Historia: HU-02 (13 pts): Como usuario, quiero poder enviar y recibir mensajes de texto a un contacto seleccionado para comunicarme.

3. Empezar el desarrollo del servicio de envío y recepción de mensajes vía Wifi-direct.

Para esto primero necesitamos crear un servicio que maneje la comunicación basándonos en las interfaces, por lo que pensamos en dividir la arquitectura de la app así:

a) Lista de contactos y la pantalla del chat.
b) Servicio wifi direct (Gestiona las conexiones y la recepción de mensajes)
c) Repositorio local(donde se guardan los mensajes y el chat).

De esta forma el servicio P2P sigue corriendo aunque se cierre la pantalla del chat.

A continuación creamos el servicio de wifi-direct, primero ponemos los permisos de AndroidManifest.xml:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>

<application ...>
    <service android:name=".WifiDirectService"
            android:exported="false" />
</application>
```

Posteriormente se crea la el servicio **Wifi Direct Service.kt** esté manejará la creación de sockets TCP para enviar y recibir mensajes entre dos dispositivos.

Importamos bibliotecas:

```
package com.example.chatp2p
import android.app.Service
import android.content.Intent
import android.os.Binder
import android.os.IBinder
import android.util.Log
import java.io.*
import java.net.*
```

Implementar wifi direct:

```
class WifiDirectService : Service() {
    private val binder = LocalBinder()
    private var serverThread: Thread? = null
    private var clientThread: Thread? = null
```

```
    private var socket: Socket? = null
    private val port = 8988
    private var messageListener: ((String) -> Unit)? = null

    inner class LocalBinder : Binder() {
        fun getService(): WifiDirectService = this@WifiDirectService }
    override fun onBind(intent: Intent?): IBinder = binder
```

Permite que la actividad se suscriba para recibir mensajes entrantes

```
    fun setOnMessageReceivedListener(listener: (String) -> Unit) {
        messageListener = listener}
```

Inicia el modo servidor (Group Owner)

```
    fun startServer() {
        serverThread = Thread {
            try {
                val serverSocket = ServerSocket(port)
                Log.d("P2P", "Servidor esperando conexión...")
                socket = serverSocket.accept()
                Log.d("P2P", "Cliente conectado")
                listenForMessages(socket!!)
            } catch (e: IOException) {
                e.printStackTrace()   }}
        serverThread?.start()    }
```

Inicia el modo cliente

```
    fun startClient(host: String) {
        clientThread = Thread {
            try {
                socket = Socket()
                socket!!.connect(InetSocketAddress(host, port), 5000)
                Log.d("P2P", "Conectado al servidor $host")
                listenForMessages(socket!!)
            } catch (e: IOException) {
                e.printStackTrace()  }  }
        clientThread?.start()     }
```

Envía mensaje

```
    fun sendMessage(message: String) {
        Thread {
            try {
val out = PrintWriter(BufferedWriter(OutputStreamWriter(socket?.getOutputStream())), true)
                out.println(message)
                out.flush()
            } catch (e: IOException) {
                e.printStackTrace() }
        }.start() }
```

Mensajes entrantes

```
    private fun listenForMessages(socket: Socket) {
        try {
            val reader = BufferedReader(InputStreamReader(socket.getInputStream()))
```

```
            var line: String?
            while (true) {
                line = reader.readLine() ?: break
                Log.d("P2P", "Mensaje recibido: $line")
                messageListener?.invoke(line)  }
        } catch (e: IOException) {
            e.printStackTrace()      }  }

    override fun onDestroy() {
        socket?.close()
        super.onDestroy()  }  }
```

Por último inicializamos el servicio en chat activity.kt:

```
class ChatActivity : AppCompatActivity() {
    private var wifiService: WifiDirectService? = null
    private var bound = false
    private var isGroupOwner = false
    private var groupOwnerAddress: String? = null
    private val connection = object : ServiceConnection {
        override fun onServiceConnected(name: ComponentName?, service: IBinder?){
            val binder = service as WifiDirectService.LocalBinder
            wifiService = binder.getService()
            bound = true

            // Configurar listener para recibir mensajes
            wifiService?.setOnMessageReceivedListener { msg ->
                runOnUiThread {
                    // Mostrar mensaje recibido
                    findViewById<TextView>(R.id.tvMessages).append("\nAmigo: $msg")
                } }
            // Iniciar socket según rol
            if (isGroupOwner) {
                wifiService?.startServer()
            } else {
                wifiService?.startClient(groupOwnerAddress!!)
            }  }
        override fun onServiceDisconnected(name: ComponentName?) {
            bound = false
        }  }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_chat)
//datos del Wifi P2p:
        isGroupOwner = intent.getBooleanExtra("isGroupOwner", false)
        groupOwnerAddress = intent.getStringExtra("groupOwnerAddress")
```

```
bindService(Intent(this, WifiDirectService::class.java), connection,
Context.BIND_AUTO_CREATE)

        findViewById<Button>(R.id.btnSend).setOnClickListener {
            val msg = findViewById<EditText>(R.id.etMessage).text.toString()
            wifiService?.sendMessage(msg)
            findViewById<TextView>(R.id.tvMessages).append("\nYo: $msg")
        }
    } override fun onDestroy() {
        super.onDestroy()
        if (bound) unbindService(connection)
    } }
```

Historia: -HU-03 (5 pts): Como usuario, quiero que todos mis mensajes se guarden localmente en mi dispositivo para poder leer mis conversaciones aunque reinicie la app.

4. Empezar a implementar la carga de chats al abrir la app:

Primero se crean las entidades room:

```
@Entity(tableName = "chats")
data class ChatEntity(
    @PrimaryKey val chatId: String,
    val contactName: String,
    val lastMessage: String?,
    val lastTimestamp: Long)

@Entity(tableName = "messages")
data class MessageEntity(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val chatId: String,
    val sender: String,
    val body: String,
    val timestamp: Long)
```

Creamos los DAO:

```
@Dao
interface ChatDao {
    @Query("SELECT * FROM chats ORDER BY lastTimestamp DESC")
    suspend fun getAllChats(): List<ChatEntity>
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertChat(chat: ChatEntity)}

@Dao
interface MessageDao {
    @Query("SELECT * FROM messages WHERE chatId = :chatId ORDER BY timestamp ASC")
    suspend fun getMessages(chatId: String): List<MessageEntity>
    @Insert
    suspend fun insertMessage(msg: MessageEntity)}
```

Creamos la base local de los chats:

```
@Database(entities = [ChatEntity::class, MessageEntity::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun chatDao(): ChatDao
    abstract fun messageDao(): MessageDao
    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "chat_db"
                ).build()
                INSTANCE = instance
                instance
            } } } }
```

Integramos la base y el wifi direct con los siguientes comandos en el programa:

```
// Dentro de WifiDirectService.kt
private fun saveMessageToLocal(chatId: String, sender: String, message: String) {
    val db = AppDatabase.getDatabase(applicationContext)
    Thread {
        val timestamp = System.currentTimeMillis()
            val msg = MessageEntity(chatId = chatId, sender = sender, body = message, timestamp =
timestamp)
        db.messageDao().insertMessage(msg)
        val chat = ChatEntity(
            chatId = chatId,
            contactName = sender,
            lastMessage = message,
            lastTimestamp = timestamp )
        db.chatDao().insertChat(chat)
    }.start() }
```

Después se incorpora esto:

a) Al recibir un mensaje:

```
messageListener?.invoke(line)
SaveMessageToLocal(chatId = "contacto1", sender = "peer", message = line)
```

b) Al enviar un mensaje:

```
sendMessage(msg)
saveMessageToLocal(chatId = "contacto1", sender = "yo", message = msg)
```

Cargamos los chats:

```
class MainActivity : AppCompatActivity() {
    private lateinit var db: AppDatabase
    private lateinit var recyclerView: RecyclerView
    private lateinit var adapter: ChatAdapter
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```
        db = AppDatabase.getDatabase(this)
        recyclerView = findViewById(R.id.recyclerChats)
        adapter = ChatAdapter { chat ->
            val i = Intent(this, ChatActivity::class.java)
            i.putExtra("chatId", chat.chatId)
            i.putExtra("contactName", chat.contactName)
            startActivity(i)
        }
        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = adapter

        //Cargar los chats locales al abrir la app
        Thread {
            val chats = db.chatDao().getAllChats()
            runOnUiThread { adapter.setData(chats) }
        }.start() }   }
```

Cargamos los mensajes:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_chat)

    val chatId = intent.getStringExtra("chatId") ?: return
    val db = AppDatabase.getDatabase(this)
    val textView = findViewById<TextView>(R.id.tvMessages)

    // Cargar historial local al abrir chat
    Thread {
        val messages = db.messageDao().getMessages(chatId)
        runOnUiThread {
            messages.forEach { msg ->
                textView.append("\n${msg.sender}: ${msg.body}")
          }    }   }.start() }
```