## **ReAlign:Tracker MVP - Phased Implementation Plan**

**Document Version:** 1.0

**Date:** May 23, 2025

**Based on:** "ReAlign Product Architecture v1.0" and existing ReAlign MVP codebase.

**Project Goal:** To build the ReAlign:Tracker MVP, a focused visibility tool allowing negotiators to provide transaction participants with a real-time, shared view of short sale status and document progress. This MVP prioritizes a color-coded document status table, static phase tracking, preformatted activity updates by the negotiator, and automated weekly email digests.

**Tech Stack (Confirmed & Assumed):**

* **Frontend:** React, TypeScript, Vite, TanStack Query, Wouter, Tailwind CSS, Shadcn/ui components
* **Backend:** Node.js, Express, TypeScript
* **Database:** PostgreSQL (via Supabase or similar), Drizzle ORM
* **Authentication:** Supabase Auth (assumed based on AuthContext.tsx and magic link mentions in previous docs).
* **Styling:** Tailwind CSS, existing Shadcn UI components.

**Static Transaction Phases (New List):**

1. Transaction Initiated
2. Property Listing
3. Documentation Collection
4. Hardship Package Submitted
5. Offer Received
6. Offer Submitted to Lender
7. Initial Lender Review
8. Property Valuation Ordered
9. Lender Negotiations
10. Final Approval Received
11. In Closing

### **Phase 0: Project Setup & Refinement**

**Objective:** Align the existing codebase with the new Tracker MVP spec. Remove or mark for deprecation features not part of this MVP. Update core data structures.

**Key Tasks:**

1. **Backend - Schema Update (shared/schema.ts):**
   * Review and modify existing tables to align with the new data model:
     + transactions: Ensure it can store property\_address, title, current\_phase\_key (string to link to static phases), negotiator\_id.
     + transaction\_phase\_history: New table or adapt existing. Columns: id, transaction\_id, phase\_key (string), timestamp, set\_by\_negotiator\_id.
     + doc\_requests: Columns: id, transaction\_id, document\_name, assigned\_party\_role (e.g., 'Seller', 'Buyer Agent'), status ('Pending', 'Complete', 'Overdue'), requested\_at, completed\_at (nullable), due\_date (nullable).
     + doc\_uploads: Simplify if existing. Columns: id, doc\_request\_id (optional, if directly fulfilling a request), transaction\_id, uploader\_id (negotiator only for MVP), file\_name, file\_path, uploaded\_at, visibility (default 'Private' - negotiator only).
     + tracker\_notes: Columns: id, transaction\_id, note\_text (can store predefined or custom), created\_at, negotiator\_id.
     + email\_subscriptions: Columns: id, transaction\_id, party\_email, party\_role, is\_subscribed (boolean), magic\_link\_token (for view-only access), token\_expires\_at.
   * Ensure Drizzle ORM configurations (drizzle.config.ts) are updated and migrations are planned.
2. **Frontend - Static Phase Definition:**
   * Create a new constants file (e.g., client/src/lib/phases.ts) to store the 11 static phases with keys and display names.
   * Example: export const TRANSACTION\_PHASES = [ { key: 'TRANSACTION\_INITIATED', name: 'Transaction Initiated' }, ... ];
3. **Code Review & Removal/Deprecation Marking:**
   * **Messaging System:**
     + Frontend: client/src/components/transactions/MessageThread.tsx, and related UI in TransactionView.tsx. Mark for removal or heavily simplify if parts can be reused for Tracker Notes.
     + Backend: server/controllers/messageController.ts and its routes in server/routes.ts.
   * **Document Upload for Non-Negotiators:**
     + Frontend: client/src/components/transactions/UploadWidget.tsx needs to be restricted or adapted for Negotiator-only use. Remove UI for non-negotiators to upload.
     + Backend: server/controllers/uploadController.ts to enforce negotiator-only uploads.
   * **Complex Notifications:**
     + Frontend: client/src/pages/NotificationSettings.tsx may need simplification to focus on email subscription preferences for the weekly digest. SMS/Push notification UI to be removed for MVP.
     + Backend: server/services/notificationService.ts and server/controllers/notificationController.ts to be refactored for weekly email digests. Remove SMS/Push logic.
   * **Parties Page (/parties link in client/src/components/layout/Sidebar.tsx):** The new spec does not define a global parties management page. Decision: Remove this link and page for Tracker MVP to maintain focus. Party management is per-transaction.
   * **Document Request System (client/src/components/transactions/DocRequestList.tsx):** Review against new "Doc Status Table" spec. Logic for assigning docs by negotiator remains, but fulfillment by non-negotiators via upload is removed for MVP.

**Testing Focus:**

* Confirm database schema changes.
* Verify removal/inactivation of out-of-scope features doesn't break remaining core functionality (login, basic transaction list).

### **Phase 1: Transaction Creation & Core Data Structures**

**Objective:** Allow negotiators to create new transactions and associate initial parties for tracking and email subscriptions. Implement backend logic for core Tracker data.

**Key Tasks:**

1. **Backend - Transaction Management (server/controllers/transactionController.ts):**
   * POST /api/v1/transactions: "Modify to capture property address, title, **and the custom welcome\_email\_body**. Set initial phase to "Transaction Initiated"."
   * "During creation, allow negotiator to add parties (email, role: 'Agent', 'Homeowner') for email\_subscriptions table (default is\_subscribed=true). Generate unique magic link tokens for each, **setting token\_expires\_at to 6 months from creation by default.**"
   * GET /api/v1/transactions/:id: Ensure it returns transaction details including current phase, property address.
   * GET /api/v1/transactions: List transactions for the negotiator (Dashboard view).
2. **Frontend - New Transaction Form (client/src/pages/NewTransaction.tsx):**
   * Simplify form: Property Address, Title.
   * "Section to add parties (email, role) who will be subscribed to Tracker updates. Include welcome email configuration (text area for a custom message, prefilled with a default template like the one in transactions.welcome\_email\_body). **Ensure this customized text is sent to the backend to be saved with the transaction.**"
3. **Frontend - Dashboard (client/src/pages/Dashboard.tsx):**
   * Update to reflect focus on Tracker: show "Total Transactions". Active/Completed distinction may change based on phase or be simplified.
   * Recent transactions should link to the new "Tracker View" for negotiators.

**Testing Focus:**

* Negotiator can create a transaction with party emails for subscription.
* Data correctly saved in transactions and email\_subscriptions.
* Initial phase correctly set.
* Dashboard lists new transactions.

### **Phase 2: Phase Tracker Implementation**

**Objective:** Enable negotiators to view and update the transaction phase from the static list. Display phase history.

**Key Tasks:**

1. **Backend - Phase Management (server/controllers/transactionController.ts):**
   * PUT /api/v1/transactions/:id/phase: Allow negotiator to update current\_phase\_key on the transaction. Record this change in transaction\_phase\_history.
   * Endpoint to fetch transaction\_phase\_history for a given transaction.
2. **Frontend - Negotiator Tracker View (Adapt client/src/pages/TransactionView.tsx):**
   * Display current transaction phase prominently.
   * Use a dropdown (Select component from client/src/components/ui/select.tsx) populated with the 11 static phases for the negotiator to update the phase.
   * Visual display of all 11 phases (e.g., a stepper or list) with the current phase highlighted. Could adapt PhaseTracker.tsx.
   * (Optional for MVP, consider for v1.1) Display phase change history with timestamps.

**Testing Focus:**

* Negotiator can update transaction phase.
* Phase changes are recorded and displayed correctly.
* UI clearly shows current phase and overall progress.

### **Phase 3: Doc Status Table & Tracker Notes (Negotiator View)**

**Objective:** Implement the core Doc Status Table and Tracker Notes functionality for the negotiator.

**Key Tasks:**

1. **Backend - Document & Note Management:**
   * **Doc Requests (server/controllers/documentController.ts - may need new controller or adapt transaction controller):**
     + POST /api/v1/transactions/:id/doc-requests: Negotiator assigns a document (custom name or predefined list if desired later), assigns to a party role. Initial status 'Pending', requested\_at set.
     + PUT /api/v1/doc-requests/:requestId: Negotiator updates status ('Complete', 'Overdue'). completed\_at set if 'Complete'.
     + DELETE /api/v1/doc-requests/:requestId: Negotiator removes a doc request.
     + GET /api/v1/transactions/:id/doc-requests: Fetch all doc requests for a transaction.
   * **Doc Uploads (Negotiator Only - server/controllers/uploadController.ts):**
     + Ensure POST /api/v1/uploads/:transactionId is restricted to negotiators. Link upload to doc\_request\_id if applicable.
   * **Tracker Notes (New controller trackerNoteController.ts):**
     + POST /api/v1/transactions/:id/tracker-notes: Negotiator adds a note (predefined or custom). Timestamp auto-applied.
     + GET /api/v1/transactions/:id/tracker-notes: Fetch notes for a transaction.
2. **Frontend - Negotiator Tracker View (client/src/pages/TransactionView.tsx):**
   * **Doc Status Table:**
     + Use Table component (client/src/components/ui/table.tsx).
     + Columns: Document Name, Party Role, Status (color-coded: 🟢 Complete, 🟡 Pending, 🔴 Overdue), Days Since Requested.
     + Negotiator can add new doc requests, edit (status), and delete existing ones.
     + Implement client-side logic for "Days Since Requested."
     + Color-coding thresholds (e.g., Pending > X days = Overdue, or allow negotiator to set thresholds later - for MVP, use fixed thresholds).
     + (Negotiator only) Link to upload document for a request, or general upload.
   * **Tracker Notes / Activity Updates:**
     + Display list of notes with timestamps, attributed to "Negotiator."
     + Input field/dropdown for negotiator to add new notes (predefined list + custom option).
     + Predefined phrases: "Called lender – docs needed", "Initiated in Aspen Grove", "BPO ordered", etc. Store these as constants."
     + Input field/dropdown for negotiator to add new notes (predefined list + custom option).
     + Predefined phrases should be stored as constants in a shared file, e.g., client/src/lib/trackerNoteOptions.ts (or shared/constants/trackerNoteOptions.ts if backend validation of these specific phrases is also required). Example content for trackerNoteOptions.ts:
     + export const TRACKER\_NOTE\_OPTIONS = [
     + "Called lender – docs needed",
     + "Initiated in Aspen Grove",
     + "BPO ordered",
     + "Equator initiated",
     + "Underwriter review",
     + "Final offer package submitted"
     + ];

**Testing Focus:**

* Negotiator can CRUD doc requests.
* Doc status table displays correctly with color-coding and day counts.
* Negotiator can upload documents (ensure ImageUpload or a new FileUpload component is used and restricted).
* Negotiator can add and view tracker notes.

### **Phase 4: Public Tracker View & Email Subscriptions**

**Objective:** Create the view-only Tracker page for subscribed parties (Agents, Homeowners) accessible via a magic link. Manage email subscriptions.

**Key Tasks:**

1. **Backend - Public Access & Subscriptions:**
   * GET /api/v1/tracker/:transactionId?token=[magic\_link\_token]: New public endpoint. Validates token against email\_subscriptions table. If valid, returns data for the public Tracker view (phases, doc status table - roles only, tracker notes).
   * POST /api/v1/tracker/unsubscribe: Endpoint for parties to unsubscribe from weekly emails (updates is\_subscribed in email\_subscriptions).
2. **Frontend - Public Tracker View (New Page: client/src/pages/PublicTrackerView.tsx):**
   * View-only version of the Negotiator Tracker View.
   * Displays: Property Address, Current Phase (and visual phase list), Doc Status Table (roles, not names; no edit controls), Tracker Notes.
   * No navigation/sidebar from AppShell.tsx, it's a standalone page.
   * Include an "Unsubscribe" link/button.  
     "**If a user accesses this page with an expired or invalid token, display a user-friendly message, e.g., 'This access link has expired or is invalid. Please contact your negotiator for a new link.'**"
3. **Frontend - Subscription Management (Part of client/src/pages/NewTransaction.tsx initially):**
   * Negotiator adds party emails and roles at transaction creation.
   * (Future) A separate interface for negotiators to manage subscriptions per transaction.
   * Add a sub-bullet: "**(Future/MVP Stretch Goal for Negotiator UI):** In the negotiator's view of a transaction (e.g., where they manage parties or subscriptions), add a 'Resend Welcome/Access Link' button for each subscribed party. Clicking this would trigger the backend to generate a new magic link token (with a new expiry) and resend the welcome email containing this new link."

**Testing Focus:**

* Magic link grants access to the correct transaction's view-only tracker.
* Public Tracker View displays data correctly and is indeed view-only.
* Unsubscribe functionality works.
* Ensure no sensitive party PII (beyond role) is exposed on the public view.

### **Phase 5: Weekly Email Digest**

**Objective:** Implement the automated weekly email digest to all subscribed parties.

**Key Tasks:**

1. **Backend - Email Service (server/services/notificationService.ts refactor):**
   * Integrate an email sending service (e.g., SendGrid, Supabase has email sending capabilities).
   * Logic to compile email content:
     + Subject: "[Tracker Update] Your Short Sale Status - [Property Address]"
     + Body: Summary of docs completed, docs overdue, recent activity updates (Tracker Notes).
     + Link to the full Tracker view (using their magic link).
     + Unsubscribe link.
   * Cron job or scheduled task (e.g., using node-cron or a Replit/Supabase cron feature) to run every Friday at 5 PM (configurable timezone, default to server time or UTC).
     + Job queries email\_subscriptions for is\_subscribed=true parties.
     + Fetches necessary data for each transaction.
     + Sends emails.
2. **Backend - Welcome Email:**
   * When a negotiator adds parties at transaction creation and configures the welcome message, send this initial email. It should also contain the magic link to the Tracker and info about weekly updates.

**Testing Focus:**

* Welcome emails are sent upon transaction creation with correct magic links.
* Weekly digest emails are sent correctly to all subscribed parties.
* Email content is accurate and includes all specified sections.
* Unsubscribe links in emails work.
* Scheduler triggers emails at the correct time.

### **General Considerations & Testing Strategy:**

* **Authentication (client/src/context/AuthContext.tsx, server/middleware/auth.ts):** Ensure all negotiator-only actions are properly protected.
* **Error Handling:** Implement robust error handling on both frontend (e.g., using toasts from client/src/hooks/use-toast.ts) and backend.
* **UI Consistency:** Leverage existing Shadcn/ui components and Tailwind CSS for a consistent look and feel. Refer to client/src/index.css for base styles.
* **Responsiveness:** Ensure all new views are responsive, especially the Public Tracker View.
* **Testing:**
  + **Unit Tests:** For critical backend logic (email compilation, status calculations) and frontend utility functions.
  + **Integration Tests:** For API endpoints and database interactions.
  + **End-to-End (E2E) Tests:** (Manual for MVP, as per original PRD non-goal for automation) covering key user flows:
    - Negotiator creates transaction, adds parties.
    - Negotiator updates phase, adds doc requests, adds tracker notes.
    - Subscribed party receives welcome email, accesses public tracker via magic link.
    - Subscribed party receives weekly digest.
    - Party unsubscribes.