Certivo

Developer Architecture Guide

Version: 8.0

Stack: Vite 5 + React 18 + Tailwind CSS + Firebase 10.7.1 + Stripe

Codebase: ~53,500 lines | 235+ files | 4 apps | 62 Cloud Functions

Last Updated: March 2026

Version: 10.0 | **Last Updated:** 2026-03-17 **Stack:** Vite 5 + React 18 + Tailwind CSS + Firebase 10.7.1 Modular SDK + Stripe + Twilio **Codebase:** ~66,200 lines | 262 files | 4 apps + landing page | 80 Cloud Functions | 44 Firestore collections

This document explains how the Certivo codebase is organized, where everything lives, why it's structured that way, and how the pieces interact. Read this before diving into the code.

Table of Contents

---

1. What Certivo Does

Certivo is a multi-tenant SaaS platform for field safety operations. It handles:

- **Scheduling** safety training classes
- **Assigning** certified instructors to teach them
- **Enrolling** students and managing rosters
- **Issuing** certificates (PDF generation, expiry tracking)
- **Invoicing** clients and collecting payments (Stripe)
- **Paying** instructors (payroll with Canadian tax deductions)
- **Compliance** tracking (cert expiry, incident reporting, audit logs)

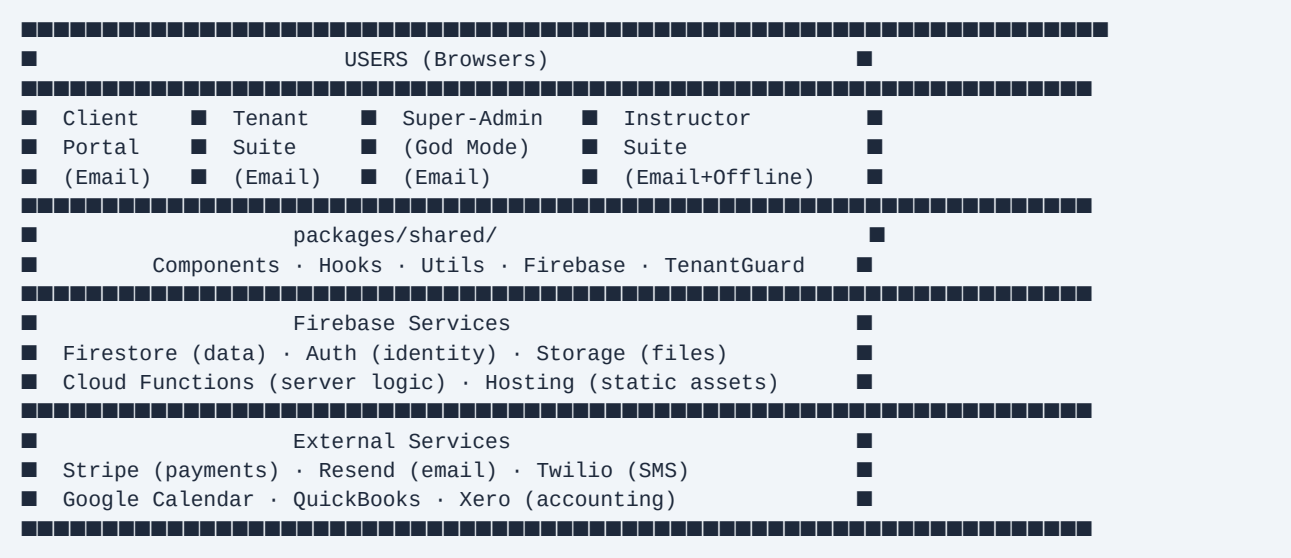
The critical business path:

```
Schedule class → Assign instructor → Enroll students → Teach class →
Issue certificates → Invoice client → Collect payment → Pay instructor
```

Every step in this chain writes to real Firestore collections with server-side financial calculations.

---

2. High-Level Architecture



Why 4 separate apps instead of 1? Each app serves a different user role with different auth methods, different data access patterns, and different security requirements. Separating them means:

- A client can never accidentally see admin UI
- Each app can be deployed independently
- Security rules are simpler (each app has a known scope)
- Bundle sizes stay small (each app only ships its own views)

3. Repository Structure

```
R4E Enterprise/
■
■■■ apps/                          ← The 4 front-end applications
■   ■■■ client-portal/             ← Client self-service (email auth)
■   ■■■ tenant-suite/              ← Admin command center (email auth)
■   ■■■ super-admin/               ← Platform god-mode (email auth)
■   ■■■ instructor-suite/          ← Field operations (email auth + offline)
■
■■■ packages/shared/               ← Shared library used by ALL 4 apps
■   ■■■ components/                ← 29 reusable React components
■   ■■■ hooks/                     ← 7 React hooks (toast, confirm, idle, feature gate, etc.)
■   ■■■ styles/                    ← Shared CSS (glass panels, animations)
■   ■■■ *.js                       ← 10 utility modules (firebase, auth, etc.)
■
■■■ functions/                     ← Firebase Cloud Functions (server-side)
■   ■■■ index.js                   ← 80 function definitions
■   ■■■ lib/                       ← Shared modules (auth, RBAC, rate-limit)
■   ■■■ email-templates.js         ← 6 HTML email templates
■   ■■■ pdf-templates.js           ← 5 PDF builders
■   ■■■ __tests__/                 ← 4 test suites
■
■■■ tests/                         ← 23 unit test suites (Vitest)
■■■ e2e/                           ← 8 E2E test files (Playwright)
■
■■■ docs/                          ← Documentation
■   ■■■ DEVELOPER-GUIDE.md         ← THIS FILE
■   ■■■ SOC2-COMPLIANCE.md         ← Auditor-ready compliance mapping
■   ■■■ STAGING-SETUP.md           ← Firebase staging environment guide
■   ■■■ RESEND-EMAIL-SETUP.md      ← Email domain verification guide
■
■■■ .github/workflows/             ← CI/CD pipelines
■   ■■■ deploy.yml                 ← Production deploy (master push)
■   ■■■ deploy-staging.yml         ← Staging deploy (develop push)
■   ■■■ promote-production.yml     ← Manual production promotion
■   ■■■ pr-check.yml               ← PR validation (test + build)
■
■■■ scripts/                       ← Developer tooling
■   ■■■ install-r4e.ps1            ← PowerShell installer
■   ■■■ install-r4e.bat            ← Batch wrapper
■   ■■■ Install-R4E.exe            ← Self-extracting installer
■   ■■■ post-build.js              ← Post-build asset copier
■
■■■ CLAUDE.md                      ← AI coding instructions (architecture + rules)
■■■ vite.config.js                 ← Multi-page Vite build config
■■■ tailwind.config.js             ← Shared Tailwind config
■■■ firebase.json                  ← Firebase hosting (4 targets)
■■■ firestore.rules                ← Firestore security rules (580 lines)
■■■ firestore.indexes.json         ← 43 composite indexes
■■■ storage.rules                  ← Storage security rules (127 lines)
■■■ package.json                   ← Dependencies + npm scripts
```

Why This Structure?

- `apps/` **separation** — Each app is independently deployable with its own `index.html`, `main.jsx`, `styles.css`, and `data-service.js`. This keeps concerns isolated.
- `packages/shared/` — Shared code lives in one place, imported via the `@shared` Vite alias. This prevents code duplication across the 4 apps while keeping each app's bundle small through tree-shaking.
- `functions/` — Server-side logic runs on Firebase Cloud Functions. Financial calculations, auth operations, and data validation all happen here — never in the browser.
- `CURRENT MAIN APPS/` — The original single-file HTML apps before the Vite migration. Kept as rollback reference. Do not modify these.

---

4. The Four Apps

App Comparison

Property	Client Portal	Tenant Suite	Super-Admin	Instructor Suite
Purpose	Client self-service	Admin command center	Platform god-mode	Field operations
Users	External clients	Org administrators	Platform operators	Instructors
Entry	`apps/client-portal/`	`apps/tenant-suite/`	`apps/super-admin/`	`apps/instructor-suite/`
Lines	~2,826	~17,101	~6,265	~5,025
Views	18	29	28	21
Auth	Email + Password	Email + Password	Email + Password	Email + Password
Theme	Dark (glass-morphism)	Light/Dark (WCAG AA)	Dark (purple gradient)	Light/Dark (CSS vars)
Offline	No	No	No	Yes (IndexedDB queue)

App Internal Structure (Same Pattern for All 4)

Each app follows the same file organization:

```
apps/{app-name}/
■■■ index.html          ← HTML entry point (loads main.jsx, CDN libs if needed)
■■■ main.jsx            ← React root: auth flow, navigation state, layout shell
■■■ styles.css          ← App-specific CSS variables and theme
■■■ data-service.js     ← DataService object: Firestore CRUD organized by domain
■■■ views/              ← One .jsx file per view/modal
    ■■■ DashboardView.jsx
    ■■■ SettingsView.jsx
    ■■■ ...
```

Why `data-service.js`? Each app has a `DataService` object that wraps all Firestore operations (reads, writes, queries). This centralizes data access in one place, making it easy to:

- Add TenantGuard scoping to every query
- Track which collections an app touches
- Swap implementations (e.g., mock for testing)

Why `views/` folder? Each view is a separate `.jsx` file. This enables code splitting via `React.lazy()`:

```
const DashboardView = lazy(() => import('./views/DashboardView'));
```

The `main.jsx` file handles navigation with `useState` (no router library).

Client Portal (apps/client-portal/)

Auth: Email + Password. Users sign in with Firebase Auth:

1. `signInWithEmailAndPassword(auth, email, password)`
2. Calls `initClientClaimsV2` Cloud Function to sync custom claims (`orgId`, `clientId`, `role: 'client'`)
3. Client refreshes token to get claims

Key views: QuoteBuilder (instant quotes with tax breakdown), ComplianceMatrix (staff cert tracking), StoreView (supply ordering + Stripe checkout), FinancialsView (invoice history + payments), SupportWidget (floating help button)

Theme: Dark-only glass-morphism. Uses `glass-panel` and `glass-input` CSS classes defined in `packages/shared/styles/shared.css`.

Tenant Suite (apps/tenant-suite/)

The largest app (~17,100 lines, 29 views). This is the admin command center where organizations manage their entire safety operation.

Key views: DashboardView (KPIs + watchtower alerts), ScheduleView (class calendar), BillingManager (invoicing + Stripe), PayrollManager (instructor payroll + Canadian taxes), ClientManager (CRM), InstructorManager (roster), CommunicationsSuite (email + drip campaigns), IntegrationsManager (Google Calendar, QuickBooks, Xero), SettingsManager (org config + MFA)

DataService: The largest data-service.js (~2,800 lines) with 11 domain modules and 80+ methods: `classes`, `students`, `instructors`, `invoices`, `expenses`, `clients`, `inventory`, `reports`, `notifications`, `wallet`, `audit`

Super-Admin (apps/super-admin/)

Cross-tenant. This app intentionally does NOT use TenantGuard — it needs to see all orgs.

Key features:

- **Ghost Mode** — Impersonate any tenant org (via `ghostLogin` Cloud Function + custom token)
- **Command Palette** — Ctrl+K keyboard shortcut for quick navigation
- **Tenant Provisioning** — Create new orgs with ProvisionModal
- **Zen Mode** — Distraction-free dashboard view

Sidebar: 13 nav items organized into 3 color groups:

- Purple (super): Admin functions (Tenants, Subscriptions, Users, Audit, Settings)
- Blue: Data/analytics (Supply Chain, Fintech, Analytics, Integrations)
- Red: Safety/security (Safety Command, Security & Access)

Instructor Suite (apps/instructor-suite/)

Offline-first. This app is designed for instructors in the field with unreliable connectivity.

Key features:

- **Service Worker** — Cache-first strategy for offline access
- **LocalActionQueue** — IndexedDB-backed queue that stores Firestore writes when offline

and flushes them when connectivity returns (7 operation types)

- **WorkflowWizard** — Step-by-step class teaching flow (check-in → attendance → certify)
- **WalletView** — Earnings, tax estimates, mileage claims, expense receipts
- **MarketView** — Available shifts to claim (transaction-based instant booking)

Offline queue pattern:

```
if (navigator.onLine) {
  await updateDoc(doc(db, "classes", id), data);
} else {
  LocalActionQueue.add({ type: 'UPDATE_CLASS', payload: { id, ...data } });
}
```

5. Shared Library (packages/shared/)

Everything in `packages/shared/` is imported via the `@shared` Vite alias:

```
import Button from '@shared/components/Button';
import { useToast } from '@shared/hooks/useToast';
import { db } from '@shared/firebase';
```

Components (29 total)

Component	Purpose	Used By
`Icon`	118 named icons (Lucide-compatible)	All 4 apps
`Button`	6 variants (primary, secondary, danger, ghost, outline, link) + loading	All 4 apps
`Card`, `StatCard`	Content containers and metric display	All 4 apps
`Badge`	Status pills (green, red, blue, yellow, slate)	All 4 apps
`Input`, `Select`, `Textarea`, `Switch`	Form controls with consistent styling	All 4 apps
`Tabs`	Tab bar navigation	All 4 apps
`ModalWrapper`	Modal overlay (sm/md/lg/xl sizes + loading state)	All 4 apps
`DataGrid`	Table with column definitions or custom render rows	Tenant, Super Admin
`FormRenderer`	Dynamic form builder from JSON schema	Tenant Suite
`StepWizard`	Multi-step wizard container	Tenant Suite
`EmptyState`	Empty data placeholder (icon + title + description + action)	All 4 apps
`Alert`	Contextual alert boxes (info, warning, error, success)	All 4 apps
`Avatar`	User avatar with initials fallback	Tenant, Super Admin
`Tooltip`	Hover tooltip	Super Admin
`Money`	Currency display with formatting	Tenant, Client Portal
`WelcomeBanner`	Greeting banner with action buttons	Client Portal
`ErrorBoundary`	React error boundary (wraps every lazy-loaded view)	All 4 apps
`LoadingScreen`	Full-page loading spinner	All 4 apps
`SessionWarningModal`	Idle timeout countdown modal	All 4 apps
`EnvironmentBanner`	Dev/staging environment indicator (hidden in production)	All 4 apps
`UpgradeModal`	Feature gate upgrade prompt with plan comparison	All 4 apps
`NPSModal`	Net Promoter Score survey collection	Tenant, Instructor
`CSATModal`	Customer satisfaction survey after key actions	All 4 apps
`InstructorProfileCard`	Instructor portfolio display (photo, certs, ratings)	Tenant, Instructor
`MarketplaceListing`	Certification listing card for marketplace browse	Client Portal, Tenant

Brand & Illustrations (`components/brand/`, `components/illustrations/`): `R4ELogo`, `AppIcon`, `LoginHeroIllustration`, `EmptyDataIllustration`, `ErrorIllustration`, `SearchIllustration`, `SuccessIllustration` — SVG components for consistent branding.

Hooks (7 total)

Hook	Purpose
`useToast()`	Show toast notifications (success, error, warning, info)
`useConfirm()`	Show confirmation dialogs (replaces `window.confirm()`)
`useIdleTimeout()`	Session timeout with warning countdown modal
`usePaginatedQuery()`	Cursor-based Firestore pagination
`usePlatformPlans()`	Fetch subscription plan data
`useSubscription()`	Current org subscription status
`useFeatureGate()`	Check feature access by plan tier, show UpgradeModal when gated

Utility Modules (10 total)

Module	Purpose
`firebase.js`	Firebase app initialization → exports `{ app, db, auth, storage, functions }`
`auth.js`	Auth helpers: `loginWithEmail`, `loginWithPin`, `logout`, `onAuthChange`
`tenant-guard.js`	`TenantGuard` (full) + `SimpleTenantGuard` (Client Portal)
`cloud-functions.js`	40 `httpsCallable` wrappers (e.g., `callRunPayroll`, `callCalculateInvoice`)
`constants.js`	`USER_ROLES`, `PERMISSIONS`, `PLATFORM_FEE_PERCENT`
`utils.js`	`formatCurrency`, `formatDateSafe`, `formatDate`
`env.js`	`getEnv()`, `isProduction()`, `isStaging()`, `isDevelopment()`
`crash-logger.js`	Black Box Recorder (crash logging for Tenant Suite)
`error-reporter.js`	Error monitoring (Sentry-ready, wired in all 4 apps)
`firestore-retry.js`	Exponential backoff wrapper for Firestore writes

Styles (shared.css)

Defines cross-app CSS classes:

- `.glass-panel` — Semi-transparent card with backdrop blur (used heavily in Client Portal)
- `.glass-card` — Standard card with theme-aware border
- Animation utilities: `.animate-fade-in`, `.animate-slide-up`, `.animate-scale-in`, etc.
- `.stagger-children` — Sequential child animation (60ms delay per child)
- `.skeleton` — Shimmer loading placeholders
- `.custom-scrollbar` — Theme-aware scrollbar styling

---

6. Cloud Functions (functions/)

Runtime: Node.js 20 | **Region:** `northamerica-northeast2` (Montreal) **Total:** 80 functions | **Lines:** ~13,000

Function Categories

Category	Count	Rate Limit	Examples
Financial	9	10/min	`runPayroll`, `calculateInvoice`, `createPaymentSession`
Auth	5	5/5min	`initUserClaims`, `initClientClaimsV2`, `ghostLogin`
Data	15	20/min	`createUser`, `generateCertificate`, `bookClassWithConflictCheck`
Integration	9	15/min	`connectGoogleCalendar`, `syncInvoiceToQBO`, `connectXero`
Platform	3	Admin only	`setPlatformSecret`, `getPlatformSecretStatus`
Webhook	1	N/A (HTTP)	`stripeWebhookHandler`
Utility	1	30/min	`logClientEvent`
Firestore Triggers	14	N/A (event)	`onClassCreated`, `onInvoiceUpdated`, `onCertificationWritten`
Scheduled	5	N/A (cron)	`cleanupRateLimits`, `onCertExpiryCheck`, `runDripEngine`

Shared Modules (functions/lib/)

Module	Purpose
`admin.js`	Firebase Admin SDK init, secret refs, region constant
`rbac.js`	`requireAuth`, `requireRole`, `requireRoleFromDoc`, `setClaimsForUser`
`rate-limit.js`	`enforceRateLimit` (Firestore-backed sliding window)
`errors.js`	`wrapHandler` (standardized error responses)
`tax.js`	`CANADA_TAX_RATES`, `resolveTaxRate` (per-province GST/HST/PST/QST)

How Client Code Calls Cloud Functions

```
// In shared/cloud-functions.js — pre-built wrappers
import { functions } from '@shared/firebase';
import { httpsCallable } from 'firebase/functions';

export const callRunPayroll = async (data) => {
  const fn = httpsCallable(functions, 'runPayroll');
  const result = await fn(data);
  return result.data;
};

// In a view component
import { callRunPayroll } from '@shared/cloud-functions';
const result = await callRunPayroll({ instructorId, month: 3, year: 2026 });
```

Security Layers on Every Cloud Function

Every callable Cloud Function enforces:

1. **Authentication** — `request.auth` must exist
2. **Rate limiting** — `enforceRateLimit(request, category, limit, windowMs)`
3. **orgId validation** — Caller's `auth.token.orgId` must match the data being accessed
4. **Role check** — Sensitive operations require specific roles (admin, super_admin)
5. **Input validation** — Parameters validated before processing
6. **Idempotency** — Financial operations use idempotency tokens to prevent duplicates

---

7. Data Flow & Security Model

Multi-Tenancy (TenantGuard)

Certivo is multi-tenant — multiple organizations share the same Firestore database, separated by `orgId`.

Three layers of enforcement:

```
Layer 1: Client-Side (TenantGuard)
  → Auto-adds orgId to every Firestore query and write
  → Prevents accidental cross-tenant data access in code

Layer 2: Firestore Security Rules (580 lines)
  → Validates request.auth.token.orgId == resource.data.orgId
  → Even if client code is tampered with, the database rejects cross-tenant access

Layer 3: Cloud Functions (server-side)
  → All 43 callable functions validate orgId from the auth token
  → Financial calculations enforce orgId scope
```

Usage in code:

```
import { TenantGuard } from '@shared/tenant-guard';

// Reading — auto-adds where("orgId", "==", currentOrgId)
const q = TenantGuard.query(collection(db, "classes"), where("status", "==", "scheduled"));

// Writing — auto-adds orgId, locationId, lastModified, integration_ids, ai_meta
await addDoc(collection(db, "classes"), TenantGuard.prepare({ courseType: "CPR" }));
```

Client Portal uses `SimpleTenantGuard` (lighter version, only adds `orgId` to collection refs). **Super Admin** intentionally skips TenantGuard (needs cross-tenant access for platform management).

Auth Flows

```
Client Portal:  Email+Password → signInWithEmailAndPassword → initClientClaimsV2 CF → custom cl
aims
Tenant Suite:   Email+Password → signInWithEmailAndPassword → initUserClaims CF → custom claims
Instructor:     Email+Password → signInWithEmailAndPassword → initUserClaims CF → custom claims
Super Admin:    Email+Password → signInWithEmailAndPassword → (optional Ghost Mode via ghostLogi
n CF)
```

Custom claims (set by Cloud Functions, stored on Firebase Auth token):

- `orgId` — Which organization this user belongs to
- `role` — User's role (admin, instructor, client, super_admin)
- `clientId` — (Client Portal only) which client record this user represents

UI role checks read the `role` field from the Firestore `users` document (for display logic). **Security rule checks** read `request.auth.token.orgId` and `request.auth.token.role` (for data access).

Firestore Collections (44)

Key collections and what they store:

Collection	Purpose	Key Fields
`users`	All user accounts	`orgId`, `role`, `email`, `displayName`
`clients`	Client organizations	`orgId`, `name`, `billingEmail`, `province`
`classes`	Training classes	`orgId`, `courseType`, `instructorId`, `status`, `dateStarted`
`students`	Student records	`orgId`, `classId`, `name`, `certExpiryDate`
`invoices`	Financial invoices	`orgId`, `clientId`, `amount`, `status`, `taxLabel`
`expenses`	Expense records	`orgId`, `instructorId`, `amount`, `receiptUrl`
`certifications`	Issued certificates	`orgId`, `studentId`, `courseType`, `expiryDate`
`notifications`	User notifications	`orgId`, `type`, `message`, `read`
`audit_log`	Audit trail	`orgId`, `action`, `userId`, `timestamp`
`tenant_stats`	Aggregated org metrics	`totalClasses`, `totalStudents`, `mrr`
`rate_limits`	Rate limit counters	`key`, `count`, `windowStart`
`drip_campaigns`	Email drip sequences	`orgId`, `name`, `steps`, `enrollments`
`ghost_sessions`	Ghost Mode audit log	`adminId`, `targetOrgId`, `enteredAt`, `exitedAt`
`sms_log`	SMS message delivery log	`orgId`, `to`, `body`, `status`, `sentAt`
`sms_consent`	CASL SMS consent records	`orgId`, `phone`, `consentType`, `consentedAt`
`api_keys`	REST API authentication keys	`orgId`, `key`, `rateLimit`, `scopes`
`webhook_subscriptions`	Webhook delivery configuration	`orgId`, `url`, `events`, `secret`
`marketplace_listings`	Certification marketplace entries	`orgId`, `courseType`, `price`, `status`
`forum_posts`	Community forum discussion threads	`orgId`, `authorId`, `title`, `category`, `pinned`
`forum_replies`	Forum post replies	`postId`, `authorId`, `body`, `createdAt`
`instructor_profiles`	Public instructor portfolio data	`userId`, `orgId`, `bio`, `specializations`, `photoUrl`
`recurring_invoices`	Recurring invoice templates	`orgId`, `clientId`, `frequency`, `nextRunDate`
`nps_responses`	Net Promoter Score survey data	`orgId`, `userId`, `score`, `feedback`
`feature_flags`	Feature gate configuration per plan	`feature`, `requiredPlan`, `enabled`

Full collection registry: `.claude/context/firestore-collections.md`

8. How the Apps Interact

Data Relationships

```
Organization (orgId)
■■■ Users (admins, staff)
■■■ Clients
■   ■■■ Students
■■■ Classes
■   ■■■ Students (enrolled)
■   ■■■ Instructor (assigned)
■   ■■■ Certifications (issued after class)
■■■ Invoices (billed to clients)
■■■ Expenses (submitted by instructors)
■■■ Inventory (supplies tracked)
■■■ Notifications (generated by triggers)
```

Cross-App Workflows

Class lifecycle (touches all 4 apps):

1. **Tenant Suite** — Admin creates a class in ScheduleView, assigns instructor
2. **Instructor Suite** — Instructor sees the class in HomeView, starts WorkflowWizard
3. **Cloud Functions** — `onClassCreated` trigger generates notification
4. **Instructor Suite** — Instructor completes class, WorkflowWizard generates certificates
5. **Cloud Functions** — `generateCertificate` creates PDF, `onCertificationWritten` tracks revenue
6. **Tenant Suite** — Admin creates invoice in BillingManager
7. **Cloud Functions** — `calculateInvoice` computes tax (per-province), sends email
8. **Client Portal** — Client sees invoice in FinancialsView, pays via Stripe checkout
9. **Cloud Functions** — `stripeWebhookHandler` processes payment, updates invoice status
10. **Tenant Suite** — Admin runs payroll in PayrollManager
11. **Cloud Functions** — `runPayroll` calculates pay with CPP/EI deductions
12. **Instructor Suite** — Instructor sees earnings in WalletView

Ghost Mode (Super Admin → Tenant Suite):

1. Super Admin clicks "Ghost" on a tenant in TenantView
2. `ghostLogin` Cloud Function creates custom token with target org's `orgId`
3. Browser signs in with custom token, navigates to Tenant Suite URL
4. TenantGuard scopes all queries to the ghosted org
5. Ghost session logged in `ghost_sessions` collection with audit trail

Real-Time Updates

Several views use `onSnapshot` for live data:

- Notification bell (all apps) — listens to `notifications` collection
- Watchtower feed (Tenant Dashboard) — alerts for overdue invoices, expiring certs
- Class status changes — instructor check-in updates visible to admin in real-time

---

9. Build System & Tooling

Vite Multi-Page Config

Vite is configured as a multi-page application in `vite.config.js`:

```
// Each app is a separate entry point
input: {
  'tenant-suite': 'apps/tenant-suite/index.html',
  'client-portal': 'apps/client-portal/index.html',
  'super-admin': 'apps/super-admin/index.html',
  'instructor-suite': 'apps/instructor-suite/index.html',
}
```

The `@shared` alias maps `@shared/` → `packages/shared/`:

```
resolve: { alias: { '@shared': path.resolve(__dirname, 'packages/shared') } }
```

Code Splitting

Each app uses `React.lazy()` to split views into separate chunks:

```
const DashboardView = lazy(() => import('./views/DashboardView'));
```

Production build generates 7 named vendor chunks to optimize caching:

- `vendor-react` — React + ReactDOM
- `vendor-firebase-core` — Firebase core
- `vendor-firebase-auth` — Firebase auth
- `vendor-firebase-firestore` — Firestore client
- `vendor-xlsx` — SheetJS (spreadsheet)
- `vendor-misc` — Other libraries
- `shared` — All packages/shared/ code

Post-Build

`scripts/post-build.js` runs after Vite build to:

1. Copy `dist/assets/` into each app's dist folder (so Firebase Hosting serves assets correctly)
2. Copy `oauth-callback.html` to Tenant Suite dist
3. Copy Service Worker files to Instructor Suite dist

npm Scripts

Script	Purpose
`npm run dev`	Start Vite dev server (localhost:3000)
`npm run build`	Production build (all 4 apps + post-build)
`npm run preview`	Preview production build locally
`npm test`	Run Vitest unit tests
`npm run test:e2e`	Run Playwright E2E tests
`npm run seed`	Seed Firestore with sample data

---

10. CI/CD & Deployment

Environments

Environment	Firebase Project	Branch	Auto-Deploy
Development	`r4e-inventory` (local)	Any	No (localhost)
Staging	`r4e-enterprise-staging`	`develop`	Yes (push to develop)
Production	`r4e-inventory`	`master`	Yes (push to master)

GitHub Actions Workflows

Workflow	Trigger	What It Does
`pr-check.yml`	PR to master/develop	Runs tests + build (validation only)
`deploy-staging.yml`	Push to `develop`	Test → Build → Deploy to staging
`deploy.yml`	Push to `master`	Test → Build → Deploy to production
`promote-production.yml`	Manual (workflow_dispatch)	Deploy to production with confirmation

All deploy workflows:

- Inject `VITE_*` environment variables from GitHub Secrets
- Run post-deploy smoke tests (HTTP 200 check on all 4 app URLs)

Production URLs

- Landing Page: `https://certivo.ca/`
- Tenant Suite: `https://app.certivo.ca/`
- Client Portal: `https://portal.certivo.ca/`
- Super Admin: `https://platform.certivo.ca/`
- Instructor Suite: `https://instructor.certivo.ca/`

---

11. Testing Strategy

Test Types

Type	Location	Framework	Count	What They Test
Unit tests	`tests/`	Vitest	23 suites	Component rendering, utility functions, data-service methods
CF tests	`functions/__tests__/`	Vitest	4 suites	Payroll calculations, auth validation, PDF templates, helpers
E2E tests	`e2e/`	Playwright	8 specs	Full user flows, WCAG AA accessibility

Running Tests

```
npm test              # Unit + CF tests
npm run test:watch    # Watch mode
npm run test:e2e      # Playwright E2E (requires dev server running)
```

---

12. Key Patterns to Know

Navigation (No Router)

All apps use `useState` for view switching — no React Router:

```
const [view, setView] = useState('dashboard');
// In render: {view === 'dashboard' && <DashboardView />}
```

Why? The apps are single-page with no need for URL-based routing. This keeps the bundle small and avoids router complexity. The trade-off: no deep linking or browser back button support.

Timestamps

All timestamps are ISO 8601 strings: `new Date().toISOString()` Never use Firestore Timestamp objects. This ensures consistency across client and server code.

Toast Notifications (Not alert())

```
import { useToast } from '@shared/hooks/useToast';
const { addToast } = useToast();
addToast('Class scheduled successfully', 'success');
addToast('Failed to save', 'error');
```

Confirmation Dialogs (Not confirm())

```
import { useConfirm } from '@shared/hooks/useConfirm';
const { confirm } = useConfirm();
const ok = await confirm('Delete this class?', { title: 'Confirm Delete', confirmText: 'Delete' }
);
if (ok) { /* proceed */ }
```

CSS Theme System

Each app defines CSS custom properties in its `styles.css`:

```
/* Tenant Suite — supports light/dark */
:root { --bg-app: #f8fafc; --text-primary: #0f172a; }
.dark { --bg-app: #020617; --text-primary: #f8fafc; }

/* Toggle dark mode */
document.documentElement.classList.toggle('dark');
```

Tailwind CSS is used for all styling (build-time via PostCSS, not CDN). App-specific Tailwind extensions live in `tailwind.config.js`.

Environment Variables

All Firebase config comes from `VITE_*` environment variables (set in `.env` locally, GitHub Secrets in CI):

```
VITE_FIREBASE_API_KEY=...
VITE_FIREBASE_AUTH_DOMAIN=...
VITE_FIREBASE_PROJECT_ID=...
```

No hardcoded fallbacks. Missing env vars produce a clear console error.

---

13. Getting Started

Quick Start

```
# Clone the repo
git clone https://github.com/ready4everythingltd/r4e-enterprise.git
cd r4e-enterprise

# Install dependencies
npm install

# Create .env from template
cp .env.example .env
# Fill in Firebase credentials (ask team lead for values)

# Start dev server
npm run dev
# Opens at http://localhost:3000

# Run tests
npm test
```

Where to Look First

1. **Understand the data model** — Read `.claude/context/firestore-collections.md`
2. **Understand the features** — Read `.claude/context/feature-spec-summary.md`
3. **Understand the rules** — Read `CLAUDE.md` (coding standards, do's and don'ts)
4. **Understand security** — Read `firestore.rules` and `functions/lib/rbac.js`
5. **Pick an app** — Start with the app you'll be working on, read its `main.jsx` first

Key Files to Read (in order)

File	Why
`CLAUDE.md`	Architecture overview, import patterns, coding rules
`packages/shared/firebase.js`	How Firebase is initialized
`packages/shared/tenant-guard.js`	How multi-tenancy works
`packages/shared/cloud-functions.js`	All server-side function wrappers
`apps/{app}/main.jsx`	App entry point, auth flow, navigation
`apps/{app}/data-service.js`	All Firestore operations for that app
`functions/index.js`	All Cloud Function definitions
`firestore.rules`	Database security rules

Dev Server URLs

When running `npm run dev`, all 4 apps are accessible:

- http://localhost:3000/apps/tenant-suite/
- http://localhost:3000/apps/client-portal/
- http://localhost:3000/apps/super-admin/
- http://localhost:3000/apps/instructor-suite/

---

14. REST API Architecture

Certivo exposes a REST API layer for external integrations and third-party tool connectivity.

Endpoints (8 total)

Endpoint	Method	Description
`/api/v1/classes`	GET, POST	List/create training classes
`/api/v1/students`	GET, POST	List/create student records
`/api/v1/certifications`	GET	List issued certifications
`/api/v1/invoices`	GET, POST	List/create invoices
`/api/v1/instructors`	GET	List instructors
`/api/v1/clients`	GET	List client organizations
`/api/v1/schedules`	GET	List scheduled classes with availability
`/api/v1/webhooks`	GET, POST, DELETE	Manage webhook subscriptions

Authentication

API requests use API key authentication via the `X-API-Key` header. Keys are stored in the `api_keys` collection and scoped to an organization (`orgId`). Each key has configurable rate limits and permission scopes (read-only, read-write, admin).

```
// Example API call
fetch('https://api.certivo.ca/api/v1/classes', {
  headers: { 'X-API-Key': 'ck_live_abc123...' }
});
```

Rate Limits

- Default: 1,000 requests/hour per API key
- Burst: 100 requests/minute
- Rate limit headers: `X-RateLimit-Limit`, `X-RateLimit-Remaining`, `X-RateLimit-Reset`

Webhook System

Webhooks deliver real-time event notifications to external URLs. Supported events: `class.created`, `class.updated`, `class.completed`, `cert.issued`, `cert.expiring`, `invoice.created`, `invoice.paid`, `student.enrolled`, `instructor.assigned`.

Webhook payloads include an HMAC signature (`X-Certivo-Signature`) for verification. Failed deliveries are retried with exponential backoff (3 attempts over 1 hour).

---

15. SMS / Twilio Integration

SMS notifications are powered by Twilio and available as a premium add-on ($25/month).

Architecture

```
User action → Cloud Function (sendSMS) → Twilio API → SMS delivered
                    ↓
               sms_log collection (delivery tracking)
               sms_consent collection (CASL compliance)
```

Cloud Function

The `sendSMS` callable Cloud Function handles all outbound SMS:

- Validates CASL consent before sending marketing messages
- Transactional messages (class reminders, cert expiry alerts) are CASL-exempt
- Logs every message to `sms_log` collection with delivery status
- Rate limited: 30 messages/minute per organization

Twilio Module (`functions/lib/twilio.js`)

Shared module for Twilio API interaction:

- `sendSMS(to, body, orgId)` — Send a single SMS
- `sendBatchSMS(recipients, body, orgId)` — Send to multiple recipients
- Twilio credentials stored in Google Secret Manager (not code)

CASL Consent

All SMS recipients must have a consent record in `sms_consent`:

```
{
  orgId: "HEADQUARTERS",
  phone: "+14035551234",
  consentType: "express",      // "express" | "implied" | "transactional"
  consentedAt: "2026-03-17T...",
  source: "onboarding_form",
  optedOut: false
}
```

16. Feature Gating System

The feature gating system controls access to premium features based on subscription plan.

How It Works

```
useFeatureGate('sms_notifications')
  → Checks feature_flags collection for required plan
  → Compares against current org subscription (useSubscription hook)
  → Returns { allowed: boolean, requiredPlan: string }
  → If !allowed, renders UpgradeModal with plan comparison
```

useFeatureGate Hook

```
import { useFeatureGate } from '@shared/hooks/useFeatureGate';

function MyComponent() {
  const { allowed, gate } = useFeatureGate('recurring_invoices');

  if (!allowed) return gate; // Renders UpgradeModal automatically

  return <RecurringInvoiceManager />;
}
```

canAccess() Enforcement

Server-side enforcement in Cloud Functions:

```
// In Cloud Function
const canAccess = await checkFeatureAccess(orgId, 'api_access');
if (!canAccess) throw new functions.https.HttpsError('permission-denied', 'Upgrade required');
```

UpgradeModal Component

`UpgradeModal` displays:

- Current plan vs required plan comparison
- Feature list for each tier
- Direct upgrade button (Stripe Checkout)
- "Contact sales" fallback for Enterprise features

---

17. Certification Marketplace

The marketplace enables training companies to publish certification offerings for public discovery.

Data Model

`marketplace_listings` **collection:**

```
{
  orgId: "HEADQUARTERS",
  courseType: "CPR-C",
  title: "CPR-C Certification — Full Day",
  description: "Red Cross CPR-C certification...",
  price: 149.00,
  duration: "8 hours",
  instructorProfileId: "prof_abc123",
  locations: ["Calgary", "Edmonton", "Red Deer"],
  status: "published",          // "draft" | "published" | "archived"
  rating: 4.8,
  reviewCount: 12,
  publishedAt: "2026-03-17T..."
}
```

Publish / Browse Flow

1. **Publish:** Training company creates a listing in Tenant Suite MarketplaceView
2. **Review:** Listing enters `draft` status, admin publishes when ready
3. **Browse:** Client Portal and public-facing marketplace display published listings
4. **Book:** Client selects listing, chooses date, completes booking via Stripe
5. **Commission:** Platform takes 5-8% commission on marketplace-facilitated bookings

---

18. Community Forum

The community forum provides a moderated discussion space for safety training professionals.

Collections

- `forum_posts` — Top-level discussion threads
- `forum_replies` — Threaded replies to posts

Post Structure

```
{
  orgId: "HEADQUARTERS",
  authorId: "user_abc123",
  authorName: "John Smith",
  title: "Best practices for confined space rescue training",
  body: "We've been updating our confined space curriculum...",
  category: "teaching_techniques",  // regulatory, equipment, techniques, events, general
  tags: ["confined-space", "rescue"],
  pinned: false,
  locked: false,
  viewCount: 47,
  replyCount: 8,
  lastReplyAt: "2026-03-17T...",
  createdAt: "2026-03-15T..."
}
```

Moderation Model

- Automated content filtering for prohibited content
- Community flagging (3 flags triggers admin review)
- Admin moderation queue in Super Admin Communications Suite
- Pin/lock/archive controls for moderators
- User reputation tracking based on post quality and community feedback

---

19. Instructor Profiles

The instructor profile system enables instructors to build public portfolios showcasing their qualifications.

Components

- `PortfolioEditor` (Instructor Suite) — Edit profile: bio, photo, specializations, certifications held
- `InstructorProfileCard` (Shared component) — Display card used in marketplace listings and instructor directories

Profile Fields (`instructor_profiles` collection)

```
{
  userId: "user_abc123",
  orgId: "HEADQUARTERS",
  displayName: "Sarah Chen",
  bio: "15 years of industrial safety training experience...",
  photoUrl: "https://storage.googleapis.com/.../profile.jpg",
  specializations: ["H2S Alive", "Confined Space", "Fall Protection"],
  certificationsHeld: ["CSO", "NCSO", "Red Cross Instructor"],
  yearsExperience: 15,
  rating: 4.9,
  reviewCount: 34,
  availability: "full-time",
  locationPreferences: ["Calgary", "Edmonton"],
  linkedIn: "https://linkedin.com/in/...",
  visibility: "public"          // "public" | "org-only" | "private"
}
```

Integration Points

- Marketplace listings link to instructor profiles via `instructorProfileId`
- Tenant Suite InstructorManager displays profile cards for roster management
- Client Portal shows instructor profiles when viewing upcoming class details

---

20. Recurring Invoice Automation

The recurring invoice system automates repetitive billing for clients with ongoing training contracts.

RecurringInvoiceManager (Tenant Suite View)

Creates and manages recurring invoice templates:

- Set frequency (weekly, bi-weekly, monthly, quarterly, annually)
- Define line items, tax treatment, and payment terms
- Preview next invoice before activation
- Pause/resume individual recurring schedules

Scheduler (`processRecurringInvoices`)

A scheduled Cloud Function that runs daily:

1. Queries `recurring_invoices` where `nextRunDate <= today` and `status == 'active'`
2. For each matching template, creates a new invoice in the `invoices` collection
3. Applies current tax rates via `resolveTaxRate()`
4. Sends invoice notification email via Resend
5. Updates `nextRunDate` based on frequency

6. Logs execution in `audit_log`

Data Model (`recurring_invoices` collection)

```
{
  orgId: "HEADQUARTERS",
  clientId: "client_abc123",
  frequency: "monthly",          // weekly, biweekly, monthly, quarterly, annually
  nextRunDate: "2026-04-01",
  lineItems: [
    { description: "Monthly Safety Training Retainer", amount: 2500.00, quantity: 1 }
  ],
  taxProvince: "AB",
  paymentTerms: 30,
  status: "active",              // active, paused, cancelled
  lastGeneratedAt: "2026-03-01T...",
  createdAt: "2026-02-15T..."
}
```

Appendix: File Sizes

Lines of Code by Area

Area	Lines	% of Total
Tenant Suite	21,200	32%
Cloud Functions	13,000	20%
Super Admin	8,400	13%
Shared Library	7,200	11%
Instructor Suite	7,800	12%
Client Portal	5,100	8%
Config + Tests	~3,500	5%
Total	**~66,200**	**100%**

Largest Views (by line count)

View	App	Lines	Purpose
StudentRegistry	Tenant	1,012	Student CRUD + cert batch + roster
PayrollManager	Tenant	981	Instructor payroll + Canadian taxes
FormBuilder	Tenant	879	Drag-and-drop form builder
IntegrationsManager	Tenant	875	OAuth connections + webhook config
SettingsManager	Tenant	838	Org config + MFA + tax settings
ClientManager	Tenant	833	Client CRM + billing
CommunicationsSuite	Tenant	797	Email + drip campaigns
PlanEditor	Super Admin	728	Subscription plan management