ChattriX - Full Project Explanation (For Interview)

🚀What is ChattriX?

ChattriX is a full-stack real-time messaging app built using **React**, **Firebase**, and **WebRTC**. It includes features like: - Realtime chat with images and emoji - Google authentication - Profile update and avatar - Typing indicators and last message preview - WebRTC-based video calling

The entire app is structured with reusable components, uses Firestore for real-time updates, and Firebase Storage for uploading media.

Major Technologies Used:

- **React**: For building the component-based UI
- **Firebase Authentication**: Email & Google login
- **Firebase Firestore**: NoSQL DB for realtime chat and user data
- **Firebase Storage**: For uploading and retrieving image files
- **WebRTC**: For video calling between users
- **emoji-picker-react**: For emoji selection
- **React Context API**: For sharing state across components

Key React Hooks Used

1. `useState()`

```
const [messages, setMessages] = useState([]);
```

- Manages state within a component - Example: Keeps track of chat messages or whether emoji picker is open

2. `useEffect()`

```
useEffect(() => {
  // runs when component mounts or dependencies change
}, [dependency]);
```

- Runs side effects (like fetching data, listening to Firebase updates) - Example: Used to subscribe to Firebase changes with `onSnapshot()`

3. `useContext()`

```
const { userData } = useContext(AppContext);
```

- Lets you access global app data (like logged-in user) without passing props

---

🚡 Firebase Functions

1. `onSnapshot()`

```
onSnapshot(doc(db, "messages", id), (docSnap) => {...})
```

- Real-time listener - Auto-updates the UI whenever Firestore data changes

2. `getDoc()`

```
const docSnap = await getDoc(doc(db, "users", userId));
```

- Fetches data from Firestore once (not real-time)

3. `updateDoc()`

```
await updateDoc(docRef, { lastSeen: Date.now() });
```

- Updates fields in a Firestore document

4. `setDoc()`

```
await setDoc(docRef, userData, { merge: true });
```

- Creates or replaces a document - `merge: true` keeps existing fields

5. `arrayUnion()`

```
arrayUnion(newMessage)
```

- Adds to an array field in Firestore without duplicates

Project Flow Summary

1. Login Flow

- User logs in (email/password or Google)
- If new, `setDoc()` creates a new user in `users/`
- Auth state is managed in `AppContext`

2. Profile Setup

- User uploads avatar (stored in Firebase Storage)
- Saves name, bio, and profile info in Firestore

3. Chat System

- Chats are stored under `chats/{userId}`
- Each chat has `rId`, `lastMessage`, and `messageId`
- Messages go in `messages/{messageId}` as an array

4. Sending Messages

- Messages sent via `updateDoc()` with `arrayUnion()`
- Realtime updates via `onSnapshot()`

5. Media Support

- Image is uploaded to Firebase Storage
- URL saved in the message payload

6. Typing Indicator

- Writing state is managed via a Firestore flag
- Displayed live using `onSnapshot()`

7. Video Call (WebRTC)

- Clicking video button sends `/call/:roomId` message
- Opens `VideoCall.jsx` which uses `RTCPeerConnection`

💈 Components Breakdown

Component	Role
`App.jsx`	Routing & auth observer
`AppContext.js`	Stores and shares app-wide user/chat state

Component	Role
`Login.jsx`	Handles login/signup with Firebase
`ProfileUpdate.jsx`	Lets users update avatar and bio
`Chat.jsx`	Main chat screen (combines all layout parts)
`LeftSidebar.jsx`	Displays chat list, search, and user avatars
`ChatBox.jsx`	Displays current conversation, messages, image preview, typing, emoji
`RightSidebar.jsx`	Shows selected user info and shared images
`upload.js`	Uploads image to Firebase Storage
`VideoCall.jsx`	Handles camera access and peer-to-peer video using WebRTC

How to Answer in an Interview:

> "This is a complete real-time chat app. I used Firebase Auth and Firestore for secure login and chat syncing. Messages update live using onSnapshot(). For images, I store them in Firebase Storage. I've also integrated WebRTC so users can initiate video calls directly from the chat. I manage all shared app state using Context API so components are clean and reusable."

Would you like this exported as a printable PDF or cheat sheet?