ReAlign MVP - AI Assisted Build Instructions (Replit) Version: 1.1 (Updated with AI Agent Specificity) Date: May 21, 2025

Objective: This document outlines the sequential steps to build the ReAlign MVP application using the Replit AI coder, guided by a human developer. Please refer to the specific .md and .docx files in the docs/ directory (or the provided source documents) for detailed specifications at each step.

General AI Guidance Additions:

"Iterative Development: We will build the application in small, manageable phases. After each significant step, code should be tested, and then committed to version control. "Document Reference: For each task, explicitly state which .md or .docx document(s) and relevant sections within them you are using as the primary source. "Replit Secrets: All environment variables (API keys, database URLs, etc.) must be stored using Replit's "Secrets" feature. Refer to docs/ReAlign MVP – Technical Architecture Overview.docx for variables needed. "Coding Standards: Adhere to ESLint and Prettier configurations (to be provided or generated based on docs/ReAlign MVP – Technical Architecture Overview.docx). All UI elements must align with docs/ReAlign MVP – Brand Kit.docx and docs/ReAlign MVP - Figma Wireframe Spec.docx. "Ask for Clarification: If any instruction is unclear or seems to conflict with a specification document, please ask for clarification. "State Derivation and Flow: When data is fetched from an API, store it in appropriate React state variables (e.g., using useState, useReducer, or React Context where global state is needed). UI elements should then derive their content and behavior from these state variables. Avoid direct DOM manipulation. Ensure state updates correctly trigger re-renders. "Error Propagation and Display: When a backend API call fails, the error handling logic in the frontend (e.g., in a .catch() block or by checking response.ok) should update a relevant state variable (e.g., an error state as seen in the scaffold ). This state variable should then be used to display user-friendly error messages. While the scaffold shows a global error display, strive to show contextual error messages within or near the component that initiated the action (e.g., 'Upload failed. File too large.' directly in the UploadWidget ). Refer to specific error messages in Supplementary Information for ReAlign MVP AI Coding Agent.docx and ReAlign MVP - Figma Wireframe Spec.docx.

"Loading States and UI Feedback: For any asynchronous operation (e.g., API calls), implement and manage loading states (like the uploading state in the scaffold ). Use these states to provide clear visual feedback to the user, such as disabling buttons, showing spinners/loaders (as per ReAlign MVP – User Onboarding Flow.docx ), or displaying progress indicators (like for file uploads ).

"Scaffold as a Primary Reference for TransactionView: The docs/ReAlign MVP - Scaffold.docx (TransactionView.tsx) is a critical reference for the structure, initial state management, mock API function signatures (which you'll replace with actual Workspace calls), and component interactions within the main transaction view. Adapt and expand upon its patterns, replacing all mock implementations with calls to the backend API endpoints defined in docs/ReAlign MVP - API Routes.docx. "Key Usage in React Lists: When rendering dynamic lists of components in React (e.g., messages, document requests, party cards), always assign a unique and stable key prop to each root element in the list. This typically should be the item's unique ID from the database (e.g., message.id). "Strict Adherence to API Contracts: Ensure all frontend API calls (e.g., using Workspace) strictly adhere to the request body formats, query parameters, and HTTP methods specified in docs/ReAlign MVP - API Routes.docx. Similarly, ensure the frontend correctly processes the JSON responses, including pagination objects and standardized error responses.

"Environment Variable Integration: Explicitly use Replit Secrets for all sensitive information. Frontend: SUPABASE\_URL and SUPABASE\_ANON\_KEY must be used to initialize the Supabase client (@supabase/supabase-js). These are typically exposed as VITE\_SUPABASE\_URL and VITE\_SUPABASE\_ANON\_KEY via .env and accessed as import.meta.env.VITE\_SUPABASE\_URL. Backend: DATABASE\_URL (from Supabase) is needed for Drizzle ORM. SUPABASE\_URL and SUPABASE\_SERVICE\_ROLE\_KEY (or SUPABASE\_ANON\_KEY if RLS is fully sufficient for backend operations) are needed for the backend Supabase admin client. Other service keys (e.g., for email/SMS/push notification services) will also be accessed from Replit Secrets. "Focus on Prop Drilling vs. Context: For MVP, prop drilling is acceptable for simpler state sharing. Use React Context primarily for global states like session/user data and theme/brand information if not handled by Tailwind directly. The useAuth hook in the scaffold is a good example for user data. "Client-Side Validation with Zod: Before submitting forms or making API calls, implement client-side validation using Zod schemas defined in Supplementary Information for ReAlign MVP AI Coding Agent.docx (e.g., LoginSchema, CreateTransactionSchema ). This provides immediate feedback to the user and reduces invalid backend requests.

Phase 0: Project Setup & Foundation

Workspace Structure: Ensure the Replit workspace has the following top-level directories: docs/ (Upload all provided specification .docx files here) frontend/ backend/ Initialize Projects: In frontend/, initialize a Vite + React (TypeScript) project. In backend/, initialize a Node.js (TypeScript, if preferred) project. Install Core Dependencies: Frontend (frontend/package.json): react, react-dom, @vitejs/plugin-react, vite, tailwindcss, autoprefixer, postcss, lucide-react, framer-motion, zod, @supabase/supabase-js, and a router (e.g., react-router-dom as a robust alternative to the specified Vite Router unless a specific vite-router package is standard). Backend (backend/package.json): express, drizzle-orm, drizzle-kit (for migrations), pg (PostgreSQL driver), zod, @supabase/supabase-js, dotenv (for local dev if needed, but Replit Secrets for deployment), and any chosen push notification SDK (e.g., firebase-admin). Configuration Files: ESLint & Prettier: Set up ESLint and Prettier in both frontend/ and backend/ projects. Use the configurations from the Supplementary Information for ReAlign MVP AI Coding Agent.docx, or ask for specific config files if provided. TailwindCSS (Frontend): Configure tailwind.config.js and postcss.config.js in frontend/. The theme should directly reflect docs/ReAlign MVP – Brand Kit.docx and the shadcn/ui setup. Translate the Color Palette (Primary Blue #263b75, Accent Yellow #f2c94c, etc.) from the Brand Kit into the theme.extend.colors section. Example: brand: { primary: '#263b75', accent: '#f2c94c' }. Map Primary Font: Inter and the fallback stack from the Brand Kit to theme.extend.fontFamily.sans.

Implement Border Radius: rounded-2xl (12px) from the Brand Kit as theme.extend.borderRadius['2xl'] = '12px'.

For shadcn/ui integration, ensure tailwind.config.js is set up to support its theming with CSS variables. The Brand Kit colors should be mapped to shadcn/ui's semantic theme variables (e.g., primary, secondary, accent, destructive, background, foreground) within the Tailwind config, as shown in the example tailwind.config.js. Include tailwindcss-animate plugin. TypeScript (tsconfig.json): Ensure basic tsconfig.json files are present in both frontend/ and backend/ if using TypeScript. Replit Secrets: Set up initial Replit Secrets for: SUPABASE\_URL, SUPABASE\_ANON\_KEY (for frontend), SUPABASE\_SERVICE\_ROLE\_KEY (for backend, if needed for admin tasks), DATABASE\_URL (from Supabase for Drizzle ORM). Add more as required (e.g., push notification service keys). Refer to docs/ReAlign MVP – Technical Architecture Overview.docx. Specifically, SUPABASE\_URL and SUPABASE\_ANON\_KEY will be used in the frontend Supabase client initialization. SUPABASE\_URL, SUPABASE\_SERVICE\_ROLE\_KEY (if used), and DATABASE\_URL will be used in the backend. Ensure these are accessed correctly in your code (e.g., process.env.DATABASE\_URL in Node.js backend, import.meta.env.VITE\_SUPABASE\_URL in Vite frontend). Phase 1: Backend - Database Schema Implementation

Primary Document: docs/ReAlign MVP - Database Schema.docx Tool: Drizzle ORM Tasks: Create Drizzle ORM schema definitions for each table: users transactions transaction\_participants (ensure UNIQUE (transaction\_id, user\_id, role\_in\_transaction) constraint ) messages document\_requests (ensure revision\_note field is included )

uploads Ensure all specified fields, types, constraints (e.g., CHECK constraints for enums like users.role, transactions.current\_phase, transaction\_participants.status, document\_requests.status, uploads.visibility ), relationships (foreign keys), and default values (e.g., gen\_random\_uuid() for ID fields, now() for timestamps) are correctly implemented.

When defining Drizzle schemas, ensure NOT NULL constraints are applied as per the SQL CREATE TABLE statements in Database Schema.docx (e.g., users.email is NOT NULL, transactions.title is NOT NULL ).

Explicitly set up ON DELETE actions (e.g., ON DELETE CASCADE for transaction\_participants.transaction\_id, ON DELETE SET NULL for messages.sender\_id ) as specified or implied by the schema descriptions.

Set up Drizzle Kit for migrations. Generate and apply the initial migration to your Supabase PostgreSQL database. Verify the table structures directly in the Supabase dashboard. Phase 2: Backend - Authentication & Core User API

Primary Documents: docs/ReAlign MVP - API Routes.docx (Auth section), docs/ReAlign MVP – Security & Privacy Plan.docx, docs/ReAlign MVP – User Onboarding Flow.docx. Technology: Express.js, Supabase Auth, Zod for validation. Tasks: Integrate Supabase Auth: Set up Supabase client for backend usage. Implement logic for email/password sign-in (negotiators).

Implement logic for magic link generation and verification (other roles). This involves calling the appropriate Supabase Auth methods. For sending the actual email/SMS containing the magic link, this step should interface with the Unified Notification Service foundation built in Phase 7. For now, it can use Supabase's built-in email for magic links if customization isn't immediately required by the notification service.

Implement API Endpoints (under /api/v1/auth): POST /auth/login: Handle negotiator login. POST /auth/magic-link: Generate and send magic link. Include rate limiting.

POST /auth/magic-link/resend: Resend magic link. Include rate limiting (max 3 per hour ).

GET /auth/me: Return current authenticated user details. Requires JWT auth middleware. The user details (name, email, role, phone) should be fetched from the users table using the id from the verified Supabase JWT. Develop JWT authentication middleware to protect routes. This middleware should verify Supabase JWTs. Use Zod for all request body validation as per API Routes.docx and Supplementary Information (e.g., LoginSchema, MagicLinkRequestSchema ).

Implement standardized error responses. Phase 3: Backend - Transaction Management API

Primary Documents: docs/ReAlign MVP - API Routes.docx (Transactions, Party Status sections), docs/ReAlign MVP - Database Schema.docx (transactions, transaction\_participants), docs/ReAlign MVP - PRD.docx (FR-4.1, FR-4.2). Tasks: Implement API Endpoints for Transactions (under /api/v1/transactions): POST /transactions: Create a new transaction. Validate input (title, property\_address, parties, initialPhase, initialMessage) using Zod schemas (e.g., CreateTransactionSchema ). Create entries in transactions and transaction\_participants tables. Store the initialMessage in the messages table with is\_seed\_message=true. The sender\_id for this seed message should be the negotiator creating the transaction.

Ensure created\_by is linked to the negotiator user.

GET /transactions: List transactions accessible to the authenticated user (role-based filtering). Implement pagination.

GET /transactions/:id: Fetch full transaction details, including parties, limited messages, document requests, and uploads (filtered by user role and upload visibility). Remember that for non-negotiator roles, uploaded files marked 'private' by other users should be filtered out. For messages and document requests, ensure they belong to the specified transaction\_id.

PATCH /transactions/:id: Update transaction metadata (negotiator only). Validate with Zod schema (e.g. UpdateTransactionSchema ).

Implement API Endpoints for Party Status (under /api/v1/transactions/:transactionId/parties): GET /transactions/:id/parties: List all parties for a transaction. PATCH /transactions/:transactionId/parties/:userId: Update a party's status (negotiator only). Implement robust role-based access control (RBAC) middleware for all transaction-related routes. This middleware must: Verify the JWT (using JWT middleware from Phase 2). Extract user\_id and role from the token. For transaction-specific routes (e.g., /transactions/:id/\*), query transaction\_participants to confirm the user\_id is linked to the transaction\_id from the path parameter. For operations restricted to 'negotiator' (e.g., PATCH /transactions/:id), additionally check if the user's role is 'negotiator'. Phase 4: Backend - Messaging API

Primary Documents: docs/ReAlign MVP - API Routes.docx (Messages section), docs/ReAlign MVP - Database Schema.docx (messages), docs/ReAlign MVP - PRD.docx (FR-4.3). Tasks: Implement API Endpoints for Messages (under /api/v1/transactions/:id/messages): GET /transactions/:id/messages: Retrieve flat message feed for a transaction, sorted by created\_at descending, with sender details. Implement pagination.

POST /transactions/:id/messages: Post a new message. Validate input (text, optional replyTo). Enforce permissions: Negotiators can create top-level posts; other roles can only reply. A top-level post will have replyTo as null. A reply must have a valid replyTo message ID that belongs to the same transaction\_id.

Store message with sender\_id, transaction\_id, and reply\_to if applicable.

Phase 5: Backend - Document Request API

Primary Documents: docs/ReAlign MVP - API Routes.docx (Doc Requests section), docs/ReAlign MVP - Database Schema.docx (document\_requests), docs/ReAlign MVP - PRD.docx (FR-4.5). Tasks: Implement API Endpoints for Document Requests: POST /transactions/:id/doc-requests (under /api/v1/transactions/:id/doc-requests): Create a document request (negotiator only). docType must be from a hardcoded preset list. This preset list is defined in ReAlign MVP - Scaffold.docx as REQUEST\_TEMPLATES. The backend should have access to a similar list for validation. The requested\_by field should be the ID of the authenticated negotiator. Validate with Zod schema (e.g. CreateDocumentRequestSchema ).

GET /transactions/:id/doc-requests (under /api/v1/transactions/:id/doc-requests): Retrieve all document requests for a transaction. Implement pagination.

PATCH /doc-requests/:requestId (under /api/v1/doc-requests/:requestId): Update a document request (e.g., status, add revisionNote) (negotiator only). Validate with Zod schema (e.g. UpdateDocumentRequestSchema ).

Phase 6: Backend - File Upload API

Primary Documents: docs/ReAlign MVP - API Routes.docx (Uploads section), docs/ReAlign MVP - Database Schema.docx (uploads), docs/ReAlign MVP – Technical Architecture Overview.docx (File Handling), docs/ReAlign MVP – Security & Privacy Plan.docx (Document & File Security), docs/ReAlign MVP - PRD.docx (FR-4.4). Technology: Supabase Storage. Tasks: Implement logic for generating signed URLs for uploads to Supabase Storage. This will likely be a separate internal endpoint called by POST /uploads/:transactionId or integrated within its logic. Implement API Endpoints for Uploads (under /api/v1/uploads): POST /uploads/:transactionId: Handle file uploads (expects multipart/form-data). Request should include docType and visibility. Validate with Zod schema (e.g. FileUploadMetadataSchema ).

Backend enforces max file size (10MB post-compression). Since compression happens client-side, the backend should primarily check the Content-Length header or the size of the received file parts against the 10MB limit.

Upload file to Supabase Storage using a signed URL. Store file metadata (including file\_url from Supabase, uploaded\_by (authenticated user's ID), doc\_type, visibility, file\_name, content\_type, size\_bytes) in the uploads table. If the frontend sends a document\_request\_id in the multipart form data, store this in the uploads.document\_request\_id field.

GET /uploads/:transactionId: List file uploads for a transaction, filtered by user role and upload visibility (negotiators see all; others see their own private and all shared). Implement pagination.

Phase 7: Backend - Unified Notification Service Foundation

Primary Documents: docs/ReAlign MVP – Notification Rules.docx, docs/ReAlign MVP – Technical Architecture Overview.docx (Push Notification Handling). Tasks: Design and implement the "Unified Notification Service" module in the backend. This service will be responsible for: Identifying recipients and triggering events. Checking notification preferences (e.g., SMS opt-in from users.phone and an assumed opt-in flag if not directly in users table). Constructing message content for different channels. Implement backend triggers for key notifications as per Notification Rules.docx. For each notification type: Recipient Identification: Determine the user\_id(s) of recipients based on the event (e.g., for 'Transaction Invitation', recipients are all parties listed in the POST /transactions request; for 'Document Request Sent', the recipient is the assignedToUserId ).

Preference Check: Fetch user details (email, phone for SMS opt-in status) from the users table. Content Construction: Use the exact message templates from Notification Rules.docx (e.g., "[Negotiator Name] requested your [Document Type]..." ), replacing placeholders like [Property Address], [MagicLinkURL], [Negotiator Name], [Document Type] with actual data. MagicLinkURL generation comes from Supabase Auth. Dispatch Logic (MVP): Email: For system emails like magic links or invitations, leverage Supabase Auth's email sending. For other notifications, if Supabase's email templates are not flexible enough, use a simple email library (like Nodemailer with an SMTP service – Replit secrets needed for credentials) or log the email content if direct sending is deferred. SMS/Push: Log the intent to send, the recipient, and the fully constructed message. The actual SDK integration for a service like Twilio (for SMS) or FCM (for push) can be a follow-up, but the service method should be structured to call these if they were present. Start with: Transaction Invitation (Email, SMS if enabled). Magic Link Delivery (Email/SMS). Document Request Sent (Email, SMS if enabled, In-App placeholder, Push placeholder). Document Request Revision Needed (Email, SMS if enabled, In-App placeholder, Push placeholder). Phase 8: Frontend - Core Setup & Shared UI Components

Primary Documents: docs/ReAlign MVP - Component Specification.docx, docs/ReAlign MVP – Brand Kit.docx, docs/ReAlign MVP – Technical Architecture Overview.docx (Frontend Architecture), docs/ReAlign MVP - Scaffold.docx (for TransactionView.tsx structure ). Technology: React 18.x + Vite 5.x, TailwindCSS 3.x, shadcn/ui, lucide-react, framer-motion. Tasks: Set up client-side routing (e.g., react-router-dom or the specified Vite Router if available and suitable) with lazy-loaded screens.

Implement global state management using React Context for session data and user role. Local state with useState/useReducer.

Create the useRoleAccess() hook for conditional UI rendering based on user role.

Implement each shared UI component defined in docs/ReAlign MVP - Component Specification.docx. Pay close attention to props, behavior, accessibility (A11y), mobile-first design, and styling with TailwindCSS and Brand Kit tokens. Examples: PhaseTracker PartyCard MessageThread DocRequestList UploadWidget. Implement client-side image compression here using browser-image-compression. The maxSizeMB: 1 in the scaffold is an example target for compression; the goal is to help files meet the overall 10MB backend limit. The widget must handle onUploadComplete callback, defaultVisibility (private ), docType prop, maxFileSizeMB prop (default 10MB ). Error messages like 'Upload failed. File too large. Please try again.' and a 'Retry' button must be implemented within this component.

When implementing components from shadcn/ui: Use the npx shadcn-ui@latest add [component-name] command within the frontend/ directory if Replit's environment supports it. Otherwise, manually copy the component code from the shadcn/ui website. Ensure these components correctly pick up the Tailwind theme defined in tailwind.config.js (colors, fonts, border radius from Brand Kit ). These components will be added to your frontend/src/components/ui directory. Ensure all components are responsive and meet tap target requirements (minimum 44x44px ).

Phase 9: Frontend - Authentication & Onboarding Screens

Primary Documents: docs/ReAlign MVP – User Onboarding Flow.docx, docs/ReAlign MVP - Figma Wireframe Spec.docx (Magic Link Login Screen), docs/ReAlign MVP - API Routes.docx (Auth endpoints). Tasks: Implement the Negotiator Login screen (email/password). Connect to POST /api/v1/auth/login API.

Implement the Magic Link request screen for other roles (email or phone input). Connect to POST /api/v1/auth/magic-link API.

Implement client-side Supabase Auth logic for: Storing JWT securely (Supabase client library handles this with cookies by default).

Handling session expiry and redirecting negotiators to login (token auto-refresh is disabled for MVP). Handling magic link callback verification. Implement UI for handling expired/invalid magic links ("This link has expired or is invalid. Please request a new one." ) and the resend magic link flow (connect to POST /api/v1/auth/magic-link/resend ). Show toast for max resends exceeded ('You have exceeded the maximum number of resend attempts. Please try again later.' ). A global toast notification system (e.g., from shadcn/ui Toaster or a simple context-based one) should be used.

Protect routes based on authentication status and role using the useRoleAccess() hook and router guards. Phase 10: Frontend - Negotiator Views Implementation

Primary Documents: docs/ReAlign MVP – User Onboarding Flow.docx (Negotiator Flow), docs/ReAlign MVP - Figma Wireframe Spec.docx (Dashboard, Create Transaction, Transaction View - Negotiator), docs/ReAlign MVP - Scaffold.docx. Tasks: Negotiator Dashboard Screen: Fetch and display a list of active transactions associated with the negotiator (connect to GET /api/v1/transactions ).

Include File title, Property address, Current phase, Last activity (Completion % if available). Implement "Start New Transaction" CTA navigating to the creation form. Create New Transaction Screen: Implement form with fields: File title, Property address, Party list (for each party: role, full name, email, phone, SMS opt-in toggle). Editable welcome message (prefilled with a default template, displayed as a multi-line textarea ).

On submit, call POST /api/v1/transactions API. Transaction View Screen (Negotiator Perspective): (Use docs/ReAlign MVP - Scaffold.docx (TransactionView.tsx) as a structural and functional reference). This means adapting the mock API calls in the scaffold to actual Workspace calls to the backend API endpoints, using the correct request bodies and handling responses (success and errors). State updates should reflect API responses. Carefully map the data fetched from GET /api/v1/transactions/:id to the props of the child components (PhaseTracker, PartyCard, DocRequestList, MessageThread, UploadWidget) as per their definitions in Component Spec.md. For example, the DocRequestList expects a requests prop, which will be transactionData.documentRequests from the API.

Fetch full transaction details (connect to GET /api/v1/transactions/:id ). Header: Display title, property address. Phase Dropdown: Allow negotiator to update transaction phase (connect to PATCH /api/v1/transactions/:id ).

Party Tracker: Display PartyCard components for each party. Allow negotiator to manually toggle party statuses (connect to GET /api/v1/transactions/:id/parties and PATCH /api/v1/transactions/:transactionId/parties/:userId ).

Document Requests (DocRequestList ): Allow negotiator to select requests from a preset list (REQUEST\_TEMPLATES from scaffold ) and assign to a party (connect to POST /api/v1/transactions/:id/doc-requests ). The UI will need a way for the negotiator to select which party from transaction\_participants the request is assignedToUserId for.

Allow negotiator to mark requests as 'complete', resend reminders (triggers notification), or reset to 'pending' with an optional revisionNote (connect to PATCH /api/v1/doc-requests/:requestId ).

Messages (MessageThread ): Allow negotiator to start new top-level messages and reply (connect to POST /api/v1/transactions/:id/messages ).

Uploads: Display all files uploaded to the transaction, with their metadata. Allow negotiator to toggle visibility of any file (private/shared). This requires a new backend endpoint (e.g., PATCH /api/v1/uploads/:uploadId/visibility body: { visibility: 'private' | 'shared' }). This endpoint should be negotiator-only or uploader-only. If creating this now is out of scope, this specific UI toggle for existing files should be omitted for MVP, and visibility set only at upload time. Implement file uploads using the UploadWidget component (connects to POST /api/v1/uploads/:transactionId ).

Phase 11: Frontend - Party Role Views Implementation

Primary Documents: docs/ReAlign MVP – User Onboarding Flow.docx (Homeowner & Party Flows), docs/ReAlign MVP - Figma Wireframe Spec.docx (Party Role View). Tasks: Transaction View Screen (Party Perspective): (This view is reached directly after magic link authentication ). Fetch transaction details relevant to the party (connect to GET /api/v1/transactions/:id ). Header: Display transaction title, property address, current phase (read-only ), list of parties (names and roles only, no contact details of others ). This data comes from the parties array within the response of GET /api/v1/transactions/:id.

Phase Tracker UI (PhaseTracker ): Collapsed by default (“You’re currently in Phase X: [Name]” ), expandable to show 9 standard phases with current phase highlighted.

"What’s Still Needed" Section (DocRequestList ): Display document requests relevant to the transaction, potentially grouped by party, highlighting those assigned to the current user.

If a request assigned to the user is 'pending' due to a negotiator revision, and a revisionNote was added, display the note clearly below the request item (e.g., 'Missing signature on page 2').

Allow users to fulfill their assigned requests by triggering the UploadWidget. Messages (MessageThread ): Allow users to view the transaction message thread and reply to existing messages (cannot start new top-level threads). Uploads (UploadWidget ): Allow users to upload documents relevant to their role or a specific request. Uploads default to 'private' visibility; user can toggle to 'shared' at time of upload. Ensure the UploadWidget component's UI allows this selection and passes the chosen visibility to the handleFileUpload function, which then sends it to the POST /api/v1/uploads/:transactionId API.

Ensure client-side image compression and error/retry handling ('Upload failed. Please try again.', 'Retry' button ) are implemented as per UploadWidget spec and Figma. Display info subtext 'Uploads are private to you and the negotiator by default.' near the upload area.

Phase 12: Frontend - Client-Side Push Notification Setup

Primary Documents: docs/ReAlign MVP – Technical Architecture Overview.docx (Frontend Push Notification Integration), docs/ReAlign MVP – Notification Rules.docx (Device Push Notification channel). Tasks: Implement logic to request user permission for push notifications via browser/OS prompts.

Upon successful permission grant, obtain the device registration token. Send this token to a new backend endpoint (e.g., POST /api/v1/users/device-token) to associate it with the authenticated user. The backend will need to create this endpoint. It should expect a body like { token: string, type: 'fcm' | 'apn' | 'web' } and store this against the authenticated user\_id. Consider a new table user\_device\_tokens (id UUID PK, user\_id UUID FK users, device\_token TEXT UNIQUE, token\_type TEXT, created\_at TIMESTAMP) or adding fields to the users table. Phase 13: Comprehensive Testing & Refinement

Primary Document: docs/ReAlign MVP - QA Test Plan.docx Tasks: Systematically go through all "Core Functional Test Areas" and "Security & Privacy Tests" outlined in the QA Test Plan. Perform manual testing across specified browsers/platforms (iOS Safari, Android Chrome, Desktop Chrome, Firefox, Edge). Test keyboard navigation and screen reader compatibility (VoiceOver, TalkBack). Identify, log, and fix bugs. Refine UI/UX based on test results and ensure adherence to all specifications.