RITZY ONE • TECHNICAL DOCUMENTATION

# Technical Specifications & System Requirements

What a technical evaluator, IT manager or hosting partner needs before deploying Ritzy One — the technology stack, the architecture, the security model, server and client requirements, ports, deployment options and the backup strategy.

Applies to the current Ritzy One web platform | Written for customer IT departments and integrators

ON THIS PAGE


1. 1. Platform overview
2. 2. Technology stack
3. 3. Architecture
4. 4. Data layer & database design
5. 5. API design
6. 6. Security model
7. 7. Licensing & entitlement
8. 8. Server requirements
9. 9. Client requirements
10. 10. Ports & network
11. 11. Deployment & updates
12. 12. Backup & recovery
13. 13. Integrations & external services
14. 14. AI capabilities
15. 15. Sizing, performance & limits
16. 16. Configuration reference


## 1 Platform overview

Ritzy One is a browser-based ERP suite, a re-platform of the long-running Ritzy desktop application. It is a **single Node.js API process** talking to a **PostgreSQL** database, with a React single-page application for the desk and a family of installable phone apps (PWAs) for the field.

| Attribute | Value |
| --- | --- |
| Application type | Single-page application over a REST API — no application server licence, no Java or .NET runtime |
| Deployment model | On-premise, private VM/VPS, or Docker — one instance per customer |
| Tenancy | Single-tenant per install; *one* PostgreSQL database per install holds every company, each company's rows isolated by Row-Level Security |
| Language | TypeScript end to end, strict mode (including `noUncheckedIndexedAccess`) |
| Codebase | npm-workspaces monorepo — 16 packages, roughly 140,000 lines of first-party TypeScript |
| Licensed modules | The Accounts core plus ten independently licensed modules: Cargo, Liner, Marine, Warehousing, Transport, Shipchandling, Inventory, Payroll, CRM, Real Estate, plus the Timesheet add-on to Payroll |

**No native dependencies**

The whole server side is pure JavaScript — including the PostgreSQL client (`pg`, without `pg-native`). There is no `libpq` to match, no compiler toolchain needed at install time, and no compiled add-on to rebuild when Node or the platform changes. That is what lets the same build run unchanged on Windows, on Linux and inside a container.

## 2 Technology stack

| Layer | Technology | Version | Notes |
|---|---|---|---|
| Runtime | Node.js | 18.x or 20.x LTS (20 recommended) | The Windows installer fetches Node 20 LTS if it is absent |
| API framework | Express | 4.21 | REST over JSON; no GraphQL |
| Language / build | TypeScript | 5.7 | `tsc` for the API; `tsc -b` plus Vite for the front ends |
| Database | PostgreSQL | 18 (17 minimum) | Installed by the Windows installer if missing, or bundled inside the Docker stack |
| Database driver | `pg` (node-postgres) | 8.23 | Pure-JavaScript PostgreSQL client with a connection pool; no native bindings |
| Front end | React + Ant Design | React 18.3, antd 5.22 | React Router 7, Axios, Day.js, Recharts |
| Bundler | Vite | 6.0 | Static output served by nginx (or by the API itself in the container build) |
| Authentication | JSON Web Tokens + TOTP | `jsonwebtoken` 9 | Sign-in passwords hashed with scrypt (Node `crypto`); two-factor backup codes with `bcryptjs` |
| Documents | `docxtemplater` + Handlebars | 3.69 / 4.7 | Real Word `.docx` templates; PDF through LibreOffice where installed |
| Mail / push | `nodemailer`, `web-push` | 9.0 / 3.6 | SMTP held per company in the database; VAPID keys for push |
| AI | Anthropic Claude SDK | 0.106 | Entirely optional — see §14 |
| Process manager | PM2 (Windows service via NSSM) / systemd / Docker | — | Survives reboot on every platform |

| | | | |
|---|---|---|---|
| Web server | nginx | current stable | Serves the static builds and proxies `/api/` to the one API process |

### Deliberate exclusions

- **No native npm modules.** Pure-JavaScript equivalents are used throughout, so an install never needs a build toolchain.
- **No ORM.** SQL is written directly against PostgreSQL, which keeps the ported business logic comparable line by line with the desktop application it replaces — the posting engines were verified to produce identical results.
- **No business logic in the database.** The desktop application's stored procedures and triggers (posting, ledger balances, stock costing) were ported to TypeScript services with regression tests; the database holds data, constraints and a few read-only SQL functions.
- **No message broker, no Redis, no separate job runner.** Scheduled work — alert digests, licence heartbeat, support-reply sweeps — runs as timers inside the API process.

## 3 Architecture

**Browser / phone** → **nginx — TLS, static, /api proxy** → **Node API (Express)** → **PostgreSQL 18** → **one database — every company**

There is exactly **one API process**. Every front end — the desk application and all fifteen phone apps — is a separate static bundle that nginx publishes on the one address — the desk application at `/`, each phone app under its own path (`/track/`, `/crm/` …) — proxying `/api/` back to that single process. One host name and one certificate cover them all. Adding a phone app adds a static site, not a service.

### Packages

| Package | Role |
|---|---|
| `packages/api` | The Express REST API — 68 route modules, the posting engines, reporting, document generation, licensing and the scheduled jobs |
| `packages/web` | The full ERP desk application (React + Ant Design) |
| `packages/*-pwa` × 15 | Track, Dues, CRM, Delivery, Notify, Ask, Support, Tenant, Jobs, Inspect, Letting, Collect, Holiday Homes, Employee, Stores |
| `packages/support` | Shared client library behind the support assistant in every front end |
| `packages/license-server` | The vendor-side licence issuing and tracking service — it runs at Ritzy, not at the customer |

### State and scaling

The API keeps no session state in memory — authentication is a signed token and session records live in the database — so it can be restarted without logging anyone out. Scaling is nonetheless **vertical by design**: the bottleneck in this workload is database I/O rather than Node CPU, and one PostgreSQL server per customer is ample. The multi-customer answer is one instance per customer, not a shared cluster.

## 4 Data layer & database design

### One database per install

Every company in an install lives in **one PostgreSQL database**. The `companies` table is the company registry and drives the login page's company dropdown. Every business table starts with `company_id`, and **Row-Level Security** limits each connection to the company it was opened for; consolidated group reports switch it to read every company. Attachments and scanned documents are held in the same database rather than in a separate file.

| Item | Detail |
| --- | --- |
| Database | PostgreSQL 18, UTF-8, created with the built-in `C.UTF-8` locale so codes and names sort the same way on every operating system |
| Schema | About 400 tables (394 in the baseline, the rest added by later migrations), 357 baseline foreign keys, 35 views |
| Keys | Each table has a surrogate key; business numbers (voucher, account code, job number) are unique columns; document numbering uses sequences plus a locked series table |
| Types | Money is exact `NUMERIC` (never floating point), flags are `boolean`, dates are `date` or `timestamptz`, text is `text`, binary is `bytea` |
| Logins | `PG_URL` (superuser) prepares and upgrades the database on start; `PG_APP_URL` (`ritzy_app`) serves every request and is subject to Row-Level Security |
| `<RITZY_ROOT>` | Install-wide state files under `Ritzy\System`, and the default backup target `Backups\PostgreSQL` |

## Schema and upgrades

On start the application creates its own database, schema, application login and upgrades from `PG_URL`, so there is no separate database-setup step. The schema is a baseline; later changes are **numbered SQL migrations** applied in order, each in its own transaction and recorded with a checksum in `schema_migrations`. The same runner upgrades an existing install when a new build is deployed, so the database and the application version stay in lockstep. A new company is seeded with its chart of accounts and defaults in one transaction, and adding a module is a data change, not a schema change.

## Business logic in TypeScript

The desktop application kept much of its logic in Firebird stored procedures and triggers. In Ritzy One that logic — voucher posting, ledger and party balances, bill allocation, inventory costing and stock posting — runs in TypeScript services, inside the same database transaction as the document it posts, and is covered by regression tests against the desktop application's results. Only a few read-only queries remain as SQL functions.

## Engineering notes

- **Test as the application login**, never as the superuser: a superuser bypasses Row-Level Security and would see every company.

- PostgreSQL is **I/O- and latency-sensitive**. Keep the data directory on local SSD/NVMe, never on a network share.
- Keep the database **local to the API server** (port 5432 is never published); if it is on a separate host, use TLS and a private network.

### Why PostgreSQL

PostgreSQL is open-source with no per-seat or per-core licence cost, runs identically on Windows, Linux and in containers, and is supported by every cloud provider as a managed service. Exact numeric types, real foreign keys and Row-Level Security give the data stronger guarantees than the desktop schema had, and standard tools — `pg_dump`, `pg_restore`, streaming replication — cover backup and standby.

### Moving from the Ritzy desktop application

Desktop customers' data is in Firebird `.FDB` files. A one-off import converts each company into the PostgreSQL database: from the administration console (*Import from Firebird*, as a new company or replacing an existing one) or with `npm run pg:convert`. Problem rows are logged rather than silently dropped. This import is the only part of Ritzy One that reads Firebird; the running server does not need Firebird installed.

## 5 API design

- **REST over JSON**, everything under `/api`, across 68 registered route modules (`/api/auth`, `/api/accounts-tran`, `/api/reports`, `/api/realestate`, `/api/payroll` and so on).
- **Bearer-token authentication** on every route except the first-run setup router and the version endpoint.
- **A separate API for the Ritzy Employee app** under `/api/employee`. Employees sign in there with their own credentials and receive a token of a distinct type: it is accepted only by that router, and an application user's token is refused by it. Every route reads the employee from the token, never from the request.
- **Four middleware layers** ahead of the handlers: authentication, module entitlement, per-form rights, and a central error handler that turns typed application errors into proper HTTP status codes.
- **Reporting endpoints return** `{ rows, totals }` — a fixed contract the grid components rely on.

- `GET /api/version` is unauthenticated and reports the running build marker. It is the canonical way to confirm which build is live after a deployment.

### Module gating

A gated module must be *both* installed in the company database *and* entitled by the subscription licence before its routes answer, and the check is on the server rather than only in the menu. It deliberately **fails open** if the entitlement lookup itself errors, so an infrastructure problem can never lock a paying customer out of their own data.

## 6 Security model

| Control | Implementation |
|---|---|
| Authentication | Username and password against the company database, issuing a signed JWT. Access token lasts **8 hours** by default, the refresh token **7 days**; both configurable per install. |
| Token signing | HMAC with a per-install secret generated randomly by the installer — never a shipped default. |
| Passwords | Hashed with scrypt, salted per password, and never stored readable — user and employee app passwords alike; two-factor backup codes are hashed with bcrypt. Accounts imported from the Ritzy 8 desktop application arrive in its reversible cipher and keep working: each is replaced by a hash the first time its owner signs in, so no password reset is forced at cut-over. |
| Employee app sign-in | Employees are not application users. HR enables the Ritzy Employee app per employee on the employee record; the employee signs in with their ID No and a password HR sets (generated, 8 characters, no look-alike characters). Passwords are hashed with scrypt and shown only while being set. Each sign-in is a session row checked on every request, so disabling the app or setting a new password ends the employee's sessions on every device at once. Five wrong passwords lock that ID No for 15 minutes. Employees see only their own record: payslips, leave, approvals for their own team, clock-in and contract signing. |
| Two-factor | TOTP (RFC 6238, any authenticator app), enrolled per user, and mandatory on the internal support account. The authenticator secret must be read back to check codes, so it is encrypted with AES-256-GCM under a server key held only in the environment (`SECRETS_KEY`, else derived from `JWT_SECRET`) — a copy of the database alone does not reveal it. Backup codes are hashed. |
| Authorisation | The desktop application's own rights model is preserved: per-group, per-form view / add / edit / delete rights, enforced server-side on every call and used to build the menu. |
| Database access | PostgreSQL listens locally and logs in with SCRAM-SHA-256. Requests run as a dedicated application login (`ritzy_app`) that does not own the tables, so Row-Level Security confines every query to the signed-in company; the superuser login is used only to prepare and upgrade the database on start. Where the database is on another host, the connection uses TLS (`sslmode` in the connection URL). |
| Browser transport | TLS terminates at nginx; a ready configuration ships in the bundle. Let's Encrypt/certbot on Linux, IIS ARR or a certificate bound in nginx on Windows. |

| | |
|---|---|
| CORS | Restricted to the configured origins. |
| Session control | Sessions are recorded per device in the database and can be ended centrally. A device that has been disabled, or whose app has been withdrawn, is rejected at the next token refresh rather than only at the next login. |
| Privileged tooling | The support console that can run SQL and replace database files sits behind three independent gates: a dedicated internal account, mandatory two-factor on it, and a rotating support password re-checked on the server for every change it makes. |

**Data residency**
Nothing leaves the customer's server by default. The optional AI features send only the text of the specific question or document being processed; the optional licence heartbeat sends an install identifier and seat counts. Both are described in §13–14, and both can be left switched off. The one exception is **company registration** (§7): when a company is created, or its Company settings are saved, the install tells the Ritzy License Server the company's code and name and its setup details — contact email and telephone, city, country, base currency, modules and the server's host name. No business data is sent.

## 7 Licensing & entitlement

Licensing is a **signed-token** scheme. A licence is issued by the vendor-side Ritzy License Server, signed with a private key, and verified locally against the matching public key — so entitlement checks keep working with no internet connection.

- **Seats are per device**, not per named user: each browser registers a device token and counts against the licensed concurrent maximum. The phone apps are seat-exempt, and employees who use only the Ritzy Employee app are not application users, so they need no seat or user licence.
- **Modules are entitled individually** — the ten add-on modules each carry their own entitlement; the Accounts core is always included.
- **A heartbeat** (optional, for subscription customers) periodically confirms the licence is current and picks up renewals, plan changes and revocations.
- **Companies are registered with the License Server**, since billing is per company database. Creating a company (first-run wizard, New Company, DBA Console) sends its code, name and setup details — contact email and telephone, city, country, base currency, modules and the server's host name — and saving Settings → Company re-sends the

stored details so the vendor's copy stays current. The License Server keeps these details per company (a blank never clears a stored value) and re-creates the customer if it is missing. Registration is best-effort: it runs after the save, and an unreachable server never blocks creating or saving a company.

- With licensing disabled, entitlement checks are inert — a perpetual on-premise install is never locked by the vendor. Company registration still runs; block outbound HTTPS to the License Server if even that must not leave the site.

# 8 Server requirements

## 8.1 Supported platforms

| Platform | Status | How it is installed |
|---|---|---|
| Windows Server 2016+, or Windows 10/11 Pro | Primary, in production | `install.ps1` — PM2 plus NSSM services, nginx, firewall rules |
| Linux (Ubuntu LTS, Debian, AlmaLinux) | Supported | `install-linux.sh` — systemd and nginx |
| Docker / Docker Compose (Linux or WSL2) | Supported, self-contained | `docker compose up -d`; PostgreSQL 18 is **inside** the bundle, so there is nothing to install |

## 8.2 Hardware

| Profile | vCPU | RAM | Storage | Suits |
|---|---|---|---|---|
| Minimum / evaluation | 2 | 4 GB | 40 GB SSD | Pilot, one company, a handful of users |
| Recommended production | 4 | 8 GB | 100 GB NVMe | One company, 10–30 concurrent users, normal reporting |
| Report-heavy or multi-company | 4–8 | 16 GB | 250 GB NVMe | Several companies, long histories, heavy trial-balance and ageing runs |

Component footprint on a lean install: idle Linux around 0.5–1 GB, the Node API 0.3–0.5 GB, and PostgreSQL with a useful cache 1–2 GB. A 3 GB tenant VM is comfortable and 4 GB generous — which is why a 32 GB host runs roughly six to eight fully isolated tenant VMs with headroom.

**Storage is the specification that matters**

This is a database workload. Insist on NVMe or provisioned-IOPS storage, and benchmark before trusting production data to it — a random read/write test *plus a real* `pg_restore` *and a report run*. Oversubscribed budget-VPS disks are the commonest cause of disappointing performance.

## 8.3 Software prerequisites

| Requirement | Version | Notes |
|---|---|---|
| PostgreSQL | 18 (17 minimum) | Installed by the Windows installer if missing (EDB installer, silent; service `postgresql-x64-18`); an older local PostgreSQL stops the installer until it is moved to 18. Logins use SCRAM-SHA-256; data checksums on. One database holds every company. The app creates the database, schema, upgrades and its own login on start, from `PG_URL`. Inside the Docker bundle already. |
| Firebird | — | *Not needed by the server.* Only the one-off import of a Ritzy desktop customer's data reads their `.FDB` files (the administration console's *Import from Firebird*, or `npm run pg:convert`), on a machine that has Firebird. |
| Node.js | 20.x LTS or later | Installed automatically by the Windows installer if missing. |
| nginx | current stable | Downloaded by the Windows installer if absent. |
| LibreOffice | any current release | *Optional.* Only needed to render Word quotation and purchase-order templates to PDF. |

## 9 Client requirements

| Client | Requirement |
|---|---|
| Desk browser | Any current evergreen browser — Chrome, Edge, Firefox or Safari. No plug-in, no Java, no ActiveX, nothing installed on the workstation. |
| Screen | 1366×768 or better for the desk application; the phone apps are built for phone widths. |
| Phones | Android with Chrome, or iPhone with Safari. The apps install to the home screen straight from the browser — no app store, no MDM package. |
| Phone features used | Camera (QR scanning, photos, visiting-card capture), microphone (spoken questions), location and signature capture — each behind the browser's own permission prompt. Several field apps hold work offline and sync when a signal returns. |
| Network | LAN, VPN or public HTTPS. Bandwidth needs are modest: JSON payloads against a cached static bundle, with no thick-client chatter and no database traffic to the workstation. |
| Push notifications | Standard Web Push (VAPID). Requires HTTPS, and on iPhone that the app has been added to the home screen. |

## 10 Ports & network

The installer opens these inbound TCP rules. For an internet-facing deployment, publish 443 only and put nginx in front of everything.

| Port | Service |
|---|---|
| 80 / 443 | Desk web application and every phone app under its path (nginx) |
| 3001 | The API process — proxied by nginx, does not need to be public |
| 5432 | PostgreSQL — local to the server; never expose it |

**Outbound** traffic is entirely optional and each item can be omitted: the licence heartbeat, company registration and the support assistant to the Ritzy License Server over HTTPS, SMTP

for sending documents, the tax authority's endpoint where e-invoicing is in use, and the browser push services.

## 11 Deployment & updates

### 11.1 Traditional install

Deployment is a **compiled bundle**, not a source checkout. The build produces a deployment zip; extracting it over the install folder and restarting the process manager is the whole update. The installer installs PostgreSQL 18 when it is missing, sets up dependencies, writes the API's `.env` with the two database logins and a freshly generated secret, schedules a nightly `pg_dump`, registers the PM2 process and the Windows services, configures nginx and opens the firewall.

- **First install:** `install.ps1` on Windows or `install-linux.sh` on Linux — it prompts once for the base folder (and, when PostgreSQL was already installed, its superuser password).
- **Application-only update:** `install-app.ps1` — no prompts, no `.env` changes; the app applies any database upgrades itself on start. Or simply extract the zip and restart the service.
- **An existing `.env` is never overwritten** by a re-install.
- **Verification:** `GET /api/version` returns the live build marker.

### 11.2 Docker

For a new standalone install the container bundle is the simplest path: two services, `postgres` (PostgreSQL 18) and `app`, each with its own volume. Updates are `pull` then `up -d`. A Ritzy desktop customer, whose data is in Firebird, is moved onto a stack with `docker/migrate/`: the companies are converted on the Windows side by the importer and restored into the stack as a `pg_dump`.

### 11.3 First run

A brand-new server has no database and no company registry, so there is nothing to log in to yet. The install therefore starts in **first-run mode**: a setup wizard — reachable only while zero companies exist, and additionally protected by the support password — creates the first company in the database (which the app has already created, with its schema and upgrades) and seeds the chart of accounts and defaults. Once a company exists the wizard is gone for good.

## 12 Backup & recovery

- **Use `pg_dump`.** It reads one consistent snapshot of the running database. The Windows installer schedules it nightly (`pg_dump -Fc`, 14 days kept under `<RITZY_ROOT>\Backups\PostgreSQL`); the Docker bundle's `backup-stacks.sh` does the same per stack. One company is backed up and restored from the administration console (`.pgbk`: download to a PC, upload, restore).
- **Do not rely on VM snapshots or hypervisor checkpoints** of a running database server — they are not database-consistent and are no substitute for a backup.
- **Copy backups offsite**, ideally to a different provider or region.
- **Recovery is a `pg_restore` plus a service restart.** The application holds no state of its own outside the database and its `.env` — keep a copy of that file with the backups.

## 13 Integrations & external services

| Integration | Detail |
|---|---|
| ZATCA e-invoicing (Saudi Arabia) | Phase-2 (Fatoora) clearance and reporting through the official SDK on the server. Invoices, cash memos, credit and debit notes are submitted on finalisation, with a locked invoice-hash and counter chain and a full submissions log. Enabled for Saudi installs only. |
| Bahrain customs manifest | The WCO-format sea cargo manifest, generated from Liner jobs and submitted to OFOQ/TFP with per-submission tracking. Requires the Liner module and the customer's own API keys. |
| Email | SMTP host, port, credentials and sender address are held per company in the database, so changing them is an application setting rather than a server file edit. |
| Web push | A standard VAPID key pair in the API's `.env`. Without it the notification inbox still works and only the push toggle is unavailable. |
| Documents | Word `.docx` templates with merge fields for quotations and purchase orders; HTML and Handlebars templates, editable inside the application, for invoices and other billing documents. QR codes are generated server-side. |
| Data import | CSV import for masters, opening balances and bank statements, with an AI-assisted column mapper. |
| Payments | PayPal, on the vendor's licence server only — not something a customer install talks to. |

## 14 AI capabilities

The AI features use Anthropic's Claude models and are strictly **opt-in**. They fall into two groups with different privacy profiles.

- **Data features** — paste-to-form extraction on around twenty-five screens, CSV column mapping, visiting-card capture, lead capture and follow-up drafting, and the management briefing — require an API key in the install's own `.env`. The key is the customer's, requests go straight from their server to the model API, and they carry only the text or image being processed. With no key configured these features report that AI is not set up and everything else carries on unchanged.

- **The product support assistant** needs no key at the customer's site. The question is forwarded over the install's licence token to the Ritzy License Server, where the model key, the knowledge base and the manual catalogue live. It reads nothing from the company database.

The management briefing is deliberately constrained: the model chooses which figures to show, but every number is computed by the server from the database. The model never produces a figure itself.

## 15 Sizing, performance & limits

| Dimension | Position |
|---|---|
| Concurrent users | Governed by the licence seat count rather than a technical ceiling. Ten to thirty concurrent users on the recommended profile is routine. |
| Companies per install | Several — all in the one database, separated by Row-Level Security, and chosen at login. Bounded by disk and RAM, not by the application. |
| Tenants per host | One VM per customer is the recommended isolation boundary; roughly six to eight 3 GB tenant VMs fit a 32 GB host. |
| Report performance | The heaviest work is trial balance, general ledger and ageing across long histories. These are sort- and I/O-bound, so NVMe storage and enough RAM for PostgreSQL's cache (`shared_buffers` plus the OS file cache) matter far more than CPU count. |
| Known constraints | The application is designed for one PostgreSQL server per customer and does not use read replicas or database scale-out. Plan capacity vertically and isolate per customer. (Standard PostgreSQL also opens the door to streaming replicas for standby and to managed PostgreSQL services, where a customer prefers them.) |

## 16 Configuration reference

Everything environment-specific lives in one file, `packages\api\.env`, which is why the same build runs unchanged on Windows, on Linux and in a container. The keys that matter most:

```
# Paths and database
RITZY_ROOT=C:\Ritzy Solutions      # the BASE folder -- not ...\Ritzy
PG_URL=postgres://postgres:********@localhost:5432/ritzy      # superuser: makes the
database ready on start
PG_APP_URL=postgres://ritzy_app:********@localhost:5432/ritzy # the login every
request uses
RITZY_BACKUP_DIR=

# API
PORT=3001
JWT_SECRET=<generated at install>
JWT_EXPIRES_IN=8h
JWT_REFRESH_EXPIRES_IN=7d
# SECRETS_KEY=<long random>   # optional: encrypts two-factor secrets; else derived
from JWT_SECRET
CORS_ORIGIN=https://erp.example.com

# Optional features
ANTHROPIC_API_KEY=                # AI data features
VAPID_PUBLIC_KEY=                 # web push
VAPID_PRIVATE_KEY=
VAPID_SUBJECT=
SOFFICE_PATH=                     # LibreOffice, for PDF rendering
FATOORA_HOME=                     # ZATCA -- Saudi installs only

# Licensing (inert unless enabled)
LICENSING_ENABLED=false
LICENSE_PUBLIC_KEY=
LICENSE_HEARTBEAT_URL=
```

**Two settings people get wrong**

`RITZY_ROOT` is the **base** folder that *contains* `Ritzy\` and `Backups\` — the application appends the rest itself, so it is `C:\Ritzy Solutions`, not `C:\Ritzy Solutions\Ritzy`. And `PG_APP_URL` must be the `ritzy_app` login, not the superuser: a superuser bypasses Row-Level Security, and the application refuses a second company on such a login.