Frontend Interview Coding Challenge: HR Feedback System

Goal: To assess the candidate's ability to build a responsive web application (React) and a cross-platform mobile application (React Native) that communicate using a shared, real-time database (Firebase Firestore).

Time Allotment: 4-6 days (The developer should prioritize core functionality and architecture over complete styling).

Task 1: React Web Application (HR Admin Panel)

Objective: Build a single-page React application for HR staff to view employee feedback data and interact with employees via chat.

Pages & Functionality

1. **Dashboard Page (`/dashboard`):**
   - **Data Display:** Fetch and display all employee feedback records from a Firebase Firestore collection named `feedback`.
     - **Schema (Firestore Document):**
       - `id` (string, generated by Firestore)
       - `date` (timestamp)
       - `employeeName` (string, e.g., "Jane Doe")
       - `score` (number, 1-5)
       - `notes` (string, the employee's written feedback)
   - **Data Table:** Display the feedback in a clean, scrollable table with columns for **Date**, **Employee Name**, **Score**, and a short snippet of the **Notes**.
   - **Real-time KPI Visualization:** Implement a **Pie Chart** that displays the distribution of the `score` field (e.g., 5-Star ratings) across all feedback records in **real-time**. Any new feedback added to the database should update the chart instantly without a page refresh.
2. **Chat Page (`/chat`):**
   - **Employee List:** Display a list of all employees (can be a hardcoded list or derived from the `feedback` collection's `employeeName` field).
   - **Chat View:** When an employee name is clicked, a chat window should open to allow the HR user to send and receive messages from that specific employee.
   - **Backend:** Messages should be stored in a Firestore collection named `messages`. The HR user is the 'sender,' and the employee (from Task 2) is the 'receiver.'

Technical Requirements (Must-Haves)

- **React:** Use functional components and React Hooks (e.g., `useState`, `useEffect`, `useMemo`).
- **Routing:** Use a library like `react-router-dom` to navigate between the two pages.
- **State Management:** Use a simple state management approach (e.g., Context API or global state for Firebase configuration, or just local component state for data fetching).
- **Data Layer:** Integrate directly with **Firebase Firestore** for all data operations (CRUD, real-time listeners for the dashboard chart and chat).
- **Data Visualization:** Use a popular React charting library (e.g., **Recharts, Nivo, Chart.js with React wrapper**) for the Pie Chart.
- **Styling:** Demonstrate basic competence in CSS/SASS/Styled Components/Tailwind CSS for a professional, clean look.

Design Mockup: HR Admin Panel (Low Fidelity)

Element	Description
Header (Top Bar)	Logo on the left. Title: "HR Feedback Admin Panel" in the center.
Navigation (Sidebar)	Vertical navigation on the left (approx. 200px wide). Two links: **Dashboard** and **Chat**.
Dashboard (Main Content)	**Section 1: Real-time Score Chart** (Top Right): A Pie Chart taking up 40% of the content width. The chart should display the percentage breakdown of each score (1, 2, 3, 4, 5). Title: "KPI Score Distribution". **Section 2: Feedback Table** (Bottom/Left): A list/table of all feedback entries with pagination or simple scrolling. Include a search/filter bar above the table.
Chat Page (Main Content)	**Left Column:** A list of employee names (e.g., 30% width). Each name is a clickable link. **Right Column:** The main chat area (e.g., 70% width). A header with the selected employee's name. A message history area. A text input field and a "Send" button at the bottom.

Dashboard UI/UX Design

Acme Co   **Admin**

Employee Feedback

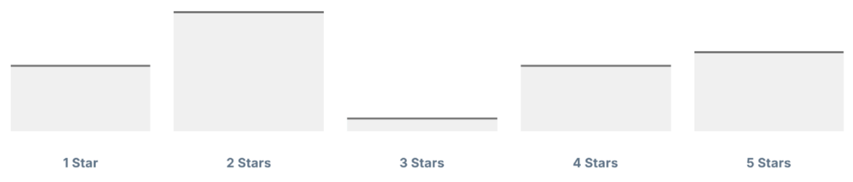
Review and analyze employee feedback data.

Date	Employee Name	Score	Notes
2024-07-26	Ethan Harper	5 stars	Exceptional performance on the recent project. Showed great leadership and problem-solving skills.
2024-07-25	Olivia Bennett	4 stars	Consistently meets expectations. Good team player and reliable.
2024-07-24	Noah Carter	3 stars	Needs improvement in communication. Technical skills are strong but needs to work on expressing ideas clearly.
2024-07-23	Ava Morgan	5 stars	Outstanding work ethic and dedication. A valuable asset to the team.
2024-07-22	Liam Foster	2 stars	Missed deadlines on multiple occasions. Needs to improve time management.
2024-07-21	Isabella Reed	4 stars	Good performance overall. Could take more initiative in projects.
2024-07-20	Jackson Hayes	3 stars	Average performance. Meets basic requirements but lacks innovation.
2024-07-19	Sophia Turner	5 stars	Exceeded expectations. Highly creative and proactive.
2024-07-18	Aiden Parker	1 star	Unsatisfactory performance. Needs significant improvement in all areas.
2024-07-17	Mia Collins	4 stars	Solid performance. Good attention to detail and quality of work.

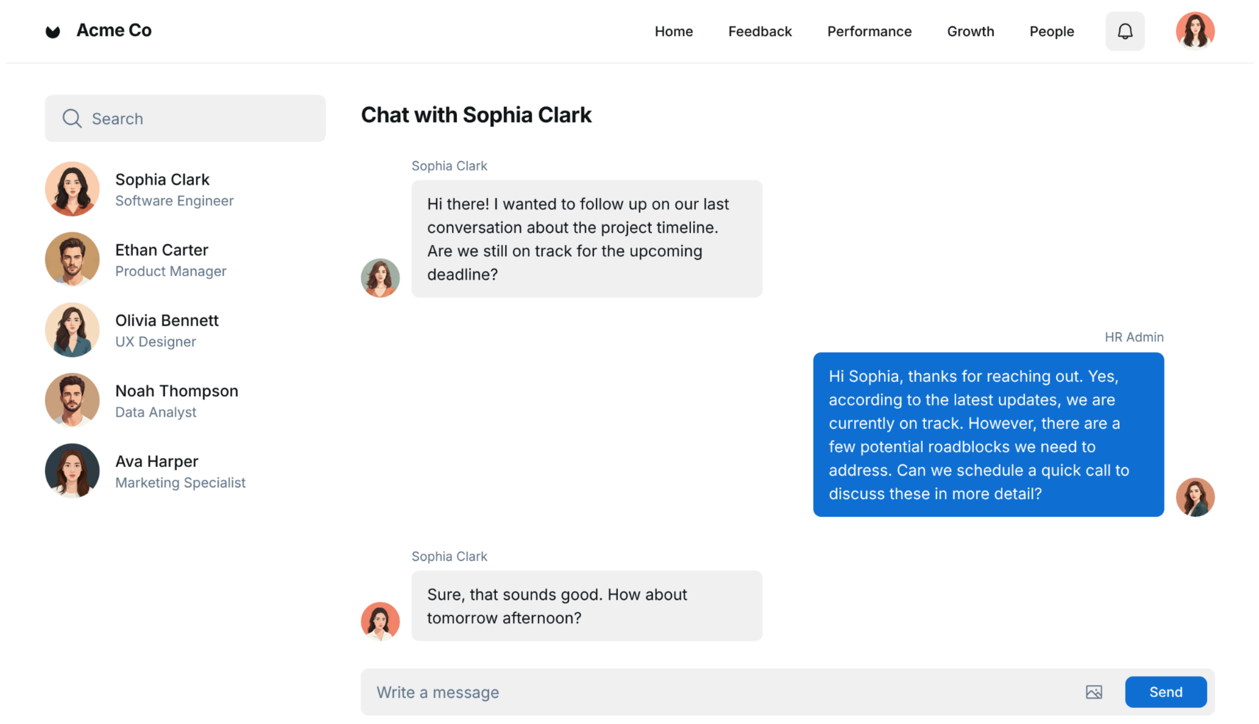
Feedback Score Distribution

4.5

Last 30 Days +10%



Chat UI/UX Design



Sophia Clark
Software Engineer

Ethan Carter
Product Manager

Olivia Bennett
UX Designer

Noah Thompson
Data Analyst

Ava Harper
Marketing Specialist

Chat with Sophia Clark

Sophia Clark

Hi there! I wanted to follow up on our last conversation about the project timeline. Are we still on track for the upcoming deadline?

HR Admin

Hi Sophia, thanks for reaching out. Yes, according to the latest updates, we are currently on track. However, there are a few potential roadblocks we need to address. Can we schedule a quick call to discuss these in more detail?

Sophia Clark

Sure, that sounds good. How about tomorrow afternoon?

Task 2: React Native Mobile Application (Employee Chat)

Objective: Build a simple React Native app for an employee to chat directly with the HR personnel using the same Firestore backend.

Pages & Functionality

1. **Home/Chat Screen:**
    - **Single Chat Interface:** Since this is a one-to-one chat with HR, the app can open directly to the chat screen. The employee's identity can be a single, pre-defined user (or the candidate can implement simple login/user selection if time permits, but it is not a core requirement).
    - **Real-time Communication:** Display messages stored in the `messages` Firestore collection (shared with the web app).
        - The app must filter messages relevant to this specific employee and the HR user.
    - **Message Sending:** An input field and button to allow the employee to send new messages to the HR user in real-time.
    - **Styling:** Clearly distinguish between sent messages (employee) and received messages (HR).

Technical Requirements (Must-Haves)

- **React Native:** Use Expo or a bare React Native project setup.
- **Component Usage:** Utilize standard React Native components (`View`, `Text`, `FlatList` or a dedicated chat UI library like `react-native-gifted-chat` is acceptable for expediting the UI, but custom implementation is preferred).
- **Data Layer:** Integrate directly with **Firebase Firestore** for all chat operations (listening for new messages, adding messages).
- **Platform-Specific Handling:** Demonstrate knowledge of basic mobile UI challenges (e.g., using `KeyboardAvoidingView` to prevent the keyboard from obscuring the message input).

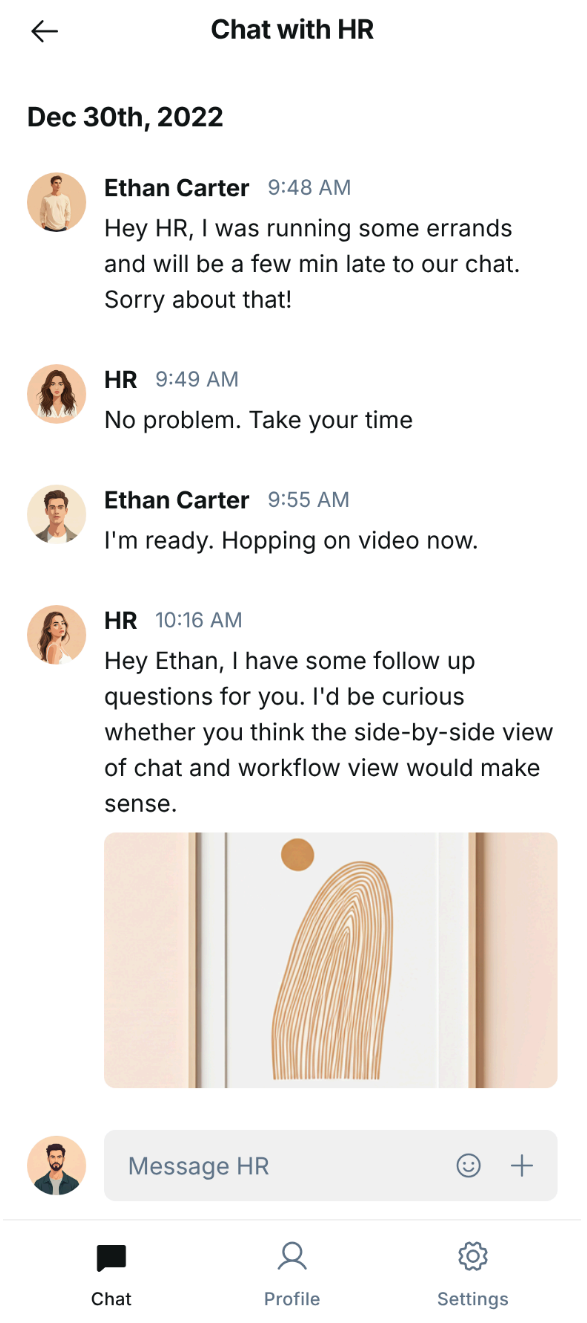
Design Mockup: Mobile Chat App (Low Fidelity)

Element	Description
Header (Top Bar)	Title: "Chat with HR" (or the HR user's name). A back/settings icon is optional.
Message Area	A vertically scrolling list (`FlatList` or equivalent) of messages. The newest message should be at the bottom. **Sent Messages:** Right-aligned bubbles with a distinct background color (e.g., blue). **Received Messages:** Left-aligned bubbles with a neutral background color (e.g., gray/white).
Input Area (Bottom)	Fixed at the bottom of the screen. A text input field for composing the message. A "Send" button (icon or text) to the right of the input field. The entire area should correctly adjust when the keyboard opens.

Evaluation Criteria for the Junior Role

Area	Focus Points
React/RN Fundamentals	Correct use of Hooks (`useEffect`, `useState`). Component structure and separation of concerns. Proper handling of props and state updates.
Firebase Integration	Successful configuration and initialization of Firebase. Correct use of Firestore queries and real-time listeners (`onSnapshot`). Proper data modeling for both feedback and chat.
Real-time Feature	The Pie Chart and the Chat must update in **real-time** on both the web and mobile apps simultaneously.
Application Architecture	Clear separation of UI components, data fetching logic, and routing. Simple, maintainable folder structure.
Code Quality	Clean, well-formatted code. Meaningful variable and function names. Basic error handling (e.g., during data fetching).
UX/UI & Styling	Basic responsiveness in the React app (looks decent on a desktop screen). Clear, accessible, and functional layout in both apps as per the mockups.

Mobile UI/UX Design



Chat with HR

Dec 30th, 2022

Ethan Carter 9:48 AM
Hey HR, I was running some errands and will be a few min late to our chat. Sorry about that!

HR 9:49 AM
No problem. Take your time

Ethan Carter 9:55 AM
I'm ready. Hopping on video now.

HR 10:16 AM
Hey Ethan, I have some follow up questions for you. I'd be curious whether you think the side-by-side view of chat and workflow view would make sense.

Message HR

Firestore Chat Schema Design

1. Top-Level Collection: `conversations`

This collection acts as the directory for all unique one-to-one chats.

Field	Type	Description
Document ID	*string*	**`[employeeId]`** (The unique ID of the employee. This ID is used to fetch the correct chat from both the HR Admin Panel and the Employee Mobile App.)
`participantNames`	*array of strings*	A simple list of names for easy display (e.g., `["HR Personnel Name", "Employee Name"]`).
`lastMessage`	*string*	A snippet of the most recent message (e.g., "OK, I will follow up..."). Used for displaying a quick preview in the Admin Panel's chat list.
`lastMessageTimestamp`	*timestamp*	The time of the last message. Used to sort the conversation list in the Admin Panel.

Example Document in `conversations`:

Field	Value
Document ID	`emp_alice_johnson_42`
`participantNames`	`["Sarah Connor (HR)", "Alice Johnson"]`
`lastMessage`	`I'll send the details by end of day.`
`lastMessageTimestamp`	`October 24, 2025 at 10:05 PM UTC`

---

2. Nested Subcollection: `messages`

This subcollection lives *inside* each conversation document and holds the chronological list of all messages for that specific chat.

Field	Type	Description
Document ID	*string*	**Auto-generated ID** by Firestore.
`senderId`	*string*	A unique identifier for the sender (e.g., `hr_sconnor` or `emp_alice_johnson_42`). This is crucial for distinguishing messages on the UI.
`text`	*string*	The actual message content.
`timestamp`	*timestamp*	The exact time the message was sent. Used to order messages within the chat thread.

Example Message Documents (within `emp_alice_johnson_42`'s `messages` subcollection):

Message 1	Value
Document ID	`msg_1A2b...`
`senderId`	`emp_alice_johnson_42`
`text`	`I had a question about my review score.`

Message 2	Value
Document ID	`msg_3C4d...`
`senderId`	`hr_sconnor`
`text`	`Hi Alice, I can help. What's the question?`

`timestamp`	`October 24, 2025 at 10:04 PM UTC`	`timestamp`	`October 24, 2025 at 10:05 PM UTC`

Implementation Flow (React & React Native)

React Admin Panel (HR User)

1. **Chat List View:** HR subscribes to the `conversations` collection and orders by `lastMessageTimestamp` to see the most active chats first.
2. **Selected Chat View:** When HR clicks on an employee (e.g., with ID `emp_alice_johnson_42`), the app establishes a real-time listener (`onSnapshot`) on the specific subcollection: $$\text{`conversations/emp\_alice\_johnson\_42/messages`}$$
3. **Sending a Message:** HR adds a new document to the same `messages` subcollection with `senderId: "hr_sconnor"`.

React Native App (Employee User)

1. **Chat View:** The Employee's app identifies its own ID (e.g., `emp_alice_johnson_42`) and establishes a real-time listener (`onSnapshot`) on the single, relevant subcollection: $$\text{`conversations/emp\_alice\_johnson\_42/messages`}$$
2. **Sending a Message:** The Employee adds a new document to their specific `messages` subcollection with their own `senderId: "emp_alice\_johnson\_42"`.
3. **UI Logic:** The app filters the messages and renders them on the left or right based on whether the `senderId` matches the current Employee's ID.