via PermissionService.getUserPermissions or createDefaultPermissions; set user in context. (3) Login: LoginForm calls login(email, password), signInWithGoogle() or signInWithMicrosoft(), or sendMagicLink(); AuthContext.login uses signInWithPassword; then fetchUserProfile. (4) Sign-up: SignUpForm calls signup(email, password, userData); AuthContext checks if user exists in users table (BambooHR); if exists, signUp only (link on first login); else signUp with metadata; profile created on first login. (5) Logout: logout() → signOut(); setUser(null).
- **Files**: App.tsx (AppContent, AuthenticatedApp), AuthContext.tsx (AuthProvider, fetchUserProfile, login, signup, signInWithGoogle, sendMagicLink, logout), LoginForm.tsx, SignUpForm.tsx, EmailConfirmationHandler.tsx, GoogleAuthCallback.tsx, permissionService.ts, users and user_permissions tables.
- **Data**: Session (Supabase Auth); users table (id, email, name, role, department, has_completed_onboarding); user_permissions (camelCase in app); User type with permissions and preferences.

Project creation and management

- **Trigger**: User clicks Create Project (from Projects page or dashboard).
- **Steps**: (1) ProjectCreationDialog opens; user fills name, client, type, budget, PM (optional). (2) On submit, ProjectService.createProject(projectData) is called; insert projects row; default phases from projectTemplates inserted; PM assignment if assignedPMId. (3) useProjects invalidated or refetched; redirect or refresh list. (4) Project detail: Projects page with id in params; useProject(projectId); ProjectDashboard shows phases; ContractManagement, FinancialManagement, RiskManagement in tabs. (5) PM assignment change

may trigger send-email-notification Edge Function (pm_assignment or pm_removal).

- **Files**: Projects.tsx, ProjectCreationDialog.tsx, ProjectCard.tsx, ProjectDashboard.tsx, projectService.ts, useProjects.ts, projectTemplates.ts, send-email-notification (Edge Function).
- **Data**: projects, project_phases, deliverables (or equivalent); clients for dropdown.

Contract approval flow

- **Trigger**: User submits contract for approval (ContractManagement or ContractApprovalDialog).
- **Steps**: (1) contractService.updateContractStatus(id, 'under-review') or contractApprovalService.submitForApproval. (2) Director (or approver) opens ContractApprovalDialog; contractApprovalService.approve or reject. (3) On approve: status → approved (or signed); may trigger send-email-notification or in-app notification. (4) Notifications table insert for assignee; useNotifications picks up via realtime.
- **Files**: Contracts.tsx, ContractManagement.tsx, ContractApprovalDialog.tsx, contractService.ts, contractApprovalService.ts, send-email-notification (if used), notificationService.ts, useNotifications.
- **Data**: contracts table; notifications table.

Staffing pipeline (candidate flow)

- **Trigger**: New application in BambooHR; or cron runs sync-new-candidates then classify-candidates-ai.
- **Steps**: (1) BambooHR webhook (employee.created) or sync-new-candidates: new_candidates row. (2) classify-candidates-ai: Mistral classifies; move to speaking_test_candidates or rejected_candidates; BambooHR status and comment. (3) send-speaking-test-invitation: SpeechAce link; email to candidate. (4) Candidate completes test; SpeechAce webhook → speechace-webhook or process-speaking-test-results: move to TestGorilla or rejected; BambooHR update. (5) send-staffing-email (TestGorilla); candidate completes TestGorilla; check-testgorilla-results (cron): parse comment; Mistral; move to

questions_cvs_candidates or rejected; send-questionnaire-email. (6) Candidate submits QuestionnaireForm; process-questionnaire-response: Mistral; move to talent_pool or rejected; BambooHR update. (7) bamboo-interviews: fetch candidates; save evaluation; move to talent pool or questions_cvs or rejected; send-dreamkeeper-interview-invitation for interview.

- **Files**: Staffing.tsx, Interviews.tsx, QuestionnaireForm.tsx, staffingService.ts, interviewsService.ts, dreamKeeperService.ts; Edge Functions: bamboo-webhook, sync-new-candidates, classify-candidates-ai, send-speaking-test-invitation, speechace-webhook, process-speaking-test-results, send-staffing-email, check-testgorilla-results, send-questionnaire-email, process-questionnaire-response, bamboo-interviews, send-dreamkeeper-interview-invitation.
- **Data**: new_candidates, speaking_test_candidates, test_gorilla_candidates, questions_cvs_candidates, talent_pool, rejected_candidates; BambooHR applications and comments.

Pod Journey sync

- **Trigger**: Frontend Layout mounts usePodJourneySync(true) (every 30 min); or user clicks Sync in PodJourneyConfig; or Pod Journey sends webhook.
- **Steps**: (1) podjourney-sync-frontend or podjourney-sync-periodic: fetchPODsFromPodJourney(); createOrUpdatePODInRivanIT(); syncPODMembers(); DB updated. (2) When POD created in RivanIT with Pod Journey link: frontend may call Pod Journey API (podJourneyService) to create POD there; Pod Journey sends pod.created webhook with dreamit_pod_id; podjourney-webhook calls update_pod_podjourney_id.
- **Files**: Layout.tsx (usePodJourneySync), PodJourneyConfig.tsx, podJourneyService.ts, podJourneySyncService.ts, podjourney-sync-frontend, podjourney-sync-periodic, podjourney-webhook; podService.ts.
- **Data**: pods, pod_members; Pod Journey API.

---

9. Root configuration and other files

File: package.json

Type: Config

Responsibility: Project name, version, scripts (dev, build, lint, preview, version:*, supabase:deploy), dependencies (React, Supabase, TanStack Query, Radix, Tailwind, etc.), devDependencies (Vite, TypeScript, ESLint, Supabase CLI).

Used by: npm/bun, Vite, CI.

File: vite.config.ts

Type: Config

Responsibility: Vite config: react plugin, path alias @ → src, define VITE_APP_VERSION from package.json, componentTagger (component-tagger) in development.

Used by: Vite build and dev server.

File: tailwind.config.ts

Type: Config

Responsibility: Tailwind: darkMode class, content paths (src/**), theme extend (colors from CSS variables, radius, keyframes, animations, sidebar colors), plugins (tailwindcss-animate).

Used by: PostCSS/Tailwind.

File: tsconfig.json, tsconfig.app.json, tsconfig.node.json

Type: Config

Responsibility: TypeScript: baseUrl and paths @/* → src; tsconfig.app.json for src (ES2020, JSX, noEmit); tsconfig.node.json for Vite/config.

Used by: TypeScript compiler, IDE.

File: components.json

Type: Config

Responsibility: shadcn/ui schema: style default, tailwind config path, baseColor slate, aliases (components, utils, ui, lib, hooks).

Used by: shadcn CLI when adding components.

File: eslint.config.js

Type: Config

Responsibility: ESLint: ignores dist; TypeScript and React; react-hooks and react-refresh; no-unused-vars off; allowShortCircuit/allowTernary for no-unused-expressions.

Used by: npm run lint, IDE.

File: index.html

Type: Other

Responsibility: HTML entry; title RivanIT; meta description; root div; script src /src/main.tsx.

Used by: Vite dev and build.

File: Dockerfile, docker-compose.yml, nginx.conf

Type: Config / Other

Responsibility: Docker build (Node build, nginx serve static); docker-compose for local run; nginx config for SPA and proxy if needed.

Used by: Deployment, local container run.

File: .env.example

Type: Config

Responsibility: Example env vars: VITE_SUPABASE_URL, VITE_SUPABASE_ANON_KEY, SUPABASE_SERVICE_ROLE_KEY, BAMBOO_WEBHOOK_SECRET, VITE_JIRA, VITE_JIRA_EMAIL, DENO_ENV, etc.

Used by: Developers to create .env.local.

Scripts folder (summary)

Purpose: Operational and one-off scripts; not part of runtime app.

Key scripts: run-bamboo-sync.mjs, run-bamboo-cleanup.mjs (BambooHR sync/cleanup); setup-bamboo-sync.mjs (Bamboo webhook setup); setup-podjourney-config.mjs, execute-podjourney-config-direct.mjs (Pod Journey config); check_api_keys.mjs, insert-api-key.sql (API keys); backup-cloud-db.sh, backup-cloud-db-cli.sh (DB backup); list-active-migrations.sh, mark-migration-applied.mjs (migrations); setup-e2e-auth.sh, setup-e2e-auth-mock-only.sh (E2E auth); update-version.mjs (version bump); test-api-endpoints.mjs, test-project-employees-endpoint.mjs (API tests). Archive folder contains old migration and fix scripts (apply-migrations.mjs, execute_migration.mjs, fix-adrian-*.mjs, etc.). Markdown: PODJOURNEY_SETUP.md, podjourney-webhook-instructions.md, create-contracts-bucket.md, SCRIPTS_AUDIT.md.

Used by: Developers, CI, cron.

Tests folder (summary)

Purpose: E2E (Playwright) and unit (Vitest) tests.

E2E: tests/e2e/ — global-setup.ts, global-teardown.ts; auth (login.spec.ts, google-auth-example.spec.ts); clients (client-crud.spec.ts); contracts (contract-crud.spec.ts, contract-rates.spec.ts, etc.); projects (project-crud.spec.ts, project-management.spec.ts); utils/database-helpers.ts. Playwright config in playwright.config.ts.

Unit: tests/unit/ — setup.ts; hooks/useAuth.test.ts; services (billingService.test.ts, clientService.test.ts, contractService.test.ts, contractApprovalService.test.ts, financialService.test.ts, projectService.test.ts); components/crud-ui.spec.tsx; fixtures/testData.ts; mocks/handlers.ts, server.ts (MSW). Vitest config in vitest.config.ts.

Used by: npm test / playwright / vitest.

Final checklist

- **Folders**: Documented (/, src, src/components, src/components/ui, src/contexts, src/data, src/hooks, src/lib, src/pages, src/types, public,

supabase, supabase/functions, supabase/migrations, supabase/templates, scripts, tests, archive_backup_baseline_migrations, coverage, playwright-report, test-results).

- **Files**: All key files in src (App, main, contexts, types, data, lib services, hooks, pages, components) documented; root config and entry documented; scripts and tests summarized.
- **Edge functions**: Each Edge Function has an entry in section 5 with location, purpose, trigger, input, processing, output, and used-by.
- **Onboarding**: A new senior developer can use this document to understand project overview, architecture, folder structure, file responsibilities, Edge Function behavior, UI/hooks usage, and core flows (auth, project, contract, staffing, Pod Journey) without reading all code first.