Relatório de Estágio - Criação de uma aplicação de recolha de dados de sensores para o sistema Android

Objetivo

O objetivo deste estágio foi desenvolver uma aplicação para o sistema Android que recolhesse dados dos sensores do dispositivo e os guardasse numa base de dados, para depois serem enviados para um servidor e processados. Uma das preocupações ao desenvolver a aplicação será minimizar o consumo de bateria.

Recolha dos dados

A recolha dos dados é feita através de um serviço (*DataAquisitionService*). O objetivo é que este serviço esteja a correr constantemente. Para tal, foi criado um *BroadcastReceiver* (*ServiceRestarterBroadcastReceiver*), cuja função é iniciar o serviço caso este não esteja a correr. Este *BroadcastReceiver* é acionado em diversas situações:

1. Quando o dispositivo é ligado (*boot*);

2. Quando a aplicação é aberta;

3. Imediatamente após o serviço ser terminado;

4. Quando é detetada uma mudança no estado da conectividade;

5. A cada dois minutos, através da classe *AlarmManager* (sempre que o *Receiver* é acionado, produz um *intent* para voltar a ser acionado daí a dois minutos).

A razão pela qual não bastam as situações 1, 2 e 3 para o serviço correr constantemente é porque nem sempre que o serviço é terminado, o *BroadcastReceiver* é acionado: para que isto aconteça é necessário que o serviço seja terminado e execute o seu método *onDestroy()*, o que quase sempre não acontece porque o sistema Android, em

situações de memória escassa, para os serviços sem os deixar executar o seu método *onDestroy()*.

(NOTA: ainda assim, existem situações em que o serviço não está a correr: por exemplo, situações em que o sistema Android entra num estado de "suspensão" e adia os alarmes criados pela classe *AlarmManager*).

Este serviço contém cinco Managers cuja função é recolher e guardar os dados necessários até que o serviço proceda à sua extração. Estes *Managers* são:

- *BluetoothCustomManager* – Gere e faz scans de dispositivos bluetooth automaticamente e quando for relevante.
- *WifiCustomManager* – Gere e faz scans de redes wifi visíveis e scans de dispositivos ligados à rede automaticamente e quando for relevante.
- *MotionCustomManager* – Gere os sensores de movimento do dispositivo.
- *LocationCustomManager* – Gere os sensores de localização.
- *VariousSensorsCustomManager* – Gere os restantes sensores/funcionalidades.

A cada minuto (DELAY_DB), o serviço *DataAquisitionService* faz uma extração dos dados dos *Managers*, isto é, coleta os dados recolhidos por estes desde a última extração.

(NOTA: o intervalo de tempo entre duas extrações consecutivas pode exceder o valor de DELAY_DB, por exemplo em situações em que o sistema Android tem escassez de memória).

Dados recolhidos

Os dados que são possíveis extrair de cada manager, e que são agrupados em cada extração num objeto da classe *SensorsEntry*, são os seguintes:

- BluetoothCustomManager:
    - **bluetoothDevices** (RealmList<BluetoothDeviceCustom>) – lista de dispositivos bluetooth visíveis. Se tiverem sido feitos mais do que um scan entre duas extrações consecutivas, esta informação é relativa apenas ao último scan efetuado. A classe BluetoothDeviceCustom tem os seguintes campos:

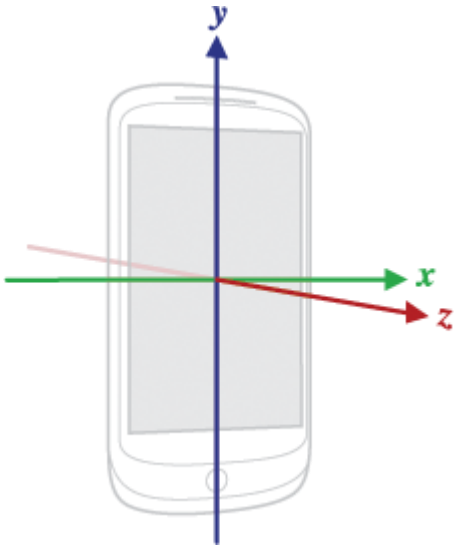
        - BluetoothDeviceCustom:
            - → **address** (String) - endereço mac do dispositivo Bluetooth.
            - → **name** (String) - nome do dispositivo Bluetooth.
            - → **type** (int) - tipo do dispositivo bluetooth (1 = Classic - BR/EDR devices, 2 = Dual Mode - BR/EDR/LE, 3 = Low Energy - LE-only, 0 = Unknown).
    - **numberBluetoothDevices** (int) - tamanho da lista acima ou –1 caso não exista informação (não tenham sido efetuados scans desde a última extração).
- WifiCustomManager:
    - **wifiDevices** (RealmList<WifiDeviceCustom>) – lista de dispositivos na rede com endereço mac (incluindo routers). Se tiverem sido feitos mais do que um scan entre duas extrações consecutivas, esta informação é relativa apenas ao último scan efetuado. A classe WifiDeviceCustom tem os seguintes campos:
        - WifiDeviceCustom:
            - → **ip** (String) - endereço ip do dispositivo na rede.
            - → **mac** (String) - endereço mac do dispositivo.
            - → **networkSSID** – SSID da rede onde foi visto o dispositivo.
    - **numberWifiDevices** (int) - tamanho da lista acima ou –1 caso não exista informação (não tenham sido efetuados scans desde a última extração).
    - **wifiNetworks** (RealmList<WifiNetworkCustom>) – lista de redes wifi visíveis. Se tiverem sido feitos mais do que um scan entre duas extrações consecutivas, esta informação é relativa apenas ao último scan efetuado. A classe WifiNetworkCustom tem os seguintes campos:
        - WifiNetworkCustom:
            - → **SSID** (String) - SSID da rede.
            - → **BSSID** (String) - endereço BSSID da rede.
    - **numberWifiNetworks** (int) - tamanho da lista acima ou –1 caso não exista informação (não tenham sido efetuados scans desde a última extração).
    - **currentNetworkSSID** (String) - SSID da rede wifi à qual estamos conectados de momento. (Esta leitura é feita apenas no momento da extração).
- MotionCustomManager:

- **motionValues** (MotionValues) - estrutura de dados que contém informações resultantes dos dados provenientes sensores de movimento do dispositivo (acelerómetro, aceleração linear, gravidade...), recolhidos desde a última extração. Esta estrutura tem os seguintes campos:
  - MotionValues:
    - → **averageAcceleration** (float) - média dos módulos dos vetores aceleração linear (m/s^2) (NOTA: nos dispositivos que não têm sensor de aceleração linear nem sensor de gravidade (a maioria dos dispositivos), isto é, possuem apenas acelerómetro, o vetor aceleração linear é calculado passando os valores obtidos pelo acelerómetro por um filtro passa-baixo, para remover a gravidade, o que faz com que os valores de aceleração linear obtidos estejam sujeitos a um maior erro).
    - → **standardDeviationAcceleration** (float) - desvio padrão dos módulos dos vetores aceleração linear (m/s^2).
    - → **averageVelocity** (float) - média dos módulos dos vetores velocidade (m/s) (NOTA: o vetor velocidade é calculado "integrando" o vetor aceleração linear, o que não dá resultados muito precisos, uma vez que estamos a integrar um conjunto discreto de dados. Se para além disto, o dispositivo possuir apenas acelerómetro, estaremos a aumentar ainda mais o erro. Por esta razão, é de evitar utilizar o campo averageVelocity, sendo preferível utilizar o campo "speed" dado pela localização, se este estiver disponível).
    - → **standardDeviationVelocity** (float) - desvio padrão dos módulos dos vetores velocidade (m/s).
    - → **averageInclinationX** (float) - média da inclinação do dispositivo relativa ao eixo x (ângulos).
    - → **standardDeviationInclinationX** (float) - desvio padrão da inclinação do dispositivo relativa ao eixo y (ângulos).

→ **averageInclinationY** (float) - média da inclinação do dispositivo relativa ao eixo x (ângulos).

→ **standardDeviationInclinationY** (float) - desvio padrão da inclinação do dispositivo relativa ao eixo y (ângulos).

→ **inMotion** (boolean) - indica se o dispositivo permaneceu em repouso (false) ou se existiu pelo menos algum movimento (existe movimento se a inclinação em algum dos eixos variou em mais de 3º) (true). Na prática, este campo é false apenas quando o dispositivo se encontra pousado numa qualquer superfície, ou em certos casos quando o utilizador está sentado com o dispositivo no bolso.



- LocationCustomManager:
  - **locationList** (RealmList<LocationCustom>) - lista de localizações obtidas pelos serviços de localização desde a última extração. A classe LocationCustom tem os seguintes campos:
    - LocationCustom:

→ **provider** (String) - fornecedor da localização (“gps”/”network”).

→ **latitude** (double) - latitude (em graus).

→ **longitude** (double – longitude (em graus).

→ **altitude** (double) - altitude (em metros), ou 0.0 se não estiver disponível.

→ **bearing** (float) - direção (em graus) no intervalo (0.0, 360.0] ou 0.0 se não estiver disponível.

→ **speed** (float) - velocidade (em m/s) ou 0.0 se não estiver disponível.

→ **accuracy** (float) - precisão horizontal da localização em metros (raio do círculo com 68% de confiança).

→ **numberOfSatellites** (int) - número de satélites que contribuíram para a localização (no caso de o fornecedor ser "gps" ou 0 caso seja "network").

→ **timestamp** (long) - instante de tempo em que ocorreu a medição (em milissegundos).

- **totalDistance** (float) - estimativa da distância total (em metros) percorrida desde o instante correspondente à última localização encontrada antes da última extração. Calculado somando todas as distâncias ente localizações consecutivas. (NOTA: este valor não representa a distância percorrida desde a ultima extração mas sim a distância percorrida desde a ultima atualização de localização anterior à ultima extração, o que é importante ter em conta em situações em que não existem atualizações de localização durante algum tempo e de repente começam a existir (por exemplo quando se sai do metro)).
- **moving** (boolean) - indica se a localização permaneceu praticamente constante desde a última extração (false) ou se existiram mudanças suficientes para se considerar que o dispositivo esteve em movimento (por exemplo o utilizador esteve a andar ou andar de transportes ou a pé) (true).

- VariousSensorsCustomManager:
  - **batteryLevel** (int) - o valor absoluto deste campo indica a percentagem de bateria do dispositivo. Se o dispositivo estiver a carregar, o valor é positivo,

caso contrário é negativo. (Esta leitura é feita apenas no momento da extração).

- **display** (boolean) - indica se o ecrã do dispositivo está aceso (true) ou apagado (false). (Esta leitura é feita apenas no momento da extração).
- **proximity** (float) - distância em cm que um objeto está do sensor de proximidade do dispositivo. (NOTA: em alguns dispositivos esta leitura é binária, isto é, 8 = longe, 0 = perto) (Esta leitura é feita apenas no momento da extração).
- **signalStrength** (int) - força do sinal de rede móvel em dBm. (Esta leitura é feita apenas no momento da extração).
- **magneticField** (float) - intensidade média do campo magnético desde a última extração (em μT).

Neste objeto da classe *SensorEntry* estão ainda presentes os seguintes campos:

- **maxSpeed** (float) - velocidade máxima (em m/s) atingida pelo dispositivo, com base nas localizações obtidas.
- **beginningTimestamp** (long) e **finalTimestamp** (long) - instantes de tempo inicial e final, respetivamente, dentro do qual foram feitas as medições.
- **onServer** (boolean) - indica se esta entrada foi introduzida no servidor (true) ou não (false).

Método de funcionamento

Os managers têm 4 modos de funcionamento (presentes na classe abstrata EnergyModes):

- MODE_HIGH_BATTERY_INMOTION: dispositivo em movimento (inMotion) e percentagem de bateria maior que 50% ou dispositivo encontra-se a carregar.
- MODE_HIGH_BATTERY_NOT_INMOTION: dispositivo parado (!inMotion) e percentagem de bateria maior que 50% ou dispositivo encontra-se a carregar.
- MODE_LOW_BATTERY_INMOTION: dispositivo em movimento (inMotion) e percentagem de bateria menor que 50%, sem estar a carregar.

- MODE_LOW_BATTERY_NOT_INMOTION: dispositivo parado (!inMotion) e percentagem de bateria menor que 50%, sem estar a carregar.

Consoante o modo em que o dispositivo se encontra, os managers têm comportamentos diferentes:

- BluetoothCustomManager:

  O tempo mínimo entre dois scans a dispositivos bluetooth depende do modo:

  - MODE_HIGH_BATTERY_INMOTION: 2 minutos
  - MODE_HIGH_BATTERY_NOT_INMOTION: 20 minutos
  - MODE_LOW_BATTERY_INMOTION: 5 minutos
  - MODE_LOW_BATTERY_NOT_INMOTION: 1 hora
- WifiCustomManager:

  O tempo mínimo entre dois scans a redes wifi depende do modo:

  - MODE_HIGH_BATTERY_INMOTION: 2 minutos
  - MODE_HIGH_BATTERY_NOT_INMOTION: 20 minutos
  - MODE_LOW_BATTERY_INMOTION: 5 minutos
  - MODE_LOW_BATTERY_NOT_INMOTION: 1 hora

  O tempo mínimo entre dois scans consecutivos a dispositivos na rede é feito quando o dispositivo se conecta a uma rede diferente da que estava conectado anteriormente ou, permanecendo na mesma rede, o intervalo de tempo mínimo entre dois scans consecutivos é o seguinte:

  - MODE_HIGH_BATTERY_INMOTION: 5 minutos
  - MODE_HIGH_BATTERY_NOT_INMOTION: 30 minutos
  - MODE_LOW_BATTERY_INMOTION: 10 minutos
  - MODE_LOW_BATTERY_NOT_INMOTION: 1 hora
- MotionCustomManager:

  Apesar de este valor ser apenas uma sugestão e não haja garantias que o Android o respeite, o intervalo de tempo entre consecutivos valores provenientes dos sensores de movimento depende do modo:

- MODE_HIGH_BATTERY_INMOTION: 0,2 segundos
- MODE_HIGH_BATTERY_NOT_INMOTION: 0,5 segundos
- MODE_LOW_BATTERY_INMOTION: 1 segundo
- MODE_LOW_BATTERY_NOT_INMOTION: 2 segundos

Este manager utiliza sempre o acelerómetro para calcular a inclinação, mas para calcular a aceleração linear (e consequentemente a velocidade), é utilizado o primeiro conjunto de sensores disponíveis no dispositivo, de entre os seguintes:

1. TYPE_LINEAR_ACCELERATION (sensor de aceleração linear)
2. TYPE_GRAVITY + TYPE_ACCELEROMETER (sensores de gravidade e acelerómetro)
3. TYPE_ACCELEROMETER (acelerómetro)

- LocationCustomManager:

Existem três fornecedores (*providers*) de localização no sistema Android. São eles: *passive*, *network* e *gps*, por ordem crescente de consumo de bateria. Destes três, é escolhido o primeiro (que estiver disponível) para ser o primário. O fornecedor primário recebe atualizações de localização num intervalo (mínimo) definido pelo modo. Para além do primário, a aplicação utiliza dois fornecedores auxiliares: fornecedor auxiliar 1 (*network*) e fornecedor auxiliar 2 (*gps*), que rebem atualizações de localização imediatamente após estas estarem disponíveis (o mais rapidamente possível).

- MODE_HIGH_BATTERY_INMOTION: Fornecedor primário sempre ativo, com intervalo mínimo entre atualizações de 5 segundos. Fornecedor auxiliar 1 é ativado se não existir uma atualização de localização há mais de 30 segundos e já estiver desligado há mais de 30 segundos, e é desligado imediatamente após ser recebida uma atualização de localização ou ao fim de 30 segundos de ter sido ligado. Fornecedor auxiliar 2 é ativado se não existir uma atualização de localização há mais de 45 segundos e já estiver desligado há mais de 45 segundos, e é desligado imediatamente após ser recebida uma atualização de localização ou ao fim de 15 segundos de ter sido ligado.
- MODE_HIGH_BATTERY_NOT_INMOTION: igual ao modo MODE_HIGH_BATTERY_INMOTION durante os primeiros 5 minutos. Após este tempo todos os fornecedores são desligados (estado *idle*).

- MODE_LOW_BATTERY_INMOTION: Fornecedor primário sempre ativo, com intervalo mínimo entre atualizações de 10 segundos. Fornecedor auxiliar 1 é ativado se não existir uma atualização de localização há mais de 90 segundos e já estiver desligado há mais de 90 segundos, e é desligado imediatamente após ser recebida uma atualização de localização ou ao fim de 30 segundos de ter sido ligado. Fornecedor auxiliar 2 é ativado se não existir uma atualização de localização há mais de 105 segundos e já estiver desligado há mais de 105 segundos, e é desligado imediatamente após ser recebida uma atualização de localização ou ao fim de 15 segundos de ter sido ligado.
- MODE_LOW_BATTERY_NOT_INMOTION: igual ao modo MODE_LOW_BATTERY_INMOTION durante os primeiros 5 minutos. Após este tempo todos os fornecedores são desligados (estado *idle*).

Existe também um estado *moving*, que é ativado sempre o utilizador atinge uma velocidade considerável (aproximadamente velocidade de caminhada) ou é deteta uma mudança significativa na localização do utilizador, nos últimos 3 minutos (ou mais, caso não exista informação acerca da localização nos últimos 3 minutos). Neste estado, há sempre pelo menos um dos fornecedores auxiliares constantemente ligado, podendo estar os dois ligados caso se determine que apenas um não é suficiente.

- VariousSensorsCustomManager:

  O funcionamento deste manager é independente do modo em que se encontra o dispositivo.

Base de Dados e Servidor

A base de dados utilizada para guardar as extrações chama-se *Realm*. Na base de dados são guardados objetos das classes *SensorEntry*, *BluetoothDeviceCustom*, *LocationCustomManager*, *MotionValues*, *WifiDeviceCustom* e *WifiNetworkCustom*, sendo que antes de cada inserção é feita uma verificação para garantir que não existem objetos duplicados das classes *BluetoothDeviceCustom, WifiDeviceCustom* e *WifiNetworkCustom.*

A cada hora, caso haja uma conexão wifi, são enviadas para o servidor as entradas

presentes na base de dados e eliminadas, exceto os objetos das classes *BluetoothDeviceCustom, WifiDeviceCustom* e *WifiNetworkCustom.*

Trabalho futuro

Um dos aspetos que terá de ser alterado nesta aplicação é a forma como se faz scans a redes wifi. Neste momento este scan é feito pela classe *WifiCustomManager* recorrendo ao método *WifiManager.startScan()*, que está descontinuado. Como neste momento não existe outra alternativa para efetuar esta operação, a única hipótese é usar este método.

Existem pelo menos duas funcionalidades que ainda não foram testadas: a aquisição do campo *magneticField* (o aparelho de teste não possuía este sensor) e a aquisição do campo *signalStrength* em dispositivos com a versão do Android anterior a JELLY_BEAN_MR1.

Um dos aspetos a melhorar será desenvolver um método para evitar que o sistema Android pare o serviço durante longos períodos de tempo. Uma ideia é utilizar, na classe *ServiceRestarterBroadcastReceiver*, em vez de o método *AlarmManager.set(…)*, o método *AlarmManager.setAndAllowWhileIdle(…)*, para definir os alarmes que irão ativar o *Receiver*, sendo que desta forma o consumo de bateria será maior.

Existe um método implementado na classe *DataAquisitionService* que permite enviar todas as entradas na base de dados em forma de *String* para um *url* (SERVER_URL, que neste momento não tem nenhum endereço) e posteriormente eliminá-las, no entanto este método ainda não foi testado e terá que ser ajustado para funcionar com o servidor.

Como utilizar

Basta extrair o ficheiro *SensorLogging.zip*. Isto irá criar uma pasta chamada *SensorLogging* onde está todo o projeto. Utilizando o software *Android Studio* (versão 3.1.4), é possível abrir esta pasta como um projeto e editar os ficheiros ou instalar a

aplicação num dispositivo. Todas as classes referidas acima encontram-se na pasta SensorLogging\app\src\main\java\com\thalesgroup\sensorlogging.

Para visualizar os conteúdos da base de dados, basta abrir o ficheiro que está localizado no dispositivo Android em /data/data/com.thalesgroup.sensorlogging/files/default.realm com o software *RealmStudio*.

Qualquer dúvida contactar diogo.m.alves@tecnico.ulisboa.pt