ConnecWrk – Messaging Module (Chat / Conversations / Meetings)

Super Detailed Spec v1

0. URL / Scope / Surfaces

- Primary route: `https://connecwrk.com/messaging/`
- Global header entry point: clicking **Messages** (chat bubble icon) in the top nav bar anywhere in the product
- Other in-product entry points that deep-link into messaging:
- "Send Enquiry" / "Contact" buttons on:
    - Assignment detail pages (`/project/?id=...`)
    - Freelancer / Talent profile cards
    - MSME business pages
- These links open `/messaging/` preloaded with that user/contact and (optionally) a draft intro message.

Messaging is a shared primitive across: - Freelancer Connect (client ↔ freelancer about an assignment) - Assignments / Bids (negotiate scope, budget, timeline) - MSME Connect (leads contacting a business page) - Talent Connect (event organizer ↔ speaker/artist)

Messaging is not just chat. Inside a conversation you can: - Send text and attachments - Negotiate deliverables and payment terms - Ask/answer clarifications about posts, bids, services - Schedule a meeting (e.g. video call) and drop the meeting info back into the chat

---

1. Product Definition

1.1 What Messaging Is

ConnecWrk Messaging is the internal DM / inbox experience that lets two users communicate in real time while staying inside the ConnecWrk ecosystem. It is designed so that: - Clients / MSME owners can reach freelancers and talent without leaving the platform. - Freelancers / talent can reply to leads fast, with proof of conversation. - All negotiation context (requirements, price, milestones, payment expectations) stays on-platform and can be referenced later. - A meeting can be set up without exchanging personal phone numbers.

1.2 High-Level Capabilities

1. Conversation list for all active chats
2. Real-time message thread for the selected contact
3. Message composer with:
4. Text box
5. Send action
6. Attachment upload (files/images up to a size limit)
7. Optional quick actions like "Schedule Meeting"

8. Contact search / New Chat flow
9. Read status, timestamps, sender info
10. Empty states and onboarding guidance when user has no chats yet

1.3 Why Messaging Matters for the Platform

- Keeps client ↔ freelancer ↔ talent ↔ MSME communication and negotiation inside ConnecWrk.
- Reduces leakage to WhatsApp / LinkedIn, which means:
- Better retention for ConnecWrk.
- Clearer audit trail if there's a dispute.
- Ability to productize later (e.g. in-chat contract, in-chat payment request, in-chat milestone approval).
- Enables an AI assistant (Connie) to help summarize or guide the conversation contextually.

---

2. Personas & Primary Goals

2.1 Client / MSME Owner / Project Owner

Examples: a small business owner posting a project, an MSME hiring a freelancer, an event planner booking a keynote speaker.

Goals: - Reach out to relevant talent/freelancer directly. - Send requirements (text or file). - Negotiate scope, budget, and dates. - Schedule a call. - Track who responded.

Must-haves for this persona: - Confidence that their message was delivered and read. - Structured context tied to assignments or services. - A simple path to escalate to a meeting.

2.2 Freelancer / Talent / Speaker / Performer

Examples: designers, developers, voiceover artists, keynote speakers, singers, trainers.

Goals: - Receive inbound leads (“Hey, saw your profile”). - Respond quickly with pitch, clarification, timeline. - Capture requirements to estimate effort. - Move toward deal closure.

Must-haves for this persona: - Fast reply UX. - Ability to send a short intro / portfolio / sample as attachment. - Proof of agreement (what was promised, what timeline).

2.3 Platform / Compliance / Safety

Goals: - Keep conversation data logged in case of abuse, harassment, or disputes. - Limit spam and protect users. - Support rate limiting and file size/content restrictions.

---

3. Global Navigation & Layout Context

3.1 Global Header Elements (visible across most pages)

- ConnecWrk logo
- Global search bar with placeholder like `Search "MSME Job"` and a submit arrow icon

• Icon menu:
• Home
• Contacts
• Messages
• Notifications
• User avatar / account menu

Clicking **Messages** in this header routes to `/messaging/`.

3.2 Messaging Page Layout (High-Level)

The Messaging page uses a 2- or 3-column pattern: 1. **Left Sidebar / Conversation List / Contacts** 2. **Center Chat Window (active thread)** 3. **(Optional) Right Sidebar / Context panel / Ads / Profile snippet**

If the user has not selected any contact yet, the center panel shows: - "Select a contact to start a conversation" - A quick CTA: "Start a new conversation"

If there are no chats at all (first-time user): - A blank state card that says something like: - "No chats available." - "Start a new conversation"

This onboarding state is important: it tells Connie what to say when user asks "Why is my inbox empty?".

---

4. Screens & Components (Detailed)

4.1 Conversation List Panel (Left Sidebar)

Purpose: - Show all ongoing 1:1 chat threads. - Let user pick which thread to open.

Visible Elements: - Search box at top: `Search by name or message...` - Button: **Start a new conversation** (opens contact selector or new-message modal) - Scrollable list of threads: - Contact avatar / placeholder avatar - Contact display name - Last message snippet (e.g. "Sure, I can start Monday") - Timestamp of last message - Unread badge (count) if messages exist that the current user has not opened/read

Behavior: - Clicking a thread loads that conversation in the Chat Window panel. - If unread, unread badge clears on open (and read receipt is updated in backend). - If user types into search box, the list filters in real-time by contact name or last-message text.

Edge Cases: - If no threads match search → show "No results". - If user has 0 conversations total → show onboarding empty state.

4.2 Chat Window / Active Conversation Panel (Center)

Purpose: - Real-time back-and-forth with the selected contact.

Visible Sections: 1. **Header** (top of conversation) - Contact avatar + name - (Optional) short role/title/ context: e.g. "Web Development", "Branding and Marketing Speaker" - A status indicator (online / last seen) – optional depending on implementation - Possible quick actions (e.g. "Schedule Meeting")

1. **Message History** (scrollable vertical list, newest at bottom)
2. Each message bubble shows:
    - Sender name or "You" label (for self)
    - Text content
    - Attachments (if any) with filename and download link / preview
    - Timestamp (HH:MM or full date if older)
    - Read status indicator for messages you sent (Delivered / Read)
3. System / info messages ("Meeting scheduled for 10 Oct, 3:00 PM IST") are rendered with a distinct neutral style (gray bubble / card).

4. **Message Composer / Input Bar (bottom)**

5. Multiline text input: placeholder like "Type your message..."
6. Attachment / upload button:
    - Allows selecting a file from device (image, pdf, doc, etc.) up to max size (e.g. 2MB)
    - On attach, preview appears inline before sending
7. Send button (paper-plane icon or arrow)
8. (Optional) "Schedule Meeting" button or calendar icon, which opens a modal (see 4.4)

Behavior: - Pressing Enter (or clicking Send) creates a new message document in Firestore for this conversation. - The new message immediately appears in the local UI (optimistic render) and marks conversation as "active" with updated last-message snippet and timestamp in the Conversation List panel. - The recipient will see it in real time.

Validation / Restrictions: - Empty messages cannot be sent (must have text or at least an attachment). - Attachment size must be below limit; if not, show inline error. - Attachment types may be restricted (e.g. block executables).

4.3 Contact Selector / New Conversation Modal

Purpose: - Start a conversation with someone not already in your conversation list.

Trigger: - "Start a new conversation" button (from left sidebar empty state or header of the chat panel). - "Send Enquiry" / "Contact" CTAs elsewhere in the site.

UI Logic: - Shows search box: type a name, company, talent, etc. - Underneath: list of possible contacts (your connections, people who messaged you before, people linked to an assignment you viewed, MSME page admins, etc.). - Selecting a contact: - Opens / creates the conversation thread object in Firestore. - Focuses the Chat Window with that contact pre-selected. - Optionally pre-fills the composer with a default pitch like: - "Hi, I'm interested in this assignment" - "Hi, I saw your MSME page on ConnecWrk"

Edge Cases: - If user tries to message themselves → block. - If user tries to message someone they are not allowed to contact (platform rules / abuse control) → show error.

4.4 Schedule Meeting Modal (In-Chat Scheduling)

Purpose: - Let two users move from text chat to a voice/video call.

Entry: - Button in chat header or in composer toolbar labeled "Schedule Meeting" or similar.

Modal Fields (example expected behavior): - Meeting Title (e.g. "Logo Design Discussion") - Proposed Date & Time (datetime picker) - Duration (e.g. 30 min, 60 min) - Location / Link Type (e.g. "Google Meet") - Notes / Agenda (textarea) - Confirm / Cancel buttons

On Confirm: 1. Meeting metadata is stored (back end can generate or store the meet link). 2. A special "system" message is injected into the chat history that contains: - Meeting title - Date & time in human-readable form - Join link or placeholder text (e.g. "Meeting link will be shared by the client") 3. Both sides can now see it in-line inside the conversation.

Edge Cases: - Invalid datetime -> show inline validation. - User cancels -> modal closes, nothing is posted to chat.

4.5 Empty State / First-Time State

If a user lands on `/messaging/` with 0 chats: - Center panel shows a neutral card: - Title: "No chats available." - Subtitle: "Select a contact to start a conversation" - CTA button: "Start a new conversation"

Connie (the AI assistant) can reuse this copy in responses like: - "You don't have any active conversations yet. Click 'Start a new conversation' to message a freelancer or client."

---

5. Core User Flows

5.1 Client → Freelancer (Assignment Inquiry)

1. Client browses assignments or freelancers and clicks "Send Enquiry" OR "Message".
2. `/messaging/` opens with that freelancer pre-selected.
3. Chat composer is pre-filled with something like:
4. "Hi, I saw your profile and I'd like to discuss a project."
5. Client types details and hits Send.
6. Conversation thread now exists; freelancer sees it under Messages.

5.2 Freelancer → Client (Bid Follow-up / Negotiation)

1. Freelancer places a bid on an assignment.
2. Freelancer wants to clarify scope or timeline.
3. Freelancer goes to Messages and opens the chat with that project owner.
4. They send clarifications:
5. "Can you confirm delivery deadline?"
6. They may attach samples / portfolio screenshots.

5.3 MSME Inquiry → Chat

1. A visitor or prospect clicks "Send Enquiry" on an MSME's profile / CBP page.

2. That triggers creation of a chat between the MSME rep and the visitor.
3. The MSME owner can respond in `/messaging/` and ask questions or send pricing.

5.4 Schedule a Meeting in Chat

1. Inside an active chat, user clicks "Schedule Meeting".
2. User fills in date/time, etc.
3. After confirming, a special meeting card is injected into the thread.
4. Other side can respond with "Works for me" or "Please reschedule" right in chat.

5.5 Attachment Sharing Flow

1. User clicks attachment icon in composer.
2. File picker opens; user selects file.
3. Client-side validation checks: file size / type.
4. Preview (filename + maybe thumbnail).
5. User hits Send; file uploads (e.g. to Firebase Storage) and the chat message is created with a URL pointing to that uploaded asset.
6. Recipient sees a message bubble with the attachment link.

---

6. Data Model (Conceptual)

Below is the conceptual model used by Messaging. Actual implementation uses: - Firebase Authentication (identities / tokens) - Firestore (messages & conversations collections) - Firebase Storage (file uploads) - ConnecWrk backend services for certain enriched actions (like meeting scheduling, abuse monitoring)

6.1 Contact Object (for search / new chat)

Minimal fields per contact in "start conversation" picker:

```
{
  "userId": 456,
  "displayName": "Kalpit Shukla",
  "headline": "Web Development",
  "avatarUrl": "https://.../userphoto/kalpit.jpg",
  "relationship": "freelancer"  // or "client", "msme_owner", etc.
}
```

- These are used to render the user card when selecting who to message. - The platform may resolve this list from: - Your network / contacts - People tied to assignments you viewed or bid on - Admins of MSME pages you visited - Talent Connect profiles you enquired about

6.2 Conversation Object

Represents one 1:1 conversation between two users.

```
{
  "conversationId": "conv_abc123",
  "participants": [101, 456],
  "lastMessage": "Sure, let's start on Monday.",
  "lastMessageAt": "2025-10-27T18:45:00Z",
  "unread": {
    "101": 0,
    "456": 2
  },
  "context": {
    "assignmentId": 789,
    "msmeId": 116,
    "talentProfileId": null
  }
}
```

Key points: - `participants`: the two user IDs. - `lastMessage` / `lastMessageAt`: speed up list rendering. - `unread`: map of unread counts per participant. - `context`: optional references that explain WHY this chat exists (assignment, MSME inquiry, talent booking). This allows Connie to answer "What is this chat about?".

6.3 Message Object

Each message sent in a conversation.

```
{
  "messageId": "msg_987",
  "conversationId": "conv_abc123",
  "senderId": 101,
  "text": "Hi, I saw your profile and I'd like to discuss a project.",
  "attachments": [
    {
      "fileName": "brand_brief.pdf",
      "fileUrl": "https://storage.connecwrk.com/.../brand_brief.pdf",
      "fileSize": 182944
    }
  ],
  "type": "user",           // "user" | "system" | "meeting"
  "createdAt": "2025-10-27T18:40:12Z",
  "readBy": [101],           // array of userIds who have read
  "metadata": {
    "milestoneId": null,
    "bidId": null
  }
}
```

Notes: - `attachments` is optional. - `type` can be: - `user`: normal chat bubble. - `system`: platform-generated info (e.g. "User X joined the conversation"). - `meeting`: structured meeting invite/card. -

`readBy` supports read receipts. - `metadata` can reference assignment/bid/milestone, useful for linking the conversation to ongoing work.

6.4 Meeting Object (System Message)

When user schedules a meeting from the chat, we also create a structured meeting record, and inject a `type: "meeting"` message into chat:

```
{
  "messageId": "msg_meet_222",
  "conversationId": "conv_abc123",
  "senderId": 101,
  "type": "meeting",
  "meeting": {
    "title": "Logo Design Discussion",
    "startTime": "2025-10-28T10:30:00+05:30",
    "durationMinutes": 30,
    "meetingLink": "https://meet.google.com/...",
    "notes": "We'll review scope and deadlines."
  },
  "createdAt": "2025-10-27T18:50:00Z",
  "readBy": [101]
}
```

This message is rendered as a card in the chat: - Title - Date/time - Join link or placeholder text - Notes / agenda

6.5 File Attachment Handling

Rule set: - Max upload size (example: 2MB) - Allowed mime types (images, PDFs, docs) - Store file in Firebase Storage under a path tied to conversationId/messageId - The message bubble shows: - file icon - filename - maybe a small preview if image - download / open link

---

7. API & Realtime Layer (Conceptual)

7.1 Authentication / Identity

- User is already logged into ConnecWrk (JWT/session).
- On loading `/messaging/`, frontend also ensures user is authenticated with Firebase Auth (this can be a behind-the-scenes token exchange).
- Firestore security rules ensure a user can only read/write messages in conversations they participate in.

7.2 Real-time Messaging

- Frontend subscribes to `conversations` collection where `participants` includes current user.
- Frontend subscribes to `messages` subcollection for the currently opened `conversationId`.

- New messages appear instantly (real-time listener).
- Once user focuses a conversation, unread count for that user is reduced to 0 and the user's id is appended to `readBy` for messages.

7.3 Sending a Message (Client Flow)

1. User types text / attaches a file.
2. Clicks Send.
3. Frontend creates a `message` document:
4. conversationId
5. senderId = current user
6. text, attachments, timestamps
7. Firestore write succeeds → optimistic UI already shows the message.
8. Conversation doc gets updated `lastMessage`, `lastMessageAt`, and increments unread counter for the other participant.

7.4 Scheduling a Meeting (Conceptual API)

Flow: 1. User opens "Schedule Meeting" modal in chat. 2. Fills in meeting title, date/time, etc. 3. Frontend calls a backend endpoint such as: - `POST /api/chat/schedule-meeting` - Payload:

```
{
  "conversationId": "conv_abc123",
  "title": "Logo Design Discussion",
  "startTime": "2025-10-28T10:30:00+05:30",
  "durationMinutes": 30,
  "notes": "We'll review scope and deadlines."
}
```

4. Backend may: - Generate a video meeting link (e.g. Google Meet / Zoom / internal meeting link) - Return a structured meeting object 5. Frontend writes a `type: "meeting"` message into Firestore with that object.

Sample response:

```
{
  "success": true,
  "meeting": {
    "meetingLink": "https://meet.google.com/abc-defg-hij",
    "startTime": "2025-10-28T10:30:00+05:30"
  }
}
```

7.5 Error / Retry

If a Firestore write or meeting scheduling call fails: - UI shows a toast like "Failed to send message. Please try again." or "Meeting could not be scheduled." - Message is not appended (or is shown in an error state until retried).

---

8. Delivery / Read / Status Semantics

8.1 Message Lifecycle States (Client-Side Labels)

- Sending... (optimistic state before Firestore confirms)
- Sent (write succeeded)
- Delivered (other user is online and fetched it OR we assume delivered on write)
- Read (the other user opened the conversation; backend appends their userId to `readBy`)

UI expectations: - For your own messages, a small status indicator can show "✔️Seen" or similar once the other user is in `readBy`. - For received messages, unread badges in the Conversation List panel show how many are not yet viewed.

8.2 Conversation-Level Unread Logic

- Each conversation keeps a per-user unread count.
- When current user opens the conversation:
- unread count for that user becomes 0
- read receipts update for all visible messages

This logic drives: - The bold/highlight style in the Conversation List - The numeric badges in headers ("Messages" icon in global nav can show a dot if there are any unread conversations)

---

9. Error Handling / Edge Cases

9.1 Auth / Permission Errors

Scenario: - User tries to open a conversation they're not part of - User tries to start a chat with someone but platform rules block it (spam prevention / user has disabled inbound contact)

Expected behavior: - Show inline error: "You don't have permission to message this user." or "This conversation is no longer available." - Do not render messages.

9.2 Attachment Errors

- File > max size → show: "This file is too large. Please upload a file up to 2MB."
- Disallowed file type → show: "This file type is not supported."
- Upload failed mid-way → show toast "Upload failed. Try again" and keep draft so user can retry.

9.3 Network / Connectivity

- If user is offline when typing:
- Composer remains available but Send is disabled OR message queues locally.
- Show a banner: "You are offline. Messages will be sent when you're back online."

9.4 Abuse / Blocking / Reporting (Future)

- A user can be blocked for harassment or spam.
- Once blocked, conversation may be locked as read-only. The UI should:

- Hide composer
- Display: "You can no longer send messages in this conversation."

(Connie must never encourage bypassing blocks or sharing personal contact details if platform policy discourages it.)

---

10. Connie (AI Assistant) Behavior for Messaging

Connie = the AI/chatbot assistant.

10.1 How Connie Should Talk in Messaging Context

Connie can: - Explain how to contact someone: "Click 'Start a new conversation' and search their name." - Explain read receipts: "If your message shows 'Seen', the other person opened it." - Explain how to negotiate: "You can discuss budget in this chat, or click 'Schedule Meeting' to arrange a call." - Summarize chat context if allowed in future (e.g. "So far you both agreed to start on Monday for INR 10,000.")

Connie must NOT: - Pretend to send a message on the user's behalf (unless we explicitly wire an action for that later). - Reveal private info from other conversations. - Invent guarantees about escrow, payment protection, etc. if not implemented. - Give legal/contract promises.

10.2 Intent Routing Cheatsheet (for Connie)

When user asks... - "Message this freelancer" → Guide them to: `/messaging/` → "Start a new conversation" → search freelancer name - "How do I follow up on my bid?" → `/messaging/` with that client, suggest sending polite clarification of scope/timeline - "Schedule a call" → Use the "Schedule Meeting" button in chat and propose time/date - "Why can't I see replies?" → Explain unread/read logic and that other user may not have seen message yet

10.3 Empty State Coaching

If the user has no chats: - Connie should say: "You don't have any active conversations yet. Click 'Start a new conversation' to reach out to a freelancer, MSME, or talent profile you've viewed."

---

11. Ingestion / RAG Notes (for Engineering)

This section defines how this Messaging spec should be chunked and stored for Connie's retrieval engine.

11.1 Namespace

- Namespace for Pinecone / vector DB: `messaging_module`
- Every chunk from this document should include metadata:

```
{
  "namespace": "messaging_module",
```

```
  "route": "/messaging/",
  "content_type": "product_spec",
  "doc_version": "v1.0",
  "updated_at": "2025-10-28"
}
```

- Rationale: keeps Messaging knowledge isolated from MSME, Freelancer Connect, Assignments, Talent Connect, etc. so retrieval can be routed by intent.

11.2 Chunking Guidance

Break into ~800 word chunks with ~100 word overlap. High-priority standalone chunks: 1. **What Messaging is / why it exists / personas** 2. **Screen layout & UI elements (Conversation List, Chat Window, Composer)** 3. **User flows: starting new chat, negotiating, scheduling a meeting** 4. **Data model examples (Conversation, Message, Meeting)** 5. **Delivery / read receipts logic** 6. **Error handling / abuse / attachment limits** 7. **Connie behavior & safety rules** 8. **RAG namespace + metadata guidance**

These chunks should remain stable in ID naming so incremental re-indexing can skip unchanged text.

11.3 ID / Hash Scheme

For each chunk we store: - `id`: `messaging_module_c{chunk_index}_{short_hash}` - `hash`: full md5/sha1 of text for delta detection - `chunk_index`: numeric order from this doc - `route`: `/messaging/` - `doc_version`: `v1.0`

By keeping this deterministic, we can: - Re-ingest only updated chunks when Messaging UI changes (e.g. max file size changes from 2MB to 5MB) - Add new features like group chat or voice notes later (new chunk indices, same namespace)

---

12. Future / Extension Hooks

(Defined now so Connie doesn't hallucinate and also so eng knows what's next)

1. Group chat / multi-participant threads
2. Conversation `participants` array >2.
3. UI changes: show list of participant avatars, allow naming the thread.
4. Voice notes / audio clips
5. Treat as attachments with a special preview player.
6. In-chat milestone approval / payment request (Assignments module tie-in)
7. A freelancer could send a "Milestone Complete" card.
8. Client could click "Approve" which triggers status update on the milestone.

9. AI Summaries / Action extraction for Connie

10. Connie could generate: "Summary of last 10 messages" or "Next steps".
11. Must never leak content from other conversations.

---

13. Summary

Messaging at `https://connecwrk.com/messaging/` is: - A unified inbox + real-time chat system used across MSME Connect, Freelancer Connect, Assignments, and Talent Connect. - Built for negotiation, clarification, document/portfolio sharing, and meeting scheduling without leaving ConnecWrk. - Backed by conversation/message/meeting objects in Firestore. - Equipped with read receipts, attachment handling, and basic abuse controls. - Designed so Connie (the AI assistant) can: - Explain how to start chats - Coach users through negotiation - Help interpret statuses and empty states - Route users to the right part of the UI quickly

This document, chunked into the `messaging_module` namespace with the metadata schema above, becomes the ground truth for how chat works, what buttons do, and what Connie is and is not allowed to claim in responses.