

Giuseppe Spathis
giuseppe.spathis@studio.unibo.it
M. 0001043077

Cono Cirone
cono.cirone@studio.unibo.it
M. 0001029785

FitnessTracker

Laboratorio di Applicazioni Mobili AA 2023/2024 UniBo



Features



Tracciamento
attività



Geofencing



Statistiche

Features



Condivisione
dati



Sistema di
autenticazione



Notifiche
periodiche

Scelte progettuali

Linguaggio



Min. SDK



Database



Design Pattern

M-V-C

Overview applicazione

Sistema autenticazione

3:55

ActivityTracker







login

Non sei ancora registrato? Registrati!

3:56

Registrati





Torna indietro Registrati

Homepage

Selezione attività da svolgere

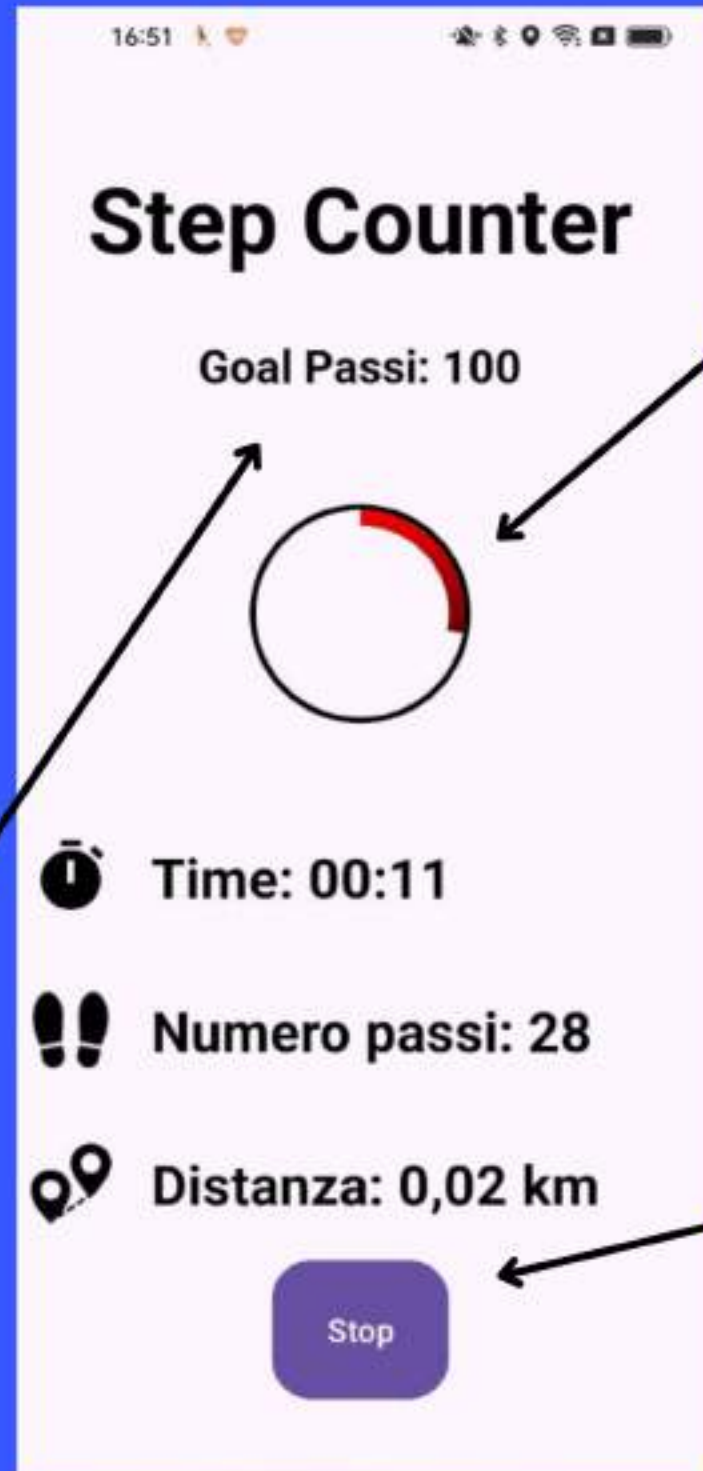
Bottone per il logout

Posizione dell'utente sulla mappa

Bottone per iniziare a tracciare nuove attività



Tracciamento Attività



ProgressBar che si riempie in base alla vicinanza al goal

Goal passi impostato dall'utente prima di avviare l'activity

Riporta l'utente alla home page e salva i dati nel db



Distanza da raggiungere impostata dall'utente

Tracciamento Attività



Azzera il timer del tempo trascorso dall'ultima volta in piedi



Geofences

Barra di ricerca per cercare nuovi luoghi

geofences

Bottone per visualizzare sulla mappa tutte le geofences salvate

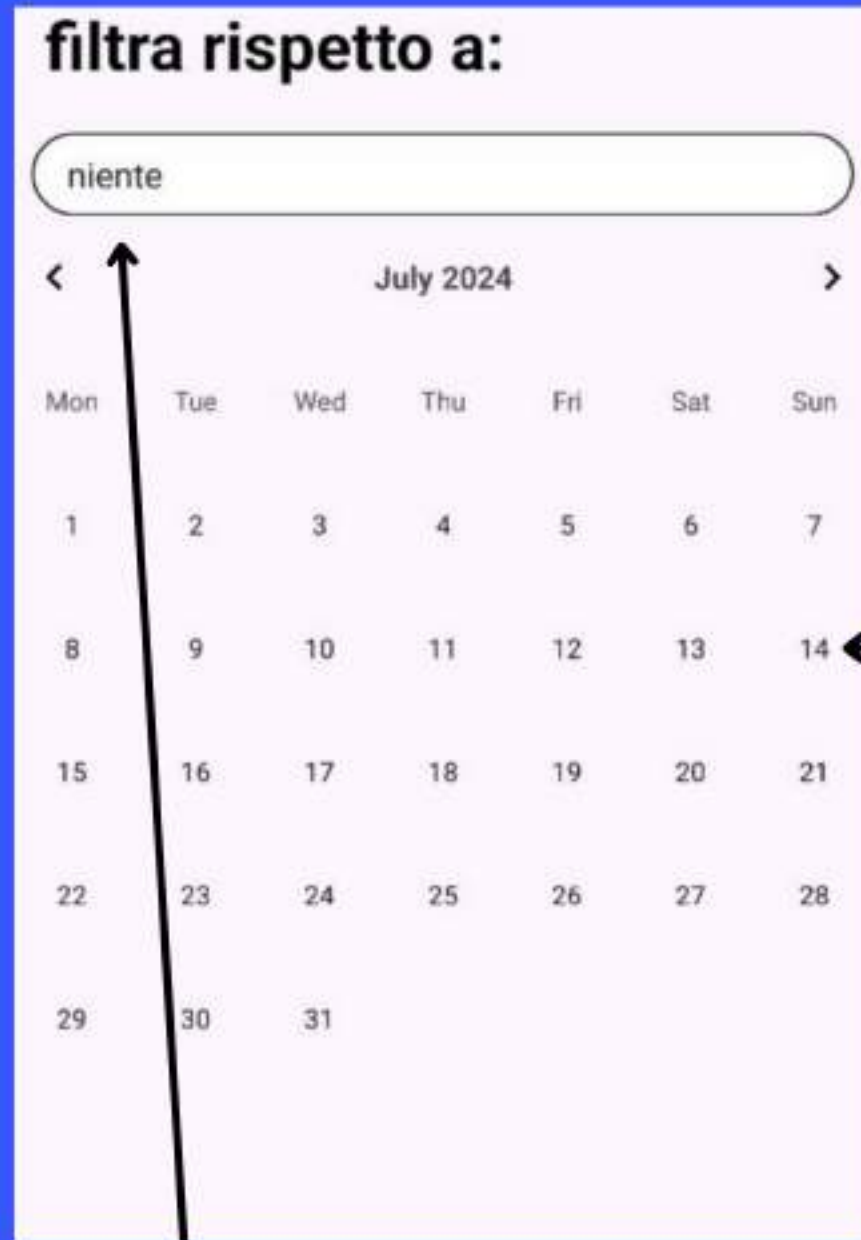


info

Bottone per far partire la ricerca

Bottone per aggiungere nuove geofences

Statistiche

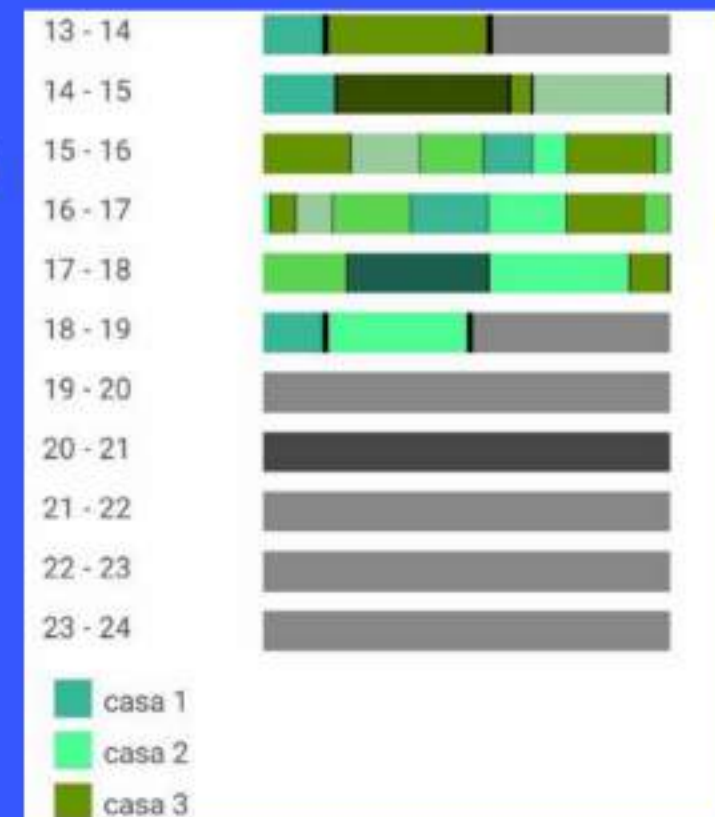


spinner per
filtrare il
calendario in
base all'attività

cliccando
su un
giorno

attività

geofencing



cliccando su
un'attività o geofence
tracciata vengono
visualizzati dettagli
approfonditi

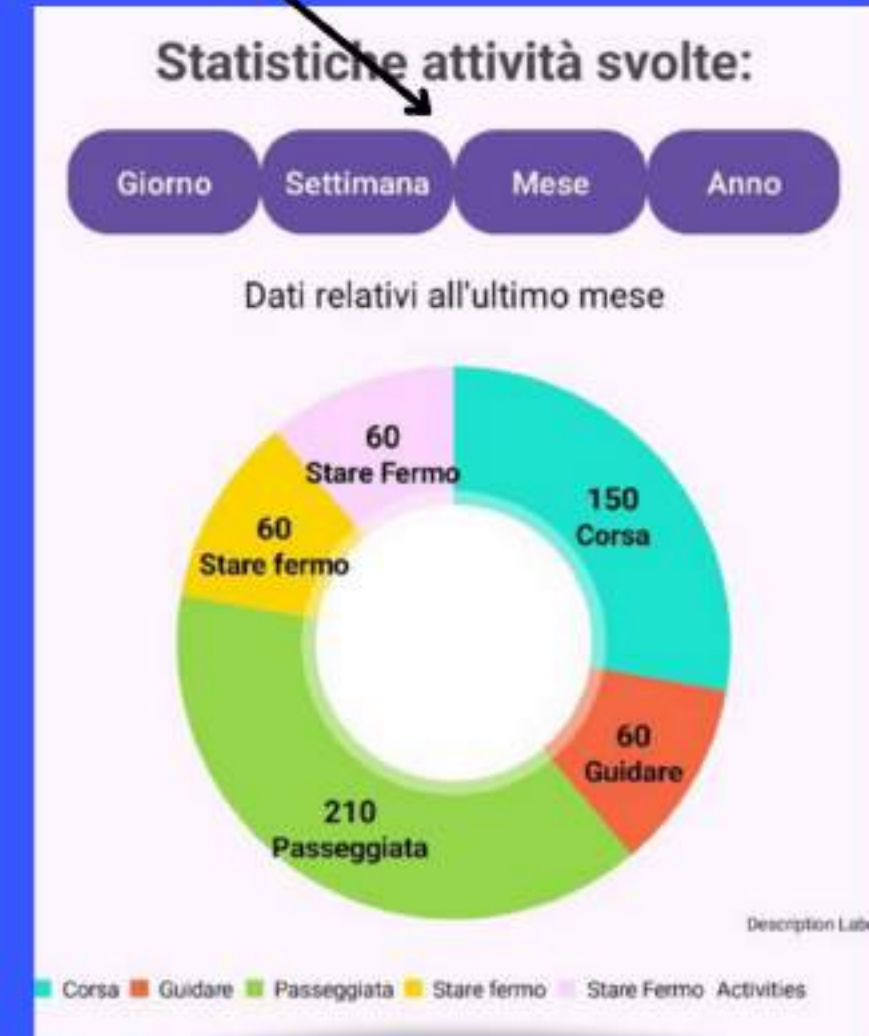
Activity Details: Passeggiata

- 🕒 Inizio: 09:00
- 🕒 Fine: 09:30
- 👣 Numero passi: 2000
- 📍 Distanza: 1.60 km
- 📅 Data: 22/07/2024

OK

Statistiche - pie charts

Bottoni che l'utente può cliccare per mostrare i dati in base al periodo selezionato



Statistiche - line chart

L'utente può scegliere il tipo di attività per cui vedere il variare dei dati

In base al tipo di attività selezionato l'utente può vedere:

- passeggiata** -> variazione del numero di passi
- corsa** -> variazione della distanza percorsa
- guidare** -> variazione della velocità media
- stare fermo** -> variazione del tempo totale passato seduto





Rilevamento persone in prossimità

bottone per diventare visibile agli altri dispositivi

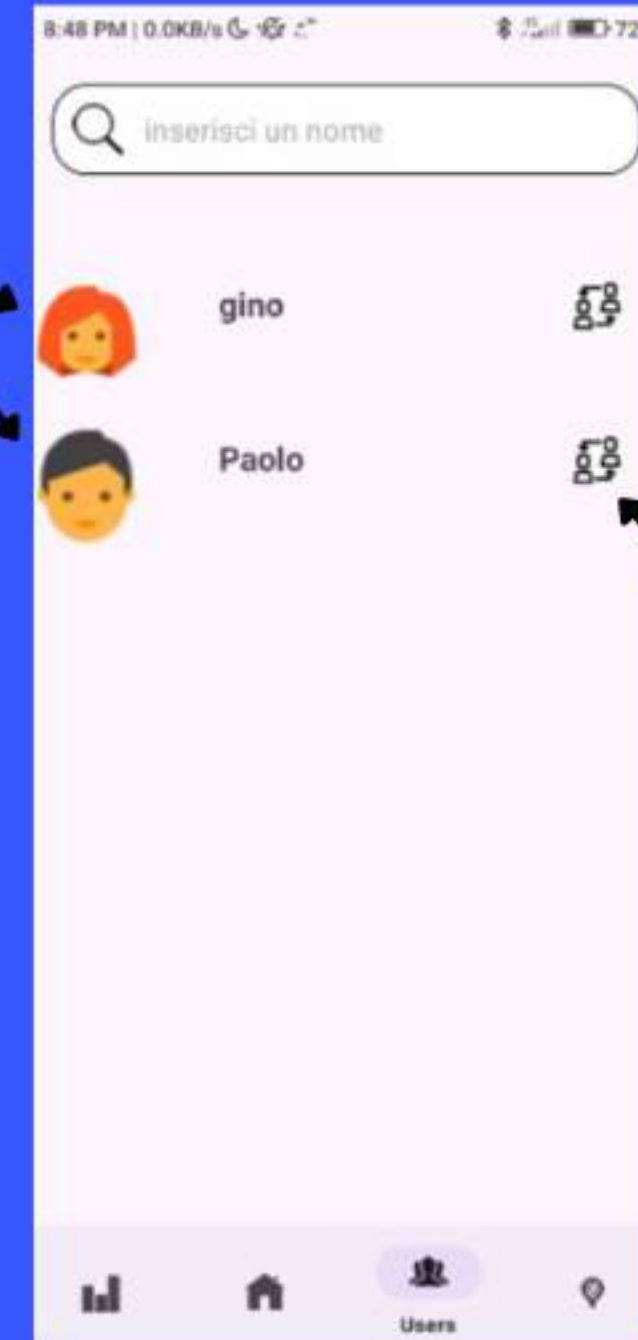
per far partire la ricerca cliccare sul bottone del bluetooth



utenti nelle vicinanze

finita la ricerca

info



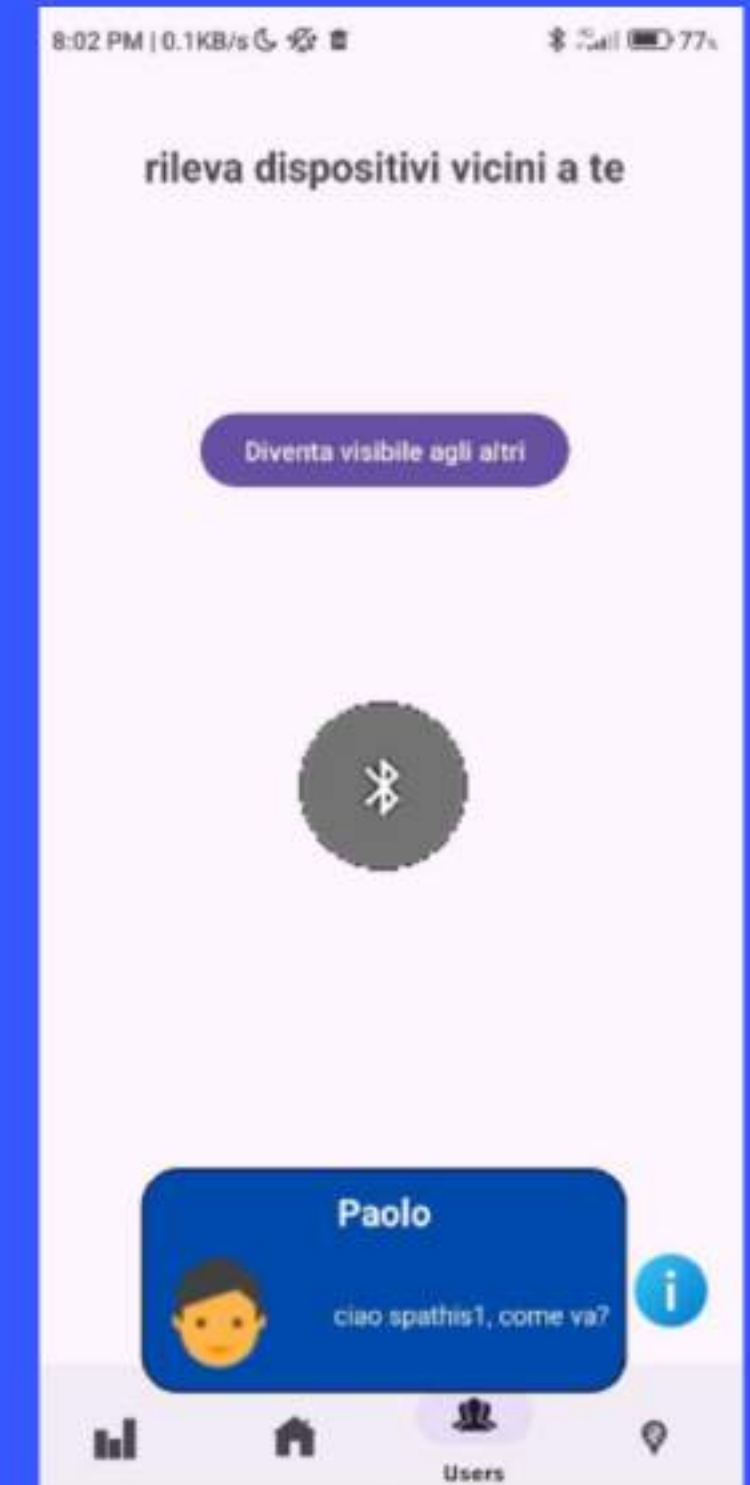
invia richiesta di pair



Condivisione dati

Client

Server



Dettagli implementativi

Servizi in background


Il **LocationUpdatesService** è responsabile del monitoraggio continuo della posizione dell'utente e della gestione delle geofence.

Caratteristiche principali:

Monitoraggio in foreground

Continuo aggiornamento posizione

Gestione geofence -> rileva automaticamente quando un utente entra o esce da una geofence

 FitnessTracker • 28 m

Location Service

La tua posizione sta venendo tracciata in background.

Gestione ingresso/uscita geofence:

```
function handleLocationUpdate(location, db, inside, callback):
    geofences = db.getAllGeofences()
    foundGeofence = false

    for each geofence in geofences:
        if distance(location, geofence) < geofence.radius:
            foundGeofence = true
            if not inside:
                inside = true
                enterTime = currentTime()
                sendNotification("Entered Geofence", geofence.placeName)
                db.insertTimeGeofence(geofence, enterTime)
            break

    if not foundGeofence and inside:
        inside = false
        exitTime = currentTime()
        sendNotification("Exited Geofence", "Exited")
        db.updateExitTimeForGeofences(exitTime)

    callback(inside)
```


Servizi in background

Il **CheckNearbyUsersWorker** è implementato utilizzando WorkManager e viene eseguito periodicamente per controllare la presenza di utenti vicini.

Caratteristiche principali:

Esecuzione Periodica -> viene eseguito ogni 15 minuti

Verifica degli Utenti Vicini -> Recupera la posizione corrente dell'utente, controlla la presenza di altri utenti attivi negli ultimi 10 minuti e Invia notifiche se trova utenti nelle vicinanze.

Gestione utenti nelle vicinanze:

```
function doWork():
    currentUserUid = getCurrentUserId()
    if currentUserUid is null:
        return failure

    currentUserLocation = getCurrentUserLocation(currentUserUid)
    if currentUserLocation is null:
        return failure

    tenMinutesAgo = currentTime - 10 minutes
    nearbyUsers = getNearbyUsers(tenMinutesAgo)

    for each user in nearbyUsers:
        if user is not currentUser:
            distance = calculateDistance(currentUserLocation, user.location)
            if distance <= 10 meters:
                sendNotification(user.username)

    return success
```


Gestione sensori

TYPE_STEP_COUNTER

```
override fun onSensorChanged(event: SensorEvent) {  
    if(event.sensor.type == Sensor.TYPE_STEP_COUNTER) {  
        val totalSteps = event.values[0].toInt()  
        if(initialStepCount == 0) {  
            initialStepCount = totalSteps  
        }  
        stepCount = totalSteps - initialStepCount  
        stepCounterText.text = "Numero passi: {stepCount}"  
        progressBar.progress = stepCount  
  
        if(stepCount >= stepCountTarget) {  
            stepCounterTargetTextView.text = "Gol raggiunto"  
        }  
  
        val distanceInKm = stepCount * steplenghtInMeters / 1000  
        distanceCounterText.text = "Distanza: {distanceInKm} km"  
    }  
}
```

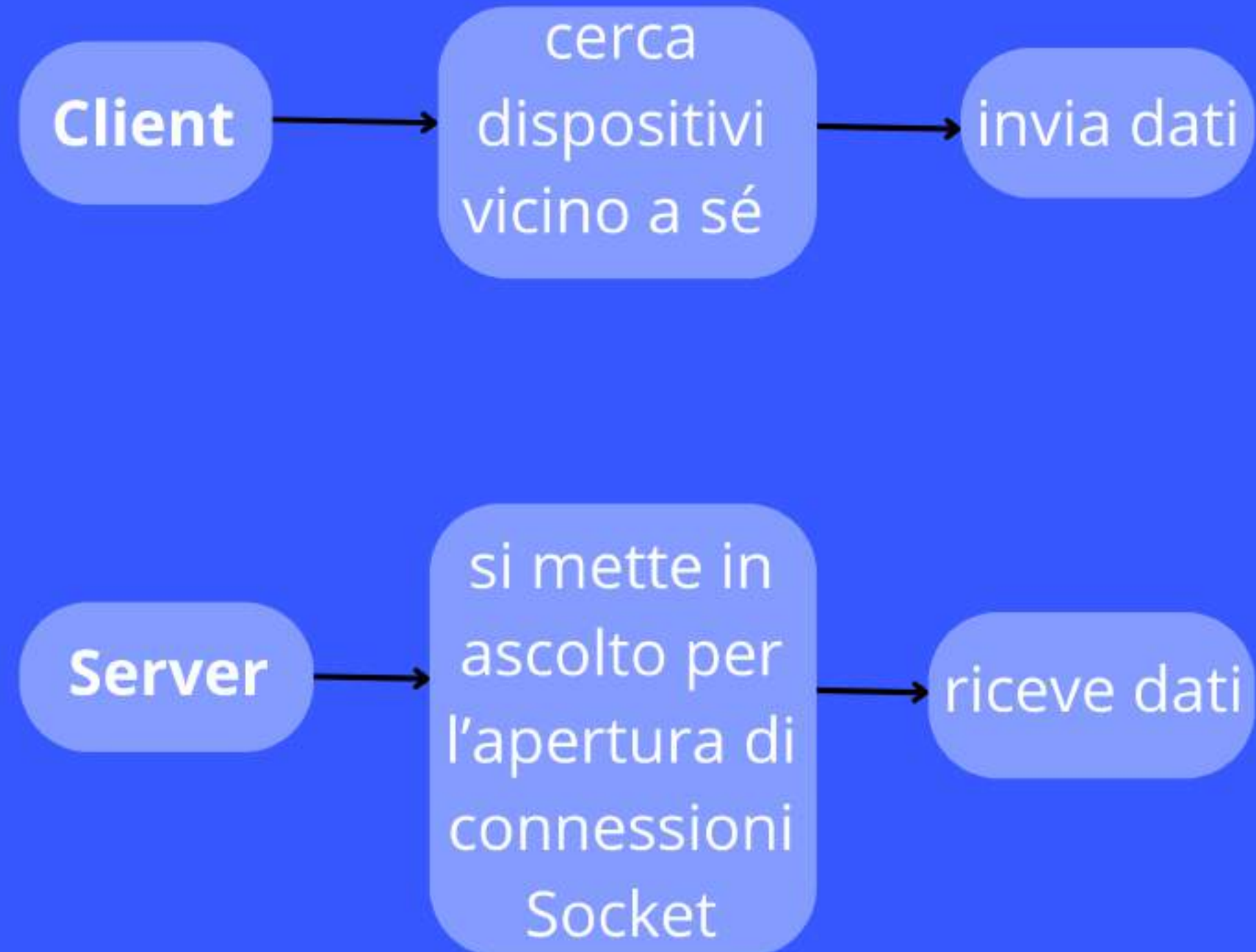
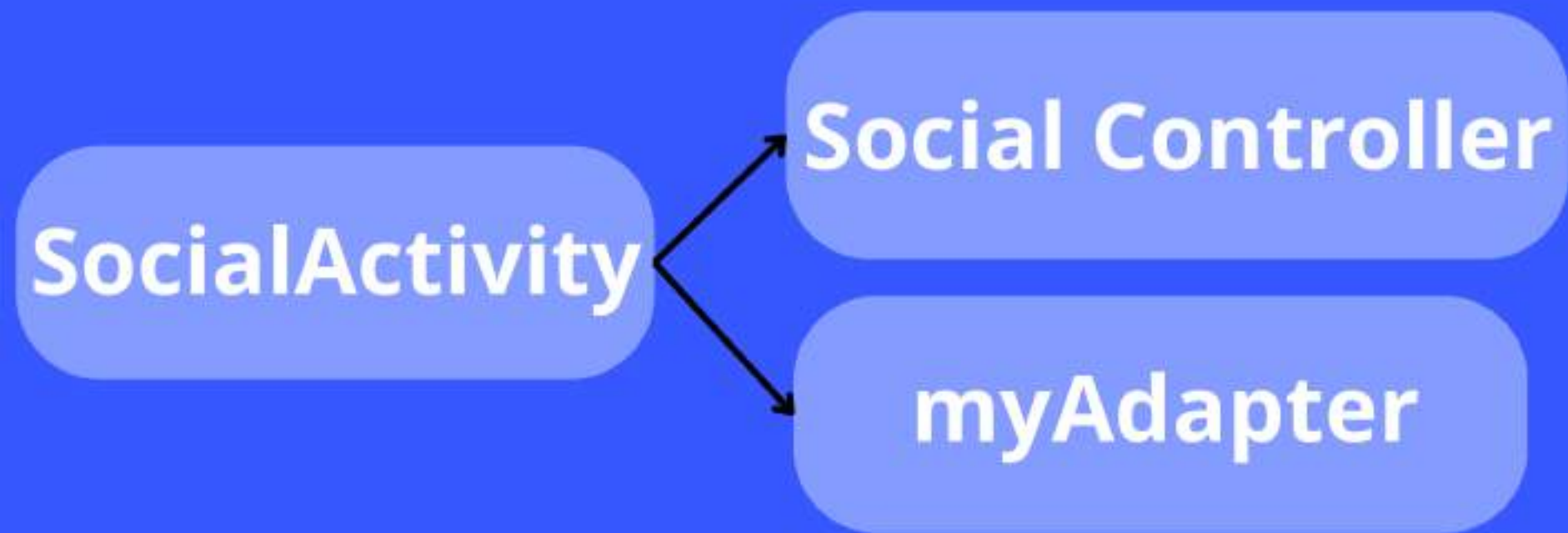
TYPE_LINEAR_ACCELERATION

```
if (event.sensor.type == Sensor.TYPE_LINEAR_ACCELERATION) {  
    val linearAcceleration = sqrt(  
        (event.values[0] * event.values[0]  
        + event.values[1] * event.values[1]  
        + event.values[2] * event.values[2])).toDouble()  
    )  
    speed = linearAcceleration * 3.6 // Converti m/s^2 in km/h
```



Bluetooth

L'activity "Social" si occupa di mettere in comunicazione più utenti attraverso il Bluetooth, consentendo loro di inviare messaggi e condividere i dati delle attività tracciate. Vengono usati i file ausiliari Social Handler per gestire tutta la questione del bluetooth e myAdapter per gestire la lista di utenti vicino a te rilevati col bluetooth.





Bluetooth

Il client cerca dispositivi vicini e, per ognuno che viene rilevato, un broadcast receiver verifica se quel dispositivo è associato a un utente su Firebase. In caso affermativo, l'utente viene aggiunto alla lista dell'adapter.

```
private val receiver = object : BroadcastReceiver() {
    @SuppressLint("MissingPermission")
    override fun onReceive(context: Context?, intent: Intent?) {
        when(intent?.action){
            BluetoothDevice.ACTION_FOUND -> {
                val device = if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU){
                    intent.getParcelableExtra(
                        BluetoothDevice.EXTRA_DEVICE,
                        BluetoothDevice::class.java
                    )
                }
                else {
                    intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)
                }

                if(device?.name != null){
                    CoroutineScope(Dispatchers.IO).launch { this: CoroutineScope
                        val found = model.updateList(device)
                        if(found){
                            withContext(Dispatchers.Main) { this: CoroutineScope
                                SocialInterface.listUpdated(getPersonList())
                            }
                        }
                    }
                }
            }
        }
    }
}
```

lato **server**, viene creato un listener RFCOMM Bluetooth socket che attende connessioni in entrata.

```
suspend fun startServer() {
    withContext(Dispatchers.IO) {
        try {
            val serverSocket = bluetoothAdapter!!.listenUsingRfcommWithServiceRecord("FitnessTrackerService", uuid)
            println("server socket: $serverSocket")
            val tmpSocket = serverSocket?.accept()
            println("tmp socket: $tmpSocket")
            withContext(Dispatchers.Main){
                MotionToast.createColorToast(
                    SocialInterface.getActivity(),
                    SocialInterface.getActivity().resources.getString(R.string.successo),
                    "aperta connessione con successo",
                    MotionToastStyle.SUCCESS,
                    MotionToast.GRAVITY_BOTTOM,
                    MotionToast.LONG_DURATION,
                    ResourcesCompat.getFont(SocialInterface.getActivity(), www.sanju.motiontoast.R.font.helvetica_regular))
            }
            if (tmpSocket != null) {
                socket = tmpSocket
                val remoteDeviceName = socket?.remoteDevice!!.name
                CoroutineScope(Dispatchers.IO).launch {
                    receiveFromSocket(SocialInterface.getActivity(), remoteDeviceName)
                }
            }
        }
    }
}
```

I dati vengono scambiati tramite gli **stream di input e output** della connessione Bluetooth. Inoltre per distinguere tra messaggi e file JSON nell'output stream, viene concatenato il carattere "#" all'inizio dei file JSON inviati.

Notifiche periodiche

Revisione Attività

Viene inviata una notifica alle 11 di sera per ricordare all'utente di rivedere le attività tracciate durante la giornata usando un worker che viene eseguito una volta al giorno, precisamente alle 11 di sera, che invia la notifica

```
fun scheduleDailyNotification(context: Context) {  
    val currentDate = Calendar.getInstance()  
    val dueDate = Calendar.getInstance()  
    dueDate.set(Calendar.HOUR_OF_DAY, 22)  
    dueDate.set(Calendar.MINUTE, 0)  
    dueDate.set(Calendar.SECOND, 0)  
    if (dueDate.before(currentDate)) {  
        dueDate.add(Calendar.DAY_OF_MONTH, 1)  
    }  
    val timeDiff = dueDate.timeInMillis - currentDate.timeInMillis  
    // Uso il PeriodicWorkRequest  
    val dailyWorkRequest =  
        PeriodicWorkRequestBuilder<NotificationWorker>(24, TimeUnit.HOURS)  
            .setInitialDelay(timeDiff, TimeUnit.MILLISECONDS)  
            .addTag("daily_notification")  
            .build()  
    WorkManager.getInstance(context).enqueueUniquePeriodicWork(  
        "daily_notification",  
        ExistingPeriodicWorkPolicy.REPLACE,  
        dailyWorkRequest  
    )  
}
```

Tracciamento Attività

Ogni quattro ore viene inviata una notifica agli utenti che non hanno utilizzato l'app durante tale intervallo di tempo. Per monitorare l'attività dell'utente, l'ultima apertura dell'applicazione viene salvata nelle SharedPreferences. Successivamente, un Worker periodico recupera questo valore e lo confronta con l'ora corrente per determinare se inviare la notifica.

```
override fun doWork(): Result {  
    val sharedPreferences =  
        applicationContext.getSharedPreferences("app_prefs", Context.MODE_PRIVATE)  
    val lastOpened = sharedPreferences.getLong("last_opened", 0)  
    val currentTime = System.currentTimeMillis()  
    if (currentTime - lastOpened >= ((3 * 60 * 60 * 1000) + (40 * 60 * 1000))) {  
        sendNotification()  
    }  
    return Result.success()  
}
```