APPOINTFRAME

AppointFrame Developer Guide

Practical maintenance, extension, debugging, test and deployment handoff handbook

Revised 13 September 2026

Current structured build: Generic + Live. Written to make ownership and safe extension discoverable without prior project history.

Handoff objective: a developer unfamiliar with the codebase should be able to trace a feature, identify its owning seam, change it without duplicating business rules, test it, and produce a safe branch-specific deployment patch.

Contents

1. 30-minute handoff path
2. Repository map and feature tracing
3. Branch discipline
4. Local/runtime prerequisites and configuration
5. Frontend composition and CSS workflow
6. Database facade, migrations and manifests
7. Appointment and Group creation flows
8. Recurring schedules V2: where to change what
9. Availability, timezone and capacity rules
10. Session Packages and accounting seams
11. Notifications, WhatsApp, Email and Session Notes
12. Calendar integration seams (Generic)
13. Secure Participant/Staff access
14. Reports, export, history and performance
15. Migration/historical maintenance
16. Extension recipes
17. Testing strategy and commands
18. Debugging map
19. Deployment/change-set workflow
20. Handoff checklist for the next developer
21. Appendix - high-value files and tests

1. 30-minute handoff path

22. Read ARCHITECTURE.md at repository root for contribution guardrails.
23. Confirm which branch you are on. Generic and Live are separate deployment baselines.
24. Read app.js, ledger-router.js and lib/database.js only to understand composition - do not start editing them for feature work.
25. Find the UI fragment in views/admin/, then its browser feature module in public/assets/app/features/.
26. Trace the HTTP call to lib/server/*-routes.js, then to the owning lib/database/* module.
27. Find the matching tests before changing code. Tests often reveal the intended invariant faster than old comments.
28. Make the smallest change in the owning seam, regenerate assembled assets if needed, run targeted tests, then package only changed files.

Fast rule: If you cannot name the owning route, database/domain module and regression test for a change, trace the feature one layer further before editing.

2. Repository map and feature tracing

Feature	UI / browser	Server	Data/domain	Tests to start with
Appointments	views/admin/tabs/appointments.html; features/appointments.js	lib/server/appointment-routes.js	lib/database/appointments-groups/*	appointment-workspace-phase1, appointment-composer-selection-ux, resource-overlap
Recurring	appointments tab/dialogs; features/appointments.js	appointment-routes.js	recurrence-rules.js; recurring-schedules.js; recurring-group-schedules.js	recurring-schedules-v2, recurring-group-schedules
Group Sessions	dialogs/group-session.html; appointments.js	appointment-routes.js	appointments-groups/group-sessions.js	recurring-group-schedules + group/appointment tests
Packages	tabs/lesson-packages.html; features/lesson-packages/*	lesson-package-routes.js	lib/database/lesson-packages/*	lesson-packages, package-consumption-accounting, package-report-reconciliation
Settings/notifications	settings fragments; settings feature modules	notification-routes.js	notifications.js + utility registry/settings	client/instructor notification preference tests
Calendar intake (Generic)	settings/integrations + Calendar Sync modal	calendar-integration-routes.js	lib/calendar-integrations/* + database/calendar-integrations.js	calendar-oauth-foundation, calendar-sync-review, google import/bulk/completion
Migration	tabs/migration.html; features/migration.js	migration-routes.js	migration staging/review/commit modules	migration-* tests
Reports	tabs/reports.html; features/reports.js	reporting-routes.js	database/reporting/*	location-aware-reporting, instructor-reporting, export-csv
Session Notes	secure note page + assets	session-notes-routes.js	database/session-notes.js + lib/session-notes.js	session-notes, settings-persistence, timezones, ui

3. Branch discipline

- Generic is the reference/product branch. It includes product-site/ and external-calendar integration in the current build.
- Live is a customer deployment. It intentionally differs in client reschedule/reminder policy and does not currently include the Generic calendar-integration stack.
- Shared appointment/recurrence changes must be applied independently to each branch baseline.
- Do not “solve” migration numbering divergence by renaming migrations. Generic head is 035; Live head is 033.
- Before a patch, keep an untouched baseline and compare any broad-suite failures against it.

4. Local/runtime prerequisites and configuration

- Node.js >= 24.14.1. Express 5.2.x. SQLite uses Node built-in node:sqlite DatabaseSync.
- Start: npm start (node app.js).
- lib/config.js is the production environment seam. Add new environment flags there and test defaulting/normalization.
- Operational Email/WhatsApp capability gates are deployment-level; business preferences remain DB-backed.
- Provider credentials/tokens/private keys never belong in browser JSON, ordinary settings tables or logs.

```
Useful baseline commands
npm run lint
npm run check
npm run smoke
node --check <changed-file.js>
node test/<targeted-test>.js
```

5. Frontend composition and CSS workflow

- Admin markup belongs in focused views/admin fragments.
- Feature-specific browser behavior belongs in public/assets/app/features; shared runtime helpers only in core/runtime.js.
- Terminology-aware UI should use the feature registry rather than hardcoded Participant/Staff/Service words where configurable.
- Source CSS lives in public/assets/app/styles. public/assets/app.css is generated/assembled via lib/admin-styles.js. Do not fix only the generated file.
- After CSS/JS changes, bump the served query version in the owning HTML shell/page and verify phone + desktop behavior.
- Generic product-site is independent of admin app styling. Its homepage/help assets live under product-site/assets/.

```
Regenerate admin CSS
node -e "const fs=require('fs'); const {loadAdminAppCss}=require('./lib/admin-styles'); fs.writeFileSync('public/assets/app.css',
loadAdminAppCss(process.cwd()))"
```

6. Database facade, migrations and manifests

- Use the next numbered migration for persisted schema changes. Published migrations are immutable.
- Keep SQL/data-domain methods inside responsibility modules. Route modules should not grow persistence logic.
- When adding/removing public LedgerDatabase methods, update database method manifests/architecture tests intentionally.
- Migration registry rejects unknown/newer DB versions and name mismatches.
- Use transactions for multi-row invariants; scheduling and package allocation use BEGIN IMMEDIATE where concurrency matters.

7. Appointment and Group creation flows

Explicit Individual multi-create and explicit Group multi-create are immediate persistence workflows. They are not planner-only recurrence features.

```
Individual explicit batch
composer -> POST /api/appointments/bulk
 -> validateAppointmentBatch()
 -> saveAppointmentsBestEffort()
 -> saveAppointments() transaction
 -> notifications after persistence

Group explicit batch
composer -> one or many POST /api/group-sessions operations
 -> saveGroupSession()
 -> optional participant appointments, revalidated per session
 -> Group/participant side effects
```

- Best-effort means valid selections survive even when another selection is blocked.
- Never validate a batch against only the initial Package/Staff state; accepted rows must affect the projection for later rows.
- On server write, recheck conflicts/package capacity under the write lock.

- The visible weekly planner Save button remains for planner drafts/inline changes; Add appointment multi-create persists directly.

8. Recurring schedules V2: where to change what

Concern	Owning file(s)
Rule normalization / date matching / descriptions	lib/database/recurrence-rules.js
Individual plan CRUD / pattern sync / occurrence ledger / planner draft preparation	lib/database/recurring-schedules.js
Group plan CRUD / occurrence ledger / shell preparation	lib/database/recurring-group-schedules.js
HTTP management endpoints	lib/server/appointment-routes.js
Create/Edit/Preview/Manage UI	public/assets/app/features/appointments.js + views/admin/tabs/appointments.html
Schema	migrations 032-035 Generic / 030-033 Live
Regression	test/recurring-schedules-v2.js; test/recurring-group-schedules.js

Before changing recurrence, answer four questions: (1) does the plan definition change? (2) do pattern rows change? (3) how are existing occurrence-ledger rows treated? (4) should already saved appointments/sessions change? The usual answer to #4 is no.

- Individual creation resolves Package eligibility against the first matching occurrence and stores the linked Package.
- Group schedules intentionally have no Participant/Package roster.
- Edit future re-anchors from the current effective date.
- Skip marks an occurrence; it does not delete history that has already been prepared/saved.
- Recurring plan creation sends no appointment notification.

9. Availability, timezone and capacity rules

Do not create an alternate availability engine in a new UI. Reuse the same domain rules that power administrator scheduling, secure client access, recurrence preparation and calendar import.

- Availability combines Organization Operating Hours, Date Overrides, Service/Offering availability, Staff availability/overrides, duration, Staff-Service mapping, Location scope, Staff capacity/Service limit, Package entitlement and conflicts.
- Physical appointment scheduling clock = selected Location timezone. Online clock = current Organization/online rule.
- Canonical UTC is the comparison/persistence identity; recipient-local time is presentation/communication.
- A one-appointment Staff-Service override is explicit and auditable; it must not silently change permanent mapping or become recurrence entitlement.

10. Session Packages and accounting seams

Task	Start in
Package eligibility/assignment	lesson-packages/assignment.js + offering-entitlements.js
Capacity/rules	lesson-packages/rules.js + model/helpers
Progress/consumption	lesson-packages/progress.js + policy.js
Payments/credits/adjustments	financials.js + finance.js + accounting-adjustments.js
Lifecycle/closure/renewal	lifecycle.js
Weekly progress	weekly-progress.js
Package settings/policy	settings.js
UI	public/assets/app/features/lesson-packages/*

Do not conflate: “scheduled/reserved capacity” with “consumed progress.” A future appointment can reserve the last package slot while the completed count still reads zero.

11. Notifications, WhatsApp, Email and Session Notes

- Keep notification event selection separate from provider delivery. Server persistence succeeds first; then event handlers decide eligible channels.
- Effective delivery = deployment capability × organization route × recipient preference/eligibility × Utility enabled/configured for WhatsApp.
- Use lib/whatsapp-utility-registry.js for Utility definitions and allowed parameter sources; do not invent private caller-only mappings.
- Generic reminder V3: Confirm QR, Cancel QR, Reschedule URL. Live preserves its own branch policy.
- Session Notes reminder is consolidated once daily when completed appointments still need notes; Staff-specific override time uses Staff timezone.
- Payment Statement Email/WhatsApp must use the same reviewed PDF snapshot bytes.

12. Calendar integration seams (Generic)

Layer	Files / responsibility
Provider transport/OAuth	lib/calendar-integrations/providers/*; crypto.js; service.js
HTTP	lib/server/calendar-integration-routes.js
DB identity/mapping memory	lib/database/calendar-integrations.js
Settings UI	views/admin/tabs/settings/integrations.html + features/calendar-integrations.js
Appointment review/import UI	features/calendar-sync.js
Tests	calendar-oauth-foundation; calendar-sync-review; calendar-google-import/bulk/completion; modal-scroll

Preserve the boundary: external calendars are read-only intake sources. AppointFrame does not silently mirror source edits into operational appointments.

13. Secure Participant/Staff access

- Client browser and WhatsApp Flow paths should converge on lib/client-self-service.js / shared scheduling rules rather than duplicate eligibility.
- Verified contacts can map to controlled candidate Participants; unknown senders must not learn private identity/availability.
- Role-scoped Staff schedule access is read-only and uses Staff-local time.
- Administrator privileges are never inferred from a client/staff contact address or phone.
- Signed notes/action URLs must retain expiry and authorization checks.

14. Reports, export, history and performance

- Keep filters compact on pointer devices but touch-safe on mobile.
- Report clocks follow report subject semantics; Location is context/filter, not a replacement timezone.
- Saved Appointments / Communication History / Audit Trail / Closed Packages use summary-first + lazy detail.
- Dashboard should use server summary endpoints; do not regress into loading the whole appointment universe in the browser.
- Export additions must include exact Location/Offering/timezone/canonical fields when needed for round-trip meaning.

15. Migration/historical maintenance

- Migration stages: Upload → Resolve → Packages → Review/Preflight → Commit.
- Analyse/Review cannot write operational records.
- Commit = fingerprint check + backup + transaction + idempotency + no normal notifications.
- Historical irregular cycles are authoritative; do not normalize them to today's package size.
- Historical date/time is immutable in correction workflows; Service/Staff/status corrections must keep package/reporting consistency.

16. Extension recipes

16.1 Add a persisted field to an existing entity

29. Identify the owning DB responsibility module and all read projections.
30. Add the next branch migration with default/backfill rules.
31. Add field to DB read/write method and route payload; keep validation server-side.
32. Add UI control in focused fragment/feature.
33. Update export/report/notification context only if the field is part of those contracts.
34. Add migration + domain + frontend regression; update manifests only for intentional public method changes.

16.2 Add a new scheduling validation rule

35. Put the invariant in the shared appointment/availability domain, not only in the composer.
36. Exercise it through direct admin create, multi-batch, copy-forward, recurrence preparation, secure client path and calendar import where relevant.
37. Make the error specific enough for best-effort batch output.
38. Add a regression that proves both single and cumulative batch behavior.

16.3 Extend recurrence

39. Change recurrence-rules.js for rule semantics/date matching.
40. Update Individual and Group implementations if the concept is shared.
41. Decide how old patterns/occurrence rows behave.
42. Update create/edit preview UI and management description.
43. Add migration only if persisted shape changes.
44. Test no-seed creation, start boundary, end mode, edit future, skip, duplicate prevention and already-created history.

16.4 Add a new notification/Utility

45. Define event and recipients before provider details.
46. Add Utility registry contract/default template and allowed parameter sources.
47. Wire organization + recipient gates.
48. Add provider send/attachment handling.
49. Write audit/communication evidence.
50. Add tests for disabled deployment, disabled organization, recipient opt-out, invalid Utility and successful send.

16.5 Add a report/export dimension

51. Define authoritative data/timezone meaning.
52. Add server/database query support first.
53. Add filter/column UI.
54. Update CSV/PDF/email delivery if applicable.
55. Test empty data, location/timezone cases and historical rows.

17. Testing strategy and commands

Run the narrowest seam first. Then syntax/static/architecture. Then broader domain suites. If a broad suite fails, run the same test against the untouched baseline before attributing it to your change.

Change area	Minimum targeted tests
Appointment composer/workspace	appointment-composer-selection-ux; appointment-workspace-phase1; appointment-compact-controls; appointment-planner-ui-stability
Recurrence	recurring-schedules-v2; recurring-group-schedules
Timezone/location	timezone-scheduling; session-locations; location-aware-reporting; migration-location-timezone-integration
Packages	lesson-packages; package-consumption-accounting; package-currency-location-offerings; package-report-reconciliation
Calendar (Generic)	calendar-oauth-foundation; calendar-sync-review; calendar-google-import/bulk/completion

Session Notes	session-notes; session-notes-timezones; session-notes-ui; settings-persistence
Migration	migration-staging/review/commit/full-history/historical invariants
Frontend structure	frontend-modules; frontend-styles; static-quality; architecture-guardrails

```
Syntax + architecture
node --check <changed-js>
node test/frontend-modules.js
node test/frontend-styles.js
node test/static-quality.js
node test/architecture-guardrails.js

Full branch confidence (when appropriate)
npm run check
```

18. Debugging map

Symptom	First checks
UI unchanged after deploy	asset query version, correct file path, Node restart if server-composed HTML changed, browser hard refresh
Multi-create accepts too many package rows	validateAppointmentBatch packageAccepted projection + transaction recheck
Recurring occurrence missing	plan active/paused/end; start/rule; occurrence ledger; linked Package (Individual); planner week preparation
Recurring duplicate	pattern/date occurrence ledger + exact existing match detection
Wrong appointment time	scheduling timezone snapshot + canonical UTC + recipient timezone formatter
Staff unavailable	Staff-Service mapping, Location scope, hours/overrides, capacity/service limit, conflicts
WhatsApp failure	deployment gate, org route, recipient preference, Utility mapping/template contract, provider status, diagnostic.log
Calendar import blocked	OAuth connection, selected calendar/range, mapping context, Package eligibility, validation result
History page slow	summary-first endpoint and lazy detail boundary; avoid eager full-history fetch
Migration blocked	review fingerprint, unresolved mappings, source timezone/showstoppers, backup/transaction diagnostics

19. Deployment/change-set workflow

56. Start from the exact current branch baseline; keep an untouched snapshot.
57. Apply earlier required patches before building a later incremental patch.
58. Modify only owning files; regenerate assembled assets when source files change.
59. Bump browser cache query versions for served CSS/JS/HTML-dependent assets.
60. Run targeted tests + syntax/static/architecture. Compare stale broad failures with baseline.
61. Build ZIP with root-relative changed files only; no wrapper directory. Include tmp/restart.txt only when deployment convention requires Node restart.
62. Never ship .env, DB/data backups, diagnostic.log, stderr.log, tokens/keys or unrelated customer data.
63. Run unzip -t and inspect the exact file list. Reapply the patch onto a clean copy and compare packaged files byte-for-byte when the change is critical.

20. Handoff checklist for the next developer

Every non-trivial change should leave a short handoff note. This prevents the next developer from rediscovering hidden branch assumptions.

```
Feature/change:
Branch + baseline:
Owning files changed:
Schema/migration:
Behavior/invariants preserved:
Notifications/side effects:
Tests run + result:
Known baseline/stale test failures:
Deployment order / restart / cache bump:
Rollback files:
Follow-up not included:
```

- If a rule exists in more than one path, document the shared owning seam rather than listing copies.
- If Generic and Live differ, state the difference explicitly.
- If a migration is added, record branch-specific migration number and whether it is data-transforming.
- If a new Utility/provider contract is introduced, record the exact external configuration required.

21. Appendix - high-value files and tests

File/test	Why it matters
ARCHITECTURE.md	Concise contribution guardrails at repo root.
package.json scripts	Authoritative lint/check/smoke command inventory.
ledger-router.js	Composition map; useful for tracing dependencies, not feature logic.
lib/database/migrations/index.js	Branch schema head/order and integrity behavior.
test/database-method-manifest.json	Intentional LedgerDatabase public surface.
test/migration-checksums.json	Migration immutability regression.
test/frontend-modules.js	Server-composed HTML/browser module contract.
test/frontend-styles.js	Responsive/admin style contract.
test/static-quality.js	Safety/size/static guardrails.
test/architecture-guardrails.js	Layering/decomposition guardrails.
diagnostic.log	Runtime technical evidence only; never package into deployment ZIP.