TECHNICAL_DOCUMENT

RivanIT - Technical Documentation

Version: 1.0.0
Last Updated: April 2026
Classification: Internal / Partner

---

1. Executive Summary

RivanIT is an enterprise-grade Agentic Delivery Platform for end-to-end project management. It provides a unified system for managing clients, projects, contracts, financials, resources, risk, recruitment, and team performance across distributed delivery organizations.

The platform is built on a modern serverless architecture (React + Supabase) and is designed for multi-tenant deployment, role-based access control, and optional integration with third-party services.

---

2. Architecture Overview

2.1 High-Level Architecture

```
+---------------------+      +-------------------+      +----
----------------+
|    Frontend (SPA)    | ---> | Supabase Platform | ---> | Ex
ternal Services  |
|  React 18 + Vite     |      | PostgreSQL + Auth  |      |
(Optional)         |
|  shadcn/ui + Tailwind|      | Edge Functions     |      | J
ira, BambooHR,    |
|  TanStack Query      |      | Storage + Realtime |      | G
itHub, Brevo, AI  |
```

```
+----------------------+      +--------------------+      +----
----------------+
```

2.2 Application Layers

Layer	Technology	Responsibility
UI	React 18, shadcn/ui, Tailwind CSS	Pages, components, layout
State	TanStack Query, AuthContext	Server state caching, auth session
Business Logic	Hooks + Service classes	Data fetching, mutations, domain logic
Data Access	Supabase JS client	PostgREST, RPC, Auth, Realtime
Backend	Supabase Edge Functions (Deno)	Webhooks, sync, email, AI, proxy
Database	PostgreSQL with RLS	Schema, row-level security, migrations

2.3 Data Flow

1. User authenticates via Supabase Auth (Google OAuth, Microsoft Azure AD, email/password, magic links)
2. AuthContext loads user profile + permissions from `users` and `user_permissions` tables
3. Pages use hooks (e.g., `useProjects`, `useFinancials`) that call service classes
4. Services use Supabase client (`supabase.from()`, `supabase.rpc()`) for DB operations
5. Edge Functions handle: webhooks, email delivery, AI processing, external API proxying
6. Notifications written to DB and delivered via in-app + email (Brevo)

---

3. Tech Stack

3.1 Frontend

Package	Version	Purpose
React	18.3.x	UI library
TypeScript	5.5.x	Type safety
Vite	5.4.x	Build tool + dev server
React Router	6.26.x	Client-side routing
TanStack Query	5.56.x	Server state management
shadcn/ui	Latest	UI component primitives (Radix + Tailwind)
Tailwind CSS	3.4.x	Utility-first CSS
Framer Motion	12.x	Animations and transitions
Recharts	2.12.x	Charts and data visualization
React Hook Form + Zod	7.x + 3.x	Form management + validation
Lucide React	0.462.x	Icon library
jsPDF + AutoTable	3.x + 5.x	Client-side PDF generation
date-fns	3.6.x	Date utilities
Gantt Task React	0.3.x	Gantt chart component

3.2 Backend

Technology	Purpose
Supabase	Backend-as-a-Service (PostgreSQL, Auth, Storage, Realtime, Edge Functions)
PostgreSQL	Primary database with Row Level Security
Deno Edge Functions	Serverless HTTP handlers for business logic
Supabase Auth	Authentication with OAuth (Google, Azure AD), magic links, email/password
Supabase Storage	File storage for contracts, avatars, documents
Supabase Realtime	Real-time data subscriptions

3.3 External Integrations (All Optional)

Service	Purpose	Configuration
Jira	Issue tracking, sprint metrics, worklogs	Per-project token in DB
GitHub	Repository metrics, PR tracking	Per-project token in DB
BambooHR	Employee data, candidate pipeline	API key in env secrets
Brevo	Transactional email delivery	API key in env secrets
Google Gemini	AI text improvement, skill extraction, candidate classification	API key in env secrets
SpeechAce	English proficiency testing	API key in env secrets
TestGorilla	Technical skill assessments	API key in env secrets

4. Database Schema

4.1 Core Tables

Table	Purpose	Key Columns
`users`	User profiles	id, email, name, role, department, bamboo_id, employee_photo
`user_permissions`	Granular RBAC permissions	user_id, can_view_all_projects, can_create_projects, ...
`clients`	Client organizations	id, name, vertical, region, tier, portfolio, email
`projects`	Project records	id, name, client_id, type, status, current_pm, github_*
`project_phases`	Phase lifecycle	id, project_id, name, status, progress, start_date, end_date
`deliverables`	Phase deliverables	id, phase_id, title, status, completion_percentage

Table	Purpose	Key Columns
`contracts`	Contract records	id, project_id, status, milestones, payment_terms
`contract_rates`	Billing rates per position	id, contract_id, bamboo_job_id, hourly_rate, currency
`contract_approvals`	Approval workflow records	id, contract_id, approver_id, status
`invoices`	Invoice records	id, contract_id, status, amount, due_date
`resources`	Team members as resources	id, user_id, availability_status
`resource_allocations`	Resource-to-project assignments	id, resource_id, project_id, percentage, start_date, end_date
`pods`	POD team definitions	id, project_id, name, category, leader_id, podjourney_id
`pod_members`	POD membership	id, pod_id, user_id, role, position
`risks`	Risk register	id, project_id, title, probability, impact, status
`mitigations`	Mitigation plans	id, risk_id, plan, owner_id, status
`change_requests`	Change request tracking	id, project_id, title, status, requested_by
`notifications`	In-app notifications	id, user_id, type, title, message, is_read
`time_tracking`	Time entries	id, user_id, project_id, hours, date
`api_keys`	API key management	id, user_id, key_hash, name, is_active
`skills`	Skill definitions	id, name, category
`user_skills`	User-to-skill mapping	id, user_id, skill_id, proficiency_level

4.2 Staffing / Recruitment Tables

Table	Purpose
`new_candidates`	Incoming candidates from BambooHR
`speaking_tests_candidates`	English proficiency test tracking
`gorilla_test_candidates`	Technical assessment tracking

Table	Purpose
`candidate_records`	Candidate evaluation history
`talent_pool`	Qualified candidates ready for placement
`rejected_candidates`	Rejected candidates with reasons
`bamboo_jobs`	Job postings (positions for billing rates)

4.3 Integration Tables

Table	Purpose
`jira_project_integrations`	Per-project Jira connection config
`jira_issues`	Synced Jira issues
`jira_worklogs`	Synced Jira time entries
`podjourney_config`	Pod Journey sync configuration
`knowbe4_*`	KnowBe4 security training data (account, groups, users, tests, campaigns)

4.4 Row Level Security (RLS)

All tables have RLS enabled. Key policies:
- **Users:** Can read own profile; directors can read all
- **Projects:** Users see projects they're assigned to; directors see all
- **Financials:** Only PM and director roles can view financial data
- **Notifications:** Users see only their own notifications
- **Service role** (edge functions) bypasses RLS for privileged operations

---

5. Authentication & Authorization

5.1 Authentication Methods

Method	Provider	Use Case
Google OAuth	Supabase + Google Cloud	Primary SSO for Google Workspace orgs
Microsoft Azure AD	Supabase + Azure	Primary SSO for Microsoft 365 orgs

Method	Provider	Use Case
Email + Password	Supabase Auth	Direct registration
Magic Links	Supabase Auth	Passwordless login for configured domains

5.2 RBAC Model

Permissions are resolved in-memory at login from the `rbac.ts` config:

Role	Access Level
Director	Full access to all modules. Short-circuits all permission checks.
PM	Project CRUD, resource allocation, contracts, financials, risk, staffing
POD Leader	Own projects + POD, risk management, talent training
Member	Own projects (timeline only), skills, time tracking

Additional additive permission: `isDreamKeeper` grants access to the Dream Keepers mentoring module.

5.3 Domain-Based Access

`ALLOWED_DOMAINS` in `AuthContext.tsx` can restrict login to specific email domains. When empty, any domain is allowed (multi-tenant mode).

---

6. Edge Functions

6.1 Overview

25+ Deno-based Edge Functions deployed on Supabase. Invoked via HTTP from the frontend (`ApiService.callEdgeFunction`) or external webhooks.

6.2 Function Categories

API & Portal:

- `api` - General API gateway
- `pods-api` - POD management API
- `portal-api` - Client portal data API
- `portal-login` - Client portal authentication
- `invite-portal-user` - Portal user invitation

- `generate-api-key` - API key generation
- `setup-portal-demo` - Demo data seeding

Notifications:
- `send-email-notification` - Generic email via Brevo
- `change-request-notifications` - Change request alerts
- `risk-notifications` - Risk alert emails
- `resource-email-notifications` - Resource assignment emails
- `weekly-risk-reports` - Scheduled risk digests
- `project-performance-alerts` - Performance threshold alerts

Jira Integration:
- `jira-proxy` - Jira API proxy (CORS bypass)
- `jira-proxy-project` - Project-aware Jira proxy
- `jira-sync-project` - Issue + worklog sync
- `jira-sprint-capacity` - Sprint capacity calculations

AI & Skills:
- `improve-text-ai` - AI-powered text enhancement (Gemini)
- `extract-skills-ai` - Skill extraction from job requirements
- `generate-skills-suggestions` - Skill recommendations
- `generate-skill-embedding` - Skill vector embeddings
- `validate-skill` - Skill validation

Staffing Pipeline:
- `bamboo-webhook` - BambooHR event listener
- `bamboo-full-sync` - Full employee data sync
- `bamboo-interviews` - Interview candidate data
- `classify-candidates-ai` - AI candidate classification
- `send-speaking-test-invitation` - English test invitations
- `send-gorilla-test-invitations` - Technical test invitations
- Plus: reminders, completions, questionnaires, sync functions

Analytics:
- `dora-metrics` - DORA performance metrics
- `project-insights` - AI-powered project analytics

---

7. Deployment

7.1 Prerequisites

- Node.js 18+ and npm/bun
- Supabase CLI (`npm install -g supabase`)
- A Supabase project (free or Pro tier)

7.2 Environment Variables

```
# Required
VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY=your-anon-key

# Optional (set in Supabase Edge Function secrets)
MISTRAL_API_KEY=...          # AI features
BREVO_API_KEY=...            # Email delivery
BAMBOO_API_KEY=...           # BambooHR integration
PODJOURNEY_API_TOKEN=...     # Pod Journey sync
```

7.3 Build & Deploy

```
# Install dependencies
npm install

# Run locally
npm run dev

# Build for production
npm run build          # outputs dist/

# Push database migrations
supabase link --project-ref <ref>
supabase db push
```

```
# Deploy edge functions
supabase functions deploy
```

7.4 Docker Deployment

```
docker-compose up -d   # Builds and runs with nginx
```

The `Dockerfile` produces a multi-stage build: Node for building, nginx for serving. The `docker-compose.yml` includes health checks and Traefik labels.

7.5 Multi-Tenant Deployment

Each customer gets:
1. A separate Supabase project (complete data isolation)
2. Frontend deployed with customer-specific env vars
3. Edge functions deployed per project
4. Optional: custom domain via DNS + SSL

---

8. API Reference

8.1 Supabase Client API

All frontend data access goes through the Supabase JS client:

```
// Direct table access (with RLS)
const { data } = await supabase.from('projects').select('*');

// RPC functions (bypass RLS where needed)
const { data } = await supabase.rpc('get_clients');
const { data } = await supabase.rpc('create_contract', { ...
});
const { data } = await supabase.rpc('get_project_contracts',
{ p_project_id: id });

// Edge function calls
```

```
const result = await ApiService.callEdgeFunction('send-email-notification', { ... });
```

8.2 Key RPC Functions

Function	Purpose
`get_clients`	List clients (RLS bypass)
`create_contract`	Create contract with milestones
`get_project_contracts`	Contracts by project
`create_resource_allocation_v2`	Allocate resource to project
`rotate_resource_allocation_v2`	Move resource between projects
`end_allocation`	End a resource allocation
`get_project_resources`	Resources for a project
`get_time_tracking_entries`	Time entries with filters
`get_pod_members_with_details`	POD members with profiles
`update_pod_podjourney_id`	Link POD to external system

8.3 External API (API Keys)

Edge functions `api` and `pods-api` accept `X-API-Key` header for external integrations:

```
GET /functions/v1/api/pods-by-project/{projectId}
Authorization: Bearer <jwt> | X-API-Key: <key>
```

---

9. Security Considerations

Area	Implementation
Authentication	Supabase Auth with PKCE flow, OAuth 2.0
Authorization	Row Level Security on all tables + in-app RBAC
Data Isolation	Per-tenant Supabase projects
API Security	JWT tokens + API key authentication
Secrets	Stored in Supabase Edge Function secrets (never in frontend)

Area	Implementation
Domain Restriction	Configurable ALLOWED_DOMAINS in AuthContext
Session Management	Auto-refresh tokens, persistent sessions
Input Validation	Zod schemas on forms, server-side validation in RPC
CORS	Edge Functions handle CORS headers; Vite proxy for dev

10. Monitoring & Observability

Feature	Tool
Edge Function logs	Supabase Dashboard > Edge Functions > Logs
Database queries	Supabase Dashboard > SQL Editor
Auth events	Supabase Dashboard > Authentication > Users
Frontend errors	Browser DevTools console
Performance	Lighthouse CI (`.lighthouserc.json` configured)
Tracing (optional)	Phoenix OTLP integration (`_shared/phoenix_otlp.ts`)

11. Testing

Type	Tool	Location
Unit tests	Vitest	`tests/unit/`
E2E tests	Playwright	`tests/e2e/`
Coverage	Vitest + Istanbul	`coverage/`

```
# Run unit tests
npx vitest

# Run E2E tests
npx playwright test

# Run with coverage
npx vitest --coverage
```

12. Project Structure

```
/
├── src/
│   ├── components/          # React components (feature + UI
primitives)
│   │   ├── ui/              # shadcn/ui components (button,
card, dialog, etc.)
│   │   └── portal/          # Client portal components
│   ├── contexts/            # React contexts (AuthContext, P
ortalAuthContext)
│   ├── hooks/               # Custom hooks (useProjects, use
Financials, etc.)
│   ├── lib/                 # Service classes + utilities (s
upabase, api, services)
│   ├── pages/               # Route-level page components
│   │   └── portal/          # Client portal pages
│   ├── types/               # TypeScript interfaces and type
s
│   ├── config/              # RBAC configuration
│   └── data/                # Mock data and templates
├── supabase/
│   ├── functions/           # Edge Functions (one folder per
function)
│   ├── migrations/          # PostgreSQL migrations (230+)
│   └── templates/           # Auth email templates
├── public/                  # Static assets
├── scripts/                 # Operational scripts
├── tests/                   # Unit and E2E tests
└── docs/                    # Documentation
```