**ReAlign MVP – Component Specification**

**Version:** 1.8

**Date:** May 21, 2025

**Focus:** Mobile-first, accessible, stable UI

**Stack:** React + TailwindCSS + shadcn/ui + Lucide Icons

This document defines the core reusable frontend components used throughout ReAlign MVP, including props, behaviors, accessibility patterns, and design rules. All components are purpose-built for MVP constraints: static values, fixed role logic, and minimized configuration, ensuring they are easy to build and use, especially for non-technical users.

**General UI/UX & Accessibility Principles:**

* **Mobile-First:** Components are designed for mobile screens first and then scale up for larger viewports.
* **Accessibility (A11y):** Adherence to WCAG 2.1 Level AA guidelines is a priority. Components will leverage shadcn/ui's accessibility features. Specific A11y considerations are noted per component.
  + **Keyboard Navigation:** All interactive elements must be fully keyboard navigable (tab order, arrow keys for selection where appropriate).
  + **Focus Management:** Visible focus indicators on all interactive elements (as per Brand Kit). Focus should be managed logically, especially in modals or dynamic content changes.
  + **ARIA Attributes:** Use appropriate ARIA roles, states, and properties to enhance screen reader compatibility where not handled by default HTML semantics or shadcn/ui.
* **Tap Targets:** Minimum 44x44px tap targets for all interactive elements on touch screens.
* **Feedback:** Provide clear visual feedback for all user interactions (e.g., hover, focus, active states, loading indicators, success/error messages via a global notification/toast system).
* **Consistency:** Components should maintain a consistent look, feel, and behavior across the application.

**📘 PhaseTracker**

* **Purpose:** Display current transaction phase visually across 9 standard steps.
* **Props:**
  + currentPhase: string (must match one of 9 fixed phase labels from PHASES constant)
  + showTimeline?: boolean (default: false)
  + creationDate?: Date (optional, for estimated calendar ranges)
* **Behavior:**
  + Collapsed by default: Displays current phase concisely (e.g., “You’re in Phase X: [Label]”).
  + Expandable (on click/tap) to show a full stepper view with descriptions for each phase.
  + Current phase is visually highlighted (e.g., using primary brand color).
  + If creationDate is provided and showTimeline is true, it can optionally show estimated week/date ranges per phase (calculation logic for MVP might be simplified, e.g., predefined durations per phase, or just showing creation date).
  + **States:**
    - Default: Collapsed.
    - Expanded: Shows timeline/stepper.
    - Hover/Focus (on interactive elements like expand toggle): Visual cue as per Brand Kit.
* **Accessibility:**
  + Full keyboard access for expansion and navigation if timeline becomes scrollable.
  + Screen-reader descriptions for each phase and current status.
  + ARIA attributes: aria-expanded on the toggle, aria-current="step" on the active phase.

**📘 PartyCard**

* **Purpose:** Display each party’s role and status within a transaction.
* **Props:**
  + role: 'Seller' | 'Buyer' | 'Listing Agent' | 'Buyer’s Agent' | 'Escrow' (Fixed list for MVP)
  + name: string
  + status: 'pending' | 'complete' | 'overdue' (Visualized by color-coded status badges)
  + lastAction?: string (e.g., “Viewed update 2h ago”)
  + isEditable?: boolean (default: false, true for negotiator view to allow status toggling)
* **Behavior:**
  + Displays party name, role, and current status.
  + Status badge color should map to Brand Kit status colors (e.g., Yellow for pending, Green for complete, Red for overdue).
  + Timestamp/last action displayed if present.
  + If isEditable is true, negotiator can click status to toggle (details in TransactionView logic).
  + **States:**
    - Hover/Focus (if interactive, e.g., for negotiator edits): Visual cue.
* **Accessibility:**
  + Content should be clearly readable. Status badge should have accessible text (e.g., via aria-label or visually hidden text) if color is the primary indicator.

**📘 MessageThread**

* **Purpose:** Show flat message list with optional replies.
* **Props:**
  + messages: Message[] (Each Message object: id, senderName, senderRole, text, timestamp, optional replyToId, isSeedMessage).
  + currentUserRole: string
  + onSendMessage: (text: string, replyToId?: string) => void
  + onUpload?: (file: File) => void (If direct upload from message area is supported)
  + initialMessageEditable?: boolean (default: false, for the very first message by negotiator)
* **Behavior:**
  + One main thread per transaction.
  + Initial seed message can be editable during transaction creation by the negotiator.
  + Replies are shown inline, visually indented or linked to the parent message.
  + Thread auto-scrolls to the latest message on load or new message.
  + Message input area available based on role (negotiator can start new, others can reply).
  + Long messages might be truncated with an option to expand, or allow scrolling within a max-height message bubble.
  + Mentions (@User Name) are plain text in MVP.
  + **States (Input Area):**
    - Default: Empty or with placeholder.
    - Typing: Text visible.
    - Disabled: If user doesn't have permission to send/reply.
    - Focus: Input field highlighted.
* **Accessibility:**
  + Each message labeled by sender and timestamp.
  + Input area should have an accessible label.
  + ARIA attributes for live regions if messages update in real-time (post-MVP).

**📘 DocRequestList**

* **Purpose:** Show and manage document requests.
* **Props:**
  + requests: Request[] (Each Request object includes id, docType, assignedToName, dueDate, status, and an optional revisionNote: string).
  + currentUserRole: string
  + onUpdateRequestStatus: (id: string, newStatus: string) => void (Negotiator action)
  + onRemind: (id: string) => void (Negotiator action)
  + onResetToPending: (id: string, note: string) => void (Negotiator action)
  + onUploadForRequest: (requestId: string) => void (Party action, likely triggers UploadWidget focus/context)
* **Behavior:**
  + Requests can be grouped by party or shown as a flat list.
  + Each request displays docType, assignedToName, status, and dueDate.
  + Status icons (e.g., pending, complete, overdue) are used.
  + If a request has status: 'pending' and a revisionNote is present, this note is displayed prominently below the request details.
  + Negotiators see controls to update status (e.g., toggle complete, resend reminder, reset to pending with note).
  + Assigned parties see a clear CTA to upload/fulfill their requests.
  + **States (for each request item):**
    - Default: Displays information.
    - Hover/Focus (on interactive elements): Visual cue.
* **Design:**
  + Stacked layout on mobile.
  + Clear visual differentiation for status using icons and/or colors from Brand Kit.
* **Accessibility:**
  + Each request and its controls should be keyboard accessible.
  + Revision notes should be clearly associated with their respective requests for screen readers.

**📘 UploadWidget**

* **Purpose:** Handle file uploads for transactions.
* **Props:**
  + transactionId: string
  + onUploadComplete: (uploadDetails: object) => void (Callback with details of the uploaded file like name, URL, docType, visibility)
  + defaultVisibility?: 'shared' | 'private' (default: private)
  + docType?: string (Optional: if the upload is for a specific pre-defined docType, otherwise user might select or it's general)
  + role?: string (Optional; used for UI personalization or defaulting visibility)
  + maxFileSizeMB?: number (default: 10MB, as per backend limit)
* **Behavior:**
  + Accepts file via drag-and-drop (desktop) and file picker (all devices), including mobile camera input.
  + Previews selected file name and type (e.g., icon for PDF, thumbnail for images if feasible in MVP).
  + Compresses images client-side before upload initiation.
  + Uploads to a signed URL provided by the backend.
  + Displays upload progress (e.g., progress bar).
  + **Error Handling:** If an upload fails, an error message (e.g., 'Upload failed. File too large. Please try again.') is displayed next to the file or in a summary area. A 'Retry' button allows re-initiation for that specific file. The widget should allow clearing a failed/selected file.
  + **Visibility Options:**
    - User can select visibility:
      * 'Private': Viewable only by the uploader and negotiator(s).
      * 'Shared': Viewable by all transaction parties.
    - Defaults to 'private'.
  + A small, persistent informational subtext (e.g., 'Uploads are private to you and the negotiator by default.') is visible near the upload area.
  + Successfully uploaded files might be shown in a temporary list within the widget or trigger onUploadComplete for parent component to handle.
  + **States:**
    - Idle: Awaiting file selection.
    - File Selected: Shows file info, ready for upload.
    - Uploading: Progress indicator visible.
    - Success: Confirmation message or visual cue.
    - Error: Error message and retry option.
    - Disabled: If user doesn't have upload permission.
* **Security:**
  + Metadata sent with upload includes userId, docType, visibility.
  + Client-side check for file size against maxFileSizeMB before attempting upload.
* **Availability:**
  + Upload permissions for all transaction participants. Uploads are scoped to the transaction.

This component specification aims to provide frontend developers with clear guidance, incorporating accessibility and detailed interaction states for a robust and user-friendly MVP.