Technical Documentation for Hybrid Chat Application

Project Overview

This React Native hybrid chat application displays real-time chat messages from a server API on both Android and iOS devices. The app opens with the latest messages loaded and allows the user to scroll up to load older messages. Key features include efficient pagination, date separation, and responsive design for optimal user experience.

---

1. Project Setup

Dependencies:

- **React Native**: The base framework for the application.
- **Expo**: For rapid development and cross-platform compatibility.
- **API Fetching Libraries** (Fetch): For communication with the backend server API.

In terminal run the below command to initialize the project and start it in the following devices:

```
npx create-expo-app ChatApp -t blank

cd ChatApp

npm run ios

npm run android
```

2. Technical Architecture

The project is structured in a modular way, with components organized in separate folders for easy scalability and readability.

- **Components**: Contains reusable components such as `ChatHeader`, `ChatInput`, `DateSeparator`, and `MessageItem`.
- **Screens**: Includes the primary `ChatScreen` for displaying chat messages.
- **Services**: Houses the API communication logic, particularly `fetchChatMessages` for fetching paginated chat data.

Folder Structure:

```
/chat-app
├── /assets
├── /src
│   ├── /components
│   │   ├── ChatBubble.js
│   │   ├── ChatHeader.js
│   │   ├── ChatInput.js
│   │   ├── DateSeparator.js
│   ├── /screens
│   │   └── ChatScreen.js
│   ├── /api
│   │   └── api.js
│   └── App.js
└── app.json
```

Key Components:

ChatScreen.js

The main screen of the application where chat messages are displayed. It includes:

- **ChatHeader**: Displays the chat header information.
- **FlatList**: Renders messages with pagination.
- **ChatInput**: Allows users to input messages.

ChatHeader.js

A static header component for showing the chat's title and group related info.

ChatInput.js

An input field for users to type and send messages. This component also handles the submission logic.

MessageItem.js

Displays individual messages based on sender/receiver roles.

DateSeparator.js

Separates messages by date, enhancing readability and helping users differentiate between different conversation days.

3. Data Fetching and Pagination

Data is fetched through an API, specifically with the `fetchChatMessages` function, using a paginated approach to avoid loading all messages at once On reaching the end of chats it calls the loadMore function that loads the new set of messages by increasing the page number.It then gets called to fetch the messages.

```
const BASE_URL = 'https://qa.corider.in/assignment';


export const fetchChatMessages = async (page = 0) => {
```

```
 try {
   const response = await
fetch(`${BASE_URL}/chat?page=${page}`);
   if (!response.ok) {
     throw new Error(`HTTP error! status: ${response.status}`);
   }
   const data = await response.json();
   return data;
 } catch (error) {
   console.error('Error fetching chat messages:', error);
   throw error;
 }
};
const handleLoadMore = () => {
    setPage(prev => prev + 1);
    loadMessages(page + 1);
 };
```

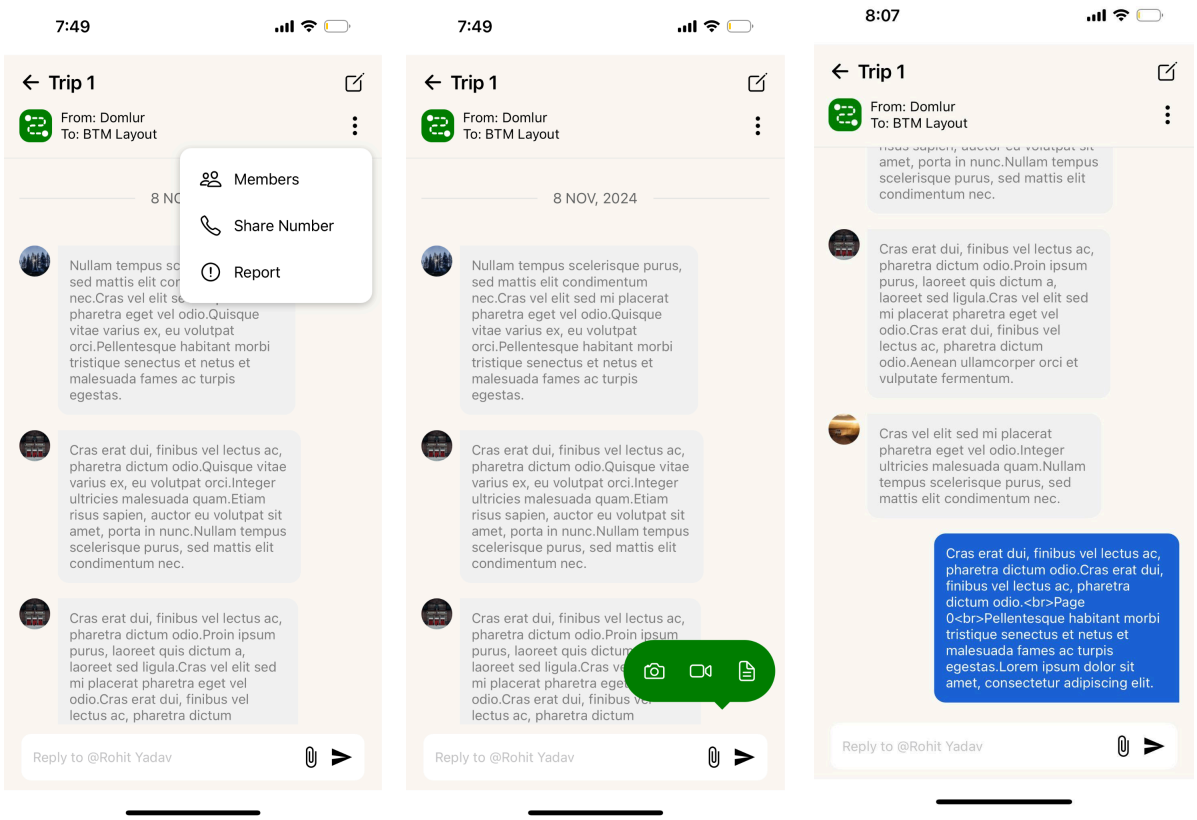
Optimizations

Pagination: Only renders a limited number of messages at once, reducing memory usage. Currently since we used mocked data we kept the page to be default to zero

```
 const loadMessages = async (pageNum = 0) => {
   if (loading) return;
   setLoading(true);
   try {
     const data = await fetchChatMessages(pageNum);
     if (pageNum === 0) {
       setMessages(data.chats);
       setTripInfo({ from: data.from, to: data.to });
     }
     else {
       setMessages(prev => [...data.chats, ...prev]);}}
catch (error) {
     console.error(error);}
finally {
     setLoading(false);}}
useEffect(() => {
   loadMessages();
 }, []);
```

```
const handleLoadMore = () => {
    setPage(prev => prev + 1);
    loadMessages(page + 1);
};
```

Screenshots and Walkthrough



Video link : Screencast_Application.MOV