SeasonPass Manager — Developer Rulebook

> *Goal: a code-referenced, operationally useful map of how this app works end-to-end.*
>
> ***Coverage note:** This document is "best-effort exhaustive" from the current repo state. Anything not found in code is marked **Unknown in codebase**.*

---

0) Scope, guarantees, and how to read this

0.1 What this rulebook covers

- Navigation and routing (Expo Router).
- Screen-by-screen behavior and every interactive element we can identify.
- Data model, persistence, backups/restores, exports.
- Schedule fetching pipeline (tRPC → Hono → ESPN/Ticketmaster).
- Operational playbooks (debug, extend, fix common issues).

0.2 What this rulebook does NOT cover

- Pixel-perfect styling and every StyleSheet constant.
- External service SLAs / data correctness from ESPN/Ticketmaster.
- Rork platform internals beyond what's visible in code.

0.3 Canonical sources in this repo

- **Routing:** `app/` (Expo Router file-based routes)
- **State + persistence:** `providers/SeasonPassProvider.tsx`
- **Backend API:** `backend/hono.ts` and `backend/trpc/**`
- **Domain types:** `constants/types.ts`
- **League/team catalog:** `constants/leagues.ts`
- **Schedule import parsing:** `lib/scheduleImporter.ts`
- **Seat parsing:** `lib/seats.ts`
- **tRPC client:** `lib/trpc.ts`

---

1) System map (high-level architecture)

1.1 Runtime architecture

- UI is a React Native app using Expo Router.
- Global state lives in a context hook created by `@nkzw/create-context-hook`:
  - `providers/SeasonPassProvider.tsx` exports `[SeasonPassProvider, useSeasonPass]`.
- Persistence is local-only via AsyncStorage:
  - `@react-native-async-storage/async-storage` is imported only in `providers/SeasonPassProvider.tsx`.
- Backend calls go through tRPC, mounted in a Hono server:
  - Client: `lib/trpc.ts` creates `trpcClient`.
  - Server: `backend/hono.ts` mounts `appRouter` at `/api/trpc`.

1.2 Data flow summary

1. UI screens call `useSeasonPass()` functions (create pass, add sale, export, etc.).

2. Provider mutates in-memory state, persists to AsyncStorage, and may also write backup artifacts.
3. Schedule fetch uses tRPC (`trpcClient.espn.getFullSchedule`) which hits the backend.
4. Backend proxies ESPN/Ticketmaster and normalizes into “events” consumed by the provider.

1.3 Environment variables

- `EXPO_PUBLIC_RORK_API_BASE_URL`
  - Used in `lib/trpc.ts` to determine the base URL for tRPC: `getBaseUrl()` .
  - Also logged/validated in `providers/SeasonPassProvider.tsx` schedule fetch.
  - In dev on web, `lib/trpc.ts` falls back to `http://localhost:8787` with a warning.
- `TICKETMASTER_API_KEY`
  - Used only in backend route `backend/trpc/routes/ticketmaster.ts` .
  - Server logs masked presence/length in `backend/hono.ts` .

---

2) Data model, persistence, backups, restores, exports

2.1 Domain types (canonical)

Defined in `constants/types.ts` :

- `SeasonPass`
  - Identifiers: `id` , `leagueId` , `teamId` (+ optional `teamAbbreviation`)
  - Branding: `teamName` , `teamLogoUrl` , `teamPrimaryColor` , `teamSecondaryColor`
  - Season: `seasonLabel` , `createdAtISO`
  - Seats: `seatPairs: SeatPair[]`
  - Schedule: `games: Game[]`
  - Sales: `salesData: Record<gameId, Record<pairId, SaleRecord>>`
  - Events: `events: Event[]`
- `SeatPair`
  - `id` , `section` , `row` , `seats` (string), `seasonCost` (number)
- `SaleRecord`
  - Links: `gameId` , `pairId`
  - Identity: `id` (currently formatted as `${gameId}_${pairId}` in Schedule modal)
  - Seat fields: `section` , `row` , `seats` , optional `seatCount`
  - Value: `price` , `paymentStatus` , `soldDate`
- `Game`
  - `id` , `date` , `month` , `day` , `opponent` , `time` , `ticketStatus` , `isPaid`
  - `type` : `'Preseason' | 'Regular' | 'Playoff'`
  - Optional: `opponentLogo` , `venueName` , `gameNumber` , `dateTimeISO`

2.2 Persistence keys (AsyncStorage)

All keys are centralized in `providers/SeasonPassProvider.tsx` .

Confirmed keys (string values):

- `SEASON_PASSES_KEY = 'season_passes'`

    - Stores the array of `SeasonPass` as JSON.
- `ACTIVE_PASS_KEY = 'active_season_pass_id'`
    - Stores the active pass id.
- `DATA_IMPORTED_KEY = 'data_imported_v1'`
    - Stores a flag string (e.g. `'true'`).
- `MASTER_BACKUP_KEY = 'master_backup_v1'`
    - Stores the "master backup" JSON.
- `ALL_PASSES_BACKUP_KEY = 'all_passes_backup_v1'`
    - Stores a full multi-pass backup JSON.
- `AUTO_RESTORE_FLAG = 'AUTO_RESTORE_COMPLETED'`
    - Stores `'true'` after auto-restore completion.

Design note: the provider uses a ref-backed state mirror (`seasonPassesRef`) to avoid "stale writes" during async persistence (especially on web).

2.3 Provider invariants and helpers

- `normalizeSeasonPass(p)` (in `providers/SeasonPassProvider.tsx`)
    - Ensures minimal fields exist and applies palette defaults from `constants/appColors.ts`.
- Seat parsing: `parseSeatsCount(seats)` in `lib/seats.ts`.
- Schedule import parsing:
    - `parseScheduleFromCSV`, `parseScheduleFromExcel`, `validateSchedule` in `lib/scheduleImporter.ts`.

2.4 Stats computation (Dashboard + Analytics)

Computed in `providers/SeasonPassProvider.tsx` as `calculateStats`:

- `seatsPerGame = Σ parseSeatsCount(pair.seats)`.
- Revenue is the sum of all `sale.price`.
- Ticket counts are computed from `sale.seatCount` if present, else parse from `sale.seats` with a fallback to 2.
- `totalTickets = games.length * seatsPerGame` (falls back to 42 games if missing).

2.5 Backups and recovery codes

- `BackupData` interface is declared in `providers/SeasonPassProvider.tsx`.
- Public recovery-code API (provider return object):
    - `createRecoveryCode()` produces a compressed recovery code string.
    - `restoreFromRecoveryCode(codeOrJson)` restores from either a compressed recovery code **or** JSON text.
- Implementation details:
    - Compression uses `lz-string` (`LZString.compressToEncodedURIComponent(JSON.stringify(data))`).
    - Parsing accepts either:
        - a compressed recovery code, or
        - raw JSON representing `BackupData`, or
        - a special "recovery context" shape under `raw.recoveryData` (used by internal recovery tooling).

2.6 Backup package / clone kit artifacts

- `buildAppSnapshotMarkdown()` (provider)
  - Generates a human-readable snapshot of app behavior.
- `buildCloneKitMarkdown({ createdAtISO, recoveryCode, backupJsonPretty, embedLogos })`
  - Produces a single Markdown "Clone Kit" containing recovery code + full JSON + snapshot.

2.7 Embedded logos (offline restore)

- `writeBackupFolder(backupObj, folderBaseName, embedAssets, fileName)` in provider
  - When `embedAssets` is true and logos are `data:` URIs:
    - On native, writes extracted files under `FileSystem.documentDirectory` and rewrites JSON references.
    - On web, cannot create a folder; triggers separate file downloads.

2.8 Exports

In `providers/SeasonPassProvider.tsx`:

- `exportAsExcel()`
  - Uses `xlsx` to write an `.xlsx` file.
- `exportAsCSV()`
  - Generates a CSV file AND copies detailed TSV rows to clipboard (for Excel paste).
- `exportAsJSON()`
  - Produces a JSON backup export.

In `app/(tabs)/settings.tsx` (UI layer):

- `handleExportExcel()` calls `exportAsExcel()`.
- `handleExportCSV()` calls `exportAsCSV()`.

2.9 Restores

- Restore from code:
  - UI: Settings modal `restoreFromRecoveryCode` is called.
  - Dedicated screen: `app/restore.tsx` reads `code` param and auto-runs restore.
- Restore from file:
  - UI triggers provider restore function (see Settings section below).

---

3) Screens and interactions (route-by-route)

3.1 Global routing

- Root navigation stack: `app/_layout.tsx`
  - Stack screens:
    - `(tabs)` (header hidden)
    - `edit-pass` (modal)
    - `setup` (fullScreenModal, gesture disabled)
    - `restore` (modal)

- Tab layout: `app/(tabs)/_layout.tsx`
  - Tabs: `index`, `schedule`, `analytics`, `events`, `settings`.
  - Redirect behavior:
    - If `needsSetup` is true, `router.replace('/setup')`.
  - Backup toast:
    - Watches `backupConfirmationMessage` and animates a toast showing success/failure.

3.2 Setup flow (route: `app/setup.tsx`)

Purpose: Create a new `SeasonPass`.

State machine: `step: 'league' | 'team' | 'season' | 'seats' | 'confirm'`.

Interactive elements and handlers:

1. League step
   - Clickable: each league card (`LEAGUES` from `constants/leagues.ts`)
     - Handler: `handleSelectLeague(league)` → sets `selectedLeague`, advances to `team`.
2. Team step
   - Clickable: team card (`getTeamsByLeague(selectedLeague.id)`)
     - Handler: `handleSelectTeam(team)` → sets `selectedTeam`, advances to `season`.
3. Season step
   - Clickable: Continue button
     - Handler: `handleSeasonNext()` → advances to `seats`.
4. Seats step
   - Clickable: mode toggle (Paired vs Individual)
     - Paired can be disabled when seat count > 2.
   - Clickable: Add button
     - Handler: `handleAddSeatPair()` → appends a `SeatPair` draft to local `seatPairs`.
   - Clickable: Trash icon per draft entry
     - Handler: `handleRemoveSeatPair(id)`.
   - Clickable: Continue
     - Handler: `handleSeatsNext()`.
5. Confirm step
   - Clickable: Start Over
     - Handler: `handleCancel()` resets local setup state.
   - Clickable: Confirm
     - Handler: `handleConfirm()` calls provider `createSeasonPass(...)` and

`router.replace('/(tabs)')`.

6. Global back
    - Clickable: Back button
        - Handler: `handleBack()` navigates step-wise or `router.back()` when appropriate.

3.3 Dashboard tab (route: `app/(tabs)/index.tsx`)

Purpose: Display season overview stats + recent sales.

Key dependencies:

- Data: `useSeasonPass()` provides `activeSeasonPass` and `calculateStats`.
- Logos: local helper `getOpponentLogo(opponentName, storedLogo?)` maps opponent names to `NHL_TEAMS`.
- Season pass selector: `components/SeasonPassSelector.tsx`.

Interactive elements:

- Seats Sold stat card
    - `onPress={openAllSales}` opens the All Sales modal.
- Recent Sales header
    - "View All (N)" `onPress={openAllSales}`.
- All Sales modal
    - Close (X) `onPress={closeAllSales}`.

Unknown in codebase: No navigation from Dashboard to edit sales directly; it's display-only besides the All Sales modal.

3.4 Schedule tab (route: `app/(tabs)/schedule.tsx`)

Purpose: Show schedule cards and allow entering sales per game.

Key dependencies:

- Data: `useSeasonPass()` provides `activeSeasonPass`, `addSaleRecord`, `removeSaleRecord`, `resyncSchedule`.
- Seat parsing: `parseSeatsCount()`.
- Auto-refresh: `refreshOnLoad()` (thin wrapper in `lib/syncGuard.ts`) triggers `resyncSchedule(passId)` on mount.

Interactive elements:

- Search input
    - `onChangeText={setSearchQuery}`.
- Filter pills
    - `onPress={() => setSelectedFilter(filter)}`.
- Game card
    - `onPress={() => openGameDetail(game)}` opens Game Sales modal.

Game Sales modal (pageSheet):

- Close (X)
  - Handler: `closeGameDetail()` resets local edit buffers.
- Save all (Check)
  - Handler: `saveAllAndClose()`:
    - For each seat pair:
      - If price string is non-empty → `saveSaleRecord(pair)` → provider `addSaleRecord(...)`.
      - If price string is explicitly empty and an existing sale exists → provider `removeSaleRecord(...)`.
- Paid/Pending toggle per seat pair
  - Handler: `togglePaymentStatus(pair.id)`.
- Price input per seat pair
  - Handler: `setEditingPrices(prev => ({...prev, [pair.id]: text}))`.
- Save single (Check)
  - Handler: `saveSaleRecord(pair)`.

Behavioral notes:

- A "sale record" is created with `soldDate = new Date().toISOString()` every time you save.
- Seat count is computed as `parseSeatsCount(pair.seats)` at save time.

3.5 Analytics tab (route: `app/(tabs)/analytics.tsx`)

Purpose: Report season revenue trends and seat pair profitability.

- Monthly revenue buckets are based on `SaleRecord.soldDate` month, not game date.
- No interactive elements (display-only).

3.6 Events tab (route: `app/(tabs)/events.tsx`)

Purpose: Track non-game event tickets (paid/sold/profit).

- Delete event
  - Handler: `handleDeleteEvent(eventId, eventName)` → provider `removeEvent(activeSeasonPassId, eventId)`.

Unknown in codebase:

- "Add Event" button exists, but no handler is wired (no navigation/modal).

3.7 Settings tab (route: `app/(tabs)/settings.tsx`)

Purpose: Data management (backup/export/restore), schedule tools, advanced troubleshooting, and navigation to edit pass.

Key helpers (local to this file):

- `markdownToPlainText(markdown)` converts `constants/userBooklet` markdown into printable plain text.
- `escapeHtml(text)` is used to render booklet text into HTML for native PDF printing.

State flags (Settings UI):

- Export/import/restore busy flags: `isResyncing`, `isExporting`, `isImporting`, `isRestoring`, `isRestoringPanthers`, `isForceReplacing`, `isReplacingSales`.
- Modals: `showRestoreModal`, `showReplaceSalesModal`.
- Advanced tools accordion: `showAdvanced`.
- Backup options: `includeLogos` (toggles embedding `data:` URIs inside backup artifacts).

Clickable inventory (in render order) and their handlers:

1. Backup status indicator
    - Retry button (only when `lastBackupStatus === 'failed'`)
        - Handler: `handleRetryBackup()` → provider `retryBackup()`.
2. SEASON PASSES section
    - "Add Season Pass" card
        - Handler: `handleAddSeasonPass()` → `router.push('/setup')`.
    - "Embed Logos in Backup (Offline Restore)" switch
        - Handler: `onValueChange={setIncludeLogos}`.
    - "Edit Active Season Pass" card (only when `activeSeasonPass`)
        - Handler: `onPress={() => router.push('/edit-pass')}`.
    - "Resync … Schedule" card (only when `activeSeasonPass`)
        - Handler: `handleResyncSchedule()`
            - Prompts user, then races provider `resyncSchedule(activeSeasonPassId)` vs a 30s hard timeout.
3. DATA MANAGEMENT section
    - "Export as Excel"
        - Handler: `handleExportExcel()` → provider `exportAsExcel()`.
    - "Export as CSV"
        - Handler: `handleExportCSV()` → provider `exportAsCSV()`.
    - "User Booklet PDF"
        - Handler: `handleDownloadUserBookletPdf()`
            - Web: dynamic import `jspdf` and `doc.save(fileName)`.
            - Native: `Print.printToFileAsync({ html })` then `Sharing.shareAsync(pdf.uri)`.
    - "Import Schedule"
        - Handler: `handleImportSchedule()` → provider `importSchedule()`.
    - "Generate & Email Clone Kit"
        - Handler: `handleGenerateRecoveryCode()` → provider `emailCloneKit(includeLogos)`.
    - "Restore from Code"
        - Handler: `onPress={() => setShowRestoreModal(true)}`.
    - "Restore from File"
        - Handler: `handleRestoreFromFile()`
            - Uses `expo-document-picker` to select JSON.
            - Reads it via `expo-file-system/legacy`.

        - Restores by calling provider `restoreFromRecoveryCode(fileText)`.

4. ADVANCED / TROUBLESHOOTING section

- "Show/Hide Advanced Tools" accordion
  - Handler: `onPress={() => setShowAdvanced(v => !v)}`.
- "Attempt ESPN Logos" (only when `activeSeasonPass`)
  - Handler: `handleFetchEspnLogos()` → provider `debugFetchLogosFromEspnForPass(activeSeasonPassId)`.
- "Restore All Season Pass Data"
  - Handler: `handleRestoreAllData()` → provider `restoreAllSeasonPassData()`.
- "Replace Sales Data" (active pass)
  - Handler: `onPress={() => setShowReplaceSalesModal(true)}`.
  - Modal:
    - Input: `onChangeText={setReplaceSalesInput}`.
    - Cancel: closes modal and clears input.
    - Replace: calls provider `replaceSalesDataFromPastedSeed(replaceSalesInput, activeSeasonPassId)`.
- "Force Replace Panthers Sales" (active pass)
  - Handler: `handleForceReplacePanthersSales()` → provider `forceReplacePanthersSales()`.
- "Email Backup"
  - Handler: `handleEmailBackup()` → provider `emailBackup(includeLogos)`.
- "Export Backup to Folder"
  - Handler: inline `onPress` that prompts for destination:
    - Email: `emailBackup(includeLogos)`
    - Messages: `prepareBackupPackage(includeLogos)` then `Share.share({ url: pkg.fileUri })` (fallback `Sharing.shareAsync`).
    - Save to Files: `prepareBackupPackage(includeLogos)`.

5. DANGER ZONE

- "Delete Current Season Pass" (active pass)
  - Handler: `handleDeleteCurrentPass()` → provider `deleteSeasonPass(activeSeasonPassId)`.
- "Clear All Data"
  - Handler: `handleClearAllData()` → provider `clearAllData()` then routes to `/setup`.

6. Restore modal (Restore from Code)

- Input: `onChangeText={setRecoveryCodeInput}`.
- Cancel: closes modal and clears input.
- Restore: `handleRestoreFromCode()` → provider `restoreFromRecoveryCode(recoveryCodeInput.trim())`.

3.8 Edit Season Pass (route: `app/edit-pass.tsx`)

Purpose: Edit seat entries and "Price Paid" (stored as `SeatPair.seasonCost`).

Entry/exit:

- Back button (header)
  - Handler: `onPress={() => router.back()}`.

Main screen clickables:

- Add seats
  - Handler: `openAdd()` → opens modal with empty fields.
- Edit seat entry (pencil icon)
  - Handler: `openEdit(pair)` → opens modal with fields pre-filled.
- Delete seat entry (trash icon)
  - Handler: `handleDelete(pair)` → confirmation `Alert`, then provider `removeSeatPair(activeSeasonPassId, pair.id)`.

Modal (Add/Edit):

- Close (X)
  - Handler: `closeModal()` (blocked while `isSaving`).
- Cancel button
  - Handler: `closeModal()`.
- Save button
  - Handler: `handleSave()`
    - Validates required fields (section/row/seats).
    - Validates cost is a non-negative number.
    - Validates seat format via `parseSeatsCount(seats)`.
    - Add path: provider `addSeatPair(activeSeasonPassId, newPair)`.
    - Edit path: provider `updateSeatPair(activeSeasonPassId, existingId, patch)`.

3.9 Restore screen (route: `app/restore.tsx`)

Purpose: Deep-link restore when opened with `?code=...`.

- If `code` param exists, auto-calls `restoreFromRecoveryCode(code)` after ~600ms.
- Restore button calls the same.

---

4) Backend API (Hono + tRPC) and schedule sources

4.1 Server mounting

- `backend/hono.ts`
  - Uses `hono/cors` with `origin: '*'`.
  - Mounts tRPC at internal route `/trpc/*` with endpoint `/api/trpc`.
  - Provides a `/` health check.

4.2 tRPC router

- `backend/trpc/app-router.ts`
  - Combines `espnRouter` and `ticketmasterRouter`.

4.3 ESPN router

- File: `backend/trpc/routes/espn.ts`
- Base URL: `ESPN_SITE_BASE = 'https://site.api.espn.com/apis/site/v2'`.

Exposed procedures:

- `espn.getTeams({ leagueId })`
  - Fetches `/sports/{sport}/{league}/teams`.
- `espn.getSchedule({ leagueId, espnTeamId })`
  - Fetches `/sports/{sport}/{league}/teams/{id}/schedule`.
- `espn.resolveTeamAndGetSchedule({ leagueId, teamAbbr?, teamName?, storedTeamId? })`
  - Resolves team id via team list or uses numeric `storedTeamId`.
- `espn.getFullSchedule({ leagueId, teamId, teamName, teamAbbreviation? })`
  - Fetches teams list then resolves a team, then fetches schedule.

Logo behavior:

- `getESPNTeamLogoUrl(leagueId, teamAbbr)` builds CDN URLs like:
  - `https://a.espncdn.com/i/teamlogos/nhl/500/{abbr}.png`
- `TEAM_LOGO_FALLBACKS` provides NHL fallbacks.

4.4 Ticketmaster router

- File: `backend/trpc/routes/ticketmaster.ts`
- Base URL: `TICKETMASTER_BASE = 'https://app.ticketmaster.com/discovery/v2'`.

Exposed procedure:

- `ticketmaster.getSchedule({ leagueId, teamId, teamName, teamAbbreviation? })`
  - Requires `process.env.TICKETMASTER_API_KEY`.
  - Uses `getSeasonDateRange(leagueId)`.
  - Builds a request to `/events.json` with query params:
    - `apikey`, `keyword`, `startDateTime`, `endDateTime`, `size`, `sort`, optionally `segmentId`, `genreId`, `city`.
  - Applies home-game heuristics via `TEAM_VENUE_MAP` and league mapping via `LEAGUE_SEGMENT_MAP`.

4.5 Client schedule fetch pipeline

Implemented in `providers/SeasonPassProvider.tsx`:

- `fetchScheduleViaESPN(pass)` → `trpcClient.espn.getFullSchedule.query(...)`.
- `fetchScheduleViaTicketmaster(pass)` → `trpcClient.ticketmaster.getSchedule.query(...)`.
- `fetchScheduleViaBackend(pass)` tries ESPN first, then Ticketmaster fallback.
- `fetchScheduleWithMasterTimeout(pass)` wraps the above with `withMasterTimeout(..., 30000, { games: [], error: 'TIMEOUT' })`.

---

5) Operational playbooks (debugging, extending, and known gaps)

5.1 Running locally

From `README.md`:

- Install: `bun i`
- Web: `bun run start-web`
- Native: `bun run start` then use Expo.

5.2 Adding a new screen

- Add a route file under `app/`.
- If it needs to be a modal, wire it in `app/_layout.tsx` Stack.
- If it needs to be a tab, wire it in `app/(tabs)/_layout.tsx` Tabs.

5.3 Adding a new league/team catalog entry

- Add the league to `constants/leagues.ts` (`LEAGUES`).
- Add the teams list and update `getTeamsByLeague(leagueId)`.
- Ensure backend ESPN league config supports it in `backend/trpc/routes/espn.ts` (`ESPN_LEAGUE_CONFIG`).
- Ensure Ticketmaster segment/genre mappings exist (optional) in `backend/trpc/routes/ticketmaster.ts`.

5.4 Common production issues and where to look

- Schedule doesn't load
  - Check `EXPO_PUBLIC_RORK_API_BASE_URL` (client) and server logs.
  - Verify `backend/hono.ts` health endpoint.
- Web share/download differences
  - Client UI uses `Platform.OS === 'web'` branching.
  - PDF generation differs between `jspdf` (web) and `expo-print` / `expo-sharing` (native).

5.5 Known gaps / TODOs (explicit)

- Events "Add Event" has no behavior in `app/(tabs)/events.tsx`.

Appendix A: Quick reference of key symbols

- Context hook: `providers/SeasonPassProvider.tsx` → `useSeasonPass()`
- Types: `constants/types.ts`
- Teams: `constants/leagues.ts`
- Seat parser: `lib/seats.ts` → `parseSeatsCount()`
- Schedule import: `lib/scheduleImporter.ts` → `parseScheduleFromCSV()`, `parseScheduleFromExcel()`, `validateSchedule()`
- tRPC client: `lib/trpc.ts` → `trpcClient`
- tRPC server: `backend/hono.ts` + `backend/trpc/app-router.ts`

Appendix B: Provider API surface (authoritative)

These fields/functions are returned from `useSeasonPass()` in `providers/SeasonPassProvider.tsx`.

- State + status

    - `seasonPasses`, `activeSeasonPass`, `activeSeasonPassId`
    - `isLoading`, `needsSetup`
    - `isLoadingSchedule`, `lastScheduleError`
    - `lastBackupTime`, `lastBackupStatus`, `backupError`, `backupConfirmationMessage`
- Season pass lifecycle
    - `createSeasonPass`, `updateSeasonPass`, `deleteSeasonPass`, `switchSeasonPass`, `clearAllData`
- Seats
    - `addSeatPair`, `removeSeatPair`, `updateSeatPair`
- Games / schedule
    - `updateGames`, `resyncSchedule`, `importSchedule`, `debugFetchLogosFromEspnForPass`
- Sales
    - `addSaleRecord`, `removeSaleRecord`, `removeAllSalesForGame`
- Events
    - `addEvent`, `removeEvent`
- Exports / backups / restore
    - `createRecoveryCode`, `restoreFromRecoveryCode`
    - `exportAsJSON`, `exportAsExcel`, `exportAsCSV`
    - `emailBackup`, `emailCloneKit`, `prepareBackupPackage`
    - `restorePanthersData`, `restoreAllSeasonPassData`, `retryBackup`
- Admin / repair
    - `forceReplacePanthersSales`, `replaceSalesDataFromPastedSeed`
- Reference data
    - `leagues`, `getTeamsByLeague`, `getLeagueById`, `getTeamById`
- Derived stats
    - `calculateStats`