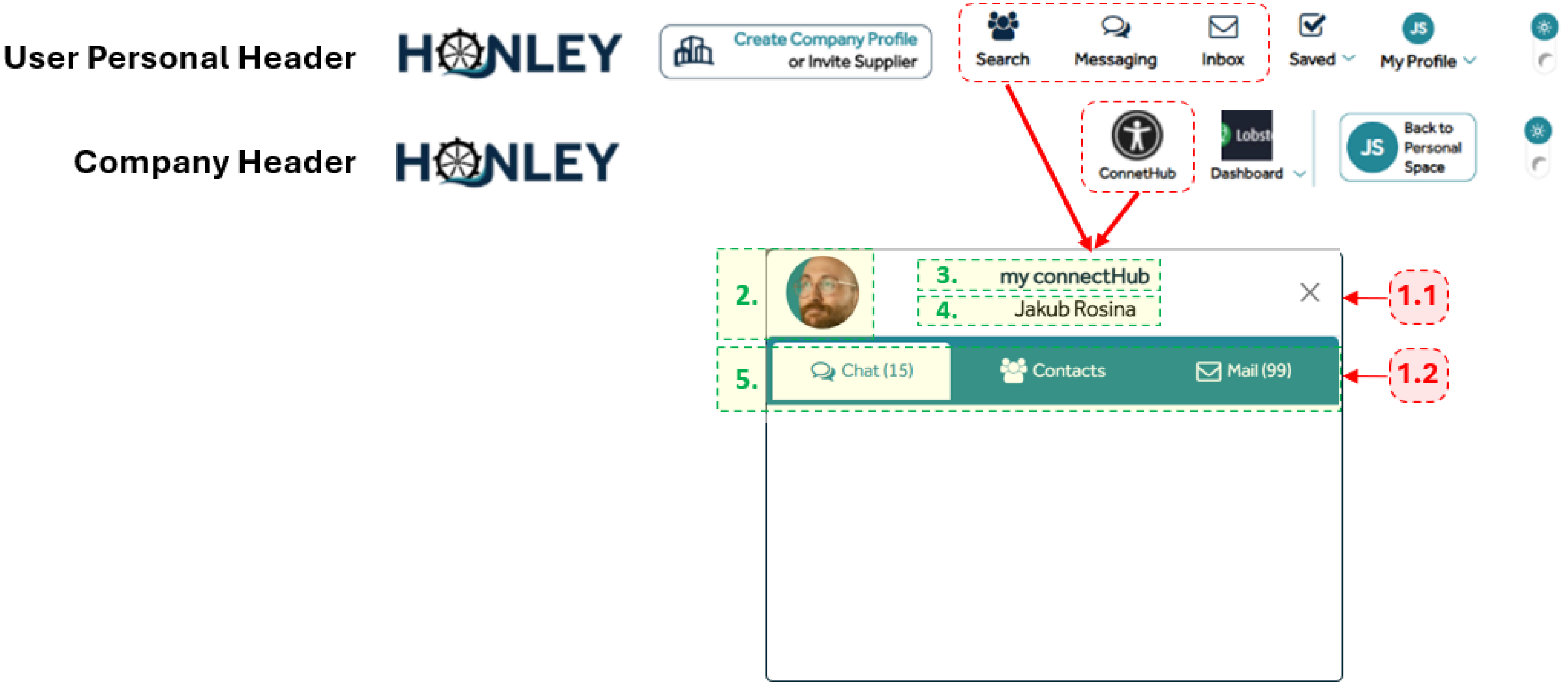
Rev	Rev Title	Description	Document [date]	Originator	Approved
0.9	**Initial Draft**		30-May-25		
0.91	**Update**	Added Contacts section 3.	04-Jun-25		
0.92	**Update**	Notifications Section	5-Jun-25		
0.93	**Update**	Added ‘Notification Message Interface’ 4.3			
1.0	**Final ready for review**	Final version updated in line with ConnectHub logic, ready for review by developer	9-Jun-25		

Contents

1. ConnectHub

1. ConnectHub Header



ID	Element	UI Display Logic	Click / Interaction Behaviour	Edge Cases & Notes	Notes

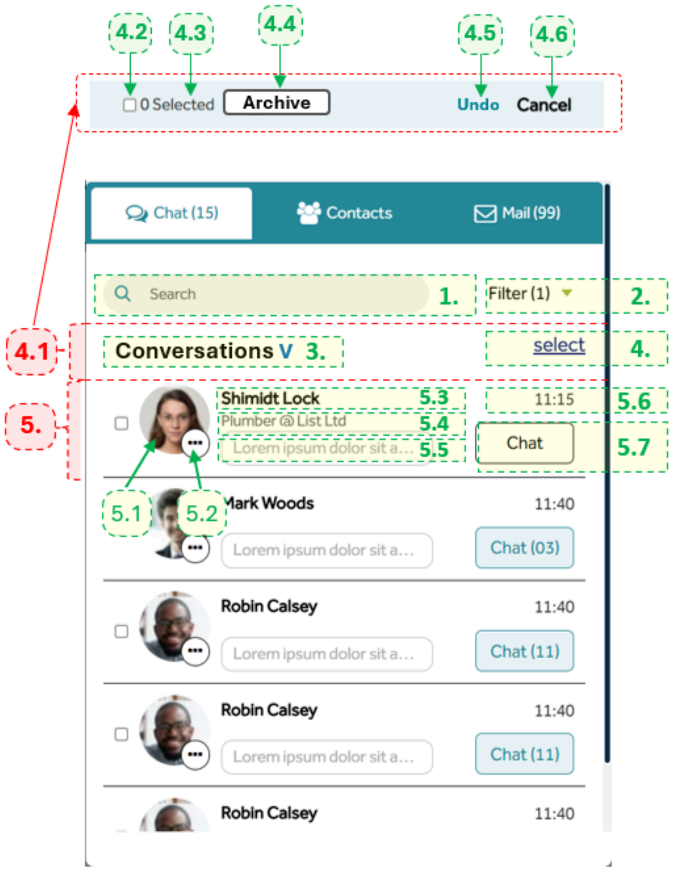
1.1	**Colour Style – Header**	Header background colour is set dynamically based on context: • If opened from **user personal header** → white background. • If opened from **company dashboard header** → dark blue background.	No direct interaction. Styling is applied during modal mount.	Determine source context using `is_company_mode` flag or header state. Ensure header colour change doesn't override internal tab colours or border. Prevent flicker by applying style before content renders.	
1.2	**Colour Style – Tabs**	Tab colours are dynamic: • **User header mode**: - Unselected tabs: green background, white text - Selected tab: white background, green text • **Company header mode**: - Unselected tabs: dark blue background, white text - Selected tab: white background, dark blue text	On tab click: – Set `activeTab` in UI state. – Apply corresponding colour class. – Load tab content via conditional rendering.	Ensure proper contrast ratios (WCAG) for both themes. Avoid reapplying tab styles if user switches back and forth. Classes must update reliably with tab switch event. Match colours with platform design tokens for maintainability.	
2	**Profile Photo**	Left-side avatar: • If in personal mode → user profile image (circle) • If in company mode → company logo (square).	No interaction. Image loads on modal open based on context metadata.	Use fallback if image fails. Detect source type to decide crop mask (circle vs square). Ensure consistent sizing between personal/company views.	
3	**Title**	Static title at top-center of modal: • Personal mode: **"my connectHub"•** Company mode: **"Company connectHub"**	No interaction. Title is rendered statically once modal is initialised.	Use `is_company_mode` boolean to determine title. Fallback to generic "ConnectHub" if mode cannot be determined.	
4	**User's Name**	Printed below or next to profile image. Fetched from current user session: `user.first_name + user.last name`.	Static display only.	If name is missing → suppress and log issue. Prevent text overflow with ellipsis.	

		Applies to both modes.			
5	**Tabs Section**	Tabs: **Chat**, **Contacts**, **Inbox**. Label correction: “Mail” → “Inbox”. • If personal mode → always show all 3 tabs. • If company mode → show only tabs where user has access permission (chat/inbox). If either is shown, also show Contacts.	Clicking a tab updates `activeTab` and renders the corresponding component. Tab titles show dynamic unread counts: – If `unread_count = 0` → show “Chat” or “Inbox” – If > 0 → show “Chat (XX)” or “Inbox (XX)”	Use `/api/chat/unread-count` and `/api/inbox/unread-count` to update labels. If user lacks permissions in company mode, suppress respective tabs. Always ensure at least Contacts tab is visible if either Chat or Inbox is. Reset activeTab to first available tab if denied access to previously active one.	

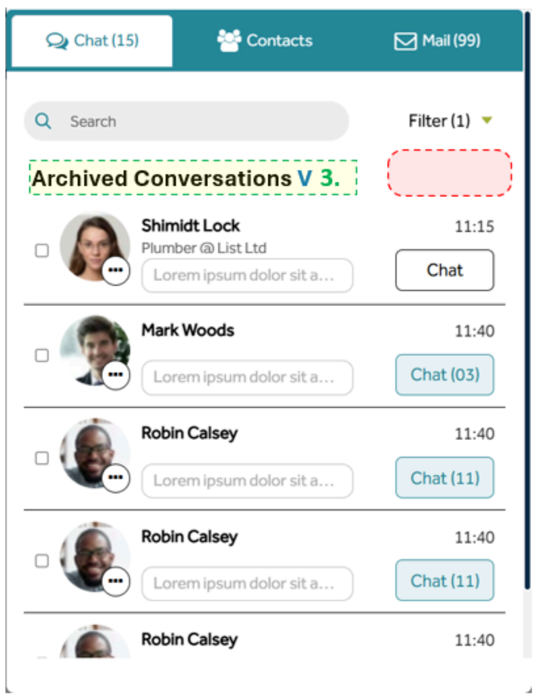
2. Chat Interface

2.1 Chat Menu

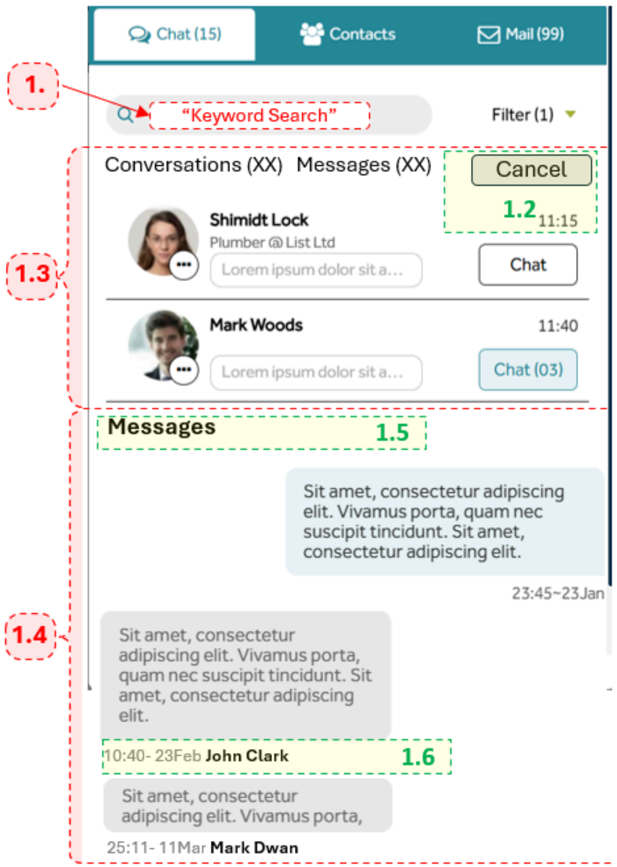
Live Conversations



Archived Conversations



Chat - Search



Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec suscipit tincidunt. Sit amet, consectetur adipiscing elit.

Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec suscipit tincidunt. Sit amet, consectetur adipiscing elit.

Sit amet, consectetur adipiscing elit. Vivamus porta,

UI Sang**:** Update **Element 3** (currently the "Chat" title) to a dropdown toggle with two options: Conversations and Archived.
Also, replace Element 4.4 (bin icon) with a label-style [Archive] button for clarity and consistency with the conversation action (Conversation cannot be deleted).

ID	Element	UI Display Logic	Click / Interaction Behaviour	Edge Cases / Notes	Notes
1	**Search Bar**	Always visible input field. Styled as single-line text. Accepts	On `input` event (with 300ms debounce), if input is valid, initiate keyword search. Do **not** trigger search for input	If network fails → show toast "Could not complete	

		alphanumeric keyword input. Minimum 2 characters to trigger. No button; triggers automatically.	length <2. API call: `GET /api/chat/search?query=KEYWORD`. Backend searches both `chat_threads` and `chat_messages` tables. System enters search mode and hides standard thread list.	search". If input cleared → cancel search. If no matches → show "No chats or messages matched your keyword".	
1.1	**Results count**	Two headers shown only after search: • `Chats (X)` – threads matched • `Messages (Y)` – messages matched. Data injected from backend response.	No click interaction. Display-only. Results count injected on API response processing. Section only visible if count > 0.	If count is 0 for both → do not render result section. System should also suppress message section title (1.5).	
1.2	**[Cancel] button - Results**	Visible only after search results appear. Static button labelled “Cancel”. Located above results container.	On click: • Clear search input. • Reset `searchMode = false`. • Hide search results view. • Re-fetch and display full chat thread list via: `GET /api/chat/threads`.	Must always be accessible when in search result state. If cancel is clicked during another search request, abort previous request if possible.	
1.3	**Chat Thread Cards Section**	Chat thread cards (Element 5 UI reused). Displayed for all threads where keyword is found either in: • Contact name • Company name • Last message body	Each card: • [Chat] button opens full chat interface modal. • […] Action dropdown options identical as explained in element 5.2. • Cards must appear before 1.4 (message matches).	Avoid duplication: if same thread appears both in summary match and in message match section, show it only once in 1.3. If card has been deleted since result fetch → disable button, show "Chat no longer exists".	
1.4	**Individual Messages - Section**	Each matching message is shown in **chat bubble format**, grouped by thread_id. • Messages from the current user (owner of the interface): right-aligned.	• Each message is clickable. • On click, the system opens the **Chat Interface modal** for the corresponding thread_id and scrolls to the **specific message** using its message_id anchor or offset. • Use message_id in URL fragment or Vue router param for precision.	• If the thread_id is archived or deleted, disable click and mark message as from “Archived Thread”. • If same message appears	

		• Messages from other users: left-aligned, **sender's full name** and **timestamp**. • **Styling and layout must match** the main Chat Interface exactly (font, padding, avatar spacing).		in multiple threads or nested replies, ensure it is rendered only once. • Handle cases where sender profile is deleted → label as "Unknown User".	
1.5	**"Messages" Title**	Appears as static section header "Messages" **only if** one or more keyword matches exist in individual messages (1.4).	None (label only). Injected dynamically if `messages.length > 0`. Hide if zero.	Ensure spacing is consistent when hiding or showing this header. If rendered, must appear after summary card section and before any messages.	
1.6	**Timestamp & User ID**	For messages from current user: show timestamp only (e.g. "10:40 – 23Feb").For others: show full name + timestamp (e.g. "10:40 – 23FebJohn Clark"). Format: 24h, 3-letter month, space-separated.	Static text element. Rendered inside each message bubble depending on direction and ownership. Laravel should return formatted timestamp + sender_name if required.	If timestamp missing or invalid → suppress rendering and log error. Handle timezone correctly – backend must localize to user if applicable.	
2	**Filter Toggle**	Dropdown menu icon next to search bar. Hidden if there are no chat threads. When clicked, opens a vertical dropdown with toggle checkboxes: (a) Show only Unread (b) User Chats (c) Business Chats. Filters should persist until cleared. Active filter badge ("Filter (1)") updates dynamically.	On change of filter: send request to endpoint: `GET /api/chat/threads?filter[unread]=true&filter[type]=user` (example). Re-render the chat thread list with filtered results only. If user unchecks all → reset view to full list. Maintain separate state for applied filters (for clearing/reapplying).	If no threads match the selected filter: show "No chats found". If WebSocket delivers new message that would match a previously filtered-out category, system should notify visually but not refresh filtered list unless toggled.	
3	Conversations / Archived	Display as a **dropdown toggle** with two options: `Conversations` and	On click, open dropdown. - Select `Conversations` → load threads where `is deleted = 0`	If a thread is archived while viewing `Conversations`, immediately remove it from	

		`Archived Conversations`. Title reflects selected view.	- Select `Archived Conversations` → load threads where `is_deleted = 1`	the list. If unarchived while in `Archived`, remove from that list.	
ID	**Element**	**UI Display Logic**	**Click / Interaction Behaviour**	**Edge Cases / Notes**	
4	[Select]	Button shown in top-right of chat list. **Visible only in `Conversations` view (is_deleted = 0)**. Hidden in `Archived Conversations`.	On click: 1. Set `selectionMode = true` 2. Show checkboxes beside each thread 3. Render Select Bar (`4.1`) in fixed position above list 4. Clear previous selection 5. Retain scroll position 6. Optionally set `aria-multiselectable = true`	Ensure select bar appears only in live conversation view. Do not show select bar or checkboxes in Archived view since bulk operations (e.g., unarchive/delete) are not allowed.	
4.1	**Select Bar (Bulk Action Bar)**	Horizontal action bar fixed above thread list. OPENs when user clicks [select] button (element 4) Includes select count (4.3), delete (4.4), undo (4.5), and cancel (4.6). Renders only when in selection mode.	Automatically appears when `selectionMode == true`. Position: sticky container, anchored to top of thread list view. System updates its content based on how many threads are selected (see 4.3). Button states update reactively.	Ensure this bar does not interfere with sticky header tabs (Chat, Contacts, Mail). Positioning must not shift UI content or overlap filter/search.	
4.2	**☐ “Select All checkbox”**	Checkbox shown inside Select Bar (left-most). When checked, selects **all visible** threads (filtered or full). Also sets internal array of selected thread IDs.	On check: loop through current list of threads and set selection state to `true`. Update select count (4.3) accordingly. On uncheck: clear all selections. Developer must bind checkbox state to master list loop with unique thread IDs.	If thread list is empty → disable this checkbox. If partial selection manually modified after “Select All”, checkbox should reflect indeterminate state (visual tick with dash line).	
4.3	**Count and Action Message**	Dynamic text label inside Select Bar: `(X) Selected`, `(X) Deleted`, `(X) Reinstated`. Content depends on last performed action (select/delete/undo).	Automatically updates on: • Checkbox change (count) • Delete API success (status = Deleted) • Undo (status = Reinstated). System must keep internal `selectionStatus` flag (e.g. 'active'		

4.4	**Archive** button	Label-style button in Select Bar (replaces trash icon). Disabled unless ≥1 chat threads selected.	On click: • If no items selected → button disabled • If selected → modal: “Archive X chat threads?” On confirm → call: `PATCH /api/chat/threads/archive` with list of `thread_ids`. Set `is_deleted = 1` for each thread. Update visible list and show Undo (4.5).	If user cancels modal → no action. If server fails → show toast “Could not archive chats. Try again.” Prevent repeated clicks by disabling post-action.	
4.5	**[Undo] button**	Appears only after a successful delete operation. Hidden by default. Active for limited window (e.g., 10 seconds) or until another user action occurs.	On click: • Call `POST /api/chat/threads/restore` with `thread_ids[]` • Update status message to `(X) Reinstated` • Re-render thread list with restored entries • Hide Undo button afterwards.	If Undo clicked after timeout → show toast “Undo expired.” If partial restore fails → show which threads were not reinstated. Reset system state after action completes.	
4.6	**[Cancel] button**	Always visible in Select Bar (far right). Closes selection mode. • Hides checkboxes • Hides Select Bar • Clears selected items and resets state	On click: • Set `selectionMode = false` • Clear selected thread ID list• Hide Select Bar • Restore default view (all items clickable, no checkbox rendered) • Reset scroll if needed	Must clear any modified visual states (e.g., highlight, counter, button enablement). Restore full click access to chat cards and remove `aria-multiselectable` roles. Ensure no data is persisted across tab reload.	
5	**Chat thread Card**	Container showing one chat. Includes: ● profile image (5.1), ● name (5.3), ● optional position/company (5.4), ● message snippet (5.5), ● timestamp (5.6),	Each card is self-contained. Clicking anywhere except […] dropdown or Image opens the Chat Interface modal. (Clicking on the image opens user profile modal (`/profile/:id`) or new tab for company profile) All data fields (names, message, unread count) must come from `/api/chat/threads`. State updates via WebSocket when new message arrives. Unread status affects styling of [Chat] button 5.7.	• A chat thread should be created and stored **at the moment the first message is sent between two users (or user & business)**. • If a thread already exists (matched by participants), reuse the existing thread ID.	

		● action dropdown (5.2), and ● Chat button (5.7). Entire card height must align with surrounding threads.		• If no messages are sent (only typing or viewing), thread is not created. • On load failure → render as "Unknown contact" with default avatar. • On missing thread ID, disable interaction. Blocked or deleted → grey out entire card and disable buttons.	
5.1	**Profile image**	Left-aligned. ● Use user profile image if chat with user (Circular mask for users), or ● Use company logo if chat with company (square crop for companies). Avatar pulled from `users.avatar_url` or `companies.logo_url`.	When clicked on, opens user profile modal (`/profile/:id`) or new tab for company profile.Consider using lazy loading for performance.	Handle `onError` to swap to placeholder image. If user has been deleted → show grey silhouette. Must not trigger interaction when clicked.	
5.2	**[…] Action Dropdown**	Three-dot dropdown on each chat card. Rendered only if user has permission. Options vary by context (user vs company): • View Profile • Chat • **Archive** • Report • Block / Unblock	- **View Profile**: Opens user modal or company public profile. - **Chat**: Loads chat modal and latest messages. - **Archive**: 1. System prompts: *"Are you sure you want to archive this conversation? It will be moved to Archived Conversations. You can restore it anytime by continuing the chat."* 2. On confirmation:	- Blocked users cannot trigger message delivery (chat or notification); enforced at API + Pusher. - Dropdown must reflect live block state: call `/api/chat/block-status/:userId`. - If thread is archived, **block/report/archive** should be disabled.	

			- A record is added to user_chat_deletions with the user ID and conversation ID. - The conversation disappears from the Live view and is loaded in the **Archived Conversations** section. - **Block / Unblock**: On dropdown open, system must **check current block state** in the **entity_blocks** table (via /api/chat/block-status/:userId) and update the button label dynamically to Block or Unblock. Button must always reflect the actual backend state — do not rely on cached UI toggle Follows unified blocking logic across chat + notifications (see *ConnectHUB System Overview – Section 5*). - **Report**: On click, system Opens 'Contact Honley' modal &: **1.** System autofills: – → **Query Category** = "Flag Issue" (readonly) – → **Subject** = "Chat: [Name] [dd/mm/yyyy HH:mm]" (readonly, based on last message) **2.** User types message (textarea input). **3.** On send: → Submit notification to admin inbox via `POST /api/admin/notifications/report-chat` → System retrieves **up to** 10 messages from the reported chat thread (in descending order). **GET/api/chat/messages/:threadId?limit=10&order=desc**	- Report cooldown should prevent spamming. - If report API fails, show fallback message and keep user input intact.	

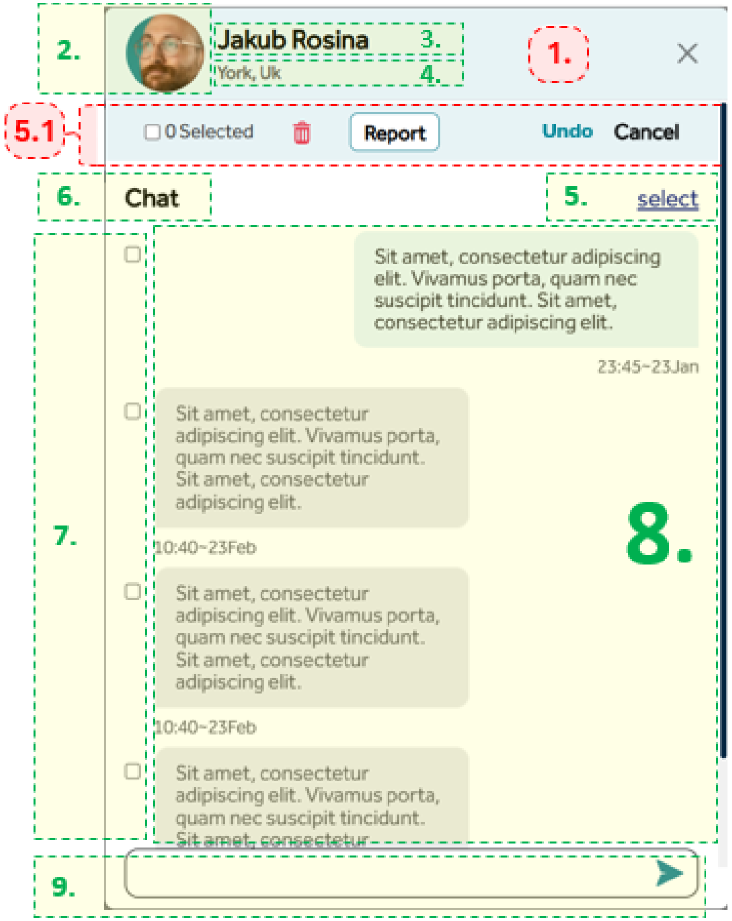
			→ Format these messages with: [HH:mm DD-MMM] Firstname Lastname on one line, followed by the message content on the next. → Attach last 10 messages from thread (`GET /api/chat/messages/:threadId?limit=10&order=desc`) in system message or internal note. → Append this list under the user’s report message, separated by a clear section divider (e.g. "----- Context: Last 10 Messages -----"). → Include all this content in the message body. **Example:** [User’s Report Message] ----- Context: Last 10 Messages ----- [10:23 15-Jun] John Smith Hi how are you! [15:23 17-Jun] Sarah Malik What’s up? – Show confirmation: “Your report has been submitted to the Honley Admin Team.”		
5.3	**Name**	Bolded name of the person or company: • IF Users: full name from `users.first_name + last_name`	No direct interaction. Content rendered statically with text truncation as needed. Entire name area can be included in the click target to open the chat modal.	If no name found → show "Unknown User" or "Unknown Company". If chat was created by a deleted profile → flag visually.	

		• IF Companies: `companies.company_name`. Must truncate with ellipsis if overflow.			
5.4	**[Position] @ [Company]**	Shown below name if user is linked to company. Format: `"Job Title @ Company Name"`. Use `user_companies.position` and `companies.company_name`. **Not shown for user-to-company chats!**	No interaction. Display-only. Data must be loaded as part of user metadata when rendering thread list.	If data missing → hide element. If position exists but company name does not, show position only. Ensure text wraps or truncates appropriately for small screens.	
5.5	**Chat Snippet**	Preview of most recent message in thread. Show max XX characters. Text must be sanitized (no HTML) and truncated with "…". Font slightly lighter than name.	No click. On hover, show full message in tooltip (optional). On new message arrival via WebSocket, replace snippet. Make sure newline characters are stripped or replaced by spaces.	If message is deleted, show "🗑️ Message deleted!". If user's message included emoji or markup, strip to plain text. On DB error → show empty string.	
5.6	**Timestamp**	Right-aligned small label. Display time of last message. Format: • Today: "14:23" • This week: "Mon" • Older: "02 Apr 2025". Auto-updates on new messages.	Display-only. Format based on message timestamp (`thread.last_message_at`). Backend should pre-format or provide raw datetime; frontend determines rendering format using user's locale.	If no timestamp → suppress element. If user's local time cannot be resolved → fallback to UTC. Watch for misalignment if clock skew across client/server.	
5.7	**[Chat] button**	Always shown at far right. Two visual styles: • Seen all messages → white background, black text: "Chat" • Has unread → green background, green text: "Chat (XX)", where XX = unread count. Styling updated live on new message / chat open.	On click: • Open Chat Interface modal `/chat/:threadId` • Call `PATCH /api/chat/threads/:id/mark-read` immediately after open to mark messages as seen. • Update button state to "Chat" and unread count to 0 • Emit WebSocket event if necessary to confirm read status with sender. Use distinct classes for both visual states to allow CSS transitions.	If thread is opened from elsewhere (e.g., from search), button state must still update globally. Ensure state is cached in client state to prevent visual desyncs.	

				If message arrives while modal is open, suppress counter increment.	

2.2 Chat Interface

Chat Interface



Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec suscipit tincidunt. Sit amet, consectetur adipiscing elit.

23:45-23Jan

Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec suscipit tincidunt. Sit amet, consectetur adipiscing elit.

10:40-23Feb

Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec suscipit tincidunt. Sit amet, consectetur adipiscing elit.

10:40-23Feb

Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec suscipit tincidunt. Sit amet, consectetur

ID	Element	UI Display Logic	Click / Interaction Behaviour	Edge Cases & Notes	Notes
1	Header Colour	When the chat interface loads, the top header background is: • **Light blue** if the conversation is with a user	No interaction (visual styling only).	Apply condition after thread metadata loads. Use `is_company` flag or `conversation_type`. If thread type	

		• **Dark blue** if the conversation is with a company Colour must apply to the entire top bar.		cannot be determined (null), default to light blue and log warning.	
2	Profile Photo	Left-aligned image. Circular for user, square for company. Pulled from `users.avatar_url` or `companies.logo_url`. Use `img` tag with `object-fit: cover`.	On click: open user profile modal (`/profile/:id`) if user; open new tab for company (`/company/:id`). Add lazy load attribute for performance.	If image fails to load, fallback to placeholder. If profile no longer exists, show generic grey avatar. Image should not trigger chat modal. Use error event to swap broken URLs automatically.	
3	Name	Bolded name of the person or company: • IF Users: full name from `users.first_name + last_name` • IF Companies: `companies.company_name`. Must truncate with ellipsis if overflow.	No direct interaction. Content rendered statically with text truncation as needed.	If no name found → show "Unknown User" or "Unknown Company". If chat was created by a deleted profile → flag visually.	
4	Location (City, Country Code)	Static text below name. Format: "City, CountryCode" (e.g., "York, UK"), sourced from `users.city` and `users.country_code`. **Only shown if the chat is with a user**. Not rendered at all for company chats.	No interaction. Display-only, populated with user metadata.	If `city` or `country_code` is missing/null → show "—". If chat is with a company, this row is entirely suppressed. Truncate city name if overly long. ISO code errors (e.g., invalid 2-letter codes) should not cause render failures—suppress silently.	
5	[select]	Button toggles checkbox mode. When active:• Show checkboxes on all messages (6).• Display Select Bar (5.1).• Visually disable this button while bar is open (covered by bar).	On click:• Set `selectionMode = true`.• Trigger message re-render with checkboxes.• Open Select Bar (anchored position).• Maintain scroll position.• Optionally set `aria-multiselectable=true`.• Should not disrupt open input.	Must persist mode unless cancelled. Prevent interaction overlap with chat entry box. If chat is reloaded while active, reset `selectionMode`.	

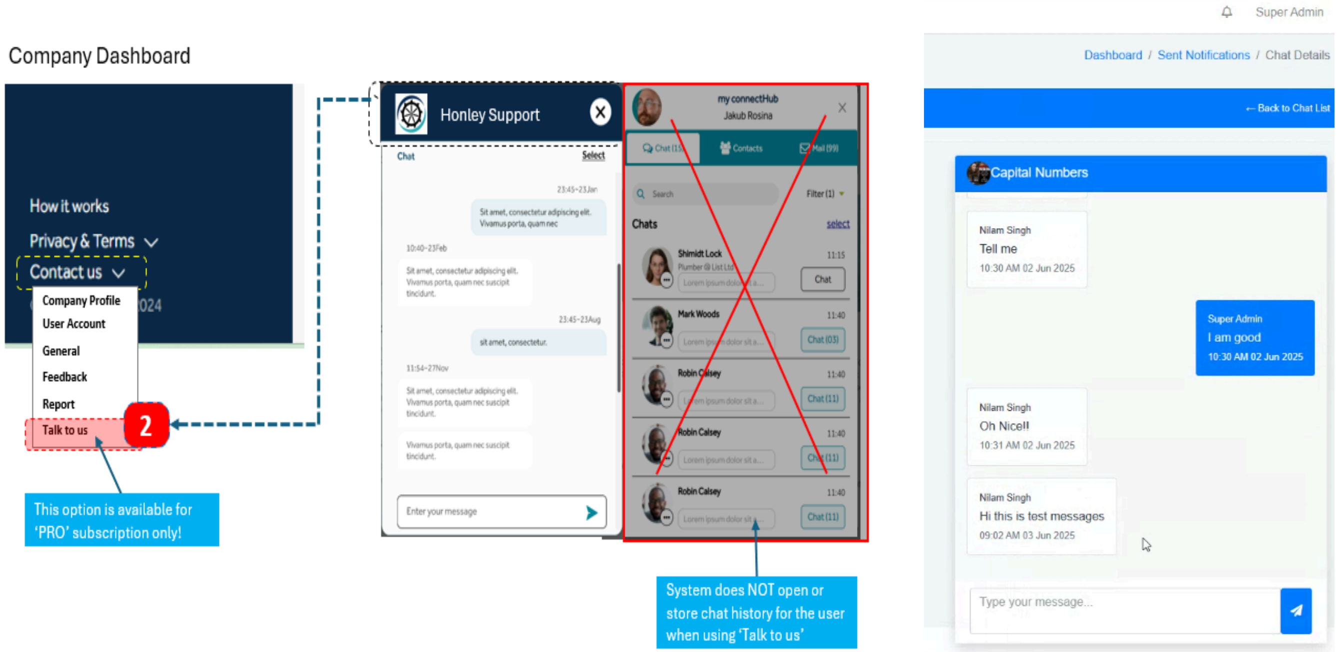
ID	Element	UI Display Logic	Click / Interaction Behaviour	Edge Cases & Notes	
5.1	Select Bar	Appears above message list when in `selectionMode`. Contains:• `Select All` checkbox• Count and status label (e.g. “3 Selected”, “2 Deleted”)• Buttons: [Delete], [Undo], [Cancel], [Report]• Report only shown if ≤10 messages selected	- □ **Select All**: selects all visible messages. - **Count Label**: updates on selection (`X Selected`), deletion (`X Deleted`), undo (`X Reinstated`). - **Delete**: 1. Show confirmation modal: “Are you sure you want to permanently delete your selected messages? This cannot be undone.” 2. On confirm: for **messages sent by the user only**, set `is_deleted = 1`, `deleted_at = NOW()`. 3. UI replaces message content with “Message deleted”. 4. Only message belonging to the user are deleted. - **Undo**: restore user-deleted messages. - **Cancel**: exit selection mode and reset UI state. - **Report**: If ≤10 messages selected, opens 'Contact Honley' modal. - Prefills Category = "Flag Issue" (readonly), - Prefills Subject = “Chat: [Name] [dd/mm/yyyy HH:mm]”. → System retrieves **the selected** messages (in descending order). **GET/api/chat/messages/:threadId?limit=10&order=desc** → Format these messages with: [HH:mm DD-MMM] Firstname Lastname on one line, followed by the message content on the next. → Attach selected messages from thread (`GET /api/chat/messages/:threadId?`	- Messages not sent by the user are never deleted. - Deletion is soft: `is_deleted = 1`, `deleted_at` set. - “Message deleted” replaces original content in UI for deleted messages. - Report is limited to 10 messages. - All actions must respect user ownership of messages. - Deleted messages stay suppressed in UI and in all real-time sync. - Block re-reporting same message twice within a short window (e.g., 30min cooldown).	

			`limit=10&order=desc`) in system message or internal note. → Append this list under the user’s report message, separated by a clear section divider (e.g. "----- Context: Reported Messages -----"). → Include all this content in the message body. **Example:** [User’s Report Message] ----- Context: Last 10 Messages ----- [10:23 15-Jun] John Smith Hi how are you! [15:23 17-Jun] Sarah Malik What’s up? On submit, call `POST /api/admin/notifications/report-chat`.		
6	“Chat” title	Static label “Chat” shown top-left of message thread. Acts as a visual anchor for the message view. Font style should match section headers across the platform.	No interaction. Pure UI label.	None. Ensure correct spacing and alignment in header bar.	
7	Checkboxes	Only shown when `selectionMode = true`. Each message row gets its checkbox aligned left. Checkbox bound to `message id`.	On select: add message ID to `selectedMessages[]`. On unselect: remove. If all checked → set “Select All” checked. Should sync with 5.1.	Must be hidden when not in selection mode. Reset on cancel. Avoid checkboxes persisting between thread switches. Checkbox should not interfere with message copy or hover interaction.	

8	Messages	• Current user messages: right-aligned in green bubble. • Other user messages: left-aligned in grey bubble. • Time format: “HH:mm – dd Mmm” or weekday if this week. • Bubbles must support dynamic updates, message states, and insertion.	● On thread load: use `GET /api/chat/messages/:threadId?after=<cursor>` for paginated scroll. ● New messages received in real time via WebSocket: `chat.thread.{threadId} → chat.new`. ● Auto-scroll unless user is scrolled upward (pause autoscroll). ● Messages marked as seen: - On modal open, call `PATCH /api/chat/messages/seen` - Emit `chat.read` WebSocket event ● Typing feedback: - On input start, emit `chat.typing` (debounced) - UI shows “User is typing…” for 3s without renewal ● Delivery indicator (✓✓) shows when recipient receives message.	• Messages from blocked users must not render live (filter at socket receive level). • Add soft deletion logic (e.g., `deleted_at`) – render "🗑️ Message deleted" bubble. • Handle duplicate messages on reconnect using `message_id` check. • Typing indicator clears if no activity in 3 seconds or message is sent. • Message bubbles should render "sent", "delivered", and "seen" states visually (✓, ✓✓, ✓✓✓ if extended). • Missed socket events (user offline) must sync from DB on reconnect. • Consider mobile UI scroll retention when history expands.	
9	Input Window	Text box with arrow button. Expands up to 3 lines. Shows placeholder: “Type a message…”. ● Disabled if: − Chat is **blocked**, **archived**, or **disconnected** − **The other party (user or company) has status = 'closed'** ● **If either side is closed**:	● On key input: validate and debounce for typing indicator → emit `chat.typing` to channel `chat.thread.{threadId}`. ● On Enter or Send: 1. Validate: non-empty, max 1000 chars, no HTML. 2. Call `POST /api/chat/messages` with `{ thread_id, message }`. 3. Optimistically insert bubble with state = “sending…” 4. On server response:	• XSS prevention: strip all tags from input. • Rate limit: throttle repeated messages (configurable, e.g. `max 3/sec`) • If WebSocket disconnected, show disabled state with tooltip: “Reconnecting…” • Input blocked if recipient has blocked sender (backend confirms before send). • On failed send (network/server), offer inline retry icon. • Track unsent messages to retry on reconnection.	

		– **Disable input field** and **Send button** – **Pre-fill input field** (read-only) with: “This contact is no longer active on the platform!”	– Replace optimistic entry with confirmed message ID – Emit `chat.new` over socket 5. Scroll to bottom. - Save draft locally if user changes thread. - Disable send if offline or rate limit exceeded (e.g. 1 msg/sec).	• Typing event suppression if modal not focused. • Draft should be stored per-thread in memory.	

2.3 User to Honley-Admin Chat support



23:45~23Jan
Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec

10:40~23Feb
Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec suscipit tincidunt.

23:45~23Aug
sit amet, consectetur.

11:54~27Nov
Sit amet, consectetur adipiscing elit. Vivamus porta, quam nec suscipit tincidunt.

Vivamus porta, quam nec suscipit tincidunt.

Nilam Singh
Tell me
10:30 AM 02 Jun 2025

Super Admin
I am good
10:30 AM 02 Jun 2025

Nilam Singh
Oh Nice!!
10:31 AM 02 Jun 2025

Nilam Singh
Hi this is test messages
09:02 AM 03 Jun 2025

This option is available for ‘PRO’ subscription only!

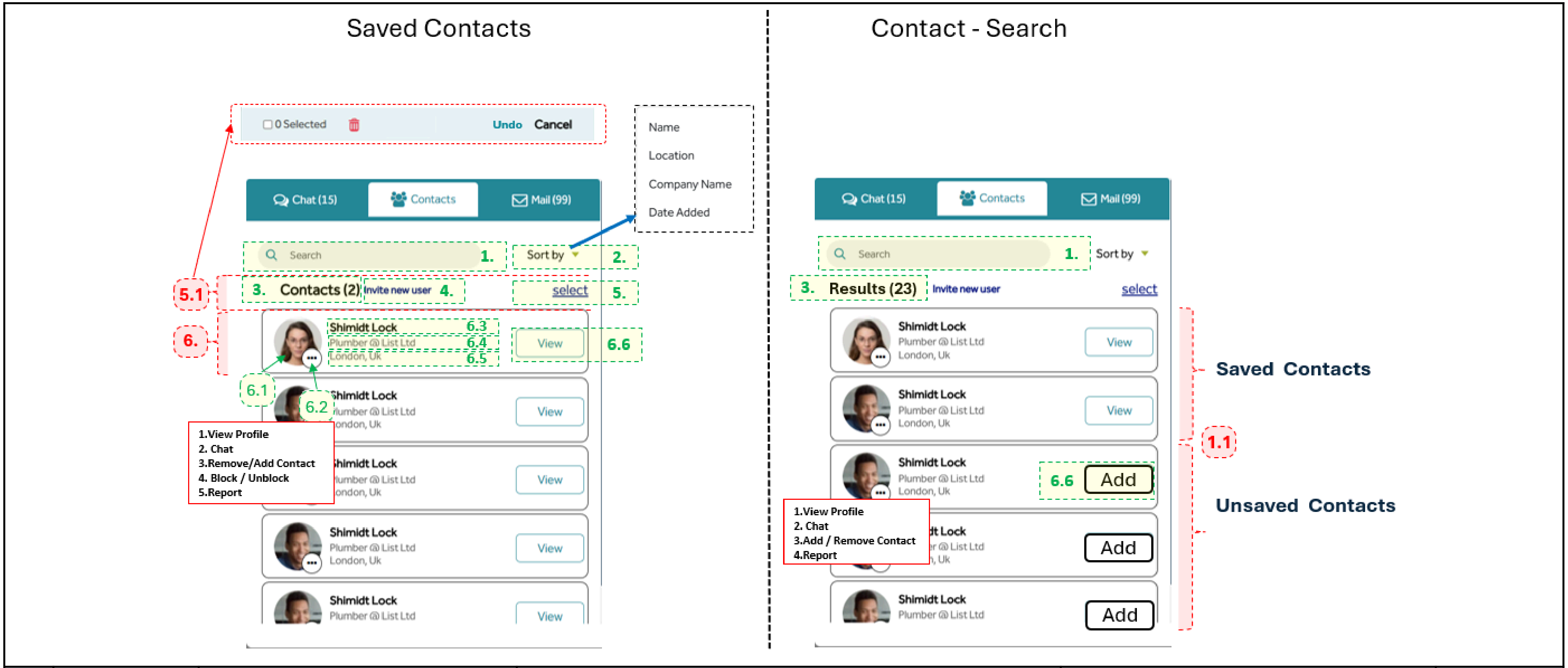
System does NOT open or store chat history for the user when using ‘Talk to us’

ID	User Action	System Response	Edge Case and Implementation Notes	Notes

1	**User clicks** Contact Us → Talk to Us **in dashboard**	Frontend checks subscription level (must be `PRO`). If valid, open chat modal with header `Honley Support` and icon.	Restrict option visibility based on plan. If not allowed, show tooltip "Available for PRO subscription only." **Chat is not stored in user's history and does not appear in user UI later — it is a temporary, session-based support interaction only.**	
2	**User sends message via input + [Send]**	System: – Checks for existing open `admin_conversations` for this user/company where `status != 'resolved'` – If none, create new record: → `user_id`, `assigned_to = NULL`, `status = 'in_progress'`, `created_at = NOW()` – Insert message into `admin_messages` with FK to conversation	Always create new conversation if last is resolved. Ensure message is inserted regardless of previous state.	
3	**Admin receives new message**	Admin interface shows: – Header: Company logo, company name, user name – New conversation visible in queue with `status = 'in_progress'`	Admin sees full history of thread. Admin can assign/reassign, filter by status, or flag.	
4	**Admin replies to user**	Message stored in `admin_messages`. Pusher event pushes message to active user socket.	If user has closed modal, no notification shown — no persistent user UI. Delivery is one-way support flow.	
5	**Admin sets status = `resolved`**	Update the `admin_conversations.status` field to `resolved`.	Important: this ends the thread. If user reopens chat via "Talk to Us," a **new conversation ID** will be created.	
6	**User closes and later restarts chat via "Talk to Us"**	System checks `admin_conversations` for unresolved status. If all are resolved → create new `admin_conversations` record and log new thread.	Prevent reusing old tickets. Threads are stateless from user's side — no history is shown to user, only admin retains the full conversation archive.	

3. Contacts

3.1 Contact Menu (and Search)



ID	Element	UI Display Logic	Click / Interaction Behaviour	Edge Cases & Notes	Notes
1	**Search Bar**	Always visible. Single-line input field. Styled consistently with platform inputs. No button. Minimum 2 characters required to trigger logic.	On input: wait 300ms (debounce), then call `GET /api/contacts/search?query=KEYWORD`. The backend searches the full users database (not just saved contacts). It also flags each result as `is_saved = true/false`. The frontend groups results into two internal arrays: `savedContacts[]` and `newContacts[]`. These are rendered in sorted order as per 1.1. Input clears = cancel request, clear results. If no match found → show message:	Backend must return all matched users with saved status. If network error, show toast “Could not complete search.” If empty input → reset results and show default saved list. If saved contact was deleted in parallel, remove from local cache. Ensure identical results are not double-rendered.	

			“No contacts matched your search— hit ‘Invite new user’ to grow your network.”		
1.1	**Search Results Display**	On valid search, results are grouped: **saved contacts first**, followed by **unsaved contacts**. Saved = white card, [View] button. Unsaved = light grey card, [Add] button. No section headers used.	On receiving data, split and render the two groups in order: • `savedContacts.sort(by active sort option)` • `newContacts.sort(by active sort option)` Sorting applies within each group independently (see ID 2). Rendering logic should maintain consistent height, card padding, and visual structure. All results shown in one list. Distinguish visually using card background and action button style. Saved contact interaction = open profile. New contact interaction = trigger Add logic.	If only one group is returned (e.g., all results are saved), show it alone with correct styles. If a saved contact is removed after search, it must be re-rendered as unsaved in grey styling. Always recheck saved status post-Add. Avoid inconsistent state caused by race condition between search and save API.	
2	**Sort Toggle**	Dropdown control visible top-right. Allows sorting by: • Name (A-Z by full name) • Location (country code, then city) • Company Name (A-Z; nulls last) • Date Added (saved contacts only, newest first).	On change: update frontend `activeSort` state.Re-sort `savedContacts[]` and `newContacts[]` arrays according to selected option. Default = Name. • **Name** = sort by full name (`first_name + last_name`). Standard practice is to use full name, not split sort. • **Location** = sort by `country_code` then `city` (both A-Z) • **Company** = A-Z by `company_name`, entries with null shown last • **Date Added** = sort descending by `saved_at` timestamp (only applied to `savedContacts[]`)	If Date Added is selected but only unsaved contacts are shown → ignore sort or disable option. Ensure sort changes trigger re-render only if data exists. Do not reload data from API on each sort toggle. Maintain last-used sort in component state.	
3	**Contact (X) Title**	Static label: “Contacts (X)”. “Contacts” is always visible. (X) shows count of all visible contacts (saved + unsaved). In search mode → reflects result total. In default state → shows count of saved only.	Auto-updated whenever: • Search results change • Contact is added or removed • Default contacts are fetched. Count must sum `savedContacts.length + newContacts.length`. Display label updates reactively.	If zero contacts are shown, display “Contacts (0)” and empty state graphic. If one group is hidden (e.g., only saved), count still reflects total shown. No click interaction. Should be coded as a reactive component label.	
4	Invite New User Button	Text button “Invite new user” appears next to Contact title. Always visible if user is authenticated.	This functionality should already be developed (Process_User Account Related Processes_1.3, Section 7) so just needs reusing here. On click: 1. Open Invite Modal (ID: 12.1). Modal contains two sections:	● If modal fails to load → log error, show “Something went wrong.” ● If rate limit exceeded → disable button and show: “You’ve reached your invite limit.”	

			• Section A: Copy Invite Link (static invite URL for Honley registration page) • Section B: Form fields: Invitee First Name, Email 2. On submit: • Validate inputs • Enforce invite limits: Max 10 invites per user per day, Max 2 per invitee email • POST to `/api/invite-user` with `{ inviter_user_id, invitee_first_name, invitee_email }` • System sends invite email via AWS SES (fallback: Gmail SMTP) • Log action in `user_invites` table • Display modal “User Invite Sent”⚠️ **Important**: If invitee signs up using the invite link and creates a profile, they must be automatically saved to the inviter’s `savedContacts[]`. Likewise, the inviter should appear in the invitee’s saved contacts.	● If input invalid → block send and show validation message. ● If email fails to send → fallback to SMTP, otherwise show: “We’re unable to send the invite.” ● Contact linkage (auto-saving the relationship) must be handled on the backend during profile creation flow using metadata from the invite record. See full logic in ‘`Process_User Account Related Processes_1.3`’, Section 7. Confirm `user_invites` table is implemented with fields: `user_id`, `invitee_email`, `invitee_first_name`, `sent_at`, `delivery_status`.	
5	**[select]**	Button shown at top-right of the contact list. When clicked, activates selection mode: • Checkbox shown beside each contact card • Select Bar (5.1) appears anchored above card list • This button is hidden while bar is open.	On click: 1. Set `selectionMode = true` 2. Show checkboxes bound to contact IDs 3. Display Select Bar component above contact list (sticky anchored) 4. Clear previous selection array 5. Maintain scroll 6. Apply `aria-multiselectable=\"true\"` to list container. When active, all view/add buttons should remain functional, but selection overrides single-click interactions.	Cancel must clear `selectionMode` and selected array. If user switches tabs or leaves page, reset all selection state. Developer must ensure the bar is placed in fixed position as per design. Reuse same checkbox component logic from Chat tab (already implemented). Track contact IDs, not index, to avoid state mismatch.	
5.1	**Select Bar**	Appears only when selection mode is active. Contains: • Select All checkbox • Dynamic count & status	• **Select All** → selects/deselects all currently rendered contacts • **Count** updates on every checkbox change (e.g., “3 Selected”)	Soft-delete only (mark `deleted_at`). Use toast for all actions (delete success, undo, report sent). Prevent duplicate reports via contact ID cooldown. Handle network/API errors gracefully. If any	

		• Buttons: [Delete], [Undo], [Cancel], [Report]. Actions behave as described below.	• **Delete**: shows confirm modal → on confirm, `DELETE /api/contacts/bulk` with contact_ids[]. On success, hide cards and update count (text = “3 Deleted”). • **Undo:** `POST /api/contacts/restore`, reinsert deleted contacts. (text = “3 Reinstated”). • **Cancel**: exits mode, clears checkboxes and selection	action fails, leave remaining items unchanged and notify user. Validate server confirmation before UI update.	
6	**Contact Card**	Container showing contact info. Two styles: • **Saved contacts**: white background, [View] button • **Unsaved contacts**: light grey background, [Add] button Includes: • Profile image (6.1) • Full name (6.3) • Position @ Company (6.4) • Location (6.5) • Action dropdown (6.2) • View/Add button (6.6)	Entire card is interactive except dropdown and profile image. • Clicking card = open User Profile modal. • Displayed fields must be loaded from `/api/contacts/search` or `/api/contacts/saved` depending on context. • Card layout must remain visually stable across states.	If a saved contact is removed, background and button revert to unsaved state. If profile data is missing or corrupt, show fallback visuals (“Unknown User” + grey avatar). Block click interaction on archived accounts. UI must gracefully handle transition between states after Add or Remove.	
6.1	**Profile image**	Left-aligned avatar, **circular**. Pulled from `users.avatar_url`. Company logo not applicable in contact cards. Image is clipped to uniform circle (CSS).	On click: open user profile modal (`/profile/:id`) in current tab or embedded modal depending on context. Enable lazy load to improve performance. Avatar element must not interfere with card-level click target.	On image error or null URL → use fallback grey silhouette. Alt tag should use user’s name for accessibility. If user is deleted → suppress click and show generic avatar. Confirm hover and tap events do not overlap with dropdown trigger zone.	
6.2	**[…] Action Dropdown**	3-dot icon opens dropdown aligned to right of contact card. Options depend on contact state: 1. **View Profile** → opens profile modal 2. **Chat** → opens chat modal (`/chat/:threadId`) 3. **Add/Remove Contact** depending on state 4. **Report** (user only)	• Chat = if contact is unsaved, system first calls `POST /api/contacts/add`, then opens chat modal • Add/Remove Contact = toggle state; update database and re-render card • **Block / Unblock (Add this ontion)**: On dropdown open, system must **check current block state** in the **entity_blocks** table (via /api/chat/block-status/:userId) and update the button label dynamically to Block or	Disable dropdown if user is deleted. For Report: apply 30-min cooldown per contact. If Add fails due to server conflict, show toast “Already in contacts”. Confirm Chat action creates thread only if message is sent (not on open).	

			Unblock. Button must always reflect the actual backend state — do not rely on cached UI toggle Follows unified blocking logic across chat + notifications (see *ConnectHUB System Overview – Section 5*). • Report = opens Contact Honley modal with prefilled fields: – Category: "Flag Issue" (locked) – Subject: "Contact: [Full Name], userID: [UUID]" – Message: user-entered → `POST /api/admin/notifications/report-chat`Include user ID and profile URL for admin lookup. (add user ID from database not the ‘username’ which is private)	Disable chat if user is blocked or deleted. State of Add/Remove must reflect immediately in dropdown label and card button.	
6.3	**Name**	Full name from `users.first_name + users.last_name`. Bold font. Truncated to one line with ellipsis if overflow. Aligned top-left.	Static. No interaction, but area may be part of card click zone (to open user profile). Should not interfere with dropdown trigger.		
6.4	**[Position] @ [Company]**	Rendered below name. Format: `position @ company_name`, both pulled from `user_companies` table. Truncated to fit one line. Skipped if neither value is present.	No interaction, but area may be part of card click zone (to open user profile). Display-only. Ensure text does not wrap or push content.	If company is missing, show position only. If only company present (no position), show company. Suppress if all fields are empty.	
6.5	**User Location**	Rendered below position line. Format: `[City], [Country Code]`.	No interaction , but area may be part of card click zone (to open user profile). Display city and country code in standardised casing (e.g., "London, UK").		
6.6	**[View] / [Add] button**	Right-aligned primary button. **Saved contacts:** “View” (white bg, green border/text). Opens profile modal. **Unsaved contacts:** “Add” (grey bg, black border/text). On Add →	**View** → opens `/profile/:id` in modal. **Add** → call `POST /api/contacts/add { user_id }`. On success: • Move user from `newContacts` to `savedContacts` group (if search)	Prevent Add if already added. If Add API fails → show toast “Could not add contact.” On fast Add + ‘Profile modal → handle race condition between contact save and modal open. Button state must always reflect saved status from DB, not	-

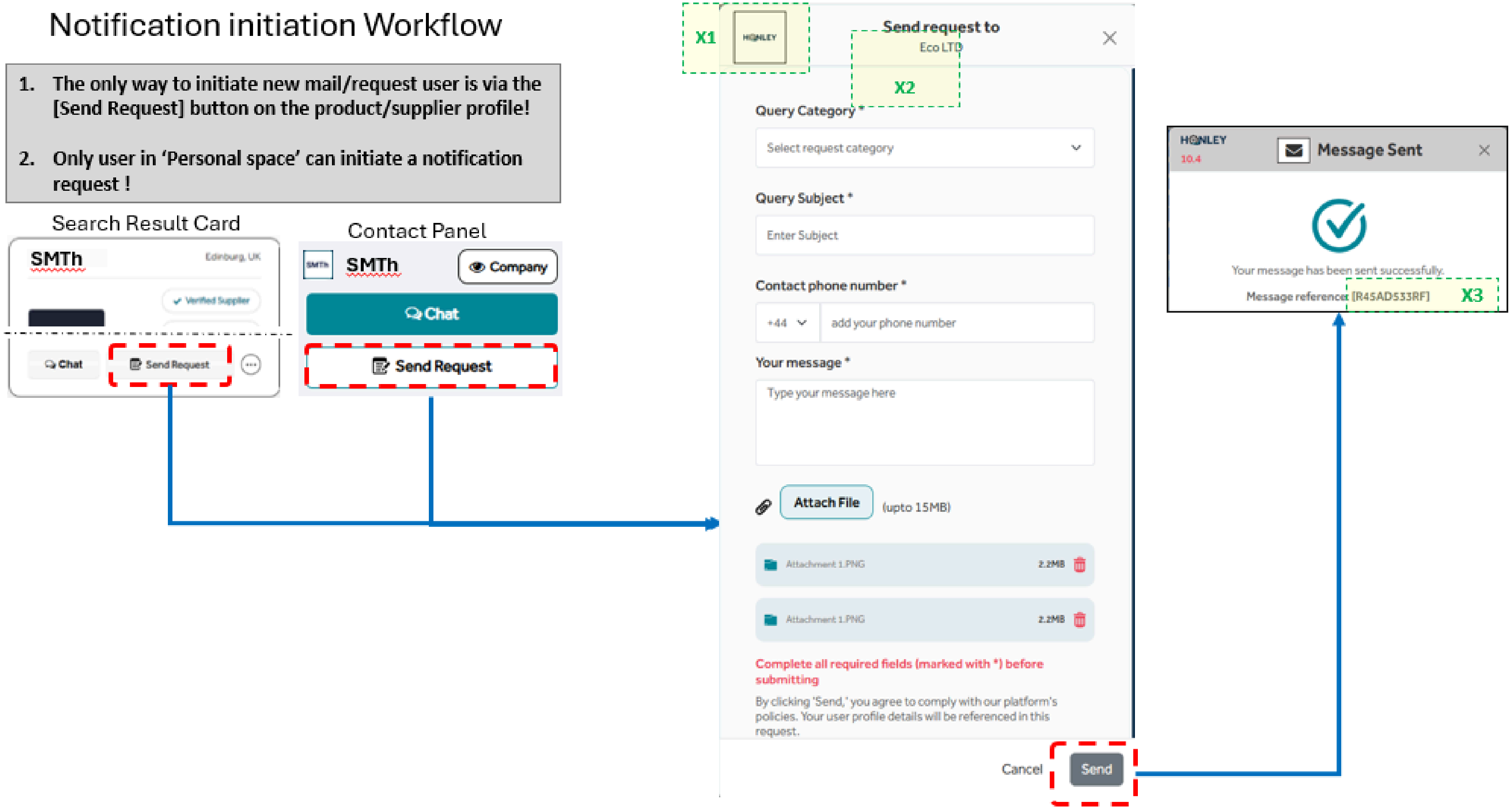
		converts card to saved state and updates button to “View”.	• Change card background to white • Update button from “Add” to “View” • Optionally show toast: “Contact added”	cached only. If contact is later removed → revert card to grey state automatically.	

4. Notifications

4.1 Notification Initiation Workflow – Technical Setup

Notification initiation Workflow

1. The only way to initiate new mail/request user is via the [Send Request] button on the product/supplier profile!
2. Only user in 'Personal space' can initiate a notification request !



Send request to
Eco LTD

Query Category *
Select request category

Query Subject *
Enter Subject

Contact phone number *
+44 add your phone number

Your message *
Type your message here

Attach File (upto 15MB)

Attachment 1.PNG 2.2MB

Attachment 1.PNG 2.2MB

Complete all required fields (marked with *) before submitting

By clicking 'Send,' you agree to comply with our platform's policies. Your user profile details will be referenced in this request.

Cancel Send

Message Sent

Your message has been sent successfully.

Message reference [R4SAD533RF]

ID	User Action	System Response	Edge Case and Implementation Notes	Notes
1	User clicks [Send Request] button on a company or product profile	System checks: • If user is **not logged in** → open Sign-In Modal • If user is logged in → open **Send Request modal**: – Set 'X1': Company logo – Set 'X2': Company name	Ensure button is only active when in **User Personal Space** (Header Type 5). Block this action if user is in Company Dashboard (Header Type 6). If user not authenticated → enforce login before opening modal.	

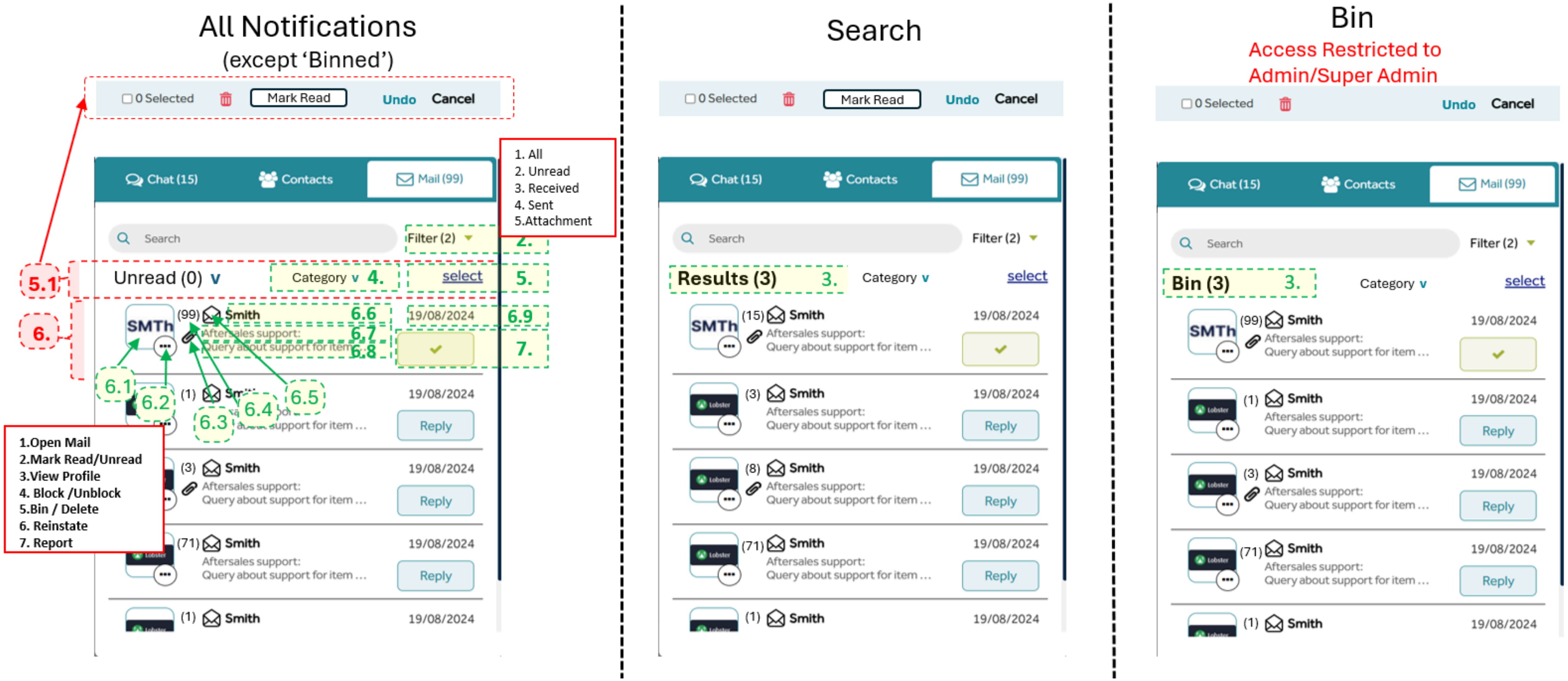
		– Load dropdown options from: notifications company categories		
2	User selects a category *	Dropdown populated from `notifications_company_categories` table:	Required field. If not selected → disable Send button and show: “Please select a category.” Backend must store reference to `category_id` for joinable reporting and inbox filters.	
3	User enters Subject *	Subject field is a free text input (max 150 characters). **If triggered from a product/service page:** 1. System auto-prefills subject with: "`[Product ID]- [Product Name] -` "– User can edit or delete it. 2. Store product_id in the ‘business_notification’ table. **If from a company profile:**– Subject field is empty and required.	Subject is always required. On submit: • Trim input • Reject if empty or >150 chars → show: “Please enter a valid subject (max 150 characters).” Ensure the source (product vs company) is detected via route or request context.	Snag: add product_id to the ‘business_notification’ table.
4	User enters phone number (optional)	Format validated using **libphonenumber-for-PHP** based on auto-detected or selected country code.	Field is optional. If entered, validate as mobile or landline. If invalid → show: “Phone number format not recognised.” Field can match existing users or companies but no linking is needed.	
5	User enters free-text message *	Max length suggested: **3000 characters**. Input supports multi-line entry. Strip scripts and disallowed characters.	Enforce XSS protection. Truncate if text exceeds limit. If empty → block send.	
6	User attaches file(s) (optional)	Allow multiple attachments. File types allowed: • .pdf, .docx, .xlsx, .png, .jpg, .jpeg Total size across all files must not exceed 15 MB. Use Laravel FileStorage to upload to AWS S3.	On upload: • Validate file extension • Enforce size limits (individual & combined) • Block upload if size > 15MB • Show: “One or more files exceed the 15MB total limit.” Sanitise file names, scan for viruses (if possible).	
7	User clicks [Send] button	Laravel backend: • Generates a **Snowflake-based reference number** with `UIB-` prefix (e.g., `UIB-1568320199381`) using `godruoyi/laravel-snowflake` • Creates new record in `business_notifications` table with all fields. System stores explicitly user’s name, **company name ‘company_admin’ table** (if available) and phone number (step 4)	Ensure the entire process runs as an **atomic transaction** — include notification save, file uploads, and reference generation. If any step fails, rollback to prevent partial data. Reference generation must use the Snowflake-based logic described in **ConnectHUB System Overview**,	

		• Check entity_blocks to see if the **company has blocked the user**. – If blocked: proceed with full notification flow but store with blocked = 1, blocked_at = NOW(). – If not blocked: store with blocked = 0, blocked_at = NULL. Then: • Notification becomes visible: → In **company ConnectHUB Inbox** → In **user ConnectHUB Sent tab** • Triggers real-time update via **Pusher** if recipient is online	Section **2.6 Reference Number Assignment (Snowflake-based)** Files (if attached) must be linked using `notification_id` and stored in `notification_attachments`. Ensure data is immediately available to filtered inbox/sent views after commit.	
8	System displays confirmation modal (10.4)	Show modal: • Title: "✅ Message Sent" • Message: "Your message has been sent successfully." • Reference ID (X3): e.g. "`UIB-1568320199381`" • Optionally: [Close] or [Go to Sent]	Reference ID must be pulled from DB record and shown in modal. Confirm it matches the stored ID for audit purposes.	

DB table: notifications_company_categories

Name	**Description**
New Inquiry	*General questions about a company's products or services*
Existing Order	*Issues or updates related to existing orders or shipments*
Technical Support	*Requests for help with using or configuring a product*
Billing	*Queries related to invoices, charges, or payment processing*
Opportunities	*Partnership, distribution, or business collaboration inquiries*
Feedback	*Complaints, dissatisfaction, or improvement suggestions*
General	*Any other communication that doesn't fit the above categories*

4.2 Notifications (All / Bin / Search)



All Notifications (except ‘Binned’)

Search

Bin

Access Restricted to Admin/Super Admin

1. All
2. Unread
3. Received
4. Sent
5.Attachment

1.Open Mail
2.Mark Read/Unread
3.View Profile
4. Block /Unblock
5.Bin / Delete
6. Reinstate
7. Report

ID	Element	UI Display Logic	Click / Interaction Behaviour	Edge Cases & Notes	Notes
1	Search Bar	Same design and behaviour as Chat/Contacts: single-line text input, always visible. Triggered by typing. Placeholder text: “Search notifications…”	On input (after 300ms debounce), system calls: `GET /api/notifications/search?query=KEYWORD`.Search must cover: • `notifications.subject` • `notifications.body` • `notifications.reference_id` • `users.first_name, last_name` • `companies.name` • `notifications.category`	If no results → display "No matching notifications found." System should sanitise all user input. Ensure index coverage on searched columns. Searching is only within active or bin view (respect toggle). Do not include soft-deleted items unless bin view is active.	

			Returned data should be **notification threads**, not individual replies. The UI replaces the current view with "Results (XX)" and updates count. Clicking a card opens the full thread in modal.		
2	Filter Toggle	Dropdown accessible on right side above card list. Default = All. Dropdown lists: • All • Unread • Received • Sent • With Attachment	● **All** = no filter; shows all notifications (except those in Bin) ● **Unread** = filter threads where current user has not opened (`read_at IS NULL` in `notification_reads`) ● **Received** = `last_message_by_id ≠ current_user_id` (i.e., someone replied to user) ● **Sent** = `last_message_by_id == current_user_id` (user sent last message) ● **Attachment** = thread has `notification_attachments.thread_id` present ● ● Trigger a new fetch to backend for each change, or apply locally if dataset already loaded.	Combine filters where possible (e.g., Unread + With Attachment). Keep state in route params or frontend store. For "Sent" and "Received", base status on the **last message** in the thread. Ensure queries use JOINs with indexed fields. When filter active, highlight selected tag in dropdown.	
3	Title with (count) + toggle	Dynamic header label above the notification card list. The label text and count update based on current view state: • **Unread (X)** – default view. X = count of notification threads where current user has not opened the last message (`notification_reads.read_at IS NULL`)	Dropdown options: • **Inbox** – default view, shows all non-deleted threads (`deleted_at IS NULL`) • **Bin** – shows soft-deleted threads only (`deleted_at IS NOT NULL`)	If non-authorised user (e.g., company member without admin role) tries to force-load Bin via `?view=bin` in URL, backend must return HTTP 403.If search is active, hide toggle and override title to "Results (X)".Bin must always be shown empty for	

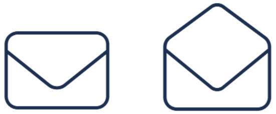
		• **Results (X)** – displayed only when a keyword search is active. X = count of matching notification threads • **Bin (X)** – displayed when Bin view is toggled. X = count of notifications with `deleted_at IS NOT NULL` (soft-deleted threads) To the right: dropdown toggle that lets user switch between "Inbox" and "Bin" views.	**Dropdown Visibility Logic:** • **Personal ConnectHUB**: toggle is always visible • **Company ConnectHUB**: toggle is **only visible to Admin and Super Admin roles** Clicking on an option reloads the thread dataset with the appropriate filters applied. Counts should update live based on result set. Toggle is hidden during search mode (`view=search`).	users with no access, even if soft-deleted threads exist. When switching between Inbox ↔ Bin, UI must maintain scroll position where possible and reset all selection/filter states. Counts must be recalculated via backend or derived from filtered dataset.	
4	Category Toggle	Positioned beside Filter toggle. Dropdown shows only categories **present in currently loaded results**. If no results, dropdown is disabled. Categories pulled from notifications via join or from indexed list at API level.	Populate using distinct query against current dataset:`SELECT DISTINCT category FROM notifications WHERE is_deleted = false AND user_id = ?` When clicked, apply client-side filter on current dataset. Optional: allow combo filter (Category + Unread, etc). Truncate long category names.	Disable when result set is empty. Provide "Clear filter" option to reset. Don't re-fetch unless necessary — use local filtering where possible. In Bin view, show only categories used by binned items.	
5	[select]	Button top-right of the card container. Triggers selection mode: • Checkboxes appear beside each thread • [Select Bar] appears (see 5.1) • This button disappears while bar is active	On click: 1. `selectionMode = true` 2. Display checkboxes (bound to notification ID) 3. Render Select Bar above list 4. Clear existing selected state 5. Preserve scroll 6. Apply `aria-multiselectable=\"true\"` for accessibility Checkboxes must not interfere with Reply/View buttons	If no messages in view → disable select button. If user leaves tab or page, clear selection mode. Use same checkbox logic as Chat and Contacts module. Selection state must persist correctly across pagination (if used).	

5. 1	Select Bar	Appears only when **selection mode** is active (`selectionMode = true`). Sticky-positioned at the top of the card list container. UI includes: • **Select All** checkbox • Dynamic **(X Selected / Deleted / Reinstated)** counter • Action buttons: – **[Delete]** or **[Bin]** (label changes based on context) – **Undo** – **Cancel** – **Report** – **Mark Read**	**Context-aware delete/bin logic:** 1. If user is in **Inbox view (default)**: → [Bin] button is shown → On click, show modal: *"Do you want to move these items to Bin?"* → On confirm: call `POST /api/notifications/bulk-soft-delete` `→ DB: set deleted_at_user = now() for sender or deleted_at_company = now() for company` → Cards removed from current view 2. If user is in **Bin view**: → [Delete] button is shown → On click, show modal: *"Do you want to permanently delete these items?"* → On confirm: call `POST /api/notifications/bulk-permanent-delete` → system soft-deletes (flags for purge or hides forever) → Cards disappear from Bin **Other Buttons:** • **Undo:** visible in Bin only → sets `deleted_at = NULL` and restores card to Inbox • **Mark Read:** applies `read_at = now()` on each selected thread • **Report:** opens Contact Honley modal (pre-filled, up to 10 threads) • **Cancel:** clears selection mode and resets selection state and bar	- Always disable action buttons when no notifications are selected - Label for [Bin] vs [Delete] must change **based on current folder view**: - Bin → user/company-specific deleted_at_* timestamp in DB. - Undo → clears that value (set deleted_at_* = null) and moves card back. - Delete → irreversible, only Admin/Super Admin. API must check user role. - [Mark Read] does nothing on already-read threads. - [Undo] should only affect items in Bin. - [Report] must enforce cooldown (e.g. 30 min per thread/user combo) and limit (10 threads max). - All actions must return toast on success or fallback alert on error. - Do not allow permanent delete unless user has required permissions (admin/superadmin). - Bin and Restore logic must sync with thread state on server side and UI view filters.	

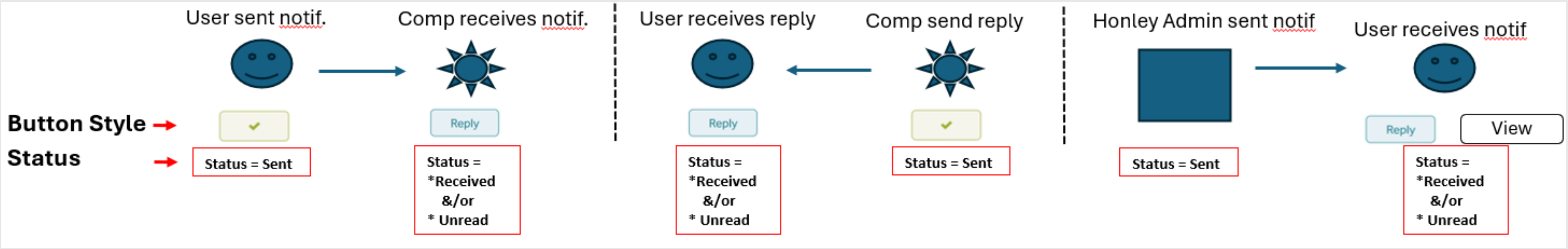
6	Notification Thread Card	Each card represents a **notification thread** (not individual messages). Created when a user sends a notification (to a business or Honley Admin), or when Honley Admin sends a platform notification. The thread is assigned a **Snowflake-based reference ID**, stored in `reference_id`. All replies (user ↔ company, user ↔ admin) are bound to the same thread ID. The UI card always reflects: • Sender/receiver identity • Last message snippet • Category • Subject • Timestamp of last message • Read/unread state • Status button (e.g., Reply / ✔️ / View) No new card is created on replies. Status and preview auto-update. The UI mimics chat, but is structured for traceability, categorisation, and long-form interaction.	Clicking anywhere on the card (except image, dropdown or checkbox) opens the **notification thread modal**, loading all messages tied to that thread via `thread_id`. Last message data is shown dynamically, updated via WebSocket (Pusher). Attachments are indicated by an icon, subject is truncated in preview, and the card updates immediately if user replies. State changes (read/unread, responded) are applied in real-time and updated visually on the card.	If the user is the sender, the name shown is the recipient (company or Honley); If the user is the receiver, the name is the sender. Card must handle: • Read status • Role context (user vs admin) • Preview formatting Use ellipsis for long text in subject/category. Ensure thread modal correctly shows all messages, attachments, sender metadata, and reply button (if allowed). Use card layout consistent with Chat UI. Read/unread must sync across views. Never show duplicate cards.	
6.1	Profile image	Left side avatar. Shows: • **User profile image** (for company viewing a user message) • **Company logo** (for user viewing a company notification) • **Honley favicon** (for system-generated or admin notifications) Styled circular or square based on context. Uses lazy loading with fallback default.	Clicking the avatar opens: • User profile → modal window • Company profile → opens in new browser tab • 'About Honley' → opens in new browser tab.	Avatar must never break layout. If user is deleted or has no image, fallback to generic grey icon. Ensure click targets are not overlapping with dropdown or checkbox elements. Do not allow drag/select interaction over the image. If company is no longer published,	

				profile still opens in read-only state.	
6.2	[…] Action Dropdown	Appears as a three-dot button, top-right on each card. Opens a dropdown with context-aware options: 1. **Open** – opens thread modal 2. **Mark Read/Unread** – toggles current status 3. **View Profile** – opens profile based on sender (user profile, company, Honley-About) 4. **Bin** – for active notifications, moves thread to Bin (soft delete) 5. **Reinstate** – only shown in Bin view, moves thread back to Inbox 6. **Delete** – only shown in Bin view, permanently deletes (admin only) 7. **Report** – opens Contact Honley form, prefilled with thread data	● **[Bin]** button is shown If user is in **Inbox view.** → On click, show modal: *“Do you want to move these items to Bin?”* → On confirm: call `POST /api/notifications/bulk-soft-delete` → `DB: set deleted_at_user = now() for sender or deleted_at_company = now() for company` → Cards removed from current view ● **[Delete]** button is shown If user is in **Bin view:** → On click, show modal: *“Do you want to permanently delete these items?* → On confirm: call `POST /api/notifications/bulk-permanent-delete` → system soft-deletes: `DB: set permanently_deleted_at_user = now() for sender or permanently_deleted_at_company = now() for company`) → Cards disappear from Bin • **Block / Unblock (Add this ontion)**: On dropdown open, system must **check current block state** in the **entity_blocks** table (via /api/chat/block-status/:userId) and update	Developer must validate action visibility: • **Bin** only in Inbox • **Reinstate/Delete** only in Bin • **Report** only outside Bin Prevent duplicate report spam (e.g. cooldown by thread_id and user_id). On deletion, soft-delete only (mark in DB). UI should not reflect deletion until backend confirms. All actions must update UI state immediately with proper toast messages. Handle failed delete/restore via fallback alerts.	

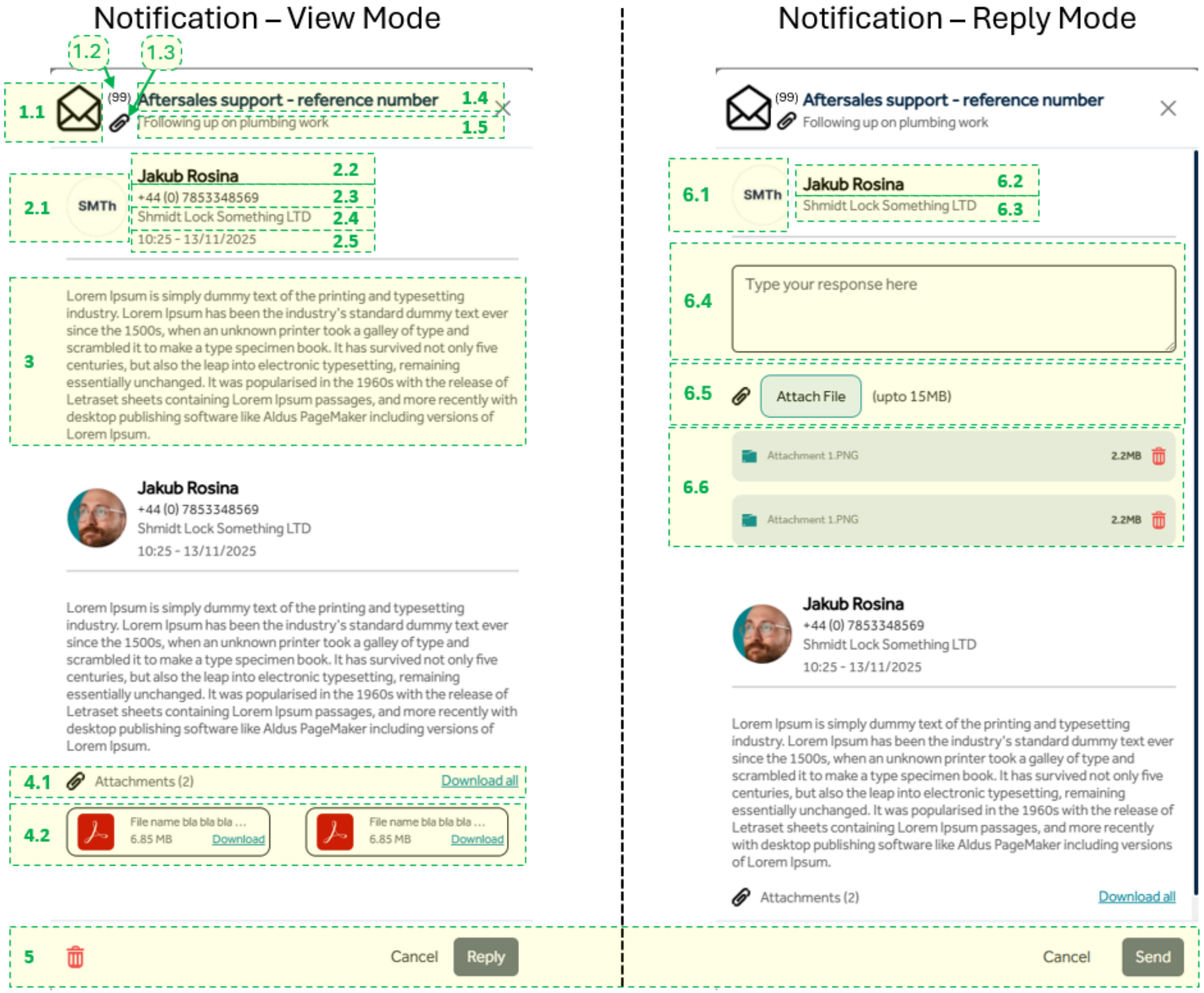
			the button label dynamically to Block or Unblock. Button must always reflect the actual backend state — do not rely on cached UI toggle Follows unified blocking logic across chat + notifications (see *ConnectHUB System Overview – Section 5*). • **Reinstate** sets `deleted_at = null` and restores card. • **Report** opens Contact Honley modal (same as other modules) with: - Category = Flag Issue (locked) - Subject = "Notification: [Category] -[Reference ID] " - Message = required field On submit → POST `/api/admin/notifications/report-notification` with thread ID, full message history + attachments. Show confirmation toast: “Your report has been submitted to the Honley Admin Team.”		
6.3	Attachment icon	Small paperclip icon next to subject area, visible only if thread contains one or more attachments. File check is based on presence of entries in `notification_attachments` for the given `thread_id`.	No interaction. Visual indicator only. Tooltip shows “Attachment included.” Should not be clickable. Icon spacing should not affect text truncation.	Icon must not appear if attachments were deleted. Only show if `attachment_count > 0`. For threads with large files, ensure file handling is deferred until thread modal is opened. Ensure consistency with similar	

				icon use in Sent tab, Mail modal, and Chat.	
6. 4	Thread Count	- Small counter badge displayed left of the name field in the notification thread card (e.g. **(3)**). - Reflects the number of posted messages in the thread (`thread_id`). - Does **not** include drafts or unsent messages. - Count auto-updates dynamically after each successfully posted message. - Value is capped at **99** – once it reaches 99, it stops incrementing and shows **(99)**.	No interaction. Purely visual indicator. Count updates in real-time via Pusher or polling. API must return thread-level `message_count` field as part of the card list payload.	• Do not increment count for typing/draft states. • Include both inbound and outbound messages. • If count exceeds 99, freeze display at (99) — do not display 100+. • Hide badge entirely if count = 0. • For migrated/legacy threads with missing count, show no badge or fallback to approximate count from backend.	UI Snag: Add (XX) message count to notification thread card (left of name). Move attachment icon below name, aligned with subject.
6. 5	Mail Icon	Envelope icon toggles based on read state for the viewer: • **Open envelope** → user has already read the latest message in the thread (`notification_reads.read_at IS NOT NULL`) • **Closed envelope** → user has not read the latest message (`read_at IS NULL`). Bold styling is applied to name and subject for unread threads. 	Icon updates automatically when user opens the thread (read_at set). System must visually update the icon immediately and remove bold styling from text elements. Tooltip (optional) can show "Unread" / "Read".	Must sync across views (Search, Inbox, Bin). Read state stored per user per thread in `notification_reads`. Do not show different icon for sender's own thread. If message is system-generated, same logic applies.	
6. 6	Name	Name of the person or entity on the **other side** of the conversation: • For user: company name or "Honley Team" • For company: user's full name	No interaction. Text shown as part of card metadata. Clicking card opens thread, but name is static.	Ensure correct resolution of user ↔ recipient logic. Do not show user's own name. Use `users.first_name + last_name`, `companies.name`, or static "Honley Team" if sent by	

		Styled in bold if unread, normal if read. Truncated with ellipsis if too long.		system. If data missing (e.g., deleted sender), show “Unknown Sender”.	
6.7	Notification ID and Category	Printed together in the format: **[Category Name] – [Reference ID]**. Example: “Billing Support – UIB-1890234982”. Truncated with ellipsis on overflow. Uses `notifications.category` and `notifications.reference_id` fields.	Static display text. No interaction. Tooltip shows full value if truncated. Should be distinguishable by styling (e.g. secondary text).	Ensure backend always assigns category and reference before save. Do not allow reference ID reuse. Reference ID format follows Snowflake-based ID with prefix (e.g., UIB-, UIA-, ADM-).	
6.8	Subject	Main subject of the notification. Truncated to one line on card. Can be clicked inside modal, not on card. Subject is set during notification creation and cannot be changed after.	Display only. No action. Used to help recipient triage notifications. Full subject is shown inside thread modal.	Max length: 100 chars. Sanitise input. Subject should never contain links or script content. Use `htmlspecialchars()` to escape before display.	
6.9	Timestamp	Shows time of last message in the thread, right-aligned. Format: • Same-day = `HH:mm` • Same-week = day of week label (e.g. Mon) • Older = `DD MMM YYYY` (e.g. 02 Apr 2025).Field is based on `notifications.last_message_at` or `notifications.updated_at`	Auto-updates when new message added. Used for sorting card order (most recent on top).	Timezone aware (convert to user local time). Always use UTC in DB. Ensure polling or Pusher updates value in real time.	
7.	[✔] / [Reply] / [View] Button	Always visible, right side of each card. **Behaviour depends on who sent the last message**: • [✔] = current user sent last message (no reply needed) • [Reply] = other party sent last message (user should reply) • [View] = Honley notification with **reply disabled**. Button opens thread modal.	Clicking button opens modal with full thread and reply box (if allowed). Button style updates dynamically based on `last_message_by_id` compared to `current_user_id`. If last message was sent by Honley and `response_allowed = false`, disable send field and change button to “View”.	Ensure buttons update automatically after message is sent or received. Prevent reply action if modal disables response (admin logic). Use consistent styling: lime green = ✔, green = Reply, grey-outline = View. If message is auto-closed or expired, reply may also be blocked by rule.	



4.3 Notification – Message Interface



Notification – View Mode

Notification – Reply Mode

1.1
1.2
1.3
(99) Aftersales support - reference number
1.4
Following up on plumbing work
1.5
2.1
SMTh
Jakub Rosina
2.2
+44 (0) 7853348569
2.3
Shmidt Lock Something LTD
2.4
10:25 - 13/11/2025
2.5
3

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

ID	Element	UI Display Logic	Click / Interaction Behaviour	Edge Cases & Notes	Notes
0	**View / Reply Mode**	When the notification thread modal opens, it defaults to '**View Mode'**. This mode displays the full thread history (chronologically sorted) in read-only format.	Click **[Reply]** button → transition into Reply Mode (element 5). Modal height adjusts to show	Must prevent reply field appearing on modal open.	

1		To respond, user must click the [**Reply**] button at the bottom. This toggles the UI into **Reply Mode**, revealing the input field and attachment controls.	input area. Pressing [**Cancel**] returns to View Mode.	For system-generated (Honley) notifications with reply disabled (`response_allowed = false`), Reply button should not be visible at all. Handle scroll focus when toggling into reply.	
1	**Notification Header Section**	Header row pinned at the top of the modal. Provides core notification metadata — not message-specific — and remains constant as user scrolls through messages.	No interactive behaviour. Static visual.	All fields must be populated from thread metadata (`notification_threads` or equivalent object). Use fallback values (e.g., "—") only if data unavailable. This header is independent of the individual messages shown below. For long values (ref ID or subject), allow line wrapping (no truncation).	
1.1	**Envelope Icon**	Always shows as **Open Envelope** icon, symbolising this is a "viewed" thread. This is not tied to read/unread message state — it's purely a static UI signal that the full thread is being viewed.	None. Static image.	No dynamic change. Make sure icon renders consistently in both View and Reply modes.	
1.2	**Thread Count**	Badge shown left of category. Displays number of actual messages (`notification_messages`) linked to current thread ID. Only posted messages are counted (excludes drafts or typing). Max value shown is (99).	None. Visual only. Backend must return message count as part of thread header API response (`message_count`). Updates in real-time via WebSocket or polling.	Hide if count is 0. If count > 99, always show (99). Ensure backend count excludes deleted or failed messages. Count should reflect combined user and company/admin messages. Use same count logic as shown in the notification thread card (outside modal).	UI Snag: Add (XX) message count
1.3	**Attachment Icon**	Paperclip icon shown only if thread contains one or more uploaded attachments. Detection is based on attachments stored in `notification_attachments` table linked by `thread_id`.	None. Visual indicator. Tooltip optional: "Attachments included".	Hide if no attachments present (i.e. `attachment_count = 0`). If attachments were removed from all messages in thread, icon should disappear.	UI Snag: Add (XX) message count.
1.4	**Category – Reference Number**	Print in format: **[Category Name] – [Reference ID]**. Text may wrap to second line if long, but never truncate with ellipsis. Pull: • Category from `notifications.category_name` • Ref No from `notifications.reference_id`.	None. Display only.	Ref ID is unique per thread, generated at creation time (Snowflake-based). Should never be empty. Ensure proper formatting and casing. Example: "Aftersales Support – UIB-0021994".	
1.5	**Subject**	Shows full subject entered by user during request creation.	None. Display only.	Ensure subject is escaped and does not contain unsupported characters.	

		Wrap to 2 lines if needed. Pull from `notifications.subject`. Do not truncate.		Note Subject Input is limited to 150 chars at notification creation. If field is null, show “(No Subject Provided)”.	
2	**Message Originator Section**	For each message bubble, render a metadata block above the message content. This block includes: avatar, sender name, contact number (if any), company name or location, and timestamp (handled separately). Applies to all messages in the thread, including original and replies. Pulled from metadata stored per message.	None	On render, UI loops through messages and renders metadata block using fields stored with the message. This avoids querying `users` or `companies` live. Required fields must be stored at time of send (e.g. `sender_name`, `sender_company`, `sender_phone`). Render statically — not dynamically looked up.	DB Snag: Add sender_name, sender_company, and sender_phone fields to all notification and reply tables as per **ConnectHUB System Overview – section 2.3**.
2.1	**Profile Image**	Render profile picture on left of metadata block. Source logic: • **User**: circular avatar via `users.avatar_url` • **Company**: square company logo via `companies.logo_url` • **Honley**: square favicon asset `/assets/honley.png` Fallback to neutral grey icon if image not found.	No interaction	Avatar must be rendered at fixed. Use `object-fit: cover`. Do not stretch. Do not re-fetch avatar each time — load from latest DB record or file path. Do not attempt lazy loading unless entire thread is very large. Do not open any modal on click. Ensure alignment does not break if image is missing.	
2.2	**Sender Name**	Show sender’s name as plain text. • From **user**: `first_name + last_name` (stored on message) • From **company**: name of the user who sent it • From **Honley**: literal `"Honley Team"`	None	Do not link name to any modal or page. Do not truncate on one line — allow wrapping. Text must reflect sender of that specific message, not the thread creator.	
2.3	**Contact Phone Number**	Displayed **only if available and permitted**.	None	Collapse line if number is not available. Remove excess whitespace.	

		• **User**: shown if user typed it in 'contact phone' field during request. This is stored with the original message. •**Company**: only if `display_phone = 1` in `companies` table — show `phone_prefix + phone`• •**Honley**: always hidden		Render in full international format (e.g. +44 7853 123456). Never re-fetch phone from user table once message is stored. If company disables phone display, new messages respect that; old ones preserve history.	
2.4	**Company Name**	Display `sender_company` stored at time of message send. This applies to both user and company messages. Always pulled from `companies.name` at time of send (**via `company_admins` if sender is user**).	None. Static text	Do not link. Do not update dynamically. Always render stored value, even if company is deleted or renamed later.	
2.5	**Timestamp**	Display to the right under sender info. Format: **HH:MM – YYYY-MM-DD**. Use sender's time of sending (not recipient view). Must be timezone adjusted to current user.	None. Display only.	Always use UTC in DB and convert to user timezone on frontend. Display format must be consistent across all modules (Chat, Notifications, etc). Do not round or approximate.	
3	**Message body**	Render full message body as plain text. This includes any line breaks or spacing as entered by the sender. Messages are pulled from `business_notifications.message` (for first message) or `business_notification_replies.message` (for replies). Same logic applies to admin tables. Render in full width of the message bubble container.	No interaction.	Ensure safe rendering — escape content to prevent XSS (e.g., `htmlspecialchars`). Handle very long strings by word-wrap or truncation with vertical scroll. Do not support markdown or HTML. Maintain spacing/padding consistency even with short (1-line) messages. Consider soft character limit (e.g., 5000) to protect layout.	
4	**Attachments Section**	If one or more attachments exist for the message, show a collapsed section underneath the message body. This section contains a header (4.1) and the list of individual file blocks (4.2). Attachments are loaded via relation (`message_id → notification_attachments` table). Only shown if records exist. Otherwise, collapse space entirely.	No direct interaction on container. Contains child elements for download action.	Attachments must be linked to the specific message ID (not thread). Attachment list fetched as part of message payload. If files were removed (e.g. expired), show placeholder or skip entry with log. Prevent layout shift if attachments section loads async. Must support zero, one, or multiple attachments gracefully.	

4.1	**Attachments count & Download All**	Below message bubble, show compact line: [📎] `[Attachments]` `(X)` [<u>`Download all]`</u> link right-aligned. ● Attachment icon and label are static. ● Count is dynamic (number of valid files). ● "Download all" should trigger sequential download of all attached files in that message. ● Each download should follow native browser download flow.	[Download All] triggers multi-file download. Use JS `fetch` or batch download endpoints to collect files for download. Trigger standard file download per file. If more than 5–10 files, show “Downloading...” message.	Ensure this section is not visible if no attachments exist. Count only valid, stored files. For missing files, skip and show user toast: “Some attachments could not be downloaded.” Use secure signed download URLs with expiry (avoid exposing raw file paths). Backend must validate user has access to the message before allowing file download.	
4.2	**Attached Files detail card**	For each file, display a rectangular clickable row containing: • File type icon (e.g. PDF, DOCX, XLS) • File name (truncate with ellipsis after 1 line) • File size in MB • “Download” link (right-aligned or below size) Entire card is a clickable unit that triggers download of that individual file.	Clicking anywhere in the card starts download of that one file. Use same signed download logic as above. Use `<a href>` with `download` attribute or trigger programmatically. No hover effects required.	Filename truncation must preserve full filename in tooltip. File type icon based on MIME or extension. If file is unavailable or corrupted, catch error and show user: “File not available.” Don’t allow right-click copy URL if sensitive. Handle edge case where file size is 0B with warning toast. Card layout must stay uniform even for odd file names or rare formats.	
5	**Footer Section**	Footer visible at bottom of modal at all times. **Before rendering buttons, system must check if the other party (user or company)** still exists and is status! = 'closed'. If either is closed → hide Reply button and disable Send section. Display static info: “This contact is no longer active on the platform!” Default buttons in **View Mode**: • `[Bin]` (grey outline) • `[Cancel]` (no fill) • `[Reply]` (green button). In **Reply Mode**, replace `[Reply]` with `[Send]` (green) and `[Cancel]`.	• `[Bin]`: button is shown in ‘View’ mode → On click, show modal: *“Do you want to move these items to Bin?”* → On confirm: call `POST /api/notifications/bulk-soft-delete` → `DB: set deleted_at_user = now() for sender or deleted_at_company = now() for company` → close modal and hide card from active inbox • `[Reply]`: **If current user is blocking recipient:** [Reply] button	Bin logic: soft delete only. Thread reappears in ConnectHUB if other party replies to same thread. Rehydration logic must check for `deleted_at != null AND new reply created_at > deleted_at` to re-display. Do not re-show binned threads if reply is older than deleted_at. **Send logic:** if message is empty or file > 15MB, block send with clear alert. Honley messages with `allow_replies = 0`: hide reply button entirely. On reply send, UI must reset to view mode and update cached thread list immediately.	

			remains visible but **disabled**, and on hover displays tooltip: *“Remove contact from block list to send a message.”* → *If contact no longer exists or is closed → disable button with static info: “This contact is no longer active on the platfor!”* IF NOT blocking, on click opens reply section 6 (show text area, file upload) • `[Send]`: validates reply + attachments → POST reply → update UI (thread card count, last message timestamp, etc.) • `[Cancel]`: closes modal		
6	**Reply Section**	Reply input section is **anchored at the top** of the thread modal and **always visible** when in reply mode (i.e., after clicking `[Reply]`). Messages from the other party, if received while this modal is open, are appended **below** the reply section (maintaining visual chat-like flow). Modal auto-scrolls to latest entry.	When user clicks `[Reply]`, section 6 becomes visible (shows profile, company, input, file uploader). This block remains sticky at the top. As replies arrive via WebSocket/Pusher or polling, new replies populate below. If user cancels, clear any draft text and attachments. `[Send]` triggers form validation + API POST to `/api/notifications/reply`. On success, field is cleared and UI resets to view mode.	Prevent duplicate sends. If user attempts to close modal with filled text or attachments → show “Discard unsent message?” confirm dialog. Do not allow access to this section if notification has `allow_reply = false`. All attachments must be validated client-side and re-validated server-side before upload. System must block reply if another submission is in progress.	

6.1	**Originator Profile Image**	Displays avatar image of the logged-in user (reply sender). Use same logic as display bubbles: • User → `users.avatar_url` (circular) • Company → `companies.logo_url` (square) • Honley → Honley favicon (square). Always left-aligned.	No interaction. Image is static. It is only for display purposes.	Use placeholder if no image is available. Avatar URL is obtained from the current session and not re-fetched per reply. Do not store image URL with draft. Always fetch fresh from current user state. Must not be clickable or trigger layout shift. Safe load via `<img onerror>` handler fallback.	
6.2	**Originator name**	Shows full name of the user composing the reply (first + last name). Fetched from session or `users` table. Always reflects **current identity** of the sender.	None. Display-only.	Must match sender's current identity. Ensure layout does not break with long names wrap accordingly. Font styling must match view mode bubbles.	
6.3	**Originator company**	Displays sender's current company (if applicable). Rules: • For user → if found in `company_admins`, pull `company_id` → get company name from `companies`. • For company staff → always show associated company from `companies` table. • For Honley → **hide/collapse.**	No interaction. Static. Label appears below name. Collapse row if null.	If user is no longer in company_admins, field will be blank. We do not backfill or resolve past links. Only show live company association of the current user. Ensure graceful fallback. Never link company name from this location. Safe to collapse completely if not applicable. In very long company names, truncate with tooltip for full value.	
6.4	**Message Input**	Multi-line text area, max 3000 characters. Auto-expands as user types (up to 10 lines). No formatting allowed (plain text only). Enforce input sanitation via XSS protection and remove disallowed characters/scripts. Block send if empty.	On input, live character count could be displayed (optional). `[Send]` button must be disabled if textarea is empty. Trim on submission. On input change, validate message length. Strip unsafe input on backend (Laravel validation middleware).	Do not allow HTML, script tags, or markdown. Validation: min 1 character, max 3000 characters. Show error on submission if empty or too long. Backend validation must match frontend limit. If failed backend validation, toast error and do not clear input. Clear field only on successful reply post. Do not persist draft locally unless explicitly requested.	
6.5	**Attachment Button**	Allows upload of multiple files. Accepted formats: • `.pdf`, `.docx`, `.xlsx`, `.png`, `.jpg`, `.jpeg`Total combined size = max 15MB. Display per-file blocks below as they're added (row 7). Use Laravel FileStorage for S3 upload.	Clicking [Attach File] opens file picker. Multiple selection enabled. On file select: • Validate extension • Validate total size • Upload to S3 • On success, append to UI preview list (row 7).	Scan each file extension. Trim file name length before display. If user uploads >15MB, reject and show total used space. Don't store file unless upload completes. Store in temporary table or attach ID until final submit. File list cleared if user cancels reply. Filenames must be safe (strip emojis, control chars). Maintain secure, signed download link even for drafts.	

			If fails, reject with toast: “File too large or unsupported.”		
7	**Attached Files detail**	Display each uploaded file before send. Each file shown in a removable card: • File icon (type) • File name `[name.ext]` • Size in MB • [Delete] icon (X) Cards render vertically stacked or as flex row grid.	Clicking X removes file from upload list and from pending backend storage (if already uploaded). On hover, highlight card. Clicking filename or icon does nothing.	File cards must be deletable at any time before `[Send]`. No reordering required. Backend must remove temp record if file was already uploaded. Cards removed from DOM instantly on X. If user re-adds same file, treat as new item. Prevent duplication by name+size. If attachment is required in future updates, logic should adapt accordingly.	