Employee OS Technical Manual

Version 1.2.1 · Database schema v7 · Windows x64

For future developers, IT administrators and Kraftt Digital support engineers

Made by Kraftt Digital

Contents

1. Product scope and release identity

Employee OS is an offline-first Electron desktop application for one organization and one primary local administrator. Version 1.2.1 manages employees, multiple bank accounts, controlled documents, attendance, holidays, configurable payroll, draft/finalized payroll, salary slips, reports, audit history and portable backups.

The supported packaged target is Windows 10/11 x64. The Electron Builder product name is `Employee OS`; the NSIS output is `Employee-OS-Setup-1.2.1-x64.exe`. The application ID is `in.shreeharispintex.employeeos`. That identifier is intentionally retained for upgrade compatibility; it is not used to hardcode company branding or production data.

The release is local-only: no cloud database, telemetry, analytics, advertising, online authentication, API integration or remote font/resource request is part of normal operation. The software does not submit statutory filings and does not claim automatic legal compliance.

2. Implemented technology stack

Versions below come from `package.json` /the installed lockfile for this release.

Layer	Package/runtime	Version	Responsibility
Desktop runtime	Electron	44.2.0	Windows process, BrowserWindow, dialogs, shell operations
Renderer	React / React DOM	19.2.8	Desktop user interface
Routing	React Router DOM	7.18.3	Hash-based local routes
Build	Vite	8.2.2	Renderer production bundle
Language	TypeScript	5.9.3	Main, preload, shared and renderer typing
Local database	Node `node:sqlite`	Electron's Node 24 runtime	Parameterized SQLite access and backup API
ORM mapping	Drizzle ORM sqlite-proxy	0.45.2	Typed schema/mapping layer over the local adapter
Validation	Zod	4.1.11	Shared input schemas
Forms	React Hook Form	7.62.0	Renderer forms and validation lifecycle
PDF	pdf-lib	1.17.1	Salary-slip/report generation
PDF fonts	@pdf-lib/fontkit	1.1.1	Embedded TrueType font support
Excel	ExcelJS	4.4.0	XLSX reports and bulk-import template
Archives	archiver / yauzl	7.0.1 / 3.4.0	Backup and release ZIP creation/validation
Image processing	sharp	0.35.4	Transparency analysis and black-signature PNG processing
Tests	Vitest / Testing Library	5.0.0 / 16.3.3	Domain, database and renderer regression tests
Packaging	electron-builder	26.15.3	x64 NSIS installer

Packaged fonts are Manrope Regular/Bold and IBM Plex Mono Regular/SemiBold. They live under `build/fonts/`, are embedded into PDFs, and include their OFL licence texts. Renderer fonts are provided by local `@fontsource` packages. No Windows font installation or internet access is needed for the rupee symbol or supported Indian names.

3. Runtime architecture and trust boundaries

```
 React renderer (no Node.js access)
        │ typed window.employeeOS API
        ▼
context-isolated preload
        │ named IPC request + session token
        ▼
Electron main process
  ├─ input/schema validation and authorization
  ├─ DatabaseService domain transactions
  ├─ controlled file dialogs and managed paths
  ├─ PDF / XLSX / archive generators
  └─ SQLite database and local folders
```

Electron main process

`src/main/index.ts` creates the 1440 × 900 window (minimum 1024 × 580) hidden, waits for `ready-to-show`, maximizes it and then shows it. This avoids visible startup resizing while preserving normal minimize/restore controls. It removes the menu, denies arbitrary new windows and denies browser permission requests. Production navigation is restricted to local `file:` content; development permits only the configured Vite localhost URL. The main process opens the data directory, initializes/migrates SQLite, registers IPC handlers and schedules a due automatic backup five seconds after startup.

`contextIsolation` is enabled and `nodeIntegration` is disabled. The Electron web-preferences sandbox is currently `false`; do not describe this V1 as using Chromium's renderer sandbox. The security boundary instead relies on isolated preload exposure, no Node integration, constrained navigation and validated IPC/file operations.

Preload layer

`src/preload/index.cts` exposes only `window.employeeOS`. It stores the in-memory session token in preload scope and attaches it to IPC requests. No general filesystem, process, shell or database API is exposed.

The public groups are `auth`, `dashboard`, `company`, `employees`, `designations`, `shifts`, `attendance`, `payroll`, `payslips`, `reports`, `backup`, `system` and `audit`. `system.openExternal` accepts only the exact Kraftt Digital HTTPS URL in the main process; the global footer cannot be used as an arbitrary URL launcher.

React renderer

`src/renderer/` contains the routes and forms. Routes use `HashRouter`, so no server is needed. The global Kraftt credit is mounted once by `App.tsx` and uses fixed positioning without changing page/table dimensions. The expanded/collapsed sidebar is implemented once in `Layout.tsx`; its preference is stored in localStorage. Application lock keeps the current route/component tree mounted behind an opaque, `inert`, accessibility-hidden boundary, so pending form state survives without exposing sensitive pixels or focus targets.

IPC validation

`src/main/ipc.ts` divides handlers into raw authentication/approved public operations and session-protected operations. Secure handlers call `SessionManager.require()` before domain work. Shared Zod schemas validate company, employee, login and payroll-facing structures; format handlers additionally constrain enumerations, numeric ranges, file extensions and dialog results.

Database values use prepared statements with bound parameters. Multi-record employee, attendance, payroll, migration and restore operations use SQLite or filesystem transaction/rollback patterns as appropriate. `LocalDatabase.transaction()` uses an outer `BEGIN IMMEDIATE` transaction and supports nested SQLite savepoints. Bulk employee import deliberately reuses the single-employee save operation inside one outer transaction without starting another top-level transaction.

4. Repository folder structure

```
employee-os/
├─ build/
│  ├─ icon.ico                 Windows application/installer icon
│  └─ fonts/                   PDF TTF files and OFL licences
├─ docs/
│  ├─ assets/                  Guide screenshots and rendered ER diagram
│  ├─ Employee-OS-User-Guide.md/.pdf
│  └─ Employee-OS-Technical-Manual.md/.pdf
├─ scripts/
│  ├─ dev.mjs                  Vite + main watcher + Electron development loop
│  ├─ desktop-check.mjs        CDP-based real Electron workflow/layout test
│  ├─ generate-pdf-fixtures.mjs
│  ├─ verify-pdf-fixtures.py
│  ├─ render-pdf-qa.mjs
│  ├─ build-docs.mjs
│  └─ create/verify Windows release scripts
├─ src/
│  ├─ main/
│  │  ├─ database/             SQLite adapter, schema, migrations, service and dashboard selector
│  │  ├─ exports/              Shared PDF layout, PDF and Excel export
│  │  ├─ backup.ts             Backup create/validate/apply/rollback
│  │  ├─ company-assets.ts     Transparency analysis and signature processing
│  │  ├─ file-exports.ts       Safe names and export orchestration
│  │  ├─ ipc.ts                Validated desktop operations
│  │  ├─ paths.ts              User-data paths and traversal guard
│  │  └─ security.ts           scrypt passwords and session manager
│  ├─ preload/                 Typed context bridge
│  ├─ renderer/                React pages, components, styling and assets
│  └─ shared/                  API/types, validation, payroll, formats/masking
├─ dist/                       Rebuilt production output (no source maps)
├─ release/                    Electron Builder and distribution artifacts
├─ package.json / package-lock.json
├─ tsconfig*.json
└─ vite.config.ts
```

Generated test profiles, PDF QA renders and fixtures live under `artifacts/` and are not included by Electron Builder. `node_modules`, source tests, raw development files and live data are not copied into the public distribution ZIP.

5. Application startup and lifecycle

1. Electron calls `createAppPaths(app.getPath('userData'))`.
2. `openDatabase()` creates/opens SQLite, applies busy timeout and migration safeguards.
3. `DatabaseService.initialize()` ensures singleton defaults.
4. IPC is registered against a getter so restore can replace and reload the service.
5. The BrowserWindow loads the Vite development URL or `dist/renderer/index.html`.
6. The renderer calls `auth.state()`, then shows first-run admin, company setup, login or the application.
7. If configured/due, the main process creates an automatic full backup.
8. On app close, the database is checkpointed/closed by the process lifecycle.

Production starts without fake employees/payroll. Development/QA creates synthetic data only in disposable `--user-data-dir` profiles.

6. Database implementation and migrations

`LocalDatabase` wraps Electron's Node 24 `DatabaseSync` API from `node:sqlite`. Drizzle's sqlite-proxy provides typed schema mapping, while `DatabaseService` owns domain SQL and transactions. Important pragmas are foreign keys ON, WAL journal mode, synchronous NORMAL and a 5,000 ms busy timeout.

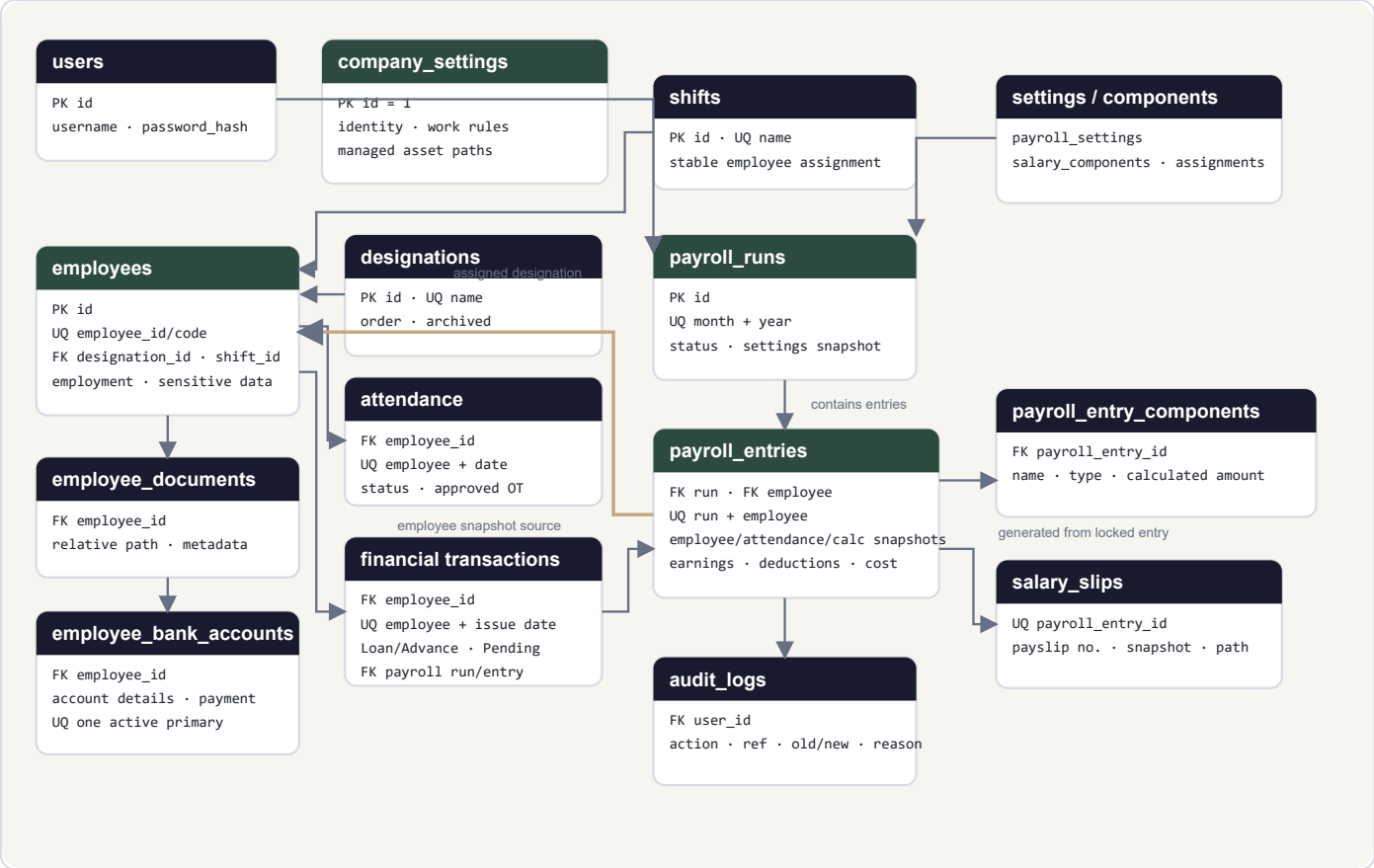
`src/main/database/migrations.ts` contains numbered immutable migrations. `schema_migrations(version, name, applied_at)` records completion. Current `LATEST_SCHEMA_VERSION` is 7:

1. Initial normalized schema.
2. Designation workflow and multiple bank accounts.
3. Active/non-archived payroll eligibility cleanup.
4. Employer contributions, whole-rupee payroll and richer audit values.
5. Employee Loan/Advance recovery records and payroll links.
6. Stable custom shifts, migration of legacy shift names and employee shift foreign keys.
7. Removal of Unpaid Leave, conversion of every legacy `UL` attendance row to `A`, and rebuilding the attendance status constraint.

Before applying a newer migration to an existing database, `openDatabase()` checkpoints WAL and writes `employee-os-pre-migration-v[old]-[timestamp].sqlite3` beside the active database. Each migration runs in a database transaction. The command `npm run migrate` applies migrations without opening the UI.

Never modify an applied migration. Add a new version, add upgrade/rollback-aware tests, and preserve earlier data semantics.

7. Entity relationships and constraints



Rendered Employee OS entity relationship diagram

Actual tables

- `users` : local administrator identity, salted scrypt hash, active flag and login timestamps.
- `company_settings` : singleton company identity, statutory references, managed asset paths and work rules.
- `departments` : retained normalized structure for data compatibility/future use; the current employee UI is designation-led.
- `designations` : unique name, archived state and display order.
- `shifts` : stable ID and case-insensitive unique name; assigned employee records block deletion.
- `employees` : unique generated employee ID/code, employment, personal, statutory and sensitive fields, archive state.
- `employee_references` : multiple references/guardians per employee.
- `employee_bank_accounts` : multiple bank accounts; a partial unique index permits only one non-archived primary account per employee.
- `employee_bank_details` : original single-account table retained for migration compatibility; active application reads/writes `employee_bank_accounts` .
- `employee_documents` : category, original/stored names, controlled relative path, MIME, size and archive state.
- `holidays` : unique date and name; current workflow treats holidays as paid.
- `attendance` : one employee/date record, status, overtime hours/approval, in/out time and remarks.
- `employee_financial_transactions` : one Loan/Advance per employee/issue date, amount/type/remarks, target recovery month/year, Pending/Recovered state and finalized payroll run/entry links.

- `payroll_settings`: singleton divisor, EPF/ESI modes/bases/rates/ceilings, overtime and rounding settings.
- `salary_components`: reusable earning/deduction definitions.
- `employee_salary_components`: unique employee/component assignment.
- `payroll_runs`: unique salary month/year, status, settings snapshot, totals, finalizer and revision reason.
- `payroll_entries`: unique run/employee, three snapshots, salary/attendance/calculation values, payment/company-cost values and the combined Loan/Advance recovery amount.
- `payroll_entry_components`: resolved named component values stored with the entry.
- `salary_slips`: unique entry/payslip number, stored snapshot/path and generation timestamps.
- `backups`: backup history metadata; the archive file itself lives on disk.
- `audit_logs`: user/action/module/reference, old/new values, reason, summary and timestamp.
- `app_preferences`: JSON preferences such as automatic-backup directory/frequency.
- `schema_migrations`: applied schema versions.

Critical database constraints include unique attendance `(employee_id, attendance_date)`, one financial transaction per `(employee_id, issue_date)`, payroll run `(salary_month, salary_year)`, payroll entry `(payroll_run_id, employee_id)`, salary slip entry/number and unique employee IDs/codes. Foreign keys protect relationships; employee subrecords use cascades where deletion is explicitly permitted.

8. Attendance and payroll domain rules

Attendance status values are `P`, `A`, `PL`, `HD`, `WO` and `H`. Overtime hours and approval are separate columns. This permits Present plus overtime on one day. The employee must be active, non-archived and within joining/leaving dates to appear in active attendance/payroll loading. Schema migration 7 rebuilds the attendance constraint and converts legacy `UL` records to `A` while preserving dates, remarks, times and overtime fields. The migration also runs after a compatible older backup is restored.

Paid units for this organization workflow are Present 1, Paid Leave 1, Weekly Off 1, Holiday 1, Half Day 0.5 and Absent 0. Unpaid leave is represented by `A` and is not a separate reporting status.

Salary divisors are calendar-day count, fixed 26, fixed 30 or actual scheduled working days. The selected method is written into the run settings snapshot.

Core calculations are:

```
Earned Basic = Monthly Basic / Divisor × Paid Attendance Days
Gross = Earned Basic + Bonus + Other Earnings + Overtime + positive adjustment + earning components
Employee Deductions = TDS + Advance/Loan + deduction components + absolute negative adjustment
Net Salary = Gross - Employee Deductions
Employer Contributions = EPF + ESI + Other Employer-Paid Costs
Company Cost = Gross + Employer Contributions
```

EPF and ESI are employer-paid costs in this implementation. They are never included in `total_deductions` and never reduce Net Salary. Both support percentage/fixed mode, earned-basic/eligible-gross/manual wage bases, optional wage ceiling, optional maximum and employee exemption. TDS is manual per employee/month.

Overtime uses approved hours only. Fixed-hourly mode computes hours × configured rate; multiplier mode derives a rate from hourly basic. A manual monthly amount replaces calculated overtime for that entry—both are not added. Time can be rounded to the configured interval.

Loan/Advance entry is initiated from Attendance Day Details but stored independently in `employee_financial_transactions`; attendance status, time and overtime columns are unchanged. Draft calculation reads every Pending transaction assigned to the employee and selected period, sums it once as `Loan / Advance Recovery`, and stores the source transaction IDs/details in `calculation_snapshot`. Recalculation replaces the draft entry, so it is idempotent. Finalization verifies the Pending sources and atomically marks them Recovered with payroll run/entry links. Unlock/revise returns those linked transactions to Pending before recalculation. A failed or negative-net finalization rolls back and leaves the transactions Pending.

Final monetary values are rounded with nearest-rupee behavior. Manual adjustment requires a reason whenever non-zero. Negative-net prevention clamps/warns according to current settings.

9. Payroll snapshots and salary slips

The monthly workflow is Calculate Draft, Review Draft, Download Draft, then Finalize & Lock. Calculation writes:

- `payroll_runs.settings_snapshot` for divisor/statutory/overtime/rounding values.
- `payroll_entries.employee_snapshot` including identity, designation, primary bank and company identity/asset paths used then.
- `attendance_snapshot` with monthly totals and source records.
- `calculation_snapshot` with the transparent calculation result/input, combined automatic/manual recovery values and source Loan/Advance transactions.
- resolved `payroll_entry_components` rows.

Finalization records `finalized_at` and `finalized_by`. Normal attendance edits that affect locked payroll are refused. Unlock/revise verifies the current administrator password, requires a reason and creates an audit entry. The reason remains normal text; only the password receives the shared show/hide control. Historical salary-slip generation reads the stored snapshots; it does not recalculate against current employee salary/settings.

Generated salary slips are stored under `payslips/YYYY-MM/` and registered in `salary_slips`. Safe non-overwriting names include `Salary-Slip_[Employee-ID]_[Month-Year].pdf`. Bulk generation creates separate PDFs in a selected folder or a ZIP.

10. PDF and Excel export system

`src/main/exports/pdf-layout.ts` is the reusable PDF design system. It owns A4 geometry, 10–14 mm margins, Employee OS colour tokens, embedded fonts, company header, document header, page footer, page numbering, wrapping, measured row heights, tables, repeated headers, total-row grouping and multi-page column partitioning.

`src/main/exports/pdf.ts` implements salary slip, monthly attendance and generic reports with those primitives. Portrait is used for salary slips/narrow reports; landscape for wide reports. A standard salary slip is constrained and tested as one A4 page; unusually long component lists use a safe continuation page rather than clipping or unreadably shrinking content. Monthly attendance uses one A3 landscape horizontal table containing every date and the monthly totals; larger employee sets continue vertically with repeated headers. Draft payroll and the Yearly Salary Register also use one A3 landscape column group and paginate only downward when the employee count requires it. Long rows wrap or apply controlled secondary ellipsis; essential text is not reduced below the designed readable size.

PDF font files are resolved from packaged app resources or the repository during development. Company logo/signature/stamp paths are optional and guarded; unreadable optional artwork does not block document creation. Upload processing checks alpha transparency. Logo/stamp revisions retain colour; signature revisions are written as PNG with visible RGB pixels black and the original alpha channel preserved. PDFs consume these managed revision paths consistently. Salary figures and authorization areas reserve separate measured space.

`src/main/exports/excel.ts` creates XLSX reports with human-readable headings/totals. `src/main/employee-spreadsheet.ts` creates and parses the bulk employee template; mandatory headers are red and validation is atomic.

QA is reproducible:

```
npm run pdf:fixtures
npm run pdf:verify
npm run pdf:render
```

The fixture set covers individual/bulk-style salary data, every report type, 100 employees, 31-day attendance, long company/employee/designation/component text, large values and missing optional fields. `verify-pdf-fixtures.py` checks valid PDFs, A4/A3 bounds, page counts, text, page headers/footers, repeated titles, embedded font resources and rupee extraction. It also verifies one-page attached-size attendance, draft-payroll, Yearly Salary Register, EPF, ESI and TDS samples. `render-pdf-qa.mjs` renders every page with Poppler and builds contact sheets for human review.

11. Managed files and data storage

At runtime the main process passes Electron's `app.getPath('userData')` to `createAppPaths()`, which appends `employee-os-data`.

Expected standard pattern:

```
%APPDATA%\Employee OS\employee-os-data\
├─ database\employee-os.sqlite3
├─ database\employee-os-pre-migration-v*.sqlite3
├─ assets\                  company logo/signature/stamp revisions
├─ documents\<employee-id>\managed files
├─ payslips\YYYY-MM\generated salary slips
├─ backups\*.eosbackup     default backups
└─ temp\                    restore/export staging
```

The exact `%APPDATA%` location depends on the signed-in Windows account and Electron configuration. The reliable method is **Settings → Security → Active data location**. Select **Open Data Folder** to ask the main process to open only the controlled root. This is read-only navigation; the renderer never receives general filesystem access.

There is no separate persistent application-log directory in V1. Expected operational events are stored in `audit_logs`; automatic-backup startup failures go to Electron process diagnostics/console rather than a managed log file.

Profile photos use the managed documents/assets mechanism and database path metadata. Company asset revisions are preserved rather than overwritten so finalized snapshots can keep resolving the artwork current at calculation time.

Dashboard attendance analytics are derived by `buildDashboardAttendance()` from `DatabaseService.getAttendanceMonth()`, not a parallel demo query. The selector applies employee lifecycle limits, implicit holidays/weekly offs, saved statuses and approved-overtime flags. Missing counts stop at today and exclude out-of-employment dates, holidays and weekly offs. Finalized dashboard financials sum `payroll_entries` snapshot columns for Finalized/Paid runs. Successful mutations increment a root data revision, forcing the selected dashboard period to reload.

Attendance uses one `saveChanges()` handler for the top-right button and Ctrl+S. The window key handler prevents the browser default even when a cell or input is focused. Navigation and window-close guards remain active while the dirty-key set is non-empty; a failed save leaves that set intact. Cell-local keyboard handling maps case-insensitive P/A/L/H to P/A/PL/HD through the same `updateCell()` path used by the picker. Ctrl+Tab moves focus to the next filtered/sorted employee at the same date without mutating the draft. Because status handling is attached to the cell button rather than the window, typing in inputs, searches, selects, textareas, dialogs or editable content cannot mark attendance.

12. Authentication, sessions and privacy

Passwords use Node `crypto.scryptSync` with a random 16-byte salt, 64-byte derived key and parameters N=16384, r=8, p=1. The encoded hash stores algorithm/parameters/salt/hash. Verification uses timing-safe comparison. Passwords and complete sensitive identifiers are excluded from audit summaries.

The session token is a random 32-byte value held in main/preload memory, not localStorage. Renderer activity events synchronize real activity. The current V1 timeout is fixed at 60 minutes with a visible warning during the final five minutes. Lock clears the in-memory session and displays the existing authentication screen without unmounting the protected application tree; a successful login restores the same route and pending UI state. Ctrl+Shift+L invokes this path globally. The handler explicitly requires Ctrl and Shift and rejects Meta/Windows-key combinations, so Windows+L remains under operating-system control.

Sensitive Aadhaar, PAN and bank values are masked in normal summaries. Intentional reveal/edit, accidental-duplicate deletion and payroll unlock require the administrator password. Login, first-run setup, change-password and payroll-unlock controls hide values by default and use one reusable accessible show/hide control without mutating the field value or losing input focus. Format validation does not query or imply verification by government or banking services.

SQLite, managed files and backup archives are **not encrypted at rest by Employee OS V1**. Password hashing is not database encryption. Protect the Windows account, device, application-data directory and backups with OS access controls and BitLocker/device encryption where policy permits.

13. Accessing local data safely

Preferred access is through Employee OS screens, PDF/XLSX reports, document Export and the backup function. Before any technical inspection:

1. Close or stop edits in Employee OS.
2. Create a full backup and verify its file size/history.
3. Close Employee OS before opening the live database.
4. Copy the database/backup to an isolated working location.
5. Use SQLite in read-only mode (`mode=ro` or a read-only GUI setting).
6. Run `PRAGMA integrity_check;` before relying on results.

Do not edit the live SQLite file while Employee OS is running. WAL/shm state, foreign keys, snapshots and managed file references can be damaged by direct updates. Direct SQLite access is appropriate only for approved diagnosis, read-only reporting or emergency recovery by an IT/support engineer. Never expose full identifiers in screenshots, support tickets or query logs.

14. Backup archive format and validation

`BackupManager.create()` requests SQLite's backup API to write a consistent snapshot, creates `metadata.json`, and archives:

```
metadata.json
database/employee-os.sqlite3
assets/**
documents/**
payslips/**
```

Metadata fields are format `employee-os-backup`, format version 1, application name, app version, database/schema version and UTC creation time. The extension is `.eosbackup`; content is ZIP-compatible. Names include backup type and an ISO-like timestamp and never silently overwrite.

Validation extracts to a unique temp directory using exclusive file creation. It rejects absolute paths, drive paths, backslashes, `..` traversal and symbolic links. It requires metadata and database, rejects unsupported format/newer schema, opens the staged DB read-only, runs `PRAGMA integrity_check`, and requires core tables (`users`, `company_settings`, `employees`, `attendance`, `payroll_runs`, `schema_migrations`).

15. Restore, rollback and recovery

The UI restore sequence is:

1. User selects an archive.
2. Archive is validated without modifying active data.
3. A full **Pre-restore** safety backup of the destination is created.
4. The active database is closed.
5. Validated database/assets/documents/payslips are staged and swapped.
6. The service is reloaded and migrations run if needed.
7. On success, rollback staging is deleted and the session is locked.
8. On failure, database/folders are restored from rollback copies and the original service is reopened.

For suspected corruption, first preserve the entire active data folder without editing it. Try the most recent known-good `.eosbackup`, then older backups. Validate employee count, company assets, document previews, attendance, payroll totals and a regenerated historical slip after restore.

If the original PC does not boot but its disk is readable, an IT administrator should recover the whole `employee-os-data` directory and all external `.eosbackup` files. Prefer restoring a valid `.eosbackup`. Emergency manual migration requires Employee OS closed on both machines, compatible/newer application version, a safety copy, and the complete folder—not only SQLite. Managed paths and files must remain aligned.

16. Moving Employee OS to another PC

Fresh installation

Install the same release ZIP and complete new administrator/company setup. No existing records are transferred.

Move application only

Copy the release ZIP, extract it and run the installer. This moves software/manuals only. The installer never contains a database, company assets or employee documents.

Move application and existing data (preferred)

On the old PC, create a full backup in Backup & Restore and copy the `.eosbackup` to protected external media. On the new PC, install the same or a compatible newer version, open Backup & Restore, validate/restore the archive, sign in with restored credentials and verify all modules.

Manual emergency migration

Only an IT/support engineer should perform this when application backup is impossible. Close Employee OS, copy the complete `employee-os-data` root, make an independent checksum inventory, install the same version, locate the new active root via Settings → Security, preserve its current data, place the recovered structure only while the app is closed, then launch and allow migrations. Immediately create a new full backup.

Do not let old/new copies continue creating independent records. V1 has no synchronization, replica IDs or conflict merge. Choose one authoritative computer after verification.

Verification checklist

- Correct company legal/display name and assets.
- Administrator sign-in/change-password behavior.
- Employee count, archive states, bank masks and document preview.
- Representative attendance month/status/OT.
- Draft and finalized payroll totals.
- Historical snapshot salary slip opens with correct values.
- PDF and Excel report export.
- New backup creation and validation on the destination.

17. Development, testing and packaging

Prerequisites for development are Windows x64, Node.js 24-compatible runtime and npm 11 or later.

```
npm install
npm run dev
npm run typecheck
npm test
npm run build
npm run test:desktop
npm run migrate
npm run build:win
```

`npm run build` first removes `dist`, then type-checks, builds the Vite renderer and compiles main/preload TypeScript. Source maps are disabled for production. `npm run build:win` invokes Electron Builder's x64 NSIS target.

`npm test` covers salary divisors, paid units, half-day/leave, EPF/ESI rates and ceilings, TDS, components, overtime, rounding, company cost/net salary, amount-to-words, attendance uniqueness, archive eligibility, duplicate deletion, snapshots, finalized locking, audit filters, backup validation/transfer and a complete employee → attendance → payroll → salary-slip/report workflow.

`npm run test:desktop` launches a real Electron renderer under a disposable `--user-data-dir` through Chrome DevTools Protocol. It verifies first-run setup, persistence, 100-row/31-day layout, dashboard charts/filters, employee refresh, overtime, employer-only EPF/ESI, sidebar reflow, payroll table scrolling and session-warning interaction. The disposable profile is not production data.

18. Release verification and checksums

The primary build command is:

```
npm run release:windows
```

The steps build the NSIS installer, create both manual PDFs, stage the client distribution, create `Employee-OS-v1.2.1-Windows-x64.zip`, and verify archive members/checksums. The public ZIP contains only:

- `Employee-OS-Setup-v1.2.1.exe`
- `Employee-OS-User-Guide.pdf`
- `Employee-OS-Technical-Manual.pdf`
- `INSTALLATION-AND-TRANSFER.txt`
- `SHA256SUMS.txt`
- required font licence notices under `licenses/`

An external release checksum file also covers the ZIP and installer. Verify on Windows with:

```
Get-FileHash .\Employee-OS-Setup-v1.2.1.exe -Algorithm SHA256
Get-FileHash .\Employee-OS-v1.2.1-Windows-x64.zip -Algorithm SHA256
```

The NSIS configuration creates Desktop/Start Menu shortcuts, a normal uninstall entry, permits directory selection and uses `deleteAppDataOnUninstall: false`. User data is intentionally preserved by configuration during uninstall, but administrators must still maintain backups. The installer is unsigned.

19. Maintenance procedures

Change statutory/payroll defaults

Use Settings → Payroll rules through the application. For code-default changes, add a migration and domain tests; never rewrite historical rows. Confirm whether old drafts need explicit recalculation. Finalized snapshots must stay stable.

Add a database field

Update `schema.ts`, append a numbered migration, update shared API/types/validation, domain service, preload/IPC, UI/export logic and upgrade/backup tests. Test an upgrade from the previous schema with files present.

Add an IPC operation

Expose the smallest typed operation in `shared/api.ts`, implement a narrow preload call, validate in main IPC, and keep filesystem/shell access in main. Decide explicitly whether it can run before authentication. Never add a generic path or shell API.

Update PDF layouts

Prefer primitives in `pdf-layout.ts`. Add stress fixture data, run the structural validator, render all pages, inspect contact sheets and sample full-size pages. Verify snapshot behavior and ₹ extraction. Do not use Windows-only fonts without embedding.

Upgrade dependencies

Use compatible exact versions, review Electron/Node ABI implications, regenerate the lockfile, run unit/build/desktop/PDF QA and rebuild the installer. Preserve `appId`, `productName`, data-folder convention and schema compatibility unless a documented migration plan exists.

20. Known V1 constraints

- Windows 10/11 x64 is the packaged target; macOS/Linux are not release targets. One organization, one primary local administrator and one active workstation are supported.
- There is no cloud sync, mobile app, biometric/GPS attendance, employee self-service, network multi-user locking, online recovery, messaging delivery, online payment or government filing.
- TDS is manual; EPF/ESI are configurable employer-cost calculations for this workflow.
- Application data and backups are not encrypted at rest. The Chromium sandbox is disabled; context isolation, disabled Node integration and validated preload/IPC are the active renderer controls.
- There is no persistent diagnostic log; use audit history and controlled reproduction for support.
- The NSIS installer is unsigned, so SmartScreen may warn. Revalidate clean-VM install/uninstall after packaging-toolchain or Windows changes.