Tars Full stack Engineer Internship Coding Challenge 2026 - Using AI-Assisted Tool is allowed

💡 Build a real-time Live chat messaging web app using Next.js, TypeScript, Convex, and Clerk, where users can sign up, find other users, and message them in real time. Then briefly explain your code in a video presentation.

🚧 Use of AI-assisted development tools like **Cursor, Claude Code, Windsurf, GitHub Copilot, Codex,** etc. is ALLOWED for assistance in making this app.

Although, you must understand every line of code you submit, as you will be asked to explain your implementation during the interview. If you don't use any AI coding tool at all, it's also fine; you won't be assessed based on them.
If you DO NOT want to use AI Assisted Tools to complete the coding challenges, then go to this other assignment made for manual coding:
🙌🏻 Tars Full stack Engineer Internship Coding Challenge 2026 - Manual coding only - NO AI-Assistance

Both types of assignments are judged in the same way. You can choose any one. Just follow the rules of using or not using the AI-Assisted Tools to complete them.

Tech Stack:

You must use the following:

- **Next.js** (App Router): Docs
- **TypeScript**: Docs
- **Convex** (backend, database, realtime): Docs: Next.js Quickstart
- **Clerk** (authentication): Docs: Convex + Clerk Setup
- For styling, use **Tailwind CSS** as the base. You may use any Tailwind-based component library: shadcn/ui (recommended), Radix UI, Headless UI, or plain Tailwind.

Convex, Clerk, and Vercel have generous free tiers: no cost to you.

Functional Requirements:

Build the following features in order. Each one builds on the previous, so it's easier to implement one by one. You can also take liberty in how to code the features if you believe you know a better solution, descriptions are just mere suggestions.

1. **Authentication**: Set up Clerk so users can sign up (email or social login), log in, and log out. Display the logged-in user's name and avatar. Store user profiles in Convex so other users can discover them.
2. **User List & Search**: Show all registered users (excluding yourself). Add a search bar that filters users by name as you type. Clicking a user opens or creates a conversation with them.
3. **One-on-One Direct Messages**: Users can have private conversations. Messages appear in real time for both sides using Convex subscriptions. Show a sidebar listing all conversations with a preview of the most recent message.
4. **Message Timestamps**: Show when each message was sent. Today's messages show time only (2:34 PM), older messages show date + time (Feb 15, 2:34 PM), and messages from a different year include the year.
5. **Empty States**: Show helpful messages when there's nothing to display: no conversations yet, no messages in a conversation, no search results. Don't leave blank screens.
6. **Responsive Layout**: Desktop: sidebar + chat area side by side. Mobile: conversation list as default view, Tapping a conversation opens full-screen chat with a back button. Use Tailwind responsive breakpoints.
7. **Online/Offline Status**: Show a green indicator next to users who currently have the app open. Update in real time when users come online or go offline.
8. **Typing Indicator**: Show "Alex is typing..." or a pulsing dots animation when the other user is typing. Disappear after ~2 seconds of inactivity or when the message is sent.
9. **Unread Message Count**: Show a badge on each conversation in the sidebar with the number of unread messages. Clear the badge when the user opens that conversation. Update in real time.
10. **Smart Auto-Scroll**: Scroll to the latest message automatically when new messages arrive. If the user has scrolled up to read older messages, don't force-scroll; show a "↓ New messages" button instead.

The following ones(11-14) are optional, but if you have time, you can implement them.

11. **Delete Own Messages**: Users can delete messages they sent. Show "This message was deleted" in italics for all users. Use soft delete: don't remove the record from Convex.
12. **Message Reactions**: Users can react to any message with a fixed set of emojis (👍 ❤️ 😂 😮 😢). Clicking the same reaction again removes it. Show reaction counts below the message.
13. **Loading & Error States**: Show skeleton loaders or spinners while data is loading. If a message fails to send, show an error with a retry option. Handle network/service errors gracefully.
14. **Group Chat**: Users can create a group conversation by picking multiple members and giving it a name. All members see messages in real time. Show group name and member count in the sidebar.

Video Explanation & Presentation:

1. Once done, record a video explanation using Loom and explain your application code in a short **5-minute video**. Loom is free to use — sign up at loom.com.
    a. **Give a short 30-second introduction about yourself at the beginning.**
    b. **Make sure your camera is on, and your face is visible when explaining your code.**
    c. Open your code in your editor, walk through one feature you're proud of, demo it working in the browser, and **make a small live code change** (modify a color, change a label, tweak some logic) to show it reflected in the app.

⚠️ **Submissions without a Loom video will not be reviewed.**

> 💡 Be confident, practice a few times before you record. We watch each video and consider clear communication an important skill. We don't care about your grammar, accent or vocabulary but rather how clearly you're able to get a point across to others.

Evaluation Criteria:

Your submission will be evaluated on the following parameters:

1. **Features Completed:** How many of the listed features work correctly? We'd rather see 10 features done well than 14 done sloppily.
2. **Code Quality:** Clean, readable code. Sensible file structure. Proper use of TypeScript. No obvious copy-paste mess.
3. **Schema Design:** How you structured your Convex tables. Is it clean and extensible?
4. **UI/UX Polish:** Does the app feel good to use? Responsive, no broken layouts, good empty/loading states.
5. **Presentation:** How you communicate and explain your code in the video. Can you explain what you built?

Hosting & Submitting:

1. Push your code to a **public GitHub repository** on github.com.
    a. **Commit frequently. We look at the git history.**
2. Deploy the app on **Vercel** (free tier). You can import your GitHub repo directly into Vercel, and it will build and deploy automatically. Here's how: Vercel — Deploying from GitHub.
    a. **Make sure the deployed version works — we will test it.**
3. Once everything is ready, send an email to `vaibhav@hellotars.com` and cc(`vinit@hellotars.com`):
    - **Subject:** `Fullstack Intern Code Challenge Submission`
    - **Body:**
        - **Full Name:**
        - **Email:**
        - **Link to GitHub Repo:**
        - **Link to Vercel App:**
        - **Link to Loom Video:**
        - **Link to LinkedIn Profile:**

- **AI-Assisted Coding Tool you used: (e.g. Cursor, Claude Code, Codex, etc.)**

You're allowed to submit multiple times if you work on new features later, and want to show them off, We will consider your latest submission, Just send it on the same email thread.

Notes:

1. We think you should be able to build this in 4**-5 hours** with AI tools.
2. Once you submit, we will review the app and code and respond within **3-5 days** with the next steps.
   - If qualified, we will send you an email on what to do next. Make sure to check your emails daily end of the day including spam folder.
3. If you have questions about the assignment, email `vaibhav@hellotars.com` with the subject: `Query about Tars Code Challenge`.