Filename: AuthManager.java

```
package com.example.protegotinyever.util;


import android.content.Context;

import androidx.annotation.NonNull;

import com.google.android.gms.tasks.Task;

import com.google.firebase.auth.AuthResult;

import com.google.firebase.auth.FirebaseAuth;

import com.google.firebase.auth.FirebaseUser;

import com.google.firebase.auth.UserProfileChangeRequest;


public class AuthManager {

    private static AuthManager instance;

    private final FirebaseAuth auth;

    private final Context context;


    private AuthManager(Context context) {

        this.context = context.getApplicationContext();

        this.auth = FirebaseAuth.getInstance();

    }


    public static synchronized AuthManager getInstance(Context context) {

        if (instance == null) {

            instance = new AuthManager(context.getApplicationContext());

        }

        return instance;
```

```
}


public Task<AuthResult> signUp(String email, String password, String username, String phone) {
    return auth.createUserWithEmailAndPassword(email, password)
            .addOnSuccessListener(authResult -> {
                FirebaseUser user = authResult.getUser();
                if (user != null) {
                    UserProfileChangeRequest profileUpdates = new UserProfileChangeRequest.Builder()
                            .setDisplayName(username)
                            .build();
                    user.updateProfile(profileUpdates);

                    // Save additional user info to Realtime Database
                    FirebaseClient firebaseClient = new FirebaseClient(username, phone);
                    firebaseClient.saveUser(username, phone, true, () -> {});
                }
            });
}


public Task<AuthResult> signIn(String email, String password) {
    return auth.signInWithEmailAndPassword(email, password);
}


public void signOut() {
    FirebaseUser user = auth.getCurrentUser();
    if (user != null) {
        String username = user.getDisplayName();
```

```
            // Set user offline in Realtime Database
            FirebaseClient firebaseClient = new FirebaseClient(username, "");
            firebaseClient.saveUser(username, "", false, () -> {
                auth.signOut();
                SessionManager.getInstance(context).clearSession();
            });
        } else {
            auth.signOut();
            SessionManager.getInstance(context).clearSession();
        }
    }

    public Task<Void> sendPasswordResetEmail(String email) {
        return auth.sendPasswordResetEmail(email);
    }

    public FirebaseUser getCurrentUser() {
        return auth.getCurrentUser();
    }

    public boolean isLoggedIn() {
        return auth.getCurrentUser() != null;
    }
}
```

Filename: ChatActivity.java

```
package com.example.protegotinyever.act;


import android.app.Dialog;

import android.content.Intent;

import android.net.Uri;

import android.os.Build;

import android.os.Bundle;

import android.util.Log;

import android.view.MenuItem;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ProgressBar;

import android.widget.TextView;

import android.widget.Toast;


import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import androidx.recyclerview.widget.LinearLayoutManager;

import androidx.recyclerview.widget.RecyclerView;


import com.example.protegotinyever.R;

import com.example.protegotinyever.adapt.MessageEntity;

import com.example.protegotinyever.mode.MessageModel;

import com.example.protegotinyever.tt.MessageAdapter;

import com.example.protegotinyever.util.DataChannelHandler;
import com.example.protegotinyever.webrtc.WebRTCClient;


import org.webrtc.DataChannel;


import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;


public class ChatActivity extends AppCompatActivity implements WebRTCClient.ProgressListener {
    private static final int YOUR_PERMISSION_REQUEST_CODE = 122;
    private RecyclerView chatRecyclerView;
    private EditText messageInput;
    private MessageAdapter messageAdapter;
    private List<MessageModel> messageList;
    private DataChannelHandler dataChannelHandler;
    private WebRTCClient webRTCClient;
    private String currentUser;
    private String peerUsername;
    private TextView connectionStatus;
    private Button sendButton;
    private static final int FILE_PICKER_REQUEST_CODE = 1;
    private Dialog progressDialog;
    private ProgressBar progressBar;
    private TextView progressText;
    private List<String> messageQueue = new ArrayList<>();
    private boolean isFileTransferInProgress = false;
    private ExecutorService messageQueueExecutor = Executors.newSingleThreadExecutor();


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chat);


        chatRecyclerView = findViewById(R.id.crv);
        messageInput = findViewById(R.id.messageInput);
        sendButton = findViewById(R.id.sendButton);
        connectionStatus = findViewById(R.id.connectionStatus);


        currentUser = "You";
        peerUsername = getIntent().getStringExtra("peerUsername");
        if (peerUsername == null || peerUsername.isEmpty()) {
            peerUsername = "Peer";
        }
        Log.d("ChatActivity", "Opened chat with peer: " + peerUsername);


        webRTCClient = WebRTCClient.getInstance(this, null);
        webRTCClient.setProgressListener(this);
        dataChannelHandler = DataChannelHandler.getInstance(getApplicationContext());


        getOnBackPressedDispatcher().addCallback(this, new androidx.activity.OnBackPressedCallback(true) {
            @Override
            public void handleOnBackPressed() {
                if (dataChannelHandler != null) {
                    dataChannelHandler.setOnMessageReceivedListener(null);
                    dataChannelHandler.setStateChangeListener(null);
                }
                finish();
            }
        });

        requestPermissions();
        setupToolbar();
        setupDataChannel();
        setupRecyclerView();
        setupProgressDialog();

        sendButton.setOnClickListener(view -> sendMessage());
        loadMessageHistory();

        findViewById(R.id.sendFileButton).setOnClickListener(v -> pickFile());
    }

    private void requestPermissions() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            String[] permissions;
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
                permissions = new String[]{
                        android.Manifest.permission.READ_MEDIA_IMAGES,
                    android.Manifest.permission.READ_MEDIA_VIDEO,
                    android.Manifest.permission.READ_MEDIA_AUDIO
                };
            } else {
                permissions = new String[]{
                    android.Manifest.permission.READ_EXTERNAL_STORAGE,
                    android.Manifest.permission.WRITE_EXTERNAL_STORAGE
                };
            }
            requestPermissions(permissions, YOUR_PERMISSION_REQUEST_CODE);
        }
    }
    private void setupToolbar() {
        TextView peerUsernameView = findViewById(R.id.peerUsername);
        peerUsernameView.setText(peerUsername.toUpperCase());
        androidx.appcompat.widget.Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        if (getSupportActionBar() != null) {
            getSupportActionBar().setDisplayHomeAsUpEnabled(true);
            getSupportActionBar().setDisplayShowHomeEnabled(true);
        }
    }

    private void setupProgressDialog() {
        progressDialog = new Dialog(this, R.style.TransparentDialog);
        progressDialog.setContentView(R.layout.dialog_progress);
        progressDialog.setCancelable(false);
```

```
        progressBar = progressDialog.findViewById(R.id.progressBar);
        progressText = progressDialog.findViewById(R.id.progressText);
        progressBar.setMax(100);
    }

    private void setupDataChannel() {
        dataChannelHandler.setCurrentPeer(peerUsername);
        dataChannelHandler.setOnMessageReceivedListener(message -> {
            Log.d("ChatActivity", "Received message: " + message);
                        addMessageToUI(new MessageModel(peerUsername.equals(currentUser) ? currentUser :
peerUsername, message, System.currentTimeMillis()));
            if (progressDialog.isShowing() && message.contains("Received file:")) {
                progressDialog.dismiss();
            }
        });
        dataChannelHandler.setStateChangeListener(state -> {
            Log.d("ChatActivity", "DataChannel state: " + state);
            runOnUiThread(() -> updateConnectionStatus(state));
        });

        DataChannel channel = dataChannelHandler.getDataChannel(peerUsername);
        if (channel != null) {
            Log.d("ChatActivity", "Initial DataChannel state: " + channel.state());
            updateConnectionStatus(channel.state());
        } else {
            Log.w("ChatActivity", "No DataChannel for " + peerUsername);
        }
        webRTCClient.setWebRTCListener(new WebRTCClient.WebRTCListener() {
            @Override
            public void onConnected() {
                Log.d("ChatActivity", "WebRTC connected");
                runOnUiThread(() -> updateConnectionStatus(DataChannel.State.OPEN));
            }

            @Override
            public void onConnectionFailed() {
                Log.d("ChatActivity", "WebRTC connection failed");
                runOnUiThread(() -> {
                    updateConnectionStatus(DataChannel.State.CLOSED);
                    if (progressDialog.isShowing()) {
                        progressDialog.dismiss();
                                    Toast.makeText(ChatActivity.this, "Connection lost during file transfer",
Toast.LENGTH_LONG).show();
                    }
                });
            }

            @Override
            public void onMessageReceived(String message, String peerUsername) {
                    Log.d("ChatActivity", "WebRTC message received from " + peerUsername + ": " + message + "
(ignored for UI)");
            }
            @Override
            public void onFileSent(String filePath, String fileName) {
                Log.d("ChatActivity", "File sent successfully: " + filePath);
                runOnUiThread(() -> {
                    addMessageToUI(new MessageModel(currentUser, "Sent file: " + fileName + " at " + filePath,
System.currentTimeMillis()));
                    if (progressDialog.isShowing()) {
                        progressDialog.dismiss();
                    }
                    Toast.makeText(ChatActivity.this, "File sent: " + fileName, Toast.LENGTH_SHORT).show();
                });
            }
        });
    }

    private void setupRecyclerView() {
        messageList = new ArrayList<>();
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        layoutManager.setStackFromEnd(true);
        chatRecyclerView.setLayoutManager(layoutManager);
        messageAdapter = new MessageAdapter(messageList, currentUser, this, this::openFile);
        chatRecyclerView.setAdapter(messageAdapter);
    }

    private void updateConnectionStatus(DataChannel.State state) {
        String statusText;
        int statusColor;
    boolean enableSend = true;


    switch (state) {
      case OPEN:
        statusText = "SECURE CONNECTION ACTIVE";
        statusColor = getColor(R.color.success_green);
        break;
      case CONNECTING:
          statusText = "ESTABLISHING CONNECTION - MESSAGES WILL BE DELIVERED WHEN
PEER IS ONLINE";
        statusColor = getColor(R.color.warning_yellow);
        break;
      case CLOSING:
      case CLOSED:
        statusText = "OFFLINE - MESSAGES WILL BE DELIVERED WHEN PEER IS ONLINE";
        statusColor = getColor(R.color.warning_yellow);
        break;
      default:
        statusText = "CONNECTION ERROR - MESSAGES WILL BE SAVED";
        statusColor = getColor(R.color.error_red);
        break;
    }


    connectionStatus.setText(statusText);
    connectionStatus.setTextColor(statusColor);
    sendButton.setEnabled(enableSend);
    messageInput.setEnabled(enableSend);
```

```
  }


  private void sendMessage() {
      String messageText = messageInput.getText().toString().trim();
      if (!messageText.isEmpty()) {
          if (isFileTransferInProgress) {
              // Queue the message if file transfer is in progress
              messageQueue.add(messageText);
              messageInput.setText("");
                              Toast.makeText(this, "Message will be sent after file transfer completes",
Toast.LENGTH_SHORT).show();
          } else {
              // Send message immediately if no file transfer is in progress
              Log.d("ChatActivity", "Sending message to " + peerUsername + ": " + messageText);
              webRTCClient.sendEncryptedMessage(messageText, peerUsername);
              messageInput.setText("");
              addMessageToUI(new MessageModel(currentUser, messageText, System.currentTimeMillis()));

              DataChannel channel = dataChannelHandler.getDataChannel(peerUsername);
              if (channel == null || channel.state() != DataChannel.State.OPEN) {
                  showOfflineMessageIndicator();
              }
          }
      }
  }


  private void processMessageQueue() {
      if (!messageQueue.isEmpty() && !isFileTransferInProgress) {
          messageQueueExecutor.execute(() -> {
              while (!messageQueue.isEmpty() && !isFileTransferInProgress) {
                  String message = messageQueue.remove(0);
                  webRTCClient.sendEncryptedMessage(message, peerUsername);
                          runOnUiThread(() -> addMessageToUI(new MessageModel(currentUser, message,
System.currentTimeMillis())));
                  try {
                      Thread.sleep(100); // Small delay between messages
                  } catch (InterruptedException e) {
                      Thread.currentThread().interrupt();
                      break;
                  }
              }
          });
      }
  }

  private void showOfflineMessageIndicator() {
              Toast.makeText(this, "Message will be delivered when peer comes online",
Toast.LENGTH_SHORT).show();
  }

  private void loadMessageHistory() {
      new Thread(() -> {
          List<MessageEntity> history = dataChannelHandler.getMessageHistory(peerUsername);
          Log.d("ChatActivity", "Loading message history for " + peerUsername + ", size: " + history.size());
          runOnUiThread(() -> {
              messageList.clear();
              for (MessageEntity msg : history) {
                  messageList.add(new MessageModel(msg.getSender(), msg.getMessage(),
msg.getTimestamp()));
              }
              messageAdapter.notifyDataSetChanged();
              chatRecyclerView.scrollToPosition(messageList.size() - 1);
          });
      }).start();
  }

  public void addMessageToUI(MessageModel message) {
      Log.d("ChatActivity", "Adding message to UI: " + message.getText() + " from " + message.getSender());
      runOnUiThread(() -> {
          messageList.add(message);
          messageAdapter.notifyItemInserted(messageList.size() - 1);
          chatRecyclerView.scrollToPosition(messageList.size() - 1);
      });
  }

  @Override
  public boolean onOptionsItemSelected(@NonNull MenuItem item) {
      if (item.getItemId() == android.R.id.home) {
          getOnBackPressedDispatcher().onBackPressed();
          return true;
      }
      return super.onOptionsItemSelected(item);
  }

  @Override
  protected void onDestroy() {
      super.onDestroy();
      messageQueueExecutor.shutdown();
      if (dataChannelHandler != null) {
          dataChannelHandler.setOnMessageReceivedListener(null);
          dataChannelHandler.setStateChangeListener(null);
      }
      if (progressDialog != null && progressDialog.isShowing()) {
          progressDialog.dismiss();
      }
  }

  public String getPeerUsername() {
      return peerUsername;
  }

  private void pickFile() {
      Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
      intent.setType("*/*");
      startActivityForResult(intent, FILE_PICKER_REQUEST_CODE);
  }

  @Override
```

```
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
      super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == FILE_PICKER_REQUEST_CODE && resultCode == RESULT_OK && data !=
null) {
          Uri uri = data.getData();
          sendFile(uri);
      }
  }

  private void sendFile(Uri fileUri) {
      try {
          String fileName = getFileNameFromUri(fileUri);
          String fileType = getContentResolver().getType(fileUri);
          if (fileType == null) fileType = "application/octet-stream";

          long fileSize = getContentResolver().openFileDescriptor(fileUri, "r").getStatSize();
          if (fileSize > 1024 * 1024 * 1024) { // 1 GB limit
              throw new IOException("File size exceeds 1 GB limit: " + (fileSize / (1024 * 1024)) + " MB");
          }

          // Show progress dialog immediately
          progressDialog.show();
          progressBar.setProgress(0);
          progressText.setText(String.format("Preparing to send '%s': 0%%", fileName, 0));

          // Offload file sending to a background thread
          ExecutorService executor = Executors.newSingleThreadExecutor();
        String finalFileType = fileType;
        executor.execute(() -> {
            try {
                webRTCClient.sendFile(fileUri, peerUsername, fileName, finalFileType);
            } catch (Exception e) {
                Log.e("ChatActivity", "Error sending file: " + e.getMessage());
                runOnUiThread(() -> {
                                Toast.makeText(ChatActivity.this, "Failed to send file: " + e.getMessage(),
Toast.LENGTH_LONG).show();
                    if (progressDialog.isShowing()) {
                        progressDialog.dismiss();
                    }
                });
            }
        });
    } catch (Exception e) {
        Log.e("ChatActivity", "Error initiating file send: " + e.getMessage());
        Toast.makeText(this, "Failed to send file: " + e.getMessage(), Toast.LENGTH_LONG).show();
        if (progressDialog.isShowing()) {
            progressDialog.dismiss();
        }
    }
}

private String getFileNameFromUri(Uri uri) {
    String fileName = "unknown_file";
    try {
        android.database.Cursor cursor = getContentResolver().query(uri, null, null, null, null);
        if (cursor != null && cursor.moveToFirst()) {
            int nameIndex = cursor.getColumnIndex(android.provider.OpenableColumns.DISPLAY_NAME);
            if (nameIndex != -1) {
                fileName = cursor.getString(nameIndex);
            }
            cursor.close();
        }
    } catch (Exception e) {
        Log.e("ChatActivity", "Error getting file name: " + e.getMessage());
    }
    return fileName;
}

private void openFile(String filePath) {
    try {
        File file = new File(filePath);
        if (!file.exists()) {
            Toast.makeText(this, "File not found: " + filePath, Toast.LENGTH_LONG).show();
            return;
        }

        Uri fileUri;
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.N) {
            fileUri = androidx.core.content.FileProvider.getUriForFile(
                this,
                getApplicationContext().getPackageName() + ".provider",
                file
            );
        } else {
            fileUri = Uri.fromFile(file);
        }

        String mimeType = getContentResolver().getType(fileUri);
        if (mimeType == null) {
            mimeType = getMimeTypeFromExtension(filePath);
        }

        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setDataAndType(fileUri, mimeType);
        intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

        // Create chooser to handle the file
        Intent chooser = Intent.createChooser(intent, "Open file with");
        if (intent.resolveActivity(getPackageManager()) != null) {
            startActivity(chooser);
        } else {
            Toast.makeText(this, "No app found to open this file type", Toast.LENGTH_SHORT).show();
        }
    } catch (Exception e) {
        Log.e("ChatActivity", "Error opening file: " + e.getMessage());
        Toast.makeText(this, "Error opening file: " + e.getMessage(), Toast.LENGTH_LONG).show();
    }
```

```
    }


    private String getMimeTypeFromExtension(String filePath) {
        String extension = android.webkit.MimeTypeMap.getFileExtensionFromUrl(filePath);
        if (extension != null) {
                                                                                            return
android.webkit.MimeTypeMap.getSingleton().getMimeTypeFromExtension(extension.toLowerCase());
        }
        return "application/octet-stream";
    }


    @Override
    public void onProgress(String operation, int progress, String fileName) {
        runOnUiThread(() -> {
            isFileTransferInProgress = progress < 100;
            if (progress == 0) {
                progressDialog.show();
            }
            progressBar.setProgress(progress);
            progressText.setText(operation + ": " + progress + "% - " + fileName);


            if (progress >= 100) {
                progressDialog.dismiss();
                // Process queued messages after file transfer completes
                processMessageQueue();
            }
        });
    }

}
```

Filename: ChatDatabase.java

```
package com.example.protegotinyever.db;


import android.content.Context;
import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;


import com.example.protegotinyever.adapt.MessageEntity;


@Database(entities = {MessageEntity.class}, version = 1, exportSchema = false)
public abstract class ChatDatabase extends RoomDatabase {
    private static final String DATABASE_NAME = "chat_db";
    private static ChatDatabase instance;
    private int rea = 1;

    public abstract MessageDao messageDao();

    public static synchronized ChatDatabase getInstance(Context context) {
        if (instance == null) {
            instance = Room.databaseBuilder(
                context.getApplicationContext(),
                ChatDatabase.class,
                DATABASE_NAME
            ).build();
        }
        return instance;

    }

}
```

Filename: ChatMessage.java

```
package com.example.protegotinyever.tt;


public class ChatMessage {
    private String sender;
    private String receiver;
    private String message;
    private int rea = 1;

    public ChatMessage() { }

    public ChatMessage(String sender, String receiver, String message) {
        this.sender = sender;
        this.receiver = receiver;
        this.message = message;
    }

    public String getSender() { return sender; }
    public String getReceiver() { return receiver; }
    public String getMessage() { return message; }
}
```

Filename: ChatsFragment.java

```
package com.example.protegotinyever.act;


import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.annotation.Nullable;

import androidx.fragment.app.Fragment;

import androidx.recyclerview.widget.LinearLayoutManager;

import androidx.recyclerview.widget.RecyclerView;


import com.example.protegotinyever.R;

import com.example.protegotinyever.service.ConnectionManager;

import com.example.protegotinyever.tt.UserAdapter;

import com.example.protegotinyever.tt.UserModel;

import com.example.protegotinyever.webrtc.WebRTCClient;


import java.util.ArrayList;

import java.util.List;


public class ChatsFragment extends Fragment {

    private RecyclerView recyclerView;

    private TextView noChatsText;
```

```
  private UserAdapter adapter;
  private List<UserModel> connectedUsers = new ArrayList<>();
  private WebRTCClient webRTCClient;
  private UserAdapter.OnUserClickListener userClickListener;
  private boolean isViewCreated = false;
  private List<UserModel> pendingUsers;
  private ConnectionManager connectionManager;

  @Override
  public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    connectionManager = ConnectionManager.getInstance(requireContext());
  }

  public static ChatsFragment newInstance() {
    return new ChatsFragment();
  }

  @Override
      public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_chats, container, false);
    recyclerView = view.findViewById(R.id.chatsRecyclerView);
    noChatsText = view.findViewById(R.id.noChatsText);
    return view;
  }
```

```
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    isViewCreated = true;
    setupRecyclerView();
    if (pendingUsers != null) {
        updateUsers(pendingUsers);
        pendingUsers = null;
    }
}

@Override
public void onDestroyView() {
    super.onDestroyView();
    isViewCreated = false;
    recyclerView = null;
    noChatsText = null;
    adapter = null;
}

private void setupRecyclerView() {
    if (recyclerView != null && getContext() != null) {
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
        adapter = new UserAdapter(connectedUsers, userClickListener, false);
        recyclerView.setAdapter(adapter);
    }
}
```

```
public void setWebRTCClient(WebRTCClient client) {
    this.webRTCClient = client;
}

public void setUserClickListener(UserAdapter.OnUserClickListener listener) {
    this.userClickListener = listener;
    if (adapter != null && recyclerView != null) {
        adapter = new UserAdapter(connectedUsers, listener, false);
        recyclerView.setAdapter(adapter);
    }
}

public void updateUsers(List<UserModel> users) {
    if (!isViewCreated) {
        pendingUsers = users;
        return;
    }

    if (getActivity() == null || !isAdded()) {
        return;
    }

    connectedUsers.clear();
    // Add all users that were ever connected
    for (UserModel user : users) {
        if (connectionManager.isUserConnected(user.getUsername())) {
            connectedUsers.add(user);
        }
    }


    getActivity().runOnUiThread(() -> {
        if (isViewCreated && isAdded()) {
            if (noChatsText != null) {
                noChatsText.setVisibility(connectedUsers.isEmpty() ? View.VISIBLE : View.GONE);
            }
            if (recyclerView != null) {
                recyclerView.setVisibility(connectedUsers.isEmpty() ? View.GONE : View.VISIBLE);
            }
            if (adapter != null) {
                adapter.notifyDataSetChanged();
            }
        }
    });
  }
}
```

Filename: ConnectActivity.java

```
package com.example.protegotinyever.act;


import android.Manifest;

import android.content.ContentResolver;

import android.content.Intent;

import android.content.pm.PackageManager;

import android.database.Cursor;

import android.os.Build;

import android.os.Bundle;

import android.os.Handler;

import android.os.Looper;

import android.provider.ContactsContract;

import android.util.Log;

import android.view.Menu;

import android.view.MenuItem;

import android.widget.Toast;

import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import androidx.core.app.ActivityCompat;

import androidx.core.content.ContextCompat;

import androidx.viewpager2.widget.ViewPager2;

import com.example.protegotinyever.R;

import com.example.protegotinyever.service.ConnectionManager;

import com.example.protegotinyever.service.WebRTCService;

import com.example.protegotinyever.tt.UserAdapter;
import com.example.protegotinyever.tt.UserModel;

import com.example.protegotinyever.util.FirebaseClient;

import com.example.protegotinyever.util.SessionManager;

import com.example.protegotinyever.webrtc.WebRTCClient;

import com.google.android.material.tabs.TabLayout;

import com.google.android.material.tabs.TabLayoutMediator;

import androidx.activity.OnBackPressedCallback;

import org.webrtc.DataChannel;

import java.util.ArrayList;

import java.util.List;

import android.view.View;

import com.example.protegotinyever.util.ThemeManager;


public class ConnectActivity extends AppCompatActivity {

   private WebRTCClient webRTCClient;

   private FirebaseClient firebaseClient;

   private ViewPager2 viewPager;

   private TabLayout tabLayout;

   private ConnectPagerAdapter pagerAdapter;

   private ConnectionManager connectionManager;

   private static final int CONTACTS_PERMISSION_CODE = 100;

   private static final int NOTIFICATION_PERMISSION_CODE = 101;

   private boolean hasCheckedPermissions = false;

   private int rea = 1;

   private Handler handler = new Handler(Looper.getMainLooper());

   private ThemeManager themeManager;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // Initialize theme before setting content view
    themeManager = ThemeManager.getInstance(this);
    themeManager.initializeTheme();

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_connect);

    androidx.appcompat.widget.Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    viewPager = findViewById(R.id.viewPager);
    tabLayout = findViewById(R.id.tabLayout);

    String username = getIntent().getStringExtra("username");
    String phoneNumber = getIntent().getStringExtra("phoneNumber");
    firebaseClient = new FirebaseClient(username, phoneNumber);
    webRTCClient = WebRTCClient.getInstance(this, firebaseClient);
    connectionManager = ConnectionManager.getInstance(this);

    setupViewPager();
    setupWebRTC();

    checkAndRequestPermissions();

    getOnBackPressedDispatcher().addCallback(this, new OnBackPressedCallback(true) {
        @Override
        public void handleOnBackPressed() {
            moveTaskToBack(true);
        }
    });

    // Set user online and attempt reconnection on login with retry
    firebaseClient.saveUser(username, phoneNumber, true, () -> {
        Log.d("ConnectActivity", "User set online: " + username);
        reconnectToPreviousUsersWithRetry(3, 1000); // Retry 3 times, 1s delay
    });
}

private void setupViewPager() {
    pagerAdapter = new ConnectPagerAdapter(this);
    viewPager.setAdapter(pagerAdapter);

    new TabLayoutMediator(tabLayout, viewPager, (tab, position) -> {
        tab.setText(position == 0 ? "REQUESTS" : "CHATS");
    }).attach();

    pagerAdapter.getRequestsFragment().setWebRTCClient(webRTCClient);
    pagerAdapter.getChatsFragment().setWebRTCClient(webRTCClient);

    UserAdapter.OnUserClickListener listener = new UserAdapter.OnUserClickListener() {
        @Override
        public void onConnectionButtonClick(UserModel user) {
            handleConnectionClick(user);
        }

        @Override
        public void onChatButtonClick(UserModel user) {
            navigateToChat(user.getUsername());
        }
    };

    pagerAdapter.getRequestsFragment().setUserClickListener(listener);
    pagerAdapter.getChatsFragment().setUserClickListener(listener);
}

private void setupWebRTC() {
    webRTCClient.setWebRTCListener(new WebRTCClient.WebRTCListener() {
        @Override
        public void onConnected() {
            runOnUiThread(() -> {
                fetchContactsAndCheckUsers();
            });
        }

        @Override
        public void onConnectionFailed() {
            runOnUiThread(() -> {
                fetchContactsAndCheckUsers();
            });
        }


        @Override
        public void onMessageReceived(String message, String peerUsername) {}


        @Override
        public void onFileSent(String filePath, String fileName) {


        }
    });
}


private void checkAndRequestPermissions() {
    if (hasCheckedPermissions) {
        return;
    }
    hasCheckedPermissions = true;


    boolean needsContactsPermission = ContextCompat.checkSelfPermission(this,
            Manifest.permission.READ_CONTACTS) != PackageManager.PERMISSION_GRANTED;


                              boolean     needsNotificationPermission     =     Build.VERSION.SDK_INT     >=
Build.VERSION_CODES.TIRAMISU &&
                ContextCompat.checkSelfPermission(this, Manifest.permission.POST_NOTIFICATIONS) !=
PackageManager.PERMISSION_GRANTED;


    if (needsContactsPermission && needsNotificationPermission) {
        ActivityCompat.requestPermissions(this,
                                                new    String[]{Manifest.permission.READ_CONTACTS,
Manifest.permission.POST_NOTIFICATIONS},
                    CONTACTS_PERMISSION_CODE);
    } else if (needsContactsPermission) {
        ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.READ_CONTACTS},
                    CONTACTS_PERMISSION_CODE);
    } else if (needsNotificationPermission) {
        ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.POST_NOTIFICATIONS},
                    NOTIFICATION_PERMISSION_CODE);
    } else {
        startWebRTCService();
        fetchContactsAndCheckUsers();
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[]
grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (requestCode == CONTACTS_PERMISSION_CODE) {
        boolean contactsGranted = false;
        boolean notificationsGranted = false;
        for (int i = 0; i < permissions.length; i++) {
            if (permissions[i].equals(Manifest.permission.READ_CONTACTS)) {
                contactsGranted = grantResults[i] == PackageManager.PERMISSION_GRANTED;
            } else if (permissions[i].equals(Manifest.permission.POST_NOTIFICATIONS)) {
                notificationsGranted = grantResults[i] == PackageManager.PERMISSION_GRANTED;
            }
        }

        if (contactsGranted) {
            handler.postDelayed(() -> {
                fetchContactsAndCheckUsers();
            }, 500);
        } else {
                        Toast.makeText(this, "Contacts permission is required to find your contacts",
Toast.LENGTH_LONG).show();
            // Show the no users text view
            View noUsersText = findViewById(R.id.noUsersText);
            if (noUsersText != null) {
                noUsersText.setVisibility(View.VISIBLE);
            }
        }

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            if (notificationsGranted) {
                startWebRTCService();
            } else {
                        Toast.makeText(this, "Notifications permission is required for chat messages",
Toast.LENGTH_LONG).show();
            }
        } else {
            startWebRTCService();
        }
    } else if (requestCode == NOTIFICATION_PERMISSION_CODE) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            startWebRTCService();
        } else {
                        Toast.makeText(this, "Notifications permission is required for chat messages",
Toast.LENGTH_LONG).show();
        }
    }
  }

  private void startWebRTCService() {
    try {
        Intent serviceIntent = new Intent(this, WebRTCService.class);
        serviceIntent.putExtra("username", getIntent().getStringExtra("username"));
        serviceIntent.putExtra("phoneNumber", getIntent().getStringExtra("phoneNumber"));
        startForegroundService(serviceIntent);

        new Handler(Looper.getMainLooper()).postDelayed(() -> {
            fetchContactsAndCheckUsers();
        }, 1000);
    } catch (Exception e) {
        Log.e("ConnectActivity", "Failed to start WebRTC service", e);
```

```
            Toast.makeText(this, "Failed to start chat service", Toast.LENGTH_SHORT).show();
        }
    }

    private void fetchContactsAndCheckUsers() {
        if (firebaseClient == null) {
            return;
        }

        // Double check permission before accessing contacts
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_CONTACTS)
                != PackageManager.PERMISSION_GRANTED) {
            // Hide any existing user lists since we can't access contacts
            View noUsersText = findViewById(R.id.noUsersText);
            if (noUsersText != null) {
                noUsersText.setVisibility(View.VISIBLE);
            }
            return;
        }

        List<String> phoneContacts = getDeviceContacts();
        firebaseClient.getRegisteredUsers(users -> {
            runOnUiThread(() -> {
                updateFragments(users);
                // Hide the no users text if we have users
                View noUsersText = findViewById(R.id.noUsersText);
                if (noUsersText != null) {
```

```
                noUsersText.setVisibility(users.isEmpty() ? View.VISIBLE : View.GONE);
            }
        });
    });
}

private void updateFragments(List<UserModel> users) {
    for (UserModel user : users) {
        if (user.isOnline() && connectionManager.isUserConnected(user.getUsername())) {
            DataChannel dataChannel = webRTCClient.getDataChannels().get(user.getUsername());
            if (dataChannel == null || dataChannel.state() != DataChannel.State.OPEN) {
                Log.d("ConnectActivity", "Reconnecting to " + user.getUsername());
                webRTCClient.startConnection(user.getUsername());
            }
        }
    }
    pagerAdapter.getRequestsFragment().updateUsers(users);
    pagerAdapter.getChatsFragment().updateUsers(users);
}

private List<String> getDeviceContacts() {
    List<String> contactList = new ArrayList<>();
    ContentResolver cr = getContentResolver();
     Cursor cursor = cr.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null,
null);

    if (cursor != null) {
        int columnIndex = cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER);
        while (cursor.moveToNext()) {
            String phone = cursor.getString(columnIndex);
            contactList.add(formatPhoneNumber(phone));
        }
        cursor.close();
    }
    return contactList;
}

private String formatPhoneNumber(String phone) {
    phone = phone.replaceAll("[^0-9]", "");
    return phone.length() > 10 ? phone.substring(phone.length() - 10) : phone;
}

private void handleConnectionClick(UserModel user) {
    DataChannel dataChannel = webRTCClient.getDataChannels().get(user.getUsername());

    if (dataChannel != null && dataChannel.state() == DataChannel.State.OPEN) {
        webRTCClient.disconnectPeer(user.getUsername());
        Toast.makeText(this, "Disconnected from " + user.getUsername(), Toast.LENGTH_SHORT).show();
        connectionManager.removeConnectedUser(user.getUsername());
    } else if (!webRTCClient.isAttemptingConnection(user.getUsername())) {
        webRTCClient.startConnection(user.getUsername());
        Toast.makeText(this, "Connecting to " + user.getUsername(), Toast.LENGTH_SHORT).show();
        connectionManager.addConnectedUser(user.getUsername());
        // Ensure requester is marked online
        firebaseClient.saveUser(getIntent().getStringExtra("username"),
                getIntent().getStringExtra("phoneNumber"), true, () -> {});
    }

    fetchContactsAndCheckUsers();
}

private void navigateToChat(String peerUsername) {
    Intent intent = new Intent(this, ChatActivity.class);
    intent.putExtra("peerUsername", peerUsername);
    startActivity(intent);

    // Try to establish connection if not already connected
    DataChannel dataChannel = webRTCClient.getDataChannels().get(peerUsername);
    if (dataChannel == null || dataChannel.state() != DataChannel.State.OPEN) {
        firebaseClient.getRegisteredUsers(users -> {
            for (UserModel user : users) {
                if (user.getUsername().equals(peerUsername) && user.isOnline()) {
                    webRTCClient.startConnection(peerUsername);
                }
            }
        });
    }
}

@Override
protected void onResume() {
    super.onResume();
    if (webRTCClient != null) {
        try {
            webRTCClient.onForeground();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        // Only fetch contacts if we have permission
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_CONTACTS)
                == PackageManager.PERMISSION_GRANTED) {
            fetchContactsAndCheckUsers();
            reconnectToPreviousUsersWithRetry(3, 1000); // Retry on resume
        } else {
            // Request permission if not granted
            checkAndRequestPermissions();
        }
    }
}

@Override
protected void onPause() {
    super.onPause();
    if (webRTCClient != null) {
        webRTCClient.onBackground();
    }
}
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_connect, menu);
    MenuItem themeItem = menu.findItem(R.id.action_theme);
    themeItem.setChecked(themeManager.isDarkMode());
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.action_theme) {
        boolean newDarkMode = !item.isChecked();
        item.setChecked(newDarkMode);
        themeManager.setDarkMode(newDarkMode);
        // Refresh the data without recreating the activity
        handler.postDelayed(() -> {
            fetchContactsAndCheckUsers();
        }, 100); // Small delay to ensure theme is applied
        return true;
    } else if (item.getItemId() == R.id.action_logout) {
        if (firebaseClient != null) {
            firebaseClient.saveUser(getIntent().getStringExtra("username"),
                    getIntent().getStringExtra("phoneNumber"),
                    false,
                    () -> {
                        stopService(new Intent(this, WebRTCService.class));
                        if (webRTCClient != null) {
```

```
                        webRTCClient.disconnect();
                    }
                    SessionManager.getInstance(this).clearSession();
                    Intent intent = new Intent(this, LoginActivity.class);
                    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);
                    startActivity(intent);
                    finish();
                });
            }
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    private void reconnectToPreviousUsersWithRetry(int retries, long delayMillis) {
        firebaseClient.getRegisteredUsers(users -> {
            runOnUiThread(() -> {
                boolean reconnected = false;
                for (UserModel user : users) {
                    if (user.isOnline() && connectionManager.isUserConnected(user.getUsername())) {
                        DataChannel dataChannel = webRTCClient.getDataChannels().get(user.getUsername());
                        if (dataChannel == null || dataChannel.state() != DataChannel.State.OPEN) {
                            Log.d("ConnectActivity", "Attempting to reconnect to " + user.getUsername());
                            webRTCClient.startConnection(user.getUsername());
                            reconnected = true;
                        }
                }
            }
            updateFragments(users);
            if (!reconnected && retries > 0) {
                Log.d("ConnectActivity", "No reconnections made, retrying in " + delayMillis + "ms, retries left: 
" + retries);
                        handler.postDelayed(() -> reconnectToPreviousUsersWithRetry(retries - 1, delayMillis), 
delayMillis);
            }
          });
        });
    }
}
```

Filename: ConnectionManager.java

```
package com.example.protegotinyever.service;


import android.content.Context;

import android.content.SharedPreferences;

import android.util.Log;

import java.util.HashSet;

import java.util.Set;


public class ConnectionManager {

    private static ConnectionManager instance;

    private final SharedPreferences prefs;

    private final Set<String> connectedUsers;

    private static final String PREFS_NAME = "ConnectionPrefs";

    private static final String KEY_CONNECTED_USERS = "connected_users";


    private ConnectionManager(Context context) {

        prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);

        connectedUsers = new HashSet<>(prefs.getStringSet(KEY_CONNECTED_USERS, new HashSet<>()));

        Log.d("ConnectionManager", "Loaded connected users: " + connectedUsers);

    }


    public static ConnectionManager getInstance(Context context) {

        if (instance == null) {

            instance = new ConnectionManager(context.getApplicationContext());

        }
        return instance;
    }

    public void addConnectedUser(String username) {
        connectedUsers.add(username);
        saveConnectedUsers();
        Log.d("ConnectionManager", "Added user: " + username + ", Now: " + connectedUsers);
    }

    public void removeConnectedUser(String username) {
        connectedUsers.remove(username);
        saveConnectedUsers();
        Log.d("ConnectionManager", "Removed user: " + username + ", Now: " + connectedUsers);
    }

    public Set<String> getConnectedUsers() {
        return new HashSet<>(connectedUsers);
    }

    public boolean isUserConnected(String username) {
        return connectedUsers.contains(username);
    }

    private void saveConnectedUsers() {
        SharedPreferences.Editor editor = prefs.edit();
        editor.putStringSet(KEY_CONNECTED_USERS, connectedUsers);
        editor.apply();
```

```
        Log.d("ConnectionManager", "Saved connected users: " + connectedUsers);
    }


    public void clearConnectedUsers() {
        connectedUsers.clear();
        saveConnectedUsers();
        Log.d("ConnectionManager", "Cleared connected users");
    }
}
```

Filename: ConnectPagerAdapter.java

```
package com.example.protegotinyever.act;


import androidx.annotation.NonNull;

import androidx.fragment.app.Fragment;

import androidx.fragment.app.FragmentActivity;

import androidx.viewpager2.adapter.FragmentStateAdapter;


public class ConnectPagerAdapter extends FragmentStateAdapter {

    private final RequestsFragment requestsFragment;

    private final ChatsFragment chatsFragment;


    public ConnectPagerAdapter(@NonNull FragmentActivity fragmentActivity) {

        super(fragmentActivity);

        requestsFragment = RequestsFragment.newInstance();

        chatsFragment = ChatsFragment.newInstance();

    }


    @NonNull

    @Override

    public Fragment createFragment(int position) {

        return position == 0 ? requestsFragment : chatsFragment;

    }


    @Override

    public int getItemCount() {
        return 2;
    }


    public RequestsFragment getRequestsFragment() {
        return requestsFragment;
    }


    public ChatsFragment getChatsFragment() {
        return chatsFragment;
    }
}
```

Filename: Contact.java

```
package com.example.protegotinyever.util;

public class Contact {
    private String name;
    private String phone;
    private int rea = 1;

    public Contact(String name, String phone) {
        this.name = name;
        this.phone = phone;
    }

    public String getName() {
        return name;
    }

    public String getPhone() {
        return phone;
    }
}
```

Filename: CustomSdpObserver.java

```
package com.example.protegotinyever.util;


import android.util.Log;
import org.webrtc.SdpObserver;
import org.webrtc.SessionDescription;


public class CustomSdpObserver implements SdpObserver {
    private int rea = 1;
    private static final String TAG = "CustomSdpObserver";


    @Override
    public void onCreateSuccess(SessionDescription sessionDescription) {
        Log.d(TAG, "SDP created successfully: " + sessionDescription.type);
    }


    @Override
    public void onSetSuccess() {
        Log.d(TAG, "SDP set successfully.");
    }


    @Override
    public void onCreateFailure(String error) {
        Log.e(TAG, "Failed to create SDP: " + error);
    }
```

```
   @Override
   public void onSetFailure(String error) {
      Log.e(TAG, "Failed to set SDP: " + error);
   }
}
```

Filename: DataChannelHandler.java

```
package com.example.protegotinyever.util;


import android.content.Context;

import android.util.Log;


import com.example.protegotinyever.db.ChatDatabase;

import com.example.protegotinyever.db.MessageDao;

import com.example.protegotinyever.adapt.MessageEntity;

import com.example.protegotinyever.webrtc.WebRTCClient;


import org.webrtc.DataChannel;

import java.nio.ByteBuffer;

import java.nio.charset.StandardCharsets;

import java.util.List;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.ArrayList;

import java.util.Map;

import java.util.HashMap;


public class DataChannelHandler {

    private static DataChannelHandler instance;

    private Map<String, DataChannel> dataChannels;

    private OnMessageReceivedListener messageReceivedListener;

    private OnStateChangeListener stateChangeListener;
private ChatDatabase chatDatabase;

private MessageDao messageDao;

private final ExecutorService databaseExecutor = Executors.newSingleThreadExecutor();

private String currentPeerUsername;

private WebRTCClient webRTCClient;

private int rea = 1;


public static synchronized DataChannelHandler getInstance(Context context) {

    if (instance == null) {

        instance = new DataChannelHandler();

        instance.setContext(context);

    }

    return instance;

}


private DataChannelHandler() {

    this.dataChannels = new HashMap<>();

}


public void setWebRTCClient(WebRTCClient client) {

    this.webRTCClient = client;

}


public void setCurrentPeer(String peerUsername) {

    Log.d("WebRTC", "Setting current peer to: " + peerUsername);

    this.currentPeerUsername = peerUsername;

}
```

```
private void setContext(Context context) {
   chatDatabase = ChatDatabase.getInstance(context);
   messageDao = chatDatabase.messageDao();
}

public void setDataChannel(DataChannel dataChannel) {
   if (currentPeerUsername == null) {
      Log.e("WebRTC", "Cannot set DataChannel: currentPeerUsername is null");
      return;
   }

   if (dataChannel != null) {
      Log.d("WebRTC", "Setting DataChannel for peer: " + currentPeerUsername);
      dataChannels.put(currentPeerUsername, dataChannel);
      registerDataChannelObserver(currentPeerUsername, dataChannel);
      if (stateChangeListener != null) {
         stateChangeListener.onStateChange(dataChannel.state());
      }
   } else {
      Log.d("WebRTC", "Removing DataChannel for peer: " + currentPeerUsername);
      dataChannels.remove(currentPeerUsername);
   }
}

private void registerDataChannelObserver(String peerUsername, DataChannel dataChannel) {
   dataChannel.registerObserver(new DataChannel.Observer() {
```

```
            @Override
            public void onBufferedAmountChange(long l) {}


            @Override
            public void onStateChange() {
                Log.d("WebRTC", "DataChannel state changed for " + peerUsername + ": " + dataChannel.state());
                if (stateChangeListener != null) {
                    stateChangeListener.onStateChange(dataChannel.state());
                }
            }


            @Override
            public void onMessage(DataChannel.Buffer buffer) {
                byte[] bytes = new byte[buffer.data.remaining()];
                buffer.data.get(bytes);
                String message = new String(bytes, StandardCharsets.UTF_8);
                Log.d("WebRTC", "? Plaintext message received from: " + peerUsername + ": " + message);
                onMessageReceived(peerUsername, message);
            }
        });
    }


    public void sendMessage(String message, String peerUsername) {
        databaseExecutor.execute(() -> {
            try {
                MessageEntity messageEntity = new MessageEntity("You", message, System.currentTimeMillis(),
peerUsername);
```

```
            messageDao.insert(messageEntity);
            Log.d("WebRTC", "Message saved to database for: " + peerUsername);
        } catch (Exception e) {
            Log.e("WebRTC", "Error saving sent message to database: " + e.getMessage());
        }
    });

    DataChannel channel = dataChannels.get(peerUsername);
    if (channel != null && channel.state() == DataChannel.State.OPEN) {
        try {
            ByteBuffer buffer = ByteBuffer.wrap(message.getBytes(StandardCharsets.UTF_8));
            channel.send(new DataChannel.Buffer(buffer, false));
            Log.d("WebRTC", "Plaintext message sent to " + peerUsername + ": " + message);
        } catch (Exception e) {
            Log.e("WebRTC", "Error sending message: " + e.getMessage());
        }
    } else {
        Log.d("WebRTC", "Message stored for offline delivery to: " + peerUsername);
    }
}

public void storeMessage(String messageText, String peerUsername, String sender) {
    databaseExecutor.execute(() -> {
        try {
                                MessageEntity messageEntity = new MessageEntity(sender, messageText,
System.currentTimeMillis(), peerUsername);
            messageDao.insert(messageEntity);
```

```
                Log.d("WebRTC", "Stored message for " + peerUsername + " from " + sender + ": " +
messageText);
        } catch (Exception e) {
            Log.e("WebRTC", "Error storing message: " + e.getMessage());
        }
    });
}

public List<MessageEntity> getMessageHistory(String peerUsername) {
    try {
        List<MessageEntity> messages = messageDao.getMessagesForPeer(peerUsername);
        Log.d("WebRTC", "Retrieved " + messages.size() + " messages for peer: " + peerUsername);
        return messages;
    } catch (Exception e) {
        Log.e("WebRTC", "Error retrieving message history: " + e.getMessage());
        return new ArrayList<>();
    }
}

public void setOnMessageReceivedListener(OnMessageReceivedListener listener) {
    this.messageReceivedListener = listener;
}

public void setStateChangeListener(OnStateChangeListener listener) {
    this.stateChangeListener = listener;
}
```

```
  public DataChannel getDataChannel(String peerUsername) {
    return dataChannels.get(peerUsername);
  }

  public void onMessageReceived(String peerUsername, String message) {
    databaseExecutor.execute(() -> {
      try {
                  MessageEntity messageEntity = new MessageEntity(peerUsername, message,
System.currentTimeMillis(), peerUsername);
        messageDao.insert(messageEntity);
        Log.d("WebRTC", "Message saved to database from: " + peerUsername);

        if (messageReceivedListener != null && peerUsername.equals(currentPeerUsername)) {
          messageReceivedListener.onMessageReceived(message);
        }

        if (webRTCClient != null && !peerUsername.equals(currentPeerUsername)) {
          webRTCClient.onMessageReceived(message, peerUsername);
        }
      } catch (Exception e) {
        Log.e("WebRTC", "Error saving received message to database: " + e.getMessage());
      }
    });
  }

  public interface OnMessageReceivedListener {
    void onMessageReceived(String message);
    }


    public interface OnStateChangeListener {
        void onStateChange(DataChannel.State state);
    }


    public void cleanup() {
        for (DataChannel channel : dataChannels.values()) {
            if (channel != null) {
                channel.close();
            }
        }
        dataChannels.clear();
        messageReceivedListener = null;
        stateChangeListener = null;
    }
}
```

Filename: DataModel.java

```
package com.example.protegotinyever.mode;


public class DataModel {
    private String type;  // Offer, Answer, ICE, Chat
    private String sender;
    private String target;
    private String data;
    private int rea = 1;


    public DataModel() {}  // Required for Firebase


    public DataModel(String type, String sender, String target, String data) {
        this.type = type;
        this.sender = sender;
        this.target = target;
        this.data = data;
    }


    public String getType() { return type; }
    public void setType(String type) { this.type = type; }


    public String getSender() { return sender; }
    public void setSender(String sender) { this.sender = sender; }


    public String getTarget() { return target; }
```

```
   public void setTarget(String target) { this.target = target; }



   public String getData() { return data; }

   public void setData(String data) { this.data = data; }

}
```

Filename: DataModelType.java

```
package com.example.protegotinyever.tt;


public class DataModelType {

    public static final String OFFER = "offer";

    public static final String ANSWER = "answer";

    public static final String ICE = "ice";

    public static final String CHAT = "chat";

}
```

Filename: EmailVerificationActivity.java

```
package com.example.protegotinyever.act;


import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;


import com.example.protegotinyever.R;
import com.example.protegotinyever.util.AuthManager;
import com.example.protegotinyever.util.SessionManager;
import com.google.firebase.auth.FirebaseUser;


public class EmailVerificationActivity extends AppCompatActivity {
    private TextView emailText;
    private Button resendButton;
    private Button verifyButton;
    private TextView changeEmailLink;
    private AuthManager authManager;
    private SessionManager sessionManager;
    private Handler handler = new Handler(Looper.getMainLooper());
    private static final int VERIFICATION_CHECK_INTERVAL = 3000; // 3 seconds
private boolean isCheckingVerification = false;
private Runnable verificationChecker;


@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_email_verification);


    authManager = AuthManager.getInstance(this);
    sessionManager = SessionManager.getInstance(this);


    emailText = findViewById(R.id.emailText);
    resendButton = findViewById(R.id.resendButton);
    verifyButton = findViewById(R.id.verifyButton);
    changeEmailLink = findViewById(R.id.changeEmailLink);


    FirebaseUser user = authManager.getCurrentUser();
    if (user != null) {
        emailText.setText(user.getEmail());
        sendVerificationEmail();
    } else {
        // If no user is signed in, go back to login
        startActivity(new Intent(this, LoginActivity.class));
        finish();
        return;
    }
    setupClickListeners();
    startVerificationCheck();
}

private void setupClickListeners() {
    resendButton.setOnClickListener(v -> {
        resendButton.setEnabled(false);
        sendVerificationEmail();
        handler.postDelayed(() -> resendButton.setEnabled(true), 30000); // Enable after 30 seconds
    });

    verifyButton.setOnClickListener(v -> {
        checkEmailVerification();
    });

    changeEmailLink.setOnClickListener(v -> {
        // Go back to signup
        authManager.signOut();
        startActivity(new Intent(this, SignUpActivity.class));
        finish();
    });
}

private void sendVerificationEmail() {
    FirebaseUser user = authManager.getCurrentUser();
    if (user != null) {
        user.sendEmailVerification()
```

```
            .addOnSuccessListener(aVoid -> {
                Toast.makeText(this, R.string.email_sent, Toast.LENGTH_SHORT).show();
            })
            .addOnFailureListener(e -> {
                Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT).show();
            });
    }
}

private void startVerificationCheck() {
    verificationChecker = new Runnable() {
        @Override
        public void run() {
            if (!isCheckingVerification) {
                return;
            }
            checkEmailVerification();
            handler.postDelayed(this, VERIFICATION_CHECK_INTERVAL);
        }
    };
    isCheckingVerification = true;
    handler.post(verificationChecker);
}

private void checkEmailVerification() {
    FirebaseUser user = authManager.getCurrentUser();
    if (user != null) {
        user.reload().addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                if (user.isEmailVerified()) {
                    isCheckingVerification = false;
                    Toast.makeText(this, R.string.email_verified, Toast.LENGTH_SHORT).show();
                    proceedToSecuritySetup();
                }
            }
        });
    }
}

private void proceedToSecuritySetup() {
    Intent intent = new Intent(this, SecuritySetupActivity.class);
    intent.putExtra("username", getIntent().getStringExtra("username"));
    intent.putExtra("phoneNumber", getIntent().getStringExtra("phoneNumber"));
    startActivity(intent);
    finish();
}

@Override
protected void onPause() {
    super.onPause();
    isCheckingVerification = false;
}

@Override
protected void onResume() {
    super.onResume();
    if (!isCheckingVerification) {
        startVerificationCheck();
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    isCheckingVerification = false;
    if (handler != null) {
        handler.removeCallbacks(verificationChecker);
    }
}
}
```

Filename: ErrorCallback.java

```
package com.example.protegotinyever.inte;


public interface ErrorCallback {
    int rea = 1;
    void onError(String error);
}
```

Filename: FirebaseClient.java

```
package com.example.protegotinyever.util;


import android.util.Log;

import androidx.annotation.NonNull;

import androidx.annotation.Nullable;


import com.example.protegotinyever.inte.SignalingCallback;

import com.example.protegotinyever.inte.SuccessCallback;

import com.example.protegotinyever.mode.DataModel;

import com.example.protegotinyever.mode.UserListCallback;

import com.example.protegotinyever.tt.DataModelType;

import com.example.protegotinyever.tt.UserModel;

import com.google.firebase.database.*;


import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;


public class FirebaseClient {

    private final DatabaseReference dbRef = FirebaseDatabase.getInstance().getReference();

    private final String currentUser;

    private final String currentUserPhone;

    private DatabaseReference connectedRef;

    private DatabaseReference userStatusRef;
private int rea = 1;


public FirebaseClient(String username, String currentUserPhone) {
    this.currentUser = username;
    this.currentUserPhone = currentUserPhone;
    setupOnlinePresence();
}


private void setupOnlinePresence() {
    connectedRef = FirebaseDatabase.getInstance().getReference(".info/connected");
    userStatusRef = dbRef.child("users").child(currentUser).child("isOnline");


    connectedRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            boolean connected = Boolean.TRUE.equals(snapshot.getValue(Boolean.class));
            if (connected) {
                Log.d("Firebase", "Connected to Firebase");
                userStatusRef.onDisconnect().setValue(false);
                userStatusRef.setValue(true);
            } else {
                Log.d("Firebase", "Disconnected from Firebase");
            }
        }


        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.e("Firebase", "Error getting connection state: " + error.getMessage());
        }
    });
}

public void saveUser(String username, String phoneNumber, boolean isOnline, SuccessCallback callback) {
    Map<String, Object> userMap = new HashMap<>();
    userMap.put("username", username);
    userMap.put("phoneNumber", phoneNumber);
    userMap.put("isOnline", isOnline);

    dbRef.child("users").child(username)
            .setValue(userMap)
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    Log.d("Firebase", "? User saved: " + username + ", Phone: " + phoneNumber + ", Online: " +
isOnline);
                    if (isOnline) setupOnlinePresence();
                    callback.onSuccess();
                } else {
                    Log.e("Firebase", "? Failed to save user: " + task.getException().getMessage());
                }
            });
}

public void getRegisteredUsers(UserListCallback callback) {
    dbRef.child("users").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            List<UserModel> users = new ArrayList<>();
            for (DataSnapshot userSnapshot : snapshot.getChildren()) {
                String username = userSnapshot.child("username").getValue(String.class);
                String phone = userSnapshot.child("phoneNumber").getValue(String.class);
                Boolean isOnline = userSnapshot.child("isOnline").getValue(Boolean.class);

                if (username != null && phone != null) {
                    boolean onlineStatus = isOnline != null ? isOnline : false;
                    users.add(new UserModel(username, phone, onlineStatus));
                        Log.d("Firebase", "? Fetched User: " + username + ", Phone: " + phone + ", Online: " +
onlineStatus);
                }
            }
            callback.onUsersFetched(users);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.e("Firebase", "? Error fetching users: " + error.getMessage());
        }
    });
  }

  public String getCurrentUserPhone() {
    Log.d("FirebaseClient", "? Returning Current User Phone: " + currentUserPhone);
```

```
        return this.currentUserPhone;
    }


    public void listenForSignaling(SignalingCallback callback) {
        dbRef.child("signaling").child(currentUser).child("data")
                .addChildEventListener(new ChildEventListener() {
                    @Override
                    public void onChildAdded(@NonNull DataSnapshot snapshot, @Nullable String previousChildName) {
                        if (snapshot.exists()) {
                            try {
                                DataModel message = snapshot.getValue(DataModel.class);
                                if (message != null) {
                                    Log.d("FirebaseClient", "? Received signaling message: " + message.getType() + " - " + message.getData());
                                    callback.onSignalingReceived(message.getType(), message.getData(), message.getSender());
                                    snapshot.getRef().removeValue();
                                } else {
                                    Log.e("FirebaseClient", "? DataModel is null!");
                                }
                            } catch (Exception e) {
                                Log.e("FirebaseClient", "? Error parsing DataModel: " + e.getMessage());
                            }
                        }
                    }
```

```
            @Override
            public void onChildChanged(@NonNull DataSnapshot snapshot, @Nullable String previousChildName) {}
            @Override
            public void onChildRemoved(@NonNull DataSnapshot snapshot) {}
            @Override
            public void onChildMoved(@NonNull DataSnapshot snapshot, @Nullable String previousChildName) {}
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Log.e("FirebaseClient", "? Error reading signaling data: " + error.getMessage());
            }
        });
    }

    public void sendSignalingData(String peerUsername, String type, String data) {
        if (peerUsername == null || peerUsername.isEmpty()) {
            Log.e("FirebaseClient", "? Peer username is null or empty! Cannot send signaling data.");
            return;
        }
        DataModel message = new DataModel(type, currentUser, peerUsername, data);

        dbRef.child("signaling").child(peerUsername).child("data")
            .push()
            .setValue(message)
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
```

```
                Log.d("FirebaseClient", "? Signaling data sent: " + type);
            } else {
                Log.e("FirebaseClient", "? Failed to send signaling data: " + task.getException().getMessage());
            }
        });
    }

    public static class SignalingData {
        private String fromUsername;
        private DataModelType type;
        private String data;

        public SignalingData() {}

        public String getFromUsername() { return fromUsername; }
        public void setFromUsername(String fromUsername) { this.fromUsername = fromUsername; }
        public DataModelType getType() { return type; }
        public void setType(DataModelType type) { this.type = type; }
        public String getData() { return data; }
        public void setData(String data) { this.data = data; }
    }
}
```

Filename: ForgotPasswordActivity.java

```
package com.example.protegotinyever.act;


import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;


import com.example.protegotinyever.R;
import com.example.protegotinyever.util.AuthManager;
import com.google.android.material.textfield.TextInputLayout;


public class ForgotPasswordActivity extends AppCompatActivity {
    private TextInputLayout emailInputLayout;
    private EditText emailInput;
    private Button resetButton;
    private TextView backToLoginLink;
    private AuthManager authManager;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_forgot_password);
        authManager = AuthManager.getInstance(this);


        emailInputLayout = findViewById(R.id.emailInputLayout);
        emailInput = findViewById(R.id.emailInput);
        resetButton = findViewById(R.id.resetButton);
        backToLoginLink = findViewById(R.id.backToLoginLink);


        // Pre-fill email if provided
        String prefilledEmail = getIntent().getStringExtra("email");
        if (prefilledEmail != null && !prefilledEmail.isEmpty()) {
            emailInput.setText(prefilledEmail);
        }


        resetButton.setOnClickListener(v -> {
            String email = emailInput.getText().toString().trim();
            if (validateEmail(email)) {
                resetButton.setEnabled(false);
                sendPasswordResetEmail(email);
            }
        });


        backToLoginLink.setOnClickListener(v -> {
            finish();
        });
    }
    private boolean validateEmail(String email) {

        if (email.isEmpty()) {

            emailInputLayout.setError("Email is required");

            return false;

        }

        if (!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {

            emailInputLayout.setError("Please enter a valid email");

            return false;

        }

        emailInputLayout.setError(null);

        return true;

    }


    private void sendPasswordResetEmail(String email) {

        authManager.sendPasswordResetEmail(email)

            .addOnSuccessListener(aVoid -> {

                Toast.makeText(this, R.string.reset_email_sent, Toast.LENGTH_LONG).show();

                // Return to login screen after successful send

                finish();

            })

            .addOnFailureListener(e -> {

                resetButton.setEnabled(true);

                Toast.makeText(this, R.string.reset_email_error, Toast.LENGTH_LONG).show();

            });

    }

}
```

Filename: LoginActivity.java

```
package com.example.protegotinyever.act;


import android.content.Intent;

import android.os.Bundle;

import android.widget.Button;

import android.widget.EditText;

import android.widget.TextView;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;


import com.example.protegotinyever.R;

import com.example.protegotinyever.util.AuthManager;

import com.example.protegotinyever.util.SessionManager;

import com.google.firebase.auth.FirebaseUser;


public class LoginActivity extends AppCompatActivity {

    private EditText emailInput, passwordInput;

    private Button loginButton;

    private TextView signUpLink;

    private AuthManager authManager;

    private SessionManager sessionManager;


    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
```

```
authManager = AuthManager.getInstance(this);
sessionManager = SessionManager.getInstance(this);

// Check if user is already logged in
if (authManager.isLoggedIn()) {
    FirebaseUser user = authManager.getCurrentUser();
    if (user != null && sessionManager.isLoggedIn()) {
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
        finish();
        return;
    }
}

setContentView(R.layout.activity_login);

emailInput = findViewById(R.id.emailInput);
passwordInput = findViewById(R.id.passwordInput);
loginButton = findViewById(R.id.loginButton);
signUpLink = findViewById(R.id.signUpLink);
TextView forgotPasswordLink = findViewById(R.id.forgotPasswordLink);

loginButton.setOnClickListener(v -> {
    String email = emailInput.getText().toString().trim();
    String password = passwordInput.getText().toString().trim();
    if (validateInputs(email, password)) {
        loginButton.setEnabled(false);
        authManager.signIn(email, password)
            .addOnSuccessListener(authResult -> {
                FirebaseUser user = authResult.getUser();
                if (user != null) {
                    String username = user.getDisplayName();
                    // For simplicity, we'll use email as phone number if not available
                    String phone = user.getPhoneNumber() != null ?
                            user.getPhoneNumber() : user.getEmail();

                    sessionManager.saveLoginSession(username, phone);
                    Intent intent = new Intent(LoginActivity.this, MainActivity.class);
                    startActivity(intent);
                    finish();
                }
            })
            .addOnFailureListener(e -> {
                loginButton.setEnabled(true);
                Toast.makeText(LoginActivity.this,
                    "Login failed: " + e.getMessage(),
                    Toast.LENGTH_SHORT).show();
            });
    }
});

signUpLink.setOnClickListener(v -> {
```

```
            startActivity(new Intent(this, SignUpActivity.class));
        });


        forgotPasswordLink.setOnClickListener(v -> {
            Intent intent = new Intent(this, ForgotPasswordActivity.class);
            intent.putExtra("email", emailInput.getText().toString().trim());
            startActivity(intent);
        });
    }


    private boolean validateInputs(String email, String password) {
        if (email.isEmpty()) {
            emailInput.setError("Email is required");
            return false;
        }
        if (!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
            emailInput.setError("Please enter a valid email");
            return false;
        }
        if (password.isEmpty()) {
            passwordInput.setError("Password is required");
            return false;
        }
        return true;
    }
}
```

Filename: MainActivity.java

```java
package com.example.protegotinyever.act;


import android.content.Intent;

import android.content.SharedPreferences;

import android.content.pm.PackageManager;

import android.os.Build;

import android.os.Bundle;

import android.os.Handler;

import android.os.Looper;

import android.view.View;

import android.widget.Button;

import android.widget.ImageView;

import android.widget.TextView;

import android.widget.Toast;

import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import androidx.biometric.BiometricManager;

import androidx.biometric.BiometricPrompt;

import androidx.core.app.ActivityCompat;

import androidx.core.content.ContextCompat;

import androidx.cardview.widget.CardView;


import com.example.protegotinyever.R;

import com.example.protegotinyever.util.AuthManager;

import com.example.protegotinyever.util.SessionManager;
import com.google.android.material.textfield.TextInputLayout;


import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.concurrent.Executor;


public class MainActivity extends AppCompatActivity {
    private TextView statusText;
    private ImageView securityIcon;
    private CardView securityCard;
    private Button authenticateButton;
    private TextInputLayout pinInputLayout;
    private SharedPreferences prefs;
    private SessionManager sessionManager;
    private AuthManager authManager;
    private static final String PREFS_NAME = "SecurityPrefs";
    private static final String KEY_SECURITY_ENABLED = "security_enabled";
    private static final String KEY_SECURITY_TYPE = "security_type";
    private static final String KEY_PIN_HASH = "pin_hash";
    private static final String TYPE_BIOMETRIC = "biometric";
    private static final String TYPE_PIN = "pin";
    private static final int PERMISSION_REQUEST_CODE = 1001;
    private boolean isAuthenticating = false;
    private boolean isPermissionRequested = false;
    private Handler handler = new Handler(Looper.getMainLooper());
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);


   initializeViews();
   checkAuthenticationState();
}


private void initializeViews() {
   statusText = findViewById(R.id.statusText);
   securityIcon = findViewById(R.id.securityIcon);
   securityCard = findViewById(R.id.securityCard);
   authenticateButton = findViewById(R.id.authenticateButton);
   pinInputLayout = findViewById(R.id.pinInputLayout);
   prefs = getSharedPreferences(PREFS_NAME, MODE_PRIVATE);
   sessionManager = SessionManager.getInstance(this);
   authManager = AuthManager.getInstance(this);
}


private void checkAuthenticationState() {
   if (!authManager.isLoggedIn() || !sessionManager.isLoggedIn()) {
      startActivity(new Intent(this, LoginActivity.class));
      finish();
      return;
   }
```

```
        boolean isSecurityEnabled = prefs.getBoolean(KEY_SECURITY_ENABLED, false);
        if (!isSecurityEnabled) {
            Intent intent = new Intent(this, SecuritySetupActivity.class);
            intent.putExtra("username", sessionManager.getUsername());
            intent.putExtra("phoneNumber", sessionManager.getPhone());
            startActivity(intent);
            finish();
            return;
        }

        String securityType = prefs.getString(KEY_SECURITY_TYPE, TYPE_PIN);
        if (TYPE_BIOMETRIC.equals(securityType)) {
            checkBiometricPrerequisites();
        } else {
            setupPinAuth();
        }
    }

    private void checkBiometricPrerequisites() {
        BiometricManager biometricManager = BiometricManager.from(this);
        int canAuthenticate = biometricManager.canAuthenticate(BiometricManager.Authenticators.BIOMETRIC_STRONG);

        if (canAuthenticate == BiometricManager.BIOMETRIC_SUCCESS) {
            checkAndRequestPermissions();
        } else {
            // Fallback to PIN if biometric becomes unavailable
        setupPinAuth();
    }
}

private void checkAndRequestPermissions() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
        if (checkSelfPermission(android.Manifest.permission.USE_BIOMETRIC)
                != PackageManager.PERMISSION_GRANTED) {
            if (!isPermissionRequested) {
                isPermissionRequested = true;
                ActivityCompat.requestPermissions(this,
                        new String[]{android.Manifest.permission.USE_BIOMETRIC},
                        PERMISSION_REQUEST_CODE);
            }
            return;
        }
    }
    setupBiometricAuth();
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == PERMISSION_REQUEST_CODE) {
        isPermissionRequested = false;
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            handler.postDelayed(this::setupBiometricAuth, 500);
        } else {
            setupPinAuth();
        }
    }
}

private void setupBiometricAuth() {
    if (isFinishing() || isDestroyed()) {
        return;
    }

    pinInputLayout.setVisibility(View.GONE);
    securityIcon.setImageResource(R.drawable.ic_fingerprint);
    statusText.setText(R.string.use_fingerprint);
    authenticateButton.setText(R.string.authenticate_biometric);

    Executor executor = ContextCompat.getMainExecutor(this);
    BiometricPrompt biometricPrompt = new BiometricPrompt(this, executor,
            new BiometricPrompt.AuthenticationCallback() {
                @Override
                public void onAuthenticationSucceeded(
                        @NonNull BiometricPrompt.AuthenticationResult result) {
                    super.onAuthenticationSucceeded(result);
                    if (!isAuthenticating) return;
                    isAuthenticating = false;
                    securityIcon.setImageResource(R.drawable.ic_check_circle);
                    statusText.setText(R.string.auth_successful);
                    handler.postDelayed(() -> proceedToConnectActivity(), 500);
                }


                @Override
                public void onAuthenticationError(int errorCode,
                        @NonNull CharSequence errString) {
                    super.onAuthenticationError(errorCode, errString);
                    isAuthenticating = false;
                    if (isFinishing() || isDestroyed()) return;


                    statusText.setText(getString(R.string.auth_error, errString));
                    if (errorCode == BiometricPrompt.ERROR_NO_BIOMETRICS ||
                        errorCode == BiometricPrompt.ERROR_HW_NOT_PRESENT ||
                        errorCode == BiometricPrompt.ERROR_HW_UNAVAILABLE) {
                        setupPinAuth();
                    }
                }


                @Override
                public void onAuthenticationFailed() {
                    super.onAuthenticationFailed();
                    if (isFinishing() || isDestroyed()) return;
                    securityIcon.setImageResource(R.drawable.ic_error);
                    statusText.setText(R.string.auth_failed);
                }
            });
    BiometricPrompt.PromptInfo promptInfo = new BiometricPrompt.PromptInfo.Builder()
            .setTitle(getString(R.string.unlock_app))
            .setSubtitle(getString(R.string.use_fingerprint_subtitle))
            .setNegativeButtonText(getString(R.string.use_pin))
            .build();

    authenticateButton.setOnClickListener(v -> {
        if (!isAuthenticating) {
            isAuthenticating = true;
            biometricPrompt.authenticate(promptInfo);
        }
    });
}

private void setupPinAuth() {
    if (isFinishing() || isDestroyed()) {
        return;
    }

    isAuthenticating = false;
    pinInputLayout.setVisibility(View.VISIBLE);
    securityIcon.setImageResource(R.drawable.ic_pin);
    statusText.setText(R.string.enter_pin);
    authenticateButton.setText(R.string.authenticate_pin);

    authenticateButton.setOnClickListener(v -> {
        if (isAuthenticating) return;
        isAuthenticating = true;


        String inputPin = pinInputLayout.getEditText().getText().toString();
        String storedPinHash = prefs.getString(KEY_PIN_HASH, "");


        try {
            String inputPinHash = hashPin(inputPin);
            if (storedPinHash.equals(inputPinHash)) {
                securityIcon.setImageResource(R.drawable.ic_check_circle);
                statusText.setText(R.string.auth_successful);
                handler.postDelayed(() -> proceedToConnectActivity(), 500);
            } else {
                securityIcon.setImageResource(R.drawable.ic_error);
                pinInputLayout.setError(getString(R.string.invalid_pin));
                statusText.setText(R.string.auth_failed);
                isAuthenticating = false;
            }
        } catch (NoSuchAlgorithmException e) {
            Toast.makeText(this, R.string.auth_error_generic,
                    Toast.LENGTH_SHORT).show();
            isAuthenticating = false;
        }
    });
}


private String hashPin(String pin) throws NoSuchAlgorithmException {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash = digest.digest(pin.getBytes());
    return Base64.getEncoder().encodeToString(hash);
}


private void proceedToConnectActivity() {
    if (isFinishing() || isDestroyed()) {
        return;
    }


    handler.post(() -> {
        try {
            Toast.makeText(this, R.string.app_unlocked, Toast.LENGTH_SHORT).show();
            Intent intent = new Intent(this, ConnectActivity.class);
            intent.putExtra("username", sessionManager.getUsername());
            intent.putExtra("phoneNumber", sessionManager.getPhone());
            startActivity(intent);
            finish();
        } catch (Exception e) {
            isAuthenticating = false;
            Toast.makeText(this, R.string.auth_error_generic, Toast.LENGTH_SHORT).show();
        }
    });
}


@Override
protected void onPause() {
        super.onPause();

        isAuthenticating = false;

    }


    @Override

    protected void onDestroy() {

        super.onDestroy();

        if (handler != null) {

            handler.removeCallbacksAndMessages(null);

        }

    }

}
```

Filename: MessageAdapter.java

```
package com.example.protegotinyever.tt;


import android.content.Context;

import android.graphics.Bitmap;

import android.net.Uri;

import android.provider.MediaStore;

import android.text.format.Formatter;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.FrameLayout;

import android.widget.ImageView;

import android.widget.LinearLayout;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.documentfile.provider.DocumentFile;

import androidx.recyclerview.widget.RecyclerView;


import com.bumptech.glide.Glide;

import com.example.protegotinyever.R;

import com.example.protegotinyever.mode.MessageModel;

import com.google.android.exoplayer2.SimpleExoPlayer;

import com.google.android.exoplayer2.ui.PlayerView;


import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.Locale;


import android.content.ContentUris;
import android.database.Cursor;
import android.os.Build;


public class MessageAdapter extends RecyclerView.Adapter<MessageAdapter.MessageViewHolder> {
    private final List<MessageModel> messageList;
    private final String currentUser;
    private final Context context;
    private final OnFileClickListener fileClickListener;
    private SimpleExoPlayer exoPlayer;


    public MessageAdapter(List<MessageModel> messageList, String currentUser, Context context,
OnFileClickListener fileClickListener) {
        this.messageList = messageList;
        this.currentUser = currentUser;
        this.context = context;
        this.fileClickListener = fileClickListener;
        initializePlayer();
    }


    private void initializePlayer() {
        exoPlayer = new SimpleExoPlayer.Builder(context).build();
```

```
}


@Override
public void onDetachedFromRecyclerView(@NonNull RecyclerView recyclerView) {
    super.onDetachedFromRecyclerView(recyclerView);
    if (exoPlayer != null) {
        exoPlayer.release();
        exoPlayer = null;
    }
}


@Override
public int getItemViewType(int position) {
    String sender = messageList.get(position).getSender();
    return (sender != null && sender.equals(currentUser)) ? 1 : 0;
}


@NonNull
@Override
public MessageViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(
            (viewType == 1) ? R.layout.message_sent : R.layout.message_received, parent, false);
    return new MessageViewHolder(view);
}


@Override
public void onBindViewHolder(@NonNull MessageViewHolder holder, int position) {
```

```
MessageModel message = messageList.get(position);
String messageText = message.getText();

// Reset visibility
holder.messageText.setVisibility(View.VISIBLE);
holder.previewContainer.setVisibility(View.GONE);
holder.messageImage.setVisibility(View.GONE);
holder.videoPreviewContainer.setVisibility(View.GONE);
holder.filePreviewContainer.setVisibility(View.GONE);

if (messageText.startsWith("Received file:") || messageText.startsWith("Sent file:")) {
    String[] parts = messageText.split(" at ");
    String fileName = parts[0].replace("Received file: ", "").replace("Sent file: ", "").trim();
    String filePath = parts.length > 1 ? parts[1].trim() : null;

    android.util.Log.d("MessageAdapter", "Processing file message: " + fileName);
    android.util.Log.d("MessageAdapter", "Original file path: " + filePath);

    if (filePath != null) {
        // Try to get the file using MediaStore
        Uri fileUri = getFileUri(filePath);
        android.util.Log.d("MessageAdapter", "File URI: " + fileUri);

        if (fileUri != null) {
            String mimeType = getMimeTypeFromUri(fileUri);
            android.util.Log.d("MessageAdapter", "MIME type: " + mimeType);
            holder.previewContainer.setVisibility(View.VISIBLE);
            holder.messageText.setVisibility(View.GONE);


            if (mimeType != null && mimeType.startsWith("image/")) {
                setupImagePreview(holder, fileUri);
            } else if (mimeType != null && mimeType.startsWith("video/")) {
                setupVideoPreview(holder, fileUri);
            } else {
                setupGenericFilePreview(holder, fileUri, fileName);
            }


            // Set click listener for the preview container
            holder.previewContainer.setOnClickListener(v -> fileClickListener.onFileClick(filePath));
        } else {
            showFallbackText(holder, fileName);
        }
    } else {
        showFallbackText(holder, fileName);
    }
} else {
    holder.messageText.setText(messageText);
    holder.messageText.setTextColor(holder.itemView.getResources().getColor(
        getItemViewType(position) == 1 ? android.R.color.white : android.R.color.black));
}


// Set timestamp
long timestamp = message.getTimestamp() > 0 ? message.getTimestamp() : System.currentTimeMillis();
```

```
    String time = new SimpleDateFormat("hh:mm a", Locale.getDefault()).format(new Date(timestamp));
    holder.messageTime.setText(time);
}

private Uri getFileUri(String filePath) {
    try {
        // First try direct file access for backward compatibility
        File file = new File(filePath);
        if (!file.exists() && filePath.startsWith("/")) {
            // Try without leading slash
            file = new File(filePath.substring(1));
        }

        if (file.exists() && file.canRead()) {
            android.util.Log.d("MessageAdapter", "File found using direct access");
            return Uri.fromFile(file);
        }

        // Try with external storage path
        File externalDir = android.os.Environment.getExternalStorageDirectory();
        file = new File(externalDir, filePath.startsWith("/") ? filePath.substring(1) : filePath);
        if (file.exists() && file.canRead()) {
            android.util.Log.d("MessageAdapter", "File found in external storage");
            return Uri.fromFile(file);
        }

        // Try with absolute path
```

```
file = new File("/" + filePath);
if (file.exists() && file.canRead()) {
   android.util.Log.d("MessageAdapter", "File found with absolute path");
   return Uri.fromFile(file);
}

// Try with content resolver
String fileName = filePath.substring(filePath.lastIndexOf('/') + 1);
String[] projection = {MediaStore.Files.FileColumns._ID};
String selection = MediaStore.Files.FileColumns.DISPLAY_NAME + "=?";
String[] selectionArgs = {fileName};

// Try in Downloads
Uri downloadsUri = MediaStore.Downloads.EXTERNAL_CONTENT_URI;
try (Cursor cursor = context.getContentResolver().query(
      downloadsUri,
      projection,
      selection,
      selectionArgs,
      null)) {
   if (cursor != null && cursor.moveToFirst()) {
      long id = cursor.getLong(0);
      android.util.Log.d("MessageAdapter", "File found in Downloads MediaStore");
      return ContentUris.withAppendedId(downloadsUri, id);
   }
}
```

```
// Try in MediaStore Files
Uri filesUri = MediaStore.Files.getContentUri("external");
try (Cursor cursor = context.getContentResolver().query(
        filesUri,
        projection,
        selection,
        selectionArgs,
        null)) {
    if (cursor != null && cursor.moveToFirst()) {
        long id = cursor.getLong(0);
        android.util.Log.d("MessageAdapter", "File found in Files MediaStore");
        return ContentUris.withAppendedId(filesUri, id);
    }
}

// Try using DocumentFile
File baseDir = new File(filePath).getParentFile();
if (baseDir != null && baseDir.exists()) {
    DocumentFile docFile = DocumentFile.fromFile(baseDir);
    if (docFile != null) {
        DocumentFile[] files = docFile.listFiles();
        for (DocumentFile doc : files) {
            if (doc.getName() != null && doc.getName().equals(fileName)) {
                android.util.Log.d("MessageAdapter", "File found using DocumentFile");
                return doc.getUri();
            }
        }
```

```
            }
        }

        android.util.Log.d("MessageAdapter", "File not found in any location");
        return null;

    } catch (Exception e) {
        android.util.Log.e("MessageAdapter", "Error getting file URI", e);
        return null;
    }
}

private String getMimeTypeFromUri(Uri uri) {
    if (uri.getScheme() != null && uri.getScheme().equals("file")) {
        // For file:// URIs, determine MIME type from file extension
        String extension = getFileExtension(uri.getPath());
        if (extension != null) {
            String mimeType = android.webkit.MimeTypeMap.getSingleton()
                .getMimeTypeFromExtension(extension.toLowerCase());
            android.util.Log.d("MessageAdapter", "Determined MIME type from extension: " + mimeType);
            return mimeType;
        }
        return null;
    } else {
        // For content:// URIs, use ContentResolver
        try {
            String mimeType = context.getContentResolver().getType(uri);
            android.util.Log.d("MessageAdapter", "Got MIME type from ContentResolver: " + mimeType);
            return mimeType;
        } catch (Exception e) {
            android.util.Log.e("MessageAdapter", "Error getting MIME type from ContentResolver", e);
            return null;
        }
    }
}

private String getFileExtension(String path) {
    if (path == null) return null;

    try {
        String fileName = path.substring(path.lastIndexOf('/') + 1);
        int lastDot = fileName.lastIndexOf('.');
        if (lastDot >= 0) {
            return fileName.substring(lastDot + 1);
        }
    } catch (Exception e) {
        android.util.Log.e("MessageAdapter", "Error getting file extension", e);
    }
    return null;
}

private void setupImagePreview(MessageViewHolder holder, Uri uri) {
    android.util.Log.d("MessageAdapter", "Setting up image preview for: " + uri);
    holder.messageImage.setVisibility(View.VISIBLE);

    holder.videoPreviewContainer.setVisibility(View.GONE);

    holder.filePreviewContainer.setVisibility(View.GONE);


    try {

        // Try loading with file path first

        File file = new File(uri.getPath());

        if (file.exists() && file.canRead()) {

            Glide.with(context)

                .load(file)

                .into(holder.messageImage);

        } else {

            // Fall back to URI

            Glide.with(context)

                .load(uri)

                .into(holder.messageImage);

        }

        android.util.Log.d("MessageAdapter", "Image loaded with Glide");

    } catch (Exception e) {

        android.util.Log.e("MessageAdapter", "Error loading image", e);

        showFallbackText(holder, uri.getLastPathSegment());

    }

}


private void setupVideoPreview(MessageViewHolder holder, Uri uri) {

    android.util.Log.d("MessageAdapter", "Setting up video preview for: " + uri);
```

```
holder.videoPreviewContainer.setVisibility(View.VISIBLE);
holder.messageImage.setVisibility(View.GONE);
holder.filePreviewContainer.setVisibility(View.GONE);

try {
   // Use MediaStore to create thumbnail
   Bitmap thumbnail = null;
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
      thumbnail = context.getContentResolver().loadThumbnail(
         uri, new android.util.Size(512, 384), null);
   } else {
      thumbnail = MediaStore.Video.Thumbnails.getThumbnail(
         context.getContentResolver(), ContentUris.parseId(uri),
         MediaStore.Video.Thumbnails.MINI_KIND, null);
   }

   if (thumbnail != null) {
      holder.videoThumbnail.setImageBitmap(thumbnail);
      android.util.Log.d("MessageAdapter", "Video thumbnail created successfully");
   } else {
      holder.videoThumbnail.setImageResource(R.drawable.ic_file_video);
      android.util.Log.d("MessageAdapter", "Failed to create video thumbnail, using default icon");
   }
} catch (Exception e) {
   android.util.Log.e("MessageAdapter", "Error creating video thumbnail", e);
   holder.videoThumbnail.setImageResource(R.drawable.ic_file_video);
}
```

```
}


}

private void setupGenericFilePreview(MessageViewHolder holder, Uri uri, String fileName) {
   android.util.Log.d("MessageAdapter", "Setting up generic file preview for: " + fileName);


   holder.filePreviewContainer.setVisibility(View.VISIBLE);
   holder.messageImage.setVisibility(View.GONE);
   holder.videoPreviewContainer.setVisibility(View.GONE);


   holder.fileName.setText(fileName);


   try {
      if (uri.getScheme() != null && uri.getScheme().equals("file")) {
         // For file:// URIs, get size directly from file
         File file = new File(uri.getPath());
         holder.fileSize.setText(Formatter.formatShortFileSize(context, file.length()));
      } else {
         // For content:// URIs, use ContentResolver
         String[] projection = { MediaStore.Files.FileColumns.SIZE };
         Cursor cursor = context.getContentResolver().query(uri, projection, null, null, null);
         if (cursor != null && cursor.moveToFirst()) {
            long size = cursor.getLong(0);
            holder.fileSize.setText(Formatter.formatShortFileSize(context, size));
            cursor.close();
         }
      }
   } catch (Exception e) {
```

```
            android.util.Log.e("MessageAdapter", "Error getting file size", e);
            holder.fileSize.setText("");
        }

        String mimeType = getMimeTypeFromUri(uri);
        int iconResource = getFileIconResource(mimeType != null ? mimeType : "application/octet-stream");
        holder.fileIcon.setImageResource(iconResource);

        android.util.Log.d("MessageAdapter", "Generic file preview set up with icon: " + iconResource);
    }

    private void showFallbackText(MessageViewHolder holder, String fileName) {
        android.util.Log.d("MessageAdapter", "Showing fallback text for: " + fileName);

        holder.messageText.setVisibility(View.VISIBLE);
        holder.previewContainer.setVisibility(View.GONE);
        holder.messageText.setText(fileName);
        holder.messageText.setTextColor(context.getResources().getColor(android.R.color.holo_blue_light));
    }

    private int getFileIconResource(String mimeType) {
        if (mimeType.startsWith("audio/")) {
            return R.drawable.ic_file_audio;
        } else if (mimeType.startsWith("video/")) {
            return R.drawable.ic_file_video;
        } else if (mimeType.startsWith("image/")) {
            return R.drawable.ic_file_image;
        } else if (mimeType.startsWith("text/") || mimeType.contains("document")) {
            return R.drawable.ic_file_document;
        } else if (mimeType.contains("pdf")) {
            return R.drawable.ic_file_pdf;
        } else if (mimeType.contains("zip") || mimeType.contains("rar") ||
                mimeType.contains("7z") || mimeType.contains("tar") ||
                mimeType.contains("gz")) {
            return R.drawable.ic_file_archive;
        } else {
            return R.drawable.ic_file_document;
        }
    }

    @Override
    public int getItemCount() {
        return messageList.size();
    }

    public static class MessageViewHolder extends RecyclerView.ViewHolder {
        TextView messageText, messageTime, fileName, fileSize;
        ImageView messageImage, videoThumbnail, fileIcon;
        FrameLayout previewContainer, videoPreviewContainer;
        LinearLayout filePreviewContainer;
        PlayerView playerView;

        public MessageViewHolder(View itemView) {
            super(itemView);
```

```
            messageText = itemView.findViewById(R.id.messageText);
            messageTime = itemView.findViewById(R.id.messageTime);
            messageImage = itemView.findViewById(R.id.messageImage);
            videoThumbnail = itemView.findViewById(R.id.videoThumbnail);
            fileIcon = itemView.findViewById(R.id.fileIcon);
            fileName = itemView.findViewById(R.id.fileName);
            fileSize = itemView.findViewById(R.id.fileSize);
            previewContainer = itemView.findViewById(R.id.previewContainer);
            videoPreviewContainer = itemView.findViewById(R.id.videoPreviewContainer);
            filePreviewContainer = itemView.findViewById(R.id.filePreviewContainer);
            playerView = itemView.findViewById(R.id.playerView);
        }
    }

    public interface OnFileClickListener {
        void onFileClick(String filePath);
    }

}
```

Filename: MessageDao.java

```
package com.example.protegotinyever.db;


import androidx.room.Dao;

import androidx.room.Insert;

import androidx.room.Query;


import com.example.protegotinyever.adapt.MessageEntity;


import java.util.List;


@Dao

public interface MessageDao {

   int rea = 1;


   @Insert

   void insert(MessageEntity message);


    @Query("SELECT * FROM messages WHERE peerUsername = :peerUsername ORDER BY timestamp
ASC")

   List<MessageEntity> getMessagesForPeer(String peerUsername);


   @Query("DELETE FROM messages WHERE peerUsername = :peerUsername")

   void deleteMessagesForPeer(String peerUsername);


   @Query("DELETE FROM messages")
```

```
    void deleteAllMessages();

}
```

Filename: MessageEncryptor.java

```java
package com.example.protegotinyever.util;


import org.bouncycastle.crypto.engines.ChaChaEngine;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.crypto.params.ParametersWithIV;


import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.nio.ByteBuffer;
import java.security.SecureRandom;
import android.util.Log;


public class MessageEncryptor {
    private static final int NONCE_LENGTH = 8; // ChaCha20 with Bouncy Castle uses 8-byte nonce
    private static final int IV_LENGTH = 16; // AES IV
    private static final int KEY_LENGTH = 32; // 256-bit keys in bytes


    public static class EncryptionResult {
        public byte[] combinedData;


        public EncryptionResult(byte[] combinedData) {
            this.combinedData = combinedData;
        }
    }
    public static EncryptionResult encryptMessage(String message, String senderEmail, String senderPhone) throws Exception {
        byte[] messageBytes = message.getBytes("UTF-8");
        return encryptData(messageBytes, senderEmail, senderPhone);
    }

    public static EncryptionResult encryptData(byte[] data, String senderEmail, String senderPhone) throws Exception {
        byte[] nonce = generateNonceFromUserInput(senderEmail, senderPhone);
        byte[] iv = generateIV();
        byte[] chachaKey = generateKey();
        byte[] aesKey = generateKey();

        Log.d("MessageEncryptor", "Encrypting data, length: " + data.length);

        // ChaCha20 encryption (Bouncy Castle)
        ChaChaEngine chachaEngine = new ChaChaEngine();
        ParametersWithIV chachaParams = new ParametersWithIV(new KeyParameter(chachaKey), nonce);
        chachaEngine.init(true, chachaParams);
        byte[] chachaEncrypted = new byte[data.length];
        chachaEngine.processBytes(data, 0, data.length, chachaEncrypted, 0);

        // AES-CTR encryption
        Cipher aesCipher = Cipher.getInstance("AES/CTR/NoPadding");
        SecretKeySpec aesKeySpec = new SecretKeySpec(aesKey, "AES");
        IvParameterSpec ivSpec = new IvParameterSpec(iv);
        aesCipher.init(Cipher.ENCRYPT_MODE, aesKeySpec, ivSpec);
        byte[] doubleEncrypted = aesCipher.doFinal(chachaEncrypted);


        // Combine keys and encrypted data
            ByteBuffer buffer = ByteBuffer.allocate(KEY_LENGTH + KEY_LENGTH + NONCE_LENGTH +
IV_LENGTH + doubleEncrypted.length);
        buffer.put(chachaKey);
        buffer.put(aesKey);
        buffer.put(nonce);
        buffer.put(iv);
        buffer.put(doubleEncrypted);
        byte[] combinedData = buffer.array();


        Log.d("MessageEncryptor", "Encryption complete, combined length: " + combinedData.length);
        return new EncryptionResult(combinedData);
    }


    public static String decryptMessage(byte[] combinedData) throws Exception {
        byte[] decryptedBytes = decryptData(combinedData);
        return new String(decryptedBytes, "UTF-8");
    }


    public static byte[] decryptData(byte[] combinedData) throws Exception {
        if (combinedData.length < KEY_LENGTH + KEY_LENGTH + NONCE_LENGTH + IV_LENGTH) {
            Log.e("MessageEncryptor", "Invalid combined data length: " + combinedData.length);
            throw new IllegalArgumentException("Invalid combined data length");
        }
        Log.d("MessageEncryptor", "Decrypting combined data, length: " + combinedData.length);


        byte[] chachaKey = new byte[KEY_LENGTH];
        byte[] aesKey = new byte[KEY_LENGTH];
        byte[] nonce = new byte[NONCE_LENGTH];
        byte[] iv = new byte[IV_LENGTH];
            byte[] encryptedData = new byte[combinedData.length - (KEY_LENGTH + KEY_LENGTH +
NONCE_LENGTH + IV_LENGTH)];


        System.arraycopy(combinedData, 0, chachaKey, 0, KEY_LENGTH);
        System.arraycopy(combinedData, KEY_LENGTH, aesKey, 0, KEY_LENGTH);
        System.arraycopy(combinedData, KEY_LENGTH * 2, nonce, 0, NONCE_LENGTH);
        System.arraycopy(combinedData, KEY_LENGTH * 2 + NONCE_LENGTH, iv, 0, IV_LENGTH);
             System.arraycopy(combinedData, KEY_LENGTH * 2 + NONCE_LENGTH + IV_LENGTH,
encryptedData, 0, encryptedData.length);


        // AES-CTR decryption
        Cipher aesCipher = Cipher.getInstance("AES/CTR/NoPadding");
        SecretKeySpec aesKeySpec = new SecretKeySpec(aesKey, "AES");
        IvParameterSpec ivSpec = new IvParameterSpec(iv);
        aesCipher.init(Cipher.DECRYPT_MODE, aesKeySpec, ivSpec);
        byte[] chachaEncrypted = aesCipher.doFinal(encryptedData);


        // ChaCha20 decryption (Bouncy Castle)
        ChaChaEngine chachaEngine = new ChaChaEngine();
        ParametersWithIV chachaParams = new ParametersWithIV(new KeyParameter(chachaKey), nonce);
        chachaEngine.init(false, chachaParams);
        byte[] decryptedBytes = new byte[chachaEncrypted.length];
        chachaEngine.processBytes(chachaEncrypted, 0, chachaEncrypted.length, decryptedBytes, 0);

        Log.d("MessageEncryptor", "Decryption complete, decrypted length: " + decryptedBytes.length);
        return decryptedBytes;
    }

    private static byte[] generateKey() {
        byte[] key = new byte[KEY_LENGTH];
        new SecureRandom().nextBytes(key);
        return key;
    }

    private static byte[] generateNonceFromUserInput(String email, String phoneNumber) {
        byte[] nonce = new byte[NONCE_LENGTH];
        String emailPart = email.substring(0, Math.min(4, email.length()));
        String phonePart = phoneNumber.substring(Math.max(0, phoneNumber.length() - 4));
        byte[] emailBytes = emailPart.getBytes();
        byte[] phoneBytes = phonePart.getBytes();
        System.arraycopy(emailBytes, 0, nonce, 0, Math.min(emailBytes.length, 4));
        System.arraycopy(phoneBytes, 0, nonce, 4, Math.min(phoneBytes.length, 4));
        return nonce; // 8 bytes from email (4) + phone (4)
    }

    private static byte[] generateIV() {
        byte[] iv = new byte[IV_LENGTH];
```

```
        new SecureRandom().nextBytes(iv);

        return iv;

    }

}
```

Filename: MessageEntity.java

```
package com.example.protegotinyever.adapt;


import androidx.room.Entity;

import androidx.room.PrimaryKey;

import androidx.room.Ignore;


@Entity(tableName = "messages")

public class MessageEntity {

    @PrimaryKey(autoGenerate = true)

    private long id;

    private String sender;

    private String message;

    private long timestamp;

    private String peerUsername;


    @Ignore

    public MessageEntity() {

        // Required empty constructor for Room

    }


    public MessageEntity(String sender, String message, long timestamp, String peerUsername) {

        this.sender = sender;

        this.message = message;

        this.timestamp = timestamp;

        this.peerUsername = peerUsername;
```

```
}


public long getId() {

   return id;

}


public void setId(long id) {

   this.id = id;

}


public String getSender() {

   return sender;

}


public void setSender(String sender) {

   this.sender = sender;

}


public String getMessage() {

   return message;

}


public void setMessage(String message) {

   this.message = message;

}


public long getTimestamp() {
        return timestamp;
    }


    public void setTimestamp(long timestamp) {
        this.timestamp = timestamp;
    }


    public String getPeerUsername() {
        return peerUsername;
    }


    public void setPeerUsername(String peerUsername) {
        this.peerUsername = peerUsername;
    }
}
```

Filename: MessageModel.java

```
package com.example.protegotinyever.mode;

public class MessageModel {
    private String sender;
    private String text;
    private long timestamp;
    private int rea = 1;

    public MessageModel() {
        // Default constructor for Firebase
    }

    public MessageModel(String sender, String text, long timestamp) {
        this.sender = sender;
        this.text = text;
        this.timestamp = timestamp;
    }

    public String getSender() {
        return sender;
    }

    public String getText() {
        return text;
    }
    public long getTimestamp() {

        return timestamp;

    }

}
```

Filename: NewMessageCallback.java

```
package com.example.protegotinyever.inte;


public interface NewMessageCallback {
    int rea = 1;
    void onNewMessage(String message, String fromPeer);
}
```

Filename: OnboardingActivity.java

```
package com.example.protegotinyever.act;


import android.content.Intent;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.LinearLayout;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.ContextCompat;
import androidx.viewpager2.widget.ViewPager2;
import com.example.protegotinyever.R;
import com.example.protegotinyever.adapt.OnboardingAdapter;
import com.example.protegotinyever.model.OnboardingItem;
import com.example.protegotinyever.util.SessionManager;
import java.util.ArrayList;
import java.util.List;


public class OnboardingActivity extends AppCompatActivity {
    private OnboardingAdapter onboardingAdapter;
    private LinearLayout indicatorLayout;
    private ViewPager2 onboardingViewPager;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
// Check if onboarding was completed before
if (SessionManager.getInstance(this).isOnboardingCompleted()) {
    startLoginActivity();
    return;
}

setContentView(R.layout.activity_onboarding);

indicatorLayout = findViewById(R.id.indicatorLayout);
onboardingViewPager = findViewById(R.id.onboardingViewPager);

setupOnboardingItems();
setupIndicators();
setCurrentIndicator(0);

onboardingViewPager.registerOnPageChangeCallback(new ViewPager2.OnPageChangeCallback() {
    @Override
    public void onPageSelected(int position) {
        super.onPageSelected(position);
        setCurrentIndicator(position);
    }
});

findViewById(R.id.buttonNext).setOnClickListener(view -> {
    if (onboardingViewPager.getCurrentItem() + 1 < onboardingAdapter.getItemCount()) {
        onboardingViewPager.setCurrentItem(onboardingViewPager.getCurrentItem() + 1);
```

```
            } else {
                finishOnboarding();
            }
        });

        findViewById(R.id.buttonSkip).setOnClickListener(view -> finishOnboarding());
    }

    private void setupOnboardingItems() {
        List<OnboardingItem> onboardingItems = new ArrayList<>();

        OnboardingItem secureChat = new OnboardingItem();
        secureChat.setTitle("Secure P2P Chat");
        secureChat.setDescription("Enjoy end-to-end encrypted messaging with direct peer-to-peer connections.
Your messages never pass through servers.");
        secureChat.setImage(R.drawable.ic_secure_chat);

        OnboardingItem fileSharing = new OnboardingItem();
        fileSharing.setTitle("Fast File Sharing");
        fileSharing.setDescription("Share files directly with your contacts. 200MB size limit, no compression,
and maximum speed through P2P transfer.");
        fileSharing.setImage(R.drawable.ic_file_sharing);

        OnboardingItem privacy = new OnboardingItem();
        privacy.setTitle("Privacy First");
        privacy.setDescription("Your data stays on your device. We don't store your messages or files on any
servers. Complete privacy guaranteed.");
```

```
    privacy.setImage(R.drawable.ic_privacy);


    onboardingItems.add(secureChat);
    onboardingItems.add(fileSharing);
    onboardingItems.add(privacy);


    onboardingAdapter = new OnboardingAdapter(onboardingItems);
    onboardingViewPager.setAdapter(onboardingAdapter);
}


private void setupIndicators() {
    ImageView[] indicators = new ImageView[onboardingAdapter.getItemCount()];
    LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT
    );
    layoutParams.setMargins(8, 0, 8, 0);


    for (int i = 0; i < indicators.length; i++) {
        indicators[i] = new ImageView(getApplicationContext());
        indicators[i].setImageDrawable(ContextCompat.getDrawable(
                getApplicationContext(),
                R.drawable.indicator_inactive
        ));
        indicators[i].setLayoutParams(layoutParams);
        indicatorLayout.addView(indicators[i]);
    }
}
```

```
private void setCurrentIndicator(int position) {
    int childCount = indicatorLayout.getChildCount();
    for (int i = 0; i < childCount; i++) {
        ImageView imageView = (ImageView) indicatorLayout.getChildAt(i);
        if (i == position) {
            imageView.setImageDrawable(
                    ContextCompat.getDrawable(getApplicationContext(), R.drawable.indicator_active)
            );
        } else {
            imageView.setImageDrawable(
                    ContextCompat.getDrawable(getApplicationContext(), R.drawable.indicator_inactive)
            );
        }
    }

    // Update button text for last page
    if (position == onboardingAdapter.getItemCount() - 1) {
        findViewById(R.id.buttonNext).setContentDescription("Get Started");
    } else {
        findViewById(R.id.buttonNext).setContentDescription("Next");
    }
}

private void finishOnboarding() {
    SessionManager.getInstance(this).setOnboardingCompleted(true);
    startLoginActivity();
    }



    private void startLoginActivity() {

        startActivity(new Intent(this, LoginActivity.class));

        finish();

    }

}
```

Filename: OnboardingAdapter.java

```
package com.example.protegotinyever.adapt;


import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import com.example.protegotinyever.R;
import com.example.protegotinyever.model.OnboardingItem;
import java.util.List;

public class OnboardingAdapter extends
RecyclerView.Adapter<OnboardingAdapter.OnboardingViewHolder> {

    private List<OnboardingItem> onboardingItems;

    public OnboardingAdapter(List<OnboardingItem> onboardingItems) {
        this.onboardingItems = onboardingItems;
    }

    @NonNull
    @Override
    public OnboardingViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
```

```
    return new OnboardingViewHolder(
            LayoutInflater.from(parent.getContext()).inflate(
                    R.layout.item_onboarding, parent, false
            )
    );
}

@Override
public void onBindViewHolder(@NonNull OnboardingViewHolder holder, int position) {
    holder.setOnboardingData(onboardingItems.get(position));
}

@Override
public int getItemCount() {
    return onboardingItems.size();
}

class OnboardingViewHolder extends RecyclerView.ViewHolder {

    private TextView textTitle;
    private TextView textDescription;
    private ImageView imageOnboarding;

    OnboardingViewHolder(@NonNull View itemView) {
        super(itemView);
        textTitle = itemView.findViewById(R.id.textTitle);
        textDescription = itemView.findViewById(R.id.textDescription);
```

```
            imageOnboarding = itemView.findViewById(R.id.imageOnboarding);
        }


        void setOnboardingData(OnboardingItem onboardingItem) {
            textTitle.setText(onboardingItem.getTitle());
            textDescription.setText(onboardingItem.getDescription());
            imageOnboarding.setImageResource(onboardingItem.getImage());
        }
    }
}
```

Filename: OnboardingItem.java

```
package com.example.protegotinyever.model;


public class OnboardingItem {
    private String title;
    private String description;
    private int image;


    public String getTitle() {
        return title;
    }


    public void setTitle(String title) {
        this.title = title;
    }


    public String getDescription() {
        return description;
    }


    public void setDescription(String description) {
        this.description = description;
    }


    public int getImage() {
        return image;
    }


    public void setImage(int image) {

        this.image = image;

    }

}
```

Filename: ProtegoTinyEverApp.java

```
package com.example.protegotinyever;


import android.app.Application;

import com.example.protegotinyever.util.ThemeManager;


public class ProtegoTinyEverApp extends Application {

    @Override

    public void onCreate() {

        super.onCreate();

        // Initialize theme

        ThemeManager.getInstance(this).initializeTheme();

    }

}
```

Filename: RequestsFragment.java

```
package com.example.protegotinyever.act;


import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.annotation.Nullable;

import androidx.fragment.app.Fragment;

import androidx.recyclerview.widget.LinearLayoutManager;

import androidx.recyclerview.widget.RecyclerView;

import androidx.appcompat.widget.SearchView;


import com.example.protegotinyever.R;

import com.example.protegotinyever.service.ConnectionManager;

import com.example.protegotinyever.tt.UserAdapter;

import com.example.protegotinyever.tt.UserModel;

import com.example.protegotinyever.webrtc.WebRTCClient;


import java.util.ArrayList;

import java.util.List;


public class RequestsFragment extends Fragment {

    private RecyclerView recyclerView;
    private TextView noUsersText;
    private SearchView searchView;
    private UserAdapter adapter;
    private List<UserModel> availableUsers = new ArrayList<>();
    private List<UserModel> filteredUsers = new ArrayList<>();
    private WebRTCClient webRTCClient;
    private UserAdapter.OnUserClickListener userClickListener;
    private List<UserModel> pendingUsers;
    private boolean isViewCreated = false;
    private ConnectionManager connectionManager;
    private String currentUsername;
    private String currentSearchQuery = "";

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        connectionManager = ConnectionManager.getInstance(requireContext());
        currentUsername = requireActivity().getIntent().getStringExtra("username");
    }

    public static RequestsFragment newInstance() {
        return new RequestsFragment();
    }

    @Override
        public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_requests, container, false);
    recyclerView = view.findViewById(R.id.requestsRecyclerView);
    noUsersText = view.findViewById(R.id.noRequestsText);
    searchView = view.findViewById(R.id.searchView);
    setupSearchView();
    return view;
}

@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    isViewCreated = true;
    setupRecyclerView();
    if (pendingUsers != null) {
        updateUsers(pendingUsers);
        pendingUsers = null;
    }
}

@Override
public void onDestroyView() {
    super.onDestroyView();
    isViewCreated = false;
    recyclerView = null;
    noUsersText = null;
    adapter = null;
}
```

```
private void setupSearchView() {
    searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
        @Override
        public boolean onQueryTextSubmit(String query) {
            filterUsers(query);
            return true;
        }

        @Override
        public boolean onQueryTextChange(String newText) {
            filterUsers(newText);
            return true;
        }
    });
}

private void filterUsers(String query) {
    currentSearchQuery = query.toLowerCase().trim();
    filteredUsers.clear();

    for (UserModel user : availableUsers) {
        if (user.getUsername().toLowerCase().contains(currentSearchQuery) ||
            user.getPhone().contains(currentSearchQuery)) {
            filteredUsers.add(user);
        }
    }
    updateVisibility();
    adapter.notifyDataSetChanged();
}

private void updateVisibility() {
    if (getActivity() == null || !isAdded()) return;

    getActivity().runOnUiThread(() -> {
        if (isViewCreated && isAdded()) {
            boolean isEmpty = filteredUsers.isEmpty();
            if (noUsersText != null) {
                noUsersText.setVisibility(isEmpty ? View.VISIBLE : View.GONE);
                if (isEmpty && !currentSearchQuery.isEmpty()) {
                    noUsersText.setText("NO MATCHING USERS FOUND");
                } else {
                    noUsersText.setText("NO CONNECTION REQUESTS");
                }
            }
            if (recyclerView != null) {
                recyclerView.setVisibility(isEmpty ? View.GONE : View.VISIBLE);
            }
        }
    });
}

private void setupRecyclerView() {
    if (recyclerView != null && getContext() != null) {
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
        adapter = new UserAdapter(filteredUsers, userClickListener, true);
        recyclerView.setAdapter(adapter);
    }
}

public void setWebRTCClient(WebRTCClient client) {
    this.webRTCClient = client;
}

public void setUserClickListener(UserAdapter.OnUserClickListener listener) {
    this.userClickListener = listener;
    if (adapter != null && recyclerView != null) {
        adapter = new UserAdapter(availableUsers, listener, true);
        recyclerView.setAdapter(adapter);
    }
}

public void updateUsers(List<UserModel> users) {
    if (!isViewCreated) {
        pendingUsers = users;
        return;
    }

    if (getActivity() == null || !isAdded()) {
        return;
        }


        availableUsers.clear();

        // Only show users that have never been connected and are not the current user

        for (UserModel user : users) {

            if (!connectionManager.isUserConnected(user.getUsername()) &&

                !user.getUsername().equals(currentUsername)) {

                availableUsers.add(user);

            }

        }


        // Apply current search filter

        filterUsers(currentSearchQuery);

    }

}
```

Filename: RTCDataChannelHandler.java

```
package com.example.protegotinyever.util;


import android.util.Log;

import org.webrtc.DataChannel;

import org.webrtc.PeerConnection;

import java.nio.ByteBuffer;


public class RTCDataChannelHandler {

   private static final String TAG = "RTCDataChannelHandler";

   private DataChannel dataChannel;

   private OnMessageReceivedListener listener;

   private int rea = 1;


   public interface OnMessageReceivedListener {

      void onMessageReceived(String message);

   }


   public RTCDataChannelHandler(PeerConnection peerConnection, OnMessageReceivedListener listener) {

      this.listener = listener;

      DataChannel.Init init = new DataChannel.Init();

      this.dataChannel = peerConnection.createDataChannel("chat", init);


      if (this.dataChannel != null) {

         this.dataChannel.registerObserver(new DataChannel.Observer() {

            @Override
            public void onBufferedAmountChange(long previousAmount) {}


            @Override
            public void onStateChange() {
                Log.d(TAG, "DataChannel state changed: " + dataChannel.state());
            }


            @Override
            public void onMessage(DataChannel.Buffer buffer) {
                String receivedMessage = bufferToString(buffer);
                Log.d(TAG, "Message received: " + receivedMessage);
                if (listener != null) {
                    listener.onMessageReceived(receivedMessage);
                }
            }
        });
    }
}


public void sendMessage(String message) {
    if (dataChannel != null && dataChannel.state() == DataChannel.State.OPEN) {
        ByteBuffer buffer = ByteBuffer.wrap(message.getBytes());
        DataChannel.Buffer dataBuffer = new DataChannel.Buffer(buffer, false);
        dataChannel.send(dataBuffer);
        Log.d(TAG, "Message sent: " + message);
    } else {
        Log.e(TAG, "DataChannel is not open. Message not sent.");
        }
    }


    private String bufferToString(DataChannel.Buffer buffer) {
        byte[] data = new byte[buffer.data.remaining()];
        buffer.data.get(data);
        return new String(data);
    }


    public void close() {
        if (dataChannel != null) {
            dataChannel.close();
        }
    }
}
```

Filename: SecuritySetupActivity.java

```
package com.example.protegotinyever.act;


import android.content.Intent;

import android.content.SharedPreferences;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.TextView;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import androidx.biometric.BiometricManager;

import androidx.cardview.widget.CardView;


import com.example.protegotinyever.R;

import com.google.android.material.textfield.TextInputLayout;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Base64;


public class SecuritySetupActivity extends AppCompatActivity {

    private TextView setupStatusText;

    private CardView biometricCard, pinCard;

    private Button enableBiometricButton;

    private EditText pinInput, confirmPinInput;
private Button enablePinButton;
private TextInputLayout pinInputLayout, confirmPinInputLayout;
private SharedPreferences prefs;
private static final String PREFS_NAME = "SecurityPrefs";
private static final String KEY_SECURITY_ENABLED = "security_enabled";
private static final String KEY_SECURITY_TYPE = "security_type";
private static final String KEY_PIN_HASH = "pin_hash";
private static final String TYPE_BIOMETRIC = "biometric";
private static final String TYPE_PIN = "pin";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_security_setup);

    setupStatusText = findViewById(R.id.setupStatusText);
    biometricCard = findViewById(R.id.biometricCard);
    pinCard = findViewById(R.id.pinCard);
    enableBiometricButton = findViewById(R.id.enableBiometricButton);
    pinInput = findViewById(R.id.pinInput);
    confirmPinInput = findViewById(R.id.confirmPinInput);
    enablePinButton = findViewById(R.id.enablePinButton);
    pinInputLayout = findViewById(R.id.pinInputLayout);
    confirmPinInputLayout = findViewById(R.id.confirmPinInputLayout);
    prefs = getSharedPreferences(PREFS_NAME, MODE_PRIVATE);

    setupBiometricOption();
```

```
        setupPinOption();
    }


    private void setupBiometricOption() {
        BiometricManager biometricManager = BiometricManager.from(this);
        boolean biometricAvailable = biometricManager.canAuthenticate(
                                    BiometricManager.Authenticators.BIOMETRIC_STRONG) ==
BiometricManager.BIOMETRIC_SUCCESS;


        if (!biometricAvailable) {
            biometricCard.setVisibility(View.GONE);
            pinCard.setVisibility(View.VISIBLE);
            setupStatusText.setText(R.string.pin_only_available);
            return;
        }


        enableBiometricButton.setOnClickListener(v -> {
            SharedPreferences.Editor editor = prefs.edit();
            editor.putBoolean(KEY_SECURITY_ENABLED, true);
            editor.putString(KEY_SECURITY_TYPE, TYPE_BIOMETRIC);
            editor.apply();


            Toast.makeText(this, R.string.biometric_enabled, Toast.LENGTH_SHORT).show();
            proceedToConnectActivity();
        });
    }
```

```
private void setupPinOption() {
    enablePinButton.setOnClickListener(v -> {
        String pin = pinInput.getText().toString();
        String confirmPin = confirmPinInput.getText().toString();

        if (!validatePin(pin, confirmPin)) {
            return;
        }

        try {
            String hashedPin = hashPin(pin);
            SharedPreferences.Editor editor = prefs.edit();
            editor.putBoolean(KEY_SECURITY_ENABLED, true);
            editor.putString(KEY_SECURITY_TYPE, TYPE_PIN);
            editor.putString(KEY_PIN_HASH, hashedPin);
            editor.apply();

            Toast.makeText(this, R.string.pin_enabled, Toast.LENGTH_SHORT).show();
            proceedToConnectActivity();
        } catch (NoSuchAlgorithmException e) {
            Toast.makeText(this, R.string.pin_setup_error, Toast.LENGTH_SHORT).show();
        }
    });
}

private boolean validatePin(String pin, String confirmPin) {
    if (pin.length() != 4) {
        pinInputLayout.setError(getString(R.string.pin_length_error));
        return false;
    }

    if (!pin.matches("\\d{4}")) {
        pinInputLayout.setError(getString(R.string.pin_digits_only));
        return false;
    }

    if (!pin.equals(confirmPin)) {
        confirmPinInputLayout.setError(getString(R.string.pin_mismatch));
        return false;
    }

    pinInputLayout.setError(null);
    confirmPinInputLayout.setError(null);
    return true;
}

private String hashPin(String pin) throws NoSuchAlgorithmException {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash = digest.digest(pin.getBytes());
    return Base64.getEncoder().encodeToString(hash);
}

private void proceedToConnectActivity() {
    Intent intent = new Intent(this, ConnectActivity.class);
```

```
        intent.putExtra("username", getIntent().getStringExtra("username"));

        intent.putExtra("phoneNumber", getIntent().getStringExtra("phoneNumber"));

        startActivity(intent);

        finish();

    }

}
```

Filename: SessionManager.java

```
package com.example.protegotinyever.util;


import android.content.Context;
import android.content.SharedPreferences;


public class SessionManager {
    private static final String PREFS_NAME = "login_session";
    private static final String KEY_USERNAME = "username";
    private static final String KEY_PHONE = "phone";
    private static final String KEY_IS_LOGGED_IN = "is_logged_in";
    private static final String KEY_ONBOARDING_COMPLETED = "onboarding_completed";


    private static SessionManager instance;
    private final SharedPreferences prefs;


    private SessionManager(Context context) {
        prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    }


    public static synchronized SessionManager getInstance(Context context) {
        if (instance == null) {
            instance = new SessionManager(context.getApplicationContext());
        }
        return instance;
    }
```

```
public void saveLoginSession(String username, String phone) {
    SharedPreferences.Editor editor = prefs.edit();
    editor.putString(KEY_USERNAME, username);
    editor.putString(KEY_PHONE, phone);
    editor.putBoolean(KEY_IS_LOGGED_IN, true);
    editor.apply();
}

public void clearSession() {
    SharedPreferences.Editor editor = prefs.edit();
    editor.clear();
    editor.apply();
}

public boolean isLoggedIn() {
    return prefs.getBoolean(KEY_IS_LOGGED_IN, false);
}

public String getUsername() {
    return prefs.getString(KEY_USERNAME, null);
}

public String getPhone() {
    return prefs.getString(KEY_PHONE, null);
}
```

```
  public boolean isOnboardingCompleted() {

    return prefs.getBoolean(KEY_ONBOARDING_COMPLETED, false);

  }


  public void setOnboardingCompleted(boolean completed) {

    SharedPreferences.Editor editor = prefs.edit();

    editor.putBoolean(KEY_ONBOARDING_COMPLETED, completed);

    editor.apply();

  }

}
```

Filename: SignalingCallback.java

```
package com.example.protegotinyever.inte;


public interface SignalingCallback {

    int rea = 1;

    void onSignalingReceived(String type, String data, String sender); // ? Add sender

}
```

Filename: SignUpActivity.java

```
package com.example.protegotinyever.act;


import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.example.protegotinyever.R;
import com.example.protegotinyever.util.AuthManager;
import com.example.protegotinyever.util.SessionManager;


public class SignUpActivity extends AppCompatActivity {
    private EditText emailInput, passwordInput, usernameInput, phoneInput;
    private Button signUpButton;
    private TextView loginLink;
    private AuthManager authManager;
    private SessionManager sessionManager;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_signup);
```

```
authManager = AuthManager.getInstance(this);
sessionManager = SessionManager.getInstance(this);

emailInput = findViewById(R.id.emailInput);
passwordInput = findViewById(R.id.passwordInput);
usernameInput = findViewById(R.id.usernameInput);
phoneInput = findViewById(R.id.phoneInput);
signUpButton = findViewById(R.id.signUpButton);
loginLink = findViewById(R.id.loginLink);

signUpButton.setOnClickListener(v -> {
   String email = emailInput.getText().toString().trim();
   String password = passwordInput.getText().toString().trim();
   String username = usernameInput.getText().toString().trim();
   String phone = phoneInput.getText().toString().trim();

   if (validateInputs(email, password, username, phone)) {
      signUpButton.setEnabled(false);
      authManager.signUp(email, password, username, phone)
         .addOnSuccessListener(authResult -> {
            sessionManager.saveLoginSession(username, phone);
            Intent intent = new Intent(SignUpActivity.this, EmailVerificationActivity.class);
            intent.putExtra("username", username);
            intent.putExtra("phoneNumber", phone);
            startActivity(intent);
            finish();
         })
```

```
                .addOnFailureListener(e -> {
                    signUpButton.setEnabled(true);
                    Toast.makeText(SignUpActivity.this,
                        "Registration failed: " + e.getMessage(),
                        Toast.LENGTH_SHORT).show();
                });
        }
    });

    loginLink.setOnClickListener(v -> {
        startActivity(new Intent(this, LoginActivity.class));
        finish();
    });
}

private boolean validateInputs(String email, String password, String username, String phone) {
    if (email.isEmpty()) {
        emailInput.setError("Email is required");
        return false;
    }
    if (!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        emailInput.setError("Please enter a valid email");
        return false;
    }
    if (password.isEmpty()) {
        passwordInput.setError("Password is required");
        return false;
        }
        if (password.length() < 6) {
            passwordInput.setError("Password must be at least 6 characters");
            return false;
        }
        if (username.isEmpty()) {
            usernameInput.setError("Username is required");
            return false;
        }
        if (phone.isEmpty()) {
            phoneInput.setError("Phone number is required");
            return false;
        }
        return true;
    }
}
```

Filename: SuccessCallback.java

```
package com.example.protegotinyever.inte;


public interface SuccessCallback {
    int rea = 1;
    void onSuccess();
}
```

Filename: ThemeManager.java

```
package com.example.protegotinyever.util;


import android.content.Context;
import android.content.SharedPreferences;
import androidx.appcompat.app.AppCompatDelegate;


public class ThemeManager {
    private static final String PREFS_NAME = "ThemePrefs";
    private static final String KEY_DARK_MODE = "dark_mode";
    private static ThemeManager instance;
    private final SharedPreferences prefs;


    private ThemeManager(Context context) {
        prefs = context.getApplicationContext()
                .getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    }


    public static ThemeManager getInstance(Context context) {
        if (instance == null) {
            instance = new ThemeManager(context);
        }
        return instance;
    }


    public boolean isDarkMode() {
        return prefs.getBoolean(KEY_DARK_MODE, true);
    }


    public void setDarkMode(boolean isDark) {
        prefs.edit().putBoolean(KEY_DARK_MODE, isDark).apply();
        applyTheme(isDark);
    }


    public void applyTheme(boolean isDark) {
        AppCompatDelegate.setDefaultNightMode(
            isDark ? AppCompatDelegate.MODE_NIGHT_YES : AppCompatDelegate.MODE_NIGHT_NO
        );
    }


    public void initializeTheme() {
        applyTheme(isDarkMode());
    }
}
```

Filename: UserAdapter.java

```
package com.example.protegotinyever.tt;


import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ImageView;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import com.example.protegotinyever.R;

import com.example.protegotinyever.service.ConnectionManager;

import com.example.protegotinyever.webrtc.WebRTCClient;


import org.webrtc.DataChannel;


import java.util.List;


public class UserAdapter extends RecyclerView.Adapter<UserAdapter.UserViewHolder> {

    private final List<UserModel> userList;

    private final OnUserClickListener listener;

    private final boolean isRequestsTab;

    private int rea = 1;
```

```
public UserAdapter(List<UserModel> userList, OnUserClickListener listener, boolean isRequestsTab) {
    this.userList = userList;
    this.listener = listener;
    this.isRequestsTab = isRequestsTab;
}

@NonNull
@Override
public UserViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_user, parent, false);
    return new UserViewHolder(view);
}

@Override
public void onBindViewHolder(@NonNull UserViewHolder holder, int position) {
    UserModel user = userList.get(position);

    // Set username and its first letter
    holder.usernameText.setText(user.getUsername().toUpperCase());
    if (!user.getUsername().isEmpty()) {
        holder.usernameLetter.setText(String.valueOf(user.getUsername().charAt(0)).toUpperCase());
    }

    // Set phone number
    holder.phoneText.setText(user.getPhone());

    // Set online status dot
    holder.onlineStatusDot.setVisibility(user.isOnline() ? View.VISIBLE : View.GONE);
    holder.onlineStatusDot.setColorFilter(holder.itemView.getContext().getColor(R.color.success_green));

    // Check WebRTC connection status
    WebRTCClient webRTCClient = WebRTCClient.getInstance(holder.itemView.getContext(), null);
    DataChannel dataChannel = webRTCClient.getDataChannels().get(user.getUsername());

    // Update connection button color based on status
    updateConnectionButtonStatus(holder.connectButton, dataChannel, webRTCClient, user.getUsername());

    // Set button click listeners
    holder.connectButton.setOnClickListener(v -> {
        if (listener != null) {
            listener.onConnectionButtonClick(user);
        }
    });

    // Only show and setup chat button in chats tab
    if (isRequestsTab) {
        holder.chatButton.setVisibility(View.GONE);
    } else {
        holder.chatButton.setVisibility(View.VISIBLE);
        holder.chatButton.setOnClickListener(v -> {
            if (listener != null) {
                listener.onChatButtonClick(user);
            }
            });
        }
    }

    private void updateConnectionButtonStatus(Button button, DataChannel dataChannel, WebRTCClient
webRTCClient, String username) {
        int backgroundColor;
        String statusText;

        if (dataChannel != null && dataChannel.state() == DataChannel.State.OPEN) {
            backgroundColor = R.color.success_green;
            statusText = "Connected";
        } else if (webRTCClient.isAttemptingConnection(username)) {
            backgroundColor = R.color.warning_yellow;
            statusText = "Connecting";
        } else if (!isRequestsTab &&
ConnectionManager.getInstance(button.getContext()).isUserConnected(username)) {
            backgroundColor = R.color.warning_yellow;
            statusText = "Offline";
        } else {
            backgroundColor = R.color.error_red;
            statusText = "Connect";
        }

        button.getContext().getResources().getColor(backgroundColor, button.getContext().getTheme());
        button.setBackgroundColor(button.getContext().getResources().getColor(backgroundColor,
button.getContext().getTheme()));
```

```
    button.setText(statusText);
}


@Override
public int getItemCount() {
    return userList.size();
}


public static class UserViewHolder extends RecyclerView.ViewHolder {
    TextView usernameText, phoneText, usernameLetter;
    EditText statusText;
    ImageView onlineStatusDot;
    Button connectButton, chatButton;


    public UserViewHolder(View itemView) {
        super(itemView);
        usernameText = itemView.findViewById(R.id.usernameText);
        phoneText = itemView.findViewById(R.id.phoneText);
        usernameLetter = itemView.findViewById(R.id.usernameLetter);
        onlineStatusDot = itemView.findViewById(R.id.onlineStatusDot);
        connectButton = itemView.findViewById(R.id.connectButton);
        chatButton = itemView.findViewById(R.id.chatButton);
    }
}


public interface OnUserClickListener {
    void onConnectionButtonClick(UserModel user);
```

```
        void onChatButtonClick(UserModel user);

    }

}
```

Filename: UserListCallback.java

```
package com.example.protegotinyever.mode;


import java.util.List;

import com.example.protegotinyever.tt.UserModel;


public interface UserListCallback {

    int rea = 1;

    void onUsersFetched(List<UserModel> users);

}
```

Filename: UserModel.java

```
package com.example.protegotinyever.tt;


public class UserModel {
    private String username;
    private String phone;
    private boolean isOnline;
    private int rea = 1;

    // Default constructor (important for Firebase!)
    public UserModel() {
    }

    // Constructor
    public UserModel(String username, String phone, boolean isOnline) {
        this.username = username;
        this.phone = phone;
        this.isOnline = isOnline;
    }

    public UserModel(String username, String phone) {
        this.username = username;
        this.phone = phone;
    }

    // Getters & Setters (needed for Firebase to map data)
    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public boolean isOnline() {
        return isOnline;
    }

    public void setOnline(boolean online) {
        isOnline = online;
    }
}
```

Filename: WebRTCClient.java

```
package com.example.protegotinyever.webrtc;


import android.content.ContentResolver;

import android.content.ContentValues;

import android.content.Context;

import android.net.Uri;

import android.os.Environment;

import android.provider.MediaStore;

import android.util.Log;

import android.widget.Toast;


import com.example.protegotinyever.adapt.MessageEntity;

import com.example.protegotinyever.service.ConnectionManager;

import com.example.protegotinyever.service.WebRTCService;

import com.example.protegotinyever.util.CustomSdpObserver;

import com.example.protegotinyever.util.DataChannelHandler;

import com.example.protegotinyever.util.FirebaseClient;

import com.example.protegotinyever.util.MessageEncryptor;


import org.webrtc.*;


import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;
```

```java
import java.io.OutputStream;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;


public class WebRTCClient {
    private static WebRTCClient instance;
    private Map<String, PeerConnection> peerConnections;
    private Map<String, DataChannel> dataChannels;
    private PeerConnectionFactory peerConnectionFactory;
    private FirebaseClient firebaseClient;
    private String currentPeerUsername;
    private WebRTCListener webrtcListener;
    private ProgressListener progressListener;
    private boolean isConnected = false;
    private boolean isAttemptingConnection = false;
    private final Context context;
    private Map<String, Boolean> hasSentOffers;
    private boolean isBackgroundMode = false;
    private WebRTCService webRTCService;
    private DataChannelHandler dataChannelHandler;
    private static final int CHUNK_SIZE = 16384; // 16 KB chunks
    private static final int BUFFER_SIZE = 8 * 1024 * 1024; // 16 MB buffer (reference)
    private Map<String, Long> lastSentOffsets = new HashMap<>();
    private Map<String, String> pendingFileTransfers = new HashMap<>();
    private final ExecutorService fileWriterExecutor = Executors.newSingleThreadExecutor(); // For offloading
file writes

    public static WebRTCClient getInstance(Context context, FirebaseClient firebaseClient) {
        if (instance == null) {
            instance = new WebRTCClient(context.getApplicationContext(), firebaseClient);
        } else if (instance.firebaseClient == null) {
            instance.firebaseClient = firebaseClient;
            instance.listenForSignaling();
        }
        return instance;
    }

    private WebRTCClient(Context context, FirebaseClient firebaseClient) {
        this.context = context.getApplicationContext();
        this.firebaseClient = firebaseClient;
        this.peerConnections = new HashMap<>();
        this.dataChannels = new HashMap<>();
        this.hasSentOffers = new HashMap<>();
        this.dataChannelHandler = DataChannelHandler.getInstance(context);
        initializePeerConnectionFactory(context);
        listenForSignaling();
    }

    public void setWebRTCListener(WebRTCListener listener) {
```

```
    this.webrtcListener = listener;
  }


  public void setProgressListener(ProgressListener listener) {
    this.progressListener = listener;
  }


  private void initializePeerConnectionFactory(Context context) {
    PeerConnectionFactory.initialize(
        PeerConnectionFactory.InitializationOptions.builder(context).createInitializationOptions()
    );
    peerConnectionFactory = PeerConnectionFactory.builder().createPeerConnectionFactory();
  }


  private void listenForSignaling() {
    firebaseClient.listenForSignaling((type, data, sender) -> {
      Log.d("WebRTC", "? Received signaling data: " + type + " from " + sender);
      currentPeerUsername = sender;
      switch (type) {
        case "OFFER":
          Log.d("WebRTC", "? Received Offer from " + currentPeerUsername);
          if (ConnectionManager.getInstance(context).isUserConnected(currentPeerUsername)) {
                Log.d("WebRTC", "Auto-accepting connection from previously connected user: " +
currentPeerUsername);
            acceptConnection(currentPeerUsername, data);
          } else if (!isConnected(currentPeerUsername)) {
            if (webRTCService != null) {
                  webRTCService.showConnectionRequestNotification(currentPeerUsername, data);
            } else {
                  Log.e("WebRTC", "Cannot show connection request - service not available");
            }
          }
          break;
        case "ANSWER":
          Log.d("WebRTC", "? Received Answer from " + currentPeerUsername);
          receiveAnswer(data);
          break;
        case "ICE":
          Log.d("WebRTC", "? Received ICE Candidate from " + currentPeerUsername);
          receiveIceCandidate(data);
          break;
      }
    });
  }

  public void startConnection(String peerUsername) {
    this.currentPeerUsername = peerUsername;
    setupPeerConnection(peerUsername);
    createOffer(peerUsername);
  }

  private void setupPeerConnection(String peerUsername) {
    if (!peerConnections.containsKey(peerUsername)) {
      Log.d("WebRTC", "? Setting up new PeerConnection for " + peerUsername);
```

```
        ArrayList<PeerConnection.IceServer> iceServers = new ArrayList<>();
        iceServers.add(PeerConnection.IceServer.builder("stun:stun.l.google.com:19302").createIceServer());
        iceServers.add(PeerConnection.IceServer.builder("stun:stun1.l.google.com:19302").createIceServer());
        iceServers.add(PeerConnection.IceServer.builder("stun:stun2.l.google.com:19302").createIceServer());
        iceServers.add(PeerConnection.IceServer.builder("stun:stun3.l.google.com:19302").createIceServer());

        PeerConnection.RTCConfiguration rtcConfig = new PeerConnection.RTCConfiguration(iceServers);
        rtcConfig.enableDtlsSrtp = true;
        rtcConfig.sdpSemantics = PeerConnection.SdpSemantics.UNIFIED_PLAN;
        rtcConfig.bundlePolicy = PeerConnection.BundlePolicy.MAXBUNDLE;
        rtcConfig.rtcpMuxPolicy = PeerConnection.RtcpMuxPolicy.REQUIRE;
                                                            rtcConfig.continualGatheringPolicy       =
PeerConnection.ContinualGatheringPolicy.GATHER_CONTINUALLY;
        rtcConfig.keyType = PeerConnection.KeyType.ECDSA;

            PeerConnection peerConnection = peerConnectionFactory.createPeerConnection(rtcConfig, new
PeerConnection.Observer() {
          @Override
          public void onIceCandidate(IceCandidate iceCandidate) {
              firebaseClient.sendSignalingData(peerUsername, "ICE", iceCandidate.sdp);
          }

          @Override
          public void onIceConnectionChange(PeerConnection.IceConnectionState iceConnectionState) {
                    Log.d("WebRTC", "? ICE connection state changed for " + peerUsername + ": " +
iceConnectionState);
              switch (iceConnectionState) {
```

```
            case CONNECTED:
                if (webrtcListener != null) webrtcListener.onConnected();
                break;
            case FAILED:
            case DISCONNECTED:
            case CLOSED:
                if (webrtcListener != null) webrtcListener.onConnectionFailed();
                if (ConnectionManager.getInstance(context).isUserConnected(peerUsername)) {
                    Log.d("WebRTC", "Connection lost, attempting to reconnect to " + peerUsername);
                    startConnection(peerUsername);
                }
                break;
        }
    }

    @Override
    public void onSignalingChange(PeerConnection.SignalingState signalingState) {}
    @Override
    public void onIceGatheringChange(PeerConnection.IceGatheringState iceGatheringState) {}
    @Override
    public void onIceConnectionReceivingChange(boolean b) {}
    @Override
    public void onIceCandidatesRemoved(IceCandidate[] iceCandidates) {}
    @Override
    public void onAddStream(MediaStream mediaStream) {}
    @Override
    public void onRemoveStream(MediaStream mediaStream) {}
```

```
            @Override
            public void onDataChannel(DataChannel dataChannel) {
                Log.d("WebRTC", "DataChannel received for peer: " + peerUsername);
                dataChannelHandler.setCurrentPeer(peerUsername);
                dataChannelHandler.setDataChannel(dataChannel);
                setupDataChannelObserver(dataChannel, peerUsername);
            }
            @Override
            public void onRenegotiationNeeded() {}
            @Override
            public void onAddTrack(RtpReceiver rtpReceiver, MediaStream[] mediaStreams) {}
            @Override
            public void onTrack(RtpTransceiver transceiver) {}
        });

        DataChannel.Init init = new DataChannel.Init();
        init.maxRetransmits = 0;
        init.ordered = true;
        DataChannel dataChannel = peerConnection.createDataChannel("chat", init);
        peerConnections.put(peerUsername, peerConnection);
        dataChannels.put(peerUsername, dataChannel);
        hasSentOffers.put(peerUsername, false);
        dataChannelHandler.setCurrentPeer(peerUsername);
        dataChannelHandler.setDataChannel(dataChannel);
        setupDataChannelObserver(dataChannel, peerUsername);
    }
}
```

```
   private void setupDataChannelObserver(DataChannel dataChannel, String peerUsername) {
       dataChannel.registerObserver(new DataChannel.Observer() {
           private String fileName;
           private String fileType;
           private long totalLength = -1;
           private File tempFile;
           private FileOutputStream fos;
           private int chunksReceived = 0;
           private long lastReceivedOffset = -1;

           @Override
           public void onBufferedAmountChange(long previousAmount) {
               long currentBuffer = dataChannel.bufferedAmount();
               Log.d("WebRTCClient", "Buffered amount changed for " + peerUsername + ": " + previousAmount
+ ", current: " + currentBuffer);
               if (currentBuffer > BUFFER_SIZE * 0.75) {
                   Log.w("WebRTCClient", "Buffer nearing capacity for " + peerUsername + ": " + currentBuffer);
               }
           }

           @Override
           public void onStateChange() {
                       Log.d("WebRTCClient", "DataChannel state changed for " + peerUsername + ": " +
dataChannel.state());
               onDataChannelStateChange(peerUsername, dataChannel.state());
           }
```

```java
@Override
public void onMessage(DataChannel.Buffer buffer) {
    byte[] data = new byte[buffer.data.remaining()];
    buffer.data.get(data);
    Log.d("WebRTCClient", "Received data from " + peerUsername + ", length: " + data.length);

    try {
        String header = new String(data, 0, Math.min(100, data.length), "UTF-8");
        if (header.startsWith("FILE:")) {
            String[] parts = header.split(":", 5);
            fileName = parts[1];
            fileType = parts[2];
            totalLength = Long.parseLong(parts[3]);
            tempFile = File.createTempFile("recv_", fileName, context.getCacheDir());
            fos = new FileOutputStream(tempFile);
            chunksReceived = 0;
            lastReceivedOffset = -1;
            Log.d("WebRTCClient", "Received file metadata from " + peerUsername + ": " + header);
        } else if (header.startsWith("CHUNK:") && totalLength != -1) {
            String[] parts = header.split(":", 4);
            long chunkTotalLength = Long.parseLong(parts[1]);
            int offset = Integer.parseInt(parts[2]);
            int headerLength = parts[0].length() + parts[1].length() + parts[2].length() + 3;
            byte[] chunkData = new byte[data.length - headerLength];
            System.arraycopy(data, headerLength, chunkData, 0, chunkData.length);
```

```
                    if (offset <= lastReceivedOffset) {
                        Log.w("WebRTCClient", "Duplicate chunk received from " + peerUsername + " at offset: "
+ offset + ", skipping");
                        return;
                    }

                    if (progressListener != null) {
                        int totalChunks = (int) Math.ceil((double) totalLength / CHUNK_SIZE);
                        int decryptProgress = (int) ((chunksReceived + 0.5) * 100 / totalChunks);
                        progressListener.onProgress("Decrypting", decryptProgress, fileName);
                    }
                    byte[] decryptedChunk = MessageEncryptor.decryptData(chunkData);
                    chunksReceived++;
                    lastReceivedOffset = offset;

                    fileWriterExecutor.execute(() -> {
                        try {
                            fos.write(decryptedChunk);
                            if (chunksReceived % 50 == 0) { // Flush more frequently (~800 KB)
                                fos.flush();
                                Log.d("WebRTCClient", "Flushed file output stream for " + peerUsername + " at offset:
" + offset);
                            }
                            Log.d("WebRTCClient", "Received and wrote chunk from " + peerUsername + ", offset: "
+ offset + ", length: " + decryptedChunk.length);

                            if (progressListener != null) {
```

```
            int totalChunks = (int) Math.ceil((double) totalLength / CHUNK_SIZE);
            int receiveProgress = (int) (chunksReceived * 100 / totalChunks);
            progressListener.onProgress("Receiving", receiveProgress, fileName);
        }

        if (offset + decryptedChunk.length >= totalLength) {
            fos.flush();
            fos.close();
                    File savedFile = saveFileToInternalStorage(tempFile, fileName, fileType,
peerUsername);
            if (!tempFile.delete()) {
                                Log.w("WebRTCClient", "Failed to delete temp file: " +
tempFile.getAbsolutePath());
            }
            if (savedFile != null) {
                String message = "Received file: " + fileName + " at " + savedFile.getAbsolutePath();
                dataChannelHandler.onMessageReceived(peerUsername, message);
                if (webrtcListener != null) {
                    webrtcListener.onMessageReceived(message, peerUsername);
                        dataChannel.send(new DataChannel.Buffer(ByteBuffer.wrap(("ACK:" +
savedFile.getAbsolutePath() + ":" + fileName).getBytes("UTF-8")), true));
                        Log.d("WebRTCClient", "Sent ACK for file: " + fileName + " to " +
peerUsername);
                }
            } else {
                throw new IOException("Failed to save decrypted file");
            }
```

```
                        fileName = null;
                        fileType = null;
                        totalLength = -1;
                        fos = null;
                        chunksReceived = 0;
                        lastReceivedOffset = -1;
                    }
                } catch (IOException e) {
                    Log.e("WebRTCClient", "Error writing chunk at offset " + offset + ": " + e.getMessage(),
e);
                }
            });
        } else if (header.startsWith("ACK:")) {
            String[] parts = header.split(":", 3);
            String filePath = parts[1];
            String ackFileName = parts[2];
            if (webrtcListener != null) {
                webrtcListener.onFileSent(filePath, ackFileName);
                lastSentOffsets.remove(peerUsername);
                pendingFileTransfers.remove(peerUsername);
            }
        } else if (header.startsWith("HEARTBEAT:")) {
            Log.d("WebRTCClient", "Received heartbeat from " + peerUsername);
                                                                                    dataChannel.send(new
DataChannel.Buffer(ByteBuffer.wrap("HEARTBEAT:ACK".getBytes("UTF-8")), true));
        } else {
            String message = MessageEncryptor.decryptMessage(data);
```

```
                Log.d("WebRTCClient", "Decrypted message from " + peerUsername + ": " + message);
                dataChannelHandler.onMessageReceived(peerUsername, message);
                if (webrtcListener != null) {
                    webrtcListener.onMessageReceived(message, peerUsername);
                }
            }
        } catch (Exception e) {
                 Log.e("WebRTCClient", "Error processing message from " + peerUsername + " at offset " +
lastReceivedOffset + ": " + e.getMessage(), e);
            new android.os.Handler(android.os.Looper.getMainLooper()).post(() ->
                                    Toast.makeText(context, "Failed to process message: " + e.getMessage(),
Toast.LENGTH_LONG).show()
            );
            if (progressListener != null) {
                progressListener.onProgress("Error", 0, fileName);
            }
            if (fos != null) {
                try {
                    fos.close();
                    if (tempFile != null && tempFile.exists() && !tempFile.delete()) {
                                        Log.w("WebRTCClient", "Failed to delete temp file after error: " +
tempFile.getAbsolutePath());
                    }
                } catch (IOException ex) {
                    Log.e("WebRTCClient", "Error closing temp file: " + ex.getMessage(), ex);
                }
            }
            }
        }
    });

    new Thread(() -> {
        while (dataChannel.state() == DataChannel.State.OPEN) {
            try {
                                                                    dataChannel.send(new
DataChannel.Buffer(ByteBuffer.wrap("HEARTBEAT:".getBytes("UTF-8")), true));
                Thread.sleep(5000);
            } catch (Exception e) {
                Log.e("WebRTCClient", "Heartbeat failed for " + peerUsername + ": " + e.getMessage());
                break;
            }
        }
    }).start();
  }

  public void sendEncryptedMessage(String message, String peerUsername) {
    try {
        sendEncryptedMessage(message.getBytes("UTF-8"), peerUsername, false, null, null);
    } catch (Exception e) {
        Log.e("WebRTCClient", "Error encoding string message: " + e.getMessage());
        dataChannelHandler.storeMessage(message, peerUsername, "You");
    }
  }
```

```
    public void sendEncryptedMessage(byte[] data, String peerUsername, boolean isFile, String fileName,
String fileType) {
        DataChannel dataChannel = dataChannels.get(peerUsername);
        if (dataChannel == null || dataChannel.state() != DataChannel.State.OPEN) {
            Log.e("WebRTCClient", "No open data channel for " + peerUsername + ", storing data");
                dataChannelHandler.storeMessage(isFile ? "File: " + fileName : new String(data), peerUsername,
"You");
            startConnection(peerUsername);
            return;
        }

        try {
            String senderPhone = firebaseClient.getCurrentUserPhone();
            String senderEmail = senderPhone + "@example.com";
                MessageEncryptor.EncryptionResult result = MessageEncryptor.encryptData(data, senderEmail,
senderPhone);
            dataChannel.send(new DataChannel.Buffer(ByteBuffer.wrap(result.combinedData), true));
                    Log.d("WebRTCClient", "Sent encrypted message to " + peerUsername + ", length: " +
result.combinedData.length);
            dataChannelHandler.storeMessage(new String(data), peerUsername, "You");
        } catch (Exception e) {
            Log.e("WebRTCClient", "Error sending encrypted data to " + peerUsername + ": " + e.getMessage());
                dataChannelHandler.storeMessage(isFile ? "File: " + fileName : new String(data), peerUsername,
"You");
            startConnection(peerUsername);
        }
    }
```

```
   public void sendFile(Uri fileUri, String peerUsername, String fileName, String fileType) throws Exception
{
       DataChannel dataChannel = dataChannels.get(peerUsername);
       if (dataChannel == null || dataChannel.state() != DataChannel.State.OPEN) {
           Log.e("WebRTCClient", "No open data channel for " + peerUsername + ", storing data");
           dataChannelHandler.storeMessage("File: " + fileName, peerUsername, "You");
           pendingFileTransfers.put(peerUsername, fileUri.toString());
           startConnection(peerUsername);
           return;
       }

       pendingFileTransfers.put(peerUsername, fileUri.toString());
       try (InputStream inputStream = context.getContentResolver().openInputStream(fileUri)) {
           if (inputStream == null) {
               throw new IOException("Unable to open input stream for URI: " + fileUri);
           }

           long fileSize = context.getContentResolver().openFileDescriptor(fileUri, "r").getStatSize();
           String senderPhone = firebaseClient.getCurrentUserPhone();
           String senderEmail = senderPhone + "@example.com";
           long resumeOffset = lastSentOffsets.getOrDefault(peerUsername, 0L);

           String metadata = "FILE:" + fileName + ":" + fileType + ":" + fileSize + ":";
           byte[] metadataBytes = metadata.getBytes("UTF-8");
           dataChannel.send(new DataChannel.Buffer(ByteBuffer.wrap(metadataBytes), true));
           Log.d("WebRTCClient", "Sent file metadata to " + peerUsername + ": " + metadata);
```

```
        int totalChunks = (int) Math.ceil((double) fileSize / CHUNK_SIZE);
        int chunksProcessed = (int) (resumeOffset / CHUNK_SIZE);
        inputStream.skip(resumeOffset);

        byte[] buffer = new byte[CHUNK_SIZE];
        int bytesRead;
        while ((bytesRead = inputStream.read(buffer)) != -1) {
            byte[] chunk = new byte[bytesRead];
            System.arraycopy(buffer, 0, chunk, 0, bytesRead);
            int offset = chunksProcessed * CHUNK_SIZE;

            if (progressListener != null) {
                int encryptProgress = (int) ((chunksProcessed + 0.5) * 100 / totalChunks);
                progressListener.onProgress("Encrypting", encryptProgress, fileName);
            }
            MessageEncryptor.EncryptionResult result = MessageEncryptor.encryptData(chunk, senderEmail,
senderPhone);
            chunksProcessed++;

            String chunkHeader = "CHUNK:" + fileSize + ":" + offset + ":";
            byte[] chunkHeaderBytes = chunkHeader.getBytes("UTF-8");
            byte[] chunkWithHeader = new byte[chunkHeaderBytes.length + result.combinedData.length];
            System.arraycopy(chunkHeaderBytes, 0, chunkWithHeader, 0, chunkHeaderBytes.length);
                    System.arraycopy(result.combinedData, 0, chunkWithHeader, chunkHeaderBytes.length,
result.combinedData.length);
```

```
                DataChannel.Buffer dataBuffer = new DataChannel.Buffer(ByteBuffer.wrap(chunkWithHeader),
true);
            long bufferedAmount = dataChannel.bufferedAmount();

            // Throttle sending
                        while (bufferedAmount > BUFFER_SIZE * 0.75 && dataChannel.state() ==
DataChannel.State.OPEN) {
                Thread.sleep(100); // Brief sleep to avoid tight loop
                bufferedAmount = dataChannel.bufferedAmount();
            }

            if (!dataChannel.send(dataBuffer)) {
                lastSentOffsets.put(peerUsername, (long) offset);
                throw new IOException("Failed to send chunk at offset " + offset);
            }

            // Update progress more frequently
            if (progressListener != null) {
                int sendProgress = (int) (chunksProcessed * 100 / totalChunks);
                progressListener.onProgress("Sending", sendProgress, fileName);
            }
        }

        Log.d("WebRTCClient", "Sent encrypted file to " + peerUsername + ", total length: " + fileSize);
        dataChannelHandler.storeMessage("File: " + fileName, peerUsername, "You");
        pendingFileTransfers.remove(peerUsername);
    } catch (Exception e) {
        Log.e("WebRTCClient", "Error sending file to " + peerUsername + ": " + e.getMessage(), e);
        throw e;
    }
}

private void createOffer(String peerUsername) {
    if (Boolean.TRUE.equals(hasSentOffers.get(peerUsername))) {
        Log.d("WebRTC", "? Offer already sent to " + peerUsername + ", skipping...");
        return;
    }

    PeerConnection peerConnection = peerConnections.get(peerUsername);
    if (peerConnection != null) {
        Log.d("WebRTC", "? Creating WebRTC offer for " + peerUsername);
        peerConnection.createOffer(new CustomSdpObserver() {
            @Override
            public void onCreateSuccess(SessionDescription sessionDescription) {
                peerConnection.setLocalDescription(new CustomSdpObserver(), sessionDescription);
                firebaseClient.sendSignalingData(peerUsername, "OFFER", sessionDescription.description);
                hasSentOffers.put(peerUsername, true);
            }
        }, new MediaConstraints());
    }
}

private void receiveAnswer(String sdp) {
    if (currentPeerUsername == null) {
        Log.e("WebRTC", "? Cannot process answer: currentPeerUsername is null");
        return;
    }

    PeerConnection peerConnection = peerConnections.get(currentPeerUsername);
    if (peerConnection != null) {
        Log.d("WebRTC", "? Processing answer from " + currentPeerUsername);
        SessionDescription answer = new SessionDescription(SessionDescription.Type.ANSWER, sdp);
        peerConnection.setRemoteDescription(new CustomSdpObserver(), answer);
    }
}

private void receiveIceCandidate(String sdp) {
    if (currentPeerUsername == null) {
        Log.e("WebRTC", "? Cannot process ICE candidate: currentPeerUsername is null");
        return;
    }

    PeerConnection peerConnection = peerConnections.get(currentPeerUsername);
    if (peerConnection != null) {
        Log.d("WebRTC", "? Processing ICE candidate from " + currentPeerUsername);
        IceCandidate iceCandidate = new IceCandidate("audio", 0, sdp);
        peerConnection.addIceCandidate(iceCandidate);
    }
}

public boolean isConnected(String peerUsername) {
    PeerConnection peerConnection = peerConnections.get(peerUsername);
    DataChannel dataChannel = dataChannels.get(peerUsername);
    if (peerConnection == null || dataChannel == null) return false;
    boolean isConnected = peerConnection.getLocalDescription() != null &&
            peerConnection.getRemoteDescription() != null &&
            dataChannel.state() == DataChannel.State.OPEN;
    Log.d("WebRTC", "Connection status for " + peerUsername + ": " + isConnected);
    return isConnected;
}

public void disconnectPeer(String peerUsername) {
    DataChannel dataChannel = dataChannels.remove(peerUsername);
    if (dataChannel != null) {
        dataChannel.close();
    }
    PeerConnection peerConnection = peerConnections.remove(peerUsername);
    if (peerConnection != null) {
        peerConnection.close();
    }
    hasSentOffers.remove(peerUsername);
    lastSentOffsets.remove(peerUsername);
    if (peerUsername.equals(currentPeerUsername)) {
        currentPeerUsername = null;
    }
}

public void disconnect() {
    for (DataChannel dataChannel : dataChannels.values()) {
        if (dataChannel != null) {
            dataChannel.close();
        }
    }
    for (PeerConnection peerConnection : peerConnections.values()) {
        if (peerConnection != null) {
            peerConnection.close();
        }
    }
    dataChannels.clear();
    peerConnections.clear();
    hasSentOffers.clear();
    lastSentOffsets.clear();
    currentPeerUsername = null;
    firebaseClient = null;
}

public boolean isAttemptingConnection(String peerUsername) {
    PeerConnection peerConnection = peerConnections.get(peerUsername);
    if (peerConnection == null) return false;
    return peerConnection.getRemoteDescription() == null;
}

public interface WebRTCListener {
    void onConnected();
    void onConnectionFailed();
    void onMessageReceived(String message, String peerUsername);
    void onFileSent(String filePath, String fileName);
}

public interface ProgressListener {
    void onProgress(String operation, int progress, String fileName);
}

public static void cleanup() {
    if (instance != null) {
        instance.disconnect();
        instance = null;
    }
}

public void onBackground() {
    isBackgroundMode = true;
    Log.d("WebRTC", "App going to background, maintaining connections");
}

public void onForeground() throws Exception {
    isBackgroundMode = false;
    Log.d("WebRTC", "App returning to foreground, restoring connections");
    if (!dataChannels.isEmpty() && firebaseClient != null) {
        ConnectionManager connectionManager = ConnectionManager.getInstance(context);
        for (String peerUsername : connectionManager.getConnectedUsers()) {
            if (pendingFileTransfers.containsKey(peerUsername)) {
```

```
            Log.d("WebRTC", "Resuming file transfer to " + peerUsername);
            Uri fileUri = Uri.parse(pendingFileTransfers.get(peerUsername));
            // TODO: Retrieve fileName and fileType from somewhere (e.g., stored metadata)
            sendFile(fileUri, peerUsername, "resumeFile", "application/octet-stream");
        } else if (!isConnected(peerUsername)) {
            Log.d("WebRTC", "Restoring connection to " + peerUsername);
            startConnection(peerUsername);
        }
    }
    if (webrtcListener != null) {
        for (PeerConnection peerConnection : peerConnections.values()) {
            if (peerConnection != null && peerConnection.getRemoteDescription() != null) {
                webrtcListener.onConnected();
            }
        }
    }
  }
}

public void onMessageReceived(String message, String fromPeer) {
  Log.d("WebRTC", "Message received from " + fromPeer + ": " + message);
  if (webRTCService != null) {
    webRTCService.handleMessageNotification(message, fromPeer);
  }
}

public void setWebRTCService(WebRTCService service) {
    this.webRTCService = service;

    Log.d("WebRTC", "? WebRTCService reference set");

}


public Map<String, PeerConnection> getPeerConnections() {

    return peerConnections;

}


public Map<String, DataChannel> getDataChannels() {

    return dataChannels;

}


public String getPeerUsername() {

    return currentPeerUsername;

}


public boolean isConnected() {

    for (String peerUsername : peerConnections.keySet()) {

        if (isConnected(peerUsername)) return true;

    }

    return false;

}


public boolean isAttemptingConnection() {

    for (String peerUsername : peerConnections.keySet()) {

        if (isAttemptingConnection(peerUsername)) return true;

    }
    return false;
}

public void onDataChannelStateChange(String peerUsername, DataChannel.State state) {
    Log.d("WebRTC", "DataChannel state changed for " + peerUsername + ": " + state);
    if (webrtcListener != null) {
        switch (state) {
            case OPEN:
                webrtcListener.onConnected();
                deliverStoredMessages(peerUsername);
                break;
            case CLOSED:
            case CLOSING:
                webrtcListener.onConnectionFailed();
                if (ConnectionManager.getInstance(context).isUserConnected(peerUsername)) {
                    startConnection(peerUsername);
                }
                break;
        }
    }
}

private void deliverStoredMessages(String peerUsername) {
    List<MessageEntity> storedMessages = dataChannelHandler.getMessageHistory(peerUsername);
    for (MessageEntity message : storedMessages) {
        if (message.getSender().equals("You")) {
            if (message.getMessage().startsWith("File: ")) {
                    Log.d("WebRTCClient", "Attempting to resume file transfer for " +
message.getMessage().substring(6));
            } else {
                sendEncryptedMessage(message.getMessage(), peerUsername);
            }
        }
    }
}

public void acceptConnection(String peerUsername, String offerSdp) {
    Log.d("WebRTC", "Accepting connection from: " + peerUsername);
    currentPeerUsername = peerUsername;
    setupPeerConnection(peerUsername);
    PeerConnection peerConnection = peerConnections.get(peerUsername);
    if (peerConnection != null) {
        Log.d("WebRTC", "? Processing offer for acceptance from " + peerUsername);
        SessionDescription offer = new SessionDescription(SessionDescription.Type.OFFER, offerSdp);
        peerConnection.setRemoteDescription(new CustomSdpObserver() {
            @Override
            public void onSetSuccess() {
                createAnswer(peerUsername);
            }
        }, offer);
    } else {
        Log.e("WebRTC", "? Failed to setup peer connection for " + peerUsername);
    }
}
```

```
  public void rejectConnection(String peerUsername) {
    Log.d("WebRTC", "Rejecting connection from: " + peerUsername);
    disconnectPeer(peerUsername);
    firebaseClient.sendSignalingData(peerUsername, "REJECT", "Connection rejected");
  }

  private void createAnswer(String peerUsername) {
    PeerConnection peerConnection = peerConnections.get(peerUsername);
    if (peerConnection != null) {
      peerConnection.createAnswer(new CustomSdpObserver() {
        @Override
        public void onCreateSuccess(SessionDescription sessionDescription) {
          peerConnection.setLocalDescription(new CustomSdpObserver(), sessionDescription);
          firebaseClient.sendSignalingData(peerUsername, "ANSWER", sessionDescription.description);
        }
      }, new MediaConstraints());
    }
  }

    private File saveFileToInternalStorage(File tempFile, String fileName, String fileType, String
peerUsername) throws IOException {
    ContentResolver resolver = context.getContentResolver();
    ContentValues contentValues = new ContentValues();
    contentValues.put(MediaStore.MediaColumns.DISPLAY_NAME, fileName);
    contentValues.put(MediaStore.MediaColumns.MIME_TYPE, fileType);
    Uri collection;
    if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.Q) {
                                        contentValues.put(MediaStore.MediaColumns.RELATIVE_PATH,
Environment.DIRECTORY_DOWNLOADS + "/From_" + peerUsername);
                                                                                collection          =
MediaStore.Downloads.getContentUri(MediaStore.VOLUME_EXTERNAL_PRIMARY);
    } else {
                                                                      File        downloadsDir        =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
      File peerDir = new File(downloadsDir, "From_" + peerUsername);
      if (!peerDir.exists() && !peerDir.mkdirs()) {
        throw new IOException("Failed to create directory: " + peerDir);
      }
      collection = MediaStore.Files.getContentUri("external");
                                contentValues.put(MediaStore.MediaColumns.DATA,    new    File(peerDir,
fileName).getAbsolutePath());
    }

    Uri fileUri = null;
    try {
      fileUri = resolver.insert(collection, contentValues);
      if (fileUri == null) {
        throw new IOException("Failed to create new MediaStore record for " + fileName);
      }

      try (OutputStream os = resolver.openOutputStream(fileUri);
         InputStream is = new java.io.FileInputStream(tempFile)) {
        if (os == null) {
            throw new IOException("Failed to open output stream for URI: " + fileUri);
        }
        byte[] buffer = new byte[CHUNK_SIZE];
        int bytesRead;
        while ((bytesRead = is.read(buffer)) != -1) {
            os.write(buffer, 0, bytesRead);
        }
        os.flush();
        Log.d("WebRTCClient", "File saved successfully: " + fileUri);
      }

      // Get the real file path
      String filePath = getRealPathFromUri(fileUri, fileName, peerUsername);
      File file = new File(filePath);

      new android.os.Handler(android.os.Looper.getMainLooper()).post(() ->
            Toast.makeText(context, "File saved to Downloads/From_" + peerUsername + ": " + fileName,
Toast.LENGTH_LONG).show()
      );
      return file;
    } catch (IOException e) {
      Log.e("WebRTCClient", "Error saving file: " + fileName + " - " + e.getMessage());
      if (fileUri != null) {
        resolver.delete(fileUri, null, null);
      }
      new android.os.Handler(android.os.Looper.getMainLooper()).post(() ->
                Toast.makeText(context, "Failed to save file: " + fileName + " - " + e.getMessage(),
Toast.LENGTH_LONG).show()
        );
        throw e;
      }
    }

    private String getRealPathFromUri(Uri uri, String fileName, String peerUsername) {
      if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.Q) {
        return Environment.DIRECTORY_DOWNLOADS + "/From_" + peerUsername + "/" + fileName;
      } else {
                                                              File        downloadsDir        =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
        return new File(downloadsDir, "From_" + peerUsername + "/" + fileName).getAbsolutePath();
      }
    }
}
```

Filename: WebRTCService.java

```
package com.example.protegotinyever.service;


import android.app.Activity;

import android.app.Application;

import android.app.Notification;

import android.app.NotificationChannel;

import android.app.NotificationManager;

import android.app.PendingIntent;

import android.app.Service;

import android.content.Context;

import android.content.Intent;

import android.content.pm.PackageManager;

import android.os.Build;

import android.os.Bundle;

import android.os.IBinder;

import android.util.Log;


import androidx.core.app.NotificationCompat;

import androidx.core.app.NotificationManagerCompat;


import com.example.protegotinyever.R;

import com.example.protegotinyever.act.ChatActivity;

import com.example.protegotinyever.util.DataChannelHandler;

import com.example.protegotinyever.util.FirebaseClient;

import com.example.protegotinyever.webrtc.WebRTCClient;
import java.util.HashMap;

import java.util.Map;


public class WebRTCService extends Service {

    private static final String SERVICE_CHANNEL_ID = "WebRTCServiceChannel";

    private static final String MESSAGE_CHANNEL_ID = "WebRTCMessageChannel";

    private static final int NOTIFICATION_ID = 1;

    private static final String TAG = "WebRTCService";

    private WebRTCClient webRTCClient;

    private FirebaseClient firebaseClient;

    private NotificationManager notificationManager;

    private int messageNotificationId = 100;

    private DataChannelHandler dataChannelHandler;

    private static String currentActivityName = null;

    private static String currentChatPeer = null;

    private static int activeActivities = 0;

    private static final Object activityLock = new Object();

    private static boolean isTransitioningActivities = false;

    private static final long TRANSITION_TIMEOUT = 1000; // 1 second timeout for transitions

    private static String lastStartedActivity = null;

    private static String lastPausedActivity = null;

    private Map<String, Integer> notificationIds;

    private int nextNotificationId = 2; // Start from 2 since 1 is used for foreground service

    private int rea = 1;


    @Override
```

```
public void onCreate() {
    super.onCreate();
    Log.d(TAG, "WebRTCService onCreate");
    notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    notificationIds = new HashMap<>();
    createNotificationChannel();
    startForeground(NOTIFICATION_ID, createForegroundNotification());
    dataChannelHandler = DataChannelHandler.getInstance(getApplicationContext());

    // Register activity lifecycle callbacks
    getApplication().registerActivityLifecycleCallbacks(new Application.ActivityLifecycleCallbacks() {
        @Override
        public void onActivityStarted(Activity activity) {
            synchronized (activityLock) {
                String activityName = activity.getClass().getName();
                lastStartedActivity = activityName;

                // Always increment counter on start
                activeActivities++;
                isTransitioningActivities = true;

                Log.d("WebRTC", "?? Activity started: " + activityName +
                      " (Active activities: " + activeActivities + ")");

                // Schedule transition timeout
                new android.os.Handler(android.os.Looper.getMainLooper()).postDelayed(() -> {
                    synchronized (activityLock) {
```

```
                isTransitioningActivities = false;
            }
        }, TRANSITION_TIMEOUT);
    }
}

@Override
public void onActivityStopped(Activity activity) {
    synchronized (activityLock) {
        String activityName = activity.getClass().getName();

        // Only decrement if we're not transitioning between activities
        if (!isTransitioningActivities) {
            activeActivities = Math.max(0, activeActivities - 1);

            // Clear chat state only when truly going to background
            if (activeActivities == 0) {
                Log.d("WebRTC", "? App going to background");
                currentActivityName = null;
                currentChatPeer = null;
            }

            // Clear chat peer if leaving ChatActivity
            if (activity instanceof ChatActivity) {
                Log.d("WebRTC", "? Leaving chat with: " + currentChatPeer);
                currentChatPeer = null;
            }
        }


        Log.d("WebRTC", "?? Activity stopped: " + activityName +

              " (Active activities: " + activeActivities +

              ", Transitioning: " + isTransitioningActivities + ")");

    }

}


@Override

public void onActivityResumed(Activity activity) {

    synchronized (activityLock) {

        String activityName = activity.getClass().getName();

        Log.d("WebRTC", "? Activity resumed: " + activityName);

        currentActivityName = activityName;

        isTransitioningActivities = false;


        if (activity instanceof ChatActivity) {

            String peer = ((ChatActivity) activity).getPeerUsername();

            if (peer != null) {

                currentChatPeer = peer;

                dataChannelHandler.setCurrentPeer(peer);

                Log.d("WebRTC", "? Chat opened with peer: " + peer);

            }

        } else {

            Log.d("WebRTC", "? Not in chat, clearing current peer: " + currentChatPeer);

            currentChatPeer = null;

            dataChannelHandler.setCurrentPeer(null);
            }
        }
    }

    @Override
    public void onActivityPaused(Activity activity) {
        synchronized (activityLock) {
            String activityName = activity.getClass().getName();
            lastPausedActivity = activityName;

            // Don't clear chat peer here - we'll handle it in onActivityStopped
            Log.d("WebRTC", "?? Activity paused: " + activityName);
        }
    }

    @Override
    public void onActivityCreated(Activity activity, Bundle bundle) {}

    @Override
    public void onActivitySaveInstanceState(Activity activity, Bundle bundle) {}

    @Override
    public void onActivityDestroyed(Activity activity) {
        synchronized (activityLock) {
            Log.d("WebRTC", "? Activity destroyed: " + activity.getClass().getName());
            if (activity instanceof ChatActivity) {
                Log.d("WebRTC", "? Chat activity destroyed, clearing chat peer");
                currentChatPeer = null;
            }
        }
    }
  });
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
  Log.d(TAG, "WebRTCService onStartCommand");

  if (intent != null) {
    String action = intent.getAction();
    if (action != null) {
      switch (action) {
        case "ACCEPT_CONNECTION":
          String peerUsername = intent.getStringExtra("peerUsername");
          String offerSdp = intent.getStringExtra("offerSdp");
          if (peerUsername != null && offerSdp != null) {
            webRTCClient.acceptConnection(peerUsername, offerSdp);
            // Cancel the notification
            notificationManager.cancel(peerUsername.hashCode());
          }
          break;
        case "REJECT_CONNECTION":
          String rejectPeer = intent.getStringExtra("peerUsername");
          if (rejectPeer != null) {
```

```
                    webRTCClient.rejectConnection(rejectPeer);
                    // Cancel the notification
                    notificationManager.cancel(rejectPeer.hashCode());
                }
                break;
            }
        }

        // Initialize clients if needed
        String username = intent.getStringExtra("username");
        String phoneNumber = intent.getStringExtra("phoneNumber");

        if (firebaseClient == null) {
            firebaseClient = new FirebaseClient(username, phoneNumber);
        }

        if (webRTCClient == null) {
            webRTCClient = WebRTCClient.getInstance(getApplicationContext(), firebaseClient);
            webRTCClient.setWebRTCService(this);
            setupMessageHandler();
        }
    }

    return START_STICKY;
}

private void setupMessageHandler() {
        // Set WebRTCClient in DataChannelHandler
        dataChannelHandler.setWebRTCClient(webRTCClient);

        dataChannelHandler.setOnMessageReceivedListener(message -> {
            String peerUsername = webRTCClient.getPeerUsername();

            // Log the current state
            Log.d("WebRTC", "? Message received:" +
                "\n- From: " + peerUsername +
                "\n- Message: " + message +
                "\n- Current Activity: " + currentActivityName +
                "\n- Current Chat Peer: " + currentChatPeer +
                "\n- Active Activities: " + activeActivities +
                "\n- Is Transitioning: " + isTransitioningActivities);

            // Delay notification check slightly to allow activity transitions to complete
            new android.os.Handler(android.os.Looper.getMainLooper()).postDelayed(() -> {
                synchronized (activityLock) {
                    // Check if we should show notification
                    boolean isBackground = activeActivities == 0;
                    boolean isInChatActivity = ChatActivity.class.getName().equals(currentActivityName);
                                        boolean isChattingWithSender = peerUsername != null &&
peerUsername.equals(currentChatPeer);

                    // Show notification if:
                    // 1. App is in background, OR
                    // 2. Not in chat activity, OR
            // 3. In different chat than sender
            boolean shouldShowNotification = isBackground || !isInChatActivity || !isChattingWithSender;


            Log.d("WebRTC", "? Notification check:" +
                "\n- Is Background: " + isBackground +
                "\n- In Chat Activity: " + isInChatActivity +
                "\n- Chatting with sender: " + isChattingWithSender +
                "\n- Should Show: " + shouldShowNotification +
                "\n- Active Activities: " + activeActivities +
                "\n- Is Transitioning: " + isTransitioningActivities +
                "\n- Current Activity: " + currentActivityName +
                "\n- Current Chat Peer: " + currentChatPeer);


            if (shouldShowNotification) {
                Log.d("WebRTC", "? Will show notification - Not actively chatting with: " + peerUsername);
                handleMessageNotification(message, peerUsername != null ? peerUsername : "Unknown");
            } else {
                Log.d("WebRTC", "? Skipping notification - Currently chatting with: " + peerUsername);
            }
        }
    }, 500); // Add a small delay to allow activity transitions
});


dataChannelHandler.setStateChangeListener(state -> {
    Log.d("WebRTC", "? DataChannel state changed to: " + state);
    switch (state) {
        case OPEN:
                updateServiceNotification("Secure Connection Active");
                break;
            case CLOSED:
            case CLOSING:
                updateServiceNotification("Connection Closed");
                break;
            default:
                break;
        }
    });
}

private boolean isChatActivityActive() {
    boolean isActive = currentActivityName != null && ChatActivity.class.getName().equals(currentActivityName);
    Log.d("WebRTC", "? Chat activity active: " + isActive + " (currentActivityName: " + currentActivityName + ")");
    return isActive;
}

private boolean isCurrentChatWith(String peerUsername) {
    boolean isChatting = peerUsername != null && peerUsername.equals(currentChatPeer);
    Log.d("WebRTC", "? Currently chatting with " + peerUsername + ": " + isChatting + " (currentChatPeer: " + currentChatPeer + ")");
    return isChatting;
}
```

```
private void showMessageNotification(String message, String fromPeer) {
    NotificationManager notificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    // Create notification channel for Android O and above
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(
            MESSAGE_CHANNEL_ID,
            "Messages",
            NotificationManager.IMPORTANCE_HIGH
        );
        channel.setDescription("Message notifications");
        notificationManager.createNotificationChannel(channel);
    }

    // Create intent for opening chat
    Intent intent = new Intent(this, ChatActivity.class);
    intent.putExtra("peerUsername", fromPeer);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP);

    PendingIntent pendingIntent = PendingIntent.getActivity(
        this, 0, intent,
        PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE
    );

    // Build the notification
                        NotificationCompat.Builder    builder    =    new    NotificationCompat.Builder(this,
MESSAGE_CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_notification)
            .setContentTitle("New message from " + fromPeer)
            .setContentText(message)
            .setPriority(NotificationCompat.PRIORITY_HIGH)
            .setAutoCancel(true)
            .setContentIntent(pendingIntent);

        // Show the notification
        notificationManager.notify(fromPeer.hashCode(), builder.build());
        Log.d("WebRTC", "? Showed notification for message from: " + fromPeer);
    }

    private boolean isBackground() {
        synchronized (activityLock) {
            return activeActivities == 0 && !isTransitioningActivities;
        }
    }

    private void updateServiceNotification(String text) {
        startForeground(NOTIFICATION_ID, createNotification(text));
    }

    private void createNotificationChannel() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            // Create service channel (low priority)
            NotificationChannel serviceChannel = new NotificationChannel(
                SERVICE_CHANNEL_ID,
                "WebRTC Service Channel",
                NotificationManager.IMPORTANCE_LOW
        );
        serviceChannel.setDescription("Maintains secure connection");
        serviceChannel.setShowBadge(false);
        notificationManager.createNotificationChannel(serviceChannel);
        Log.d("WebRTC", "? Service notification channel created");

        // Create message channel (high priority)
        NotificationChannel messageChannel = new NotificationChannel(
                MESSAGE_CHANNEL_ID,
                "WebRTC Message Channel",
                NotificationManager.IMPORTANCE_HIGH
        );
        messageChannel.setDescription("Shows incoming messages");
        messageChannel.enableLights(true);
        messageChannel.enableVibration(true);
        messageChannel.setLockscreenVisibility(Notification.VISIBILITY_PUBLIC);
        messageChannel.setShowBadge(true);
        notificationManager.createNotificationChannel(messageChannel);
        Log.d("WebRTC", "? Message notification channel created");
    }
}

private Notification createNotification(String text) {
    Intent notificationIntent = new Intent(this, ChatActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent,
            PendingIntent.FLAG_IMMUTABLE);

    return new NotificationCompat.Builder(this, SERVICE_CHANNEL_ID)
            .setContentTitle("Secure Chat")
            .setContentText(text)
            .setSmallIcon(R.drawable.avatar)
            .setContentIntent(pendingIntent)
            .setPriority(NotificationCompat.PRIORITY_LOW)
            .setCategory(NotificationCompat.CATEGORY_SERVICE)
            .build();
}

@Override
public IBinder onBind(Intent intent) {
    return null;
}

@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "WebRTCService onDestroy");
    if (webRTCClient != null) {
        // Don't disconnect, just cleanup resources
        webRTCClient.onBackground();
    }
}
```

```
    // Helper method to check if we're in chat with specific peer
    private boolean isInChatWith(String peerUsername) {
        boolean isInChatActivity = ChatActivity.class.getName().equals(currentActivityName);
        boolean isWithPeer = peerUsername != null && peerUsername.equals(currentChatPeer);
        return isInChatActivity && isWithPeer;
    }

    public void handleMessageNotification(String message, String fromPeer) {
        synchronized (activityLock) {
            // At this point, we know we should show a notification because:
            // 1. The message is from a peer we're not actively chatting with
            // 2. The DataChannelHandler has already filtered out messages from the active chat

            // Create chat intent
            Intent chatIntent = new Intent(this, ChatActivity.class);
            chatIntent.putExtra("peerUsername", fromPeer);
                                        chatIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK      |
Intent.FLAG_ACTIVITY_CLEAR_TOP);

            PendingIntent pendingIntent = PendingIntent.getActivity(
                this, fromPeer.hashCode(), chatIntent,
                PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE
            );

            // Build notification
                          NotificationCompat.Builder   builder   =   new   NotificationCompat.Builder(this,
MESSAGE_CHANNEL_ID)
                .setSmallIcon(R.drawable.ic_notification)
                .setContentTitle("Message from " + fromPeer)
                .setContentText(message)
                .setPriority(NotificationCompat.PRIORITY_HIGH)
                .setAutoCancel(true)
                .setContentIntent(pendingIntent);

            // Show notification
            notificationManager.notify(fromPeer.hashCode(), builder.build());
            Log.d(TAG, "Showing notification for message from " + fromPeer);
        }
    }

    private Notification createForegroundNotification() {
        Intent notificationIntent = new Intent(this, ChatActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(
            this,
            0,
            notificationIntent,
            PendingIntent.FLAG_IMMUTABLE
        );

        return new NotificationCompat.Builder(this, SERVICE_CHANNEL_ID)
            .setContentTitle("Chat Service Running")
            .setContentText("Connected to chat network")
            .setSmallIcon(R.drawable.ic_notification)
            .setContentIntent(pendingIntent)
            .build();
    }

    public void showConnectionRequestNotification(String fromPeer, String offerSdp) {
        // Create accept intent
        Intent acceptIntent = new Intent(this, WebRTCService.class);
        acceptIntent.setAction("ACCEPT_CONNECTION");
        acceptIntent.putExtra("peerUsername", fromPeer);
        acceptIntent.putExtra("offerSdp", offerSdp);

        PendingIntent acceptPendingIntent = PendingIntent.getService(
            this, 0, acceptIntent,
            PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE
        );

        // Create reject intent
        Intent rejectIntent = new Intent(this, WebRTCService.class);
        rejectIntent.setAction("REJECT_CONNECTION");
        rejectIntent.putExtra("peerUsername", fromPeer);

        PendingIntent rejectPendingIntent = PendingIntent.getService(
            this, 1, rejectIntent,
            PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE
        );

        // Build notification
```

```
                        NotificationCompat.Builder    builder    =    new    NotificationCompat.Builder(this,
MESSAGE_CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_notification)
        .setContentTitle("Connection Request")
        .setContentText(fromPeer + " wants to connect")
        .setPriority(NotificationCompat.PRIORITY_HIGH)
        .setAutoCancel(true)
        .addAction(R.drawable.ic_accept, "Accept", acceptPendingIntent)
        .addAction(R.drawable.ic_reject, "Reject", rejectPendingIntent);

      NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);

      // Check notification permission for Android 13+
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
                        if   (checkSelfPermission(android.Manifest.permission.POST_NOTIFICATIONS)   !=
PackageManager.PERMISSION_GRANTED) {
          Log.w("WebRTCService", "Notification permission not granted");
          return;
        }
      }

      notificationManager.notify(fromPeer.hashCode(), builder.build());
  }
}
```