Code Explanation

The required dependencies, including React, Firebase, and the React hooks **useAuthState** and **useCollectionData**, are imported by the code. The App component, which houses the chat room's primary layout, is then set up. Using the **useAuthState** hook, it determines if the user is logged in and, depending on the authentication state, either the **ChatRoom** or **SignIn** components are displayed.

```
import React, { useRef, useState } from 'react';
import './App.css';
import {auth, firestore} from "./firebase/firebaseConfig";
import firebase from "firebase/app";
import { useAuthState } from 'react-firebase-hooks/auth';
import { useCollectionData } from 'react-firebase-hooks/firestore';
```

Fig:1 *import libraries*

The **SignIn** component renders a button that allows users to sign in with their Google account using the **signInWithGoogle** function. The **SignOut** component renders a button that allows users to sign out if they are already signed in.

```
function SignIn() {

  const signInWithGoogle = () => {
    const provider = new firebase.auth.GoogleAuthProvider();
    auth.signInWithPopup(provider);
  }

  return (
    <>
      <button className="sign-in" onClick={signInWithGoogle}>Sign in with Google</button>

    </>
  )

}

function SignOut() {
  return auth.currentUser && (
    <button className="sign-out" onClick={() => auth.signOut()}>Sign Out</button>
  )
}
```

Fig:2 Signin And signout

The primary conversation interface is rendered by the **ChatRoom** component. In order to scroll to the bottom of the conversation when a new message is submitted, it creates a reference to the fake element using the **useRef** hook. The **useCollectionData** hook is then used to obtain the

messages from the Fire store database, and the **ChatMessage** component is used to render each message.

```
function ChatRoom() {
  const dummy = useRef();
  const messagesRef = firestore.collection('messages');
  const query = messagesRef.orderBy('createdAt');

  const [messages] = useCollectionData(query, { idField: 'id' });

  const [formValue, setFormValue] = useState('');


  const sendMessage = async (e) => {
    e.preventDefault();

    const { uid, photoURL } = auth.currentUser;

    await messagesRef.add({
      text: formValue,
      createdAt: firebase.firestore.FieldValue.serverTimestamp(),
      uid,
      photoURL
    })

    setFormValue('');
    dummy.current.scrollIntoView({ behavior: 'smooth' });
  }

  return (<>
    <main>

      {messages && messages.map(msg => <ChatMessage key={msg.id} message={msg} />)}

      <span ref={dummy}></span>

    </main>

    <form onSubmit={sendMessage}>

      <input value={formValue} onChange={(e) => setFormValue(e.target.value)} placeholder="say something nice" />

      <button type="submit" disabled={!formValue}>Send</button>

    </form>
  </>)
}
```

Fig:1 *Chat room*

When a user presses the send button, a new message is added to the Firestore database using the sendMessage function. It uses the add function to add the message to the Firestore database after first retrieving the current user's uid and photoURL. Additionally, it uses the dummy.current.scrollIntoView method to scroll to the bottom of the chat and resets the formValue state.

Every message in the chat is rendered using the ChatMessage component. It applies the proper styling and determines if the message was delivered or received using the messageClass variable.

```
function ChatMessage(props) {
  const { text, uid, photoURL } = props.message;

  const messageClass = uid === auth.currentUser.uid ? 'sent' : 'received';

  return (<>
    <div className={`message ${messageClass}`}>
      <img src={photoURL || 'https://api.adorable.io/avatars/23/abott@adorable.png'} alt = "msg" />
      <p>{text}</p>
    </div>
  </>)
}
```

Fig:1 ***Chat room and send function***