UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

MANUAL TÉCNICO DA APLICAÇÃO “PERDI?”

QUE UTILIZA RECURSOS DE IOT

BRUNO BACELAR ABE
SANDRA SOUZA RODRIGUES
RENATA PONTIN DE MATTOS FORTES

Nº 429

RELATÓRIOS TÉCNICOS

São Carlos – SP

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

MANUAL TÉCNICO DA APLICAÇÃO "PERDI?"
QUE UTILIZA RECURSOS DE IOT

BRUNO BACELAR ABE
SANDRA SOUZA RODRIGUES
RENATA PONTIN DE MATTOS FORTES

Nº 429

RELATÓRIOS TÉCNICOS

São Carlos – SP
Dez./2019

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Manual técnico da aplicação *"Perdi?"* que utiliza recursos de IoT

Bruno Bacelar Abe
Sandra Souza Rodrigues
Renata Pontin de Mattos Fortes

Departamento de Ciências de Computação (SCC)
Caixa Postal 668 – 13560-970 São Carlos – SP
Dezembro de 2019

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Os autores também agradecem ao PUB - Programa Unificado de Bolsas da USP pelo apoio financeiro.

LISTA DE CÓDIGOS

Código 1 – Identificação de um *Beacon* conhecido *smartwatch* 21
Código 2 – Código para envio de dados ao *smartwatch* 22
Código 3 – Código para recebimento de dados no *smartwatch* 22
Código 4 – Código para vibração 23
Código 5 – Adaptador de uma *RecyclerView* 23
Código 6 – Código para verificar se um objeto está no raio de interesse 26
Código 7 – Código para adicionar um objeto no raio de interesse 28
Código 8 – Código para verificar se há notificação referente a qualquer objeto que esteja fora do raio de interesse 29

SUMÁRIO

1 INTRODUÇÃO 7
1.1 Contexto e motivação 7
1.2 Objetivos 8
1.3 Organização deste manual 9

2 MANUAL DE INSTALAÇÃO E USO 11
2.1 Pré-requisitos do dispositivo 11
2.2 Instalação da aplicação *"Perdi?"* 12
2.3 Requisitos para uso da aplicação *"Perdi?"* 12
2.4 Cenário de uso 13
2.5 Tarefas propostas 14
2.5.1 Telas para interação com "Perdi?" no Smartphone *14*
2.5.2 Tela para interação com "Perdi?" no Smartwatch *16*

3 MANUAL TÉCNICO 19
3.1 Tecnologias empregadas 19
3.2 *Bluetooth beacons* 19
3.3 Implementação 21
3.3.1 Programação reativa *21*
3.3.2 RxBeacon *21*
3.3.3 RxWear *22*
3.4 Aplicação *"Perdi?"* no *smartwatch* 23
3.5 Aplicação *"Perdi?"* no *smartphone* 23
3.5.1 Envio de notificações *26*
3.5.2 Adição de objetos observados *28*
3.5.3 Configurações *29*
3.5.4 Navegação entre telas *30*
3.6 Requisitos mínimos 31
3.7 Licença 31

4 CONSIDERAÇÕES FINAIS 33

REFERÊNCIAS 35

CAPÍTULO

1

INTRODUÇÃO

1.1 Contexto e motivação

A Internet das Coisas (do inglês, *Internet of Things* (IoT)) é um novo paradigma computacional que está modificando a forma como as pessoas e computadores interagem. As tecnologias IoT representam a próxima evolução da Internet (TAIVALSAARI; MIKKONEN, 2017). Esse conceito evoluiu da Computação Ubíqua e o termo Internet das Coisas foi utilizado pela primeira vez por Kevin Ashton em 1.999 (ASHTON *et al.*, 2009).

Ao longo dos anos, a definição de IoT tornou-se mais ampla para incluir diferentes aplicações. Entre as definições encontradas na literatura, Atzori, Iera e Morabito (2010) argumentam que a ideia básica desse conceito é a presença de uma variedade de "coisas" que, por meio da Internet, são capazes de interagir entre si para alcançar objetivos comuns.

O número de dispositivos conectados a Internet das Coisas deverá atingir 75,44 bilhões em todo o mundo até 2.025, ou seja, está sendo previsto um aumento de cinco vezes, em dez anos (Statista, 2019). Esse rápido crescimento tem ocorrido de forma paralela ao envelhecimento populacional. Segundo Organização Mundial da Saúde (OMS) (*World Health Organization* (WHO)), o número de pessoas com mais de 60 anos deve dobrar até 2.050 (WHO, 2015). No Brasil, a população com 65 anos ou mais será quatro vezes maior até 2.060, de acordo com o Censo Nacional de 2.010, realizado pelo Instituto Brasileiro de Geografia e Estatística (IBGE) (IBGE, 2015).

O envelhecimento populacional apresenta desafios significativos e exige mais estudos e pesquisas que considerem as necessidades das pessoas idosas. A medida que as pessoas envelhecem, elas enfrentam alguns declínios em suas habilidades devido ao processo natural de envelhecimento. Dessa maneira, os efeitos degenerativos da idade podem lhes trazer algumas limitações, relacionados a visão, audição, habilidade motora e cognição (HAYFLICK, 1996;

FINN; JOHNSON, 2013).

As pessoas idosas podem se beneficiar das tecnologias IoT para obter uma melhor qualidade de vida, independência e autonomia, segundo Domingo (2012). Para isso, é necessário entender as demandas, necessidades, preferências e sentimentos desses usuários visto que as pessoas idosas somente poderão aproveitar o potencial das soluções IoT se forem acessíveis e fáceis de usar.

Nesse projeto, foi desenvolvida uma aplicação IoT para auxiliar as pessoas idosas, visando atender sua demanda por minimizar o esquecimento de seus pertences pessoais. A aplicação foi desenvolvida considerando-se questões de acessibilidade e usabilidade. Assim, o desenvolvimento incluiu os seguintes procedimentos:

1. revisão da literatura,
2. entrevistas com pessoas idosas,
3. prototipação e
4. avaliações de usabilidade e acessibilidade com especialistas da área.

A principal contribuição desse trabalho foi a investigação sobre as necessidades das pessoas idosas e o desenvolvimento de aplicações IoT considerando-se a acessibilidade e usabilidade para esse público-alvo.

1.2 Objetivos

O objetivo desse relatório técnico é apresentar um manual (guia de instruções técnicas) sobre a aplicação IoT, denominada *"Perdi?"*, desenvolvida durante a realização de estudos de Iniciação Científica, com apoio do Programa Unificado de Bolsas (PUB) da USP, e estudos práticos de Doutorado com apoio CAPES.

Esse manual visa auxiliar usuários no processo de instalação e uso da aplicação *"Perdi?"*. Além disso, descreve os detalhes técnicos sobre a aplicação, bem como apresenta as instruções relacionadas ao código-fonte, a arquitetura e as tecnologias empregadas para que outros desenvolvedores e interessados possam colaborar com a evolução do desenvolvimento.

1.3 Organização deste manual

O restante deste manual está organizado da seguinte forma: O Capítulo 2 fornece instruções para instalação e contém orientações sobre como utilizar a aplicação *"Perdi?"*. No Capítulo 3 são apresentados os detalhes técnicos da arquitetura, organização e tecnologias da aplicação, assim como instruções para colaboração. Finalmente, no Capítulo 4 são apresentadas as considerações finais, principais contribuições e oportunidades para trabalhos futuros.

CAPÍTULO

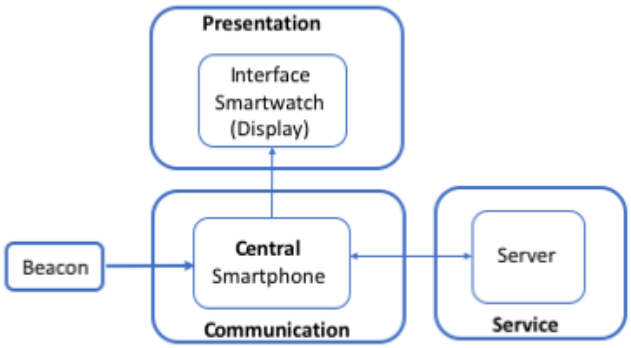
2

MANUAL DE INSTALAÇÃO E USO

2.1 Pré-requisitos do dispositivo

Para iniciar a instalação da aplicação *"Perdi?"* são necessários que determinados softwares estejam previamente disponibilizados em um dispositivo móvel. A Figura 1 mostra um esquema da arquitetura da aplicação *"Perdi?"*, onde observa-se que o componente **Central** é o responsável pela Comunicação entre as demais partes, viabilizando a interação entre as "coisas": objetos pessoas com "*Beacons*" e *Smartwatches*, por exemplo. Assim, o dispositivo móvel, um *Smartphone*, é requerido para que sejam instalados os softwares básicos, bem como os módulos centrais da aplicação.

Figura 1 – Arquitetura da aplicação *"Perdi?"*.



Fonte: Elaborada pelo autor.

Por decisões durante os estudos deste projeto, a aplicação *"Perdi?"* foi desenvolvida para funcionar apenas em plataforma Android. Android é um sistema operacional *Open Source*, amplamente disseminado nas comunidades de desenvolvedores e usuários de aplicações móveis, baseado no núcleo Linux. Foi desenvolvido por um consórcio de desenvolvedores conhecido como *Open Handset Alliance*, e a empresa Google é principal colaboradora.

Desse modo, os seguintes recursos são pré-requisitos para a instalação da aplicação *"Perdi?"* no dispositivo móvel:

(1.) Android 5.0 (*Lollipop*) ou superior - que é o sistema operacional do dispositivo;

(2.) Conexão à Internet - possibilita que o usuário realize o *download* do apk[1].

2.2 Instalação da aplicação *"Perdi?"*

A instalação da aplicação deve ser realizada por meio de *download* do apk, porque a aplicação *"Perdi?"* ainda não está disponível na *Google Play*. Essa instalação considera que os seguintes passos devem ser conduzidos no seu dispositivo móvel (*Smartphone*):

1. Permitir que o seu dispositivo móvel contenha a aplicação em questão: deve-se acessar o dispositivo, em Configurações » Segurança "habilitar" a opção de Fontes desconhecidas ou de Terceiros;

2. Para obter o pacote da aplicação *"Perdi?"*, é necessário acessar a conta no Github (<https://github.com/abe2602/Beacon_Application>) e realizar o *download* do apk em seu dispositivo móvel. Para isso, acesse a pasta apk. A seguir, realize o *download* do apk disponível;

3. Colocar a aplicação *"Perdi?"* disponível para funcionar no dispositivo móvel, ou seja, a partir do *download* realizado, deve-se clicar no item referente a aplicação e esperar a sua instalação finalizar.

2.3 Requisitos para uso da aplicação *"Perdi?"*

A aplicação *"Perdi?"* pode ser acessada a partir do *Smartphone* e *Smartwatch*, logo após concluídos os três passos da seção 2.2. Assim, para que a aplicação funcione adequadamente, é necessário o acesso à Internet, *bluetooth* ativo e os dispositivos pareados.

O uso da aplicação é possível a partir de realizada sua instalação, mas a sua utilidade é evidenciada com a utilização dos demais recursos de IoT, conforme descritos na próxima seção.

[1] Android Package (do inglês, *Android Application Pack* (apk)) é um arquivo de pacote destinado ao sistema operacional Android.

2.4 Cenário de uso

A aplicação *"Perdi?"* foi desenvolvida para oferecer um suporte à vida cotidiana das pessoas idosas, visando auxilia-las a encontrar objetos de uso pessoal, em geral aqueles pequenos e leves, os quais podem ter sidos esquecidos onde foram previamente colocados/guardados. Esse suporte foi proposto para atender uma demanda reconhecida na literatura, pois a medida que as pessoas envelhecem suas habilidades cognitivas são afetadas, causando dificuldades para lembrar de tarefas e atividades do dia-a-dia (MALONE; BASTIAN, 2016).

Assim, nesse contexto, o seguinte cenário foi elaborado:

> "Uma viúva de 67 anos (Maria) mora sozinha em um condomínio, longe do centro da cidade. Seus filhos mudaram-se para ficar mais próximos de seus empregos. Ao passar muito tempo sozinha em casa, Maria aproveita as várias atividades oferecidas pelo seu condomínio, como ginástica, dança de salão e voleibol. Maria é uma idosa cheia de energia. No entanto, recentemente, ela começou a enfrentar o declínio de algumas habilidades cognitivas, como alterações de memória, atenção e concentração devido ao processo de envelhecimento. No último mês, ela repetidamente esqueceu em casa a chave para entrar no clube do condomínio. O esquecimento causa problemas para Maria e a deixa em situações desagradáveis. Maria é uma senhora que preza por sua independência e gosta de se sentir útil, de modo que o esquecimento a deixou desanimada. Esse tipo de situação causa problemas emocionais em pessoas idosas, provocando uma sensação de dependência e até mesmo de inutilidade."

Nesse cenário, verifica-se o potencial de utilidade de suporte com recursos e tecnologias IoT, para ajudar esses usuários a lembrarem de seus pertences. Portanto, uma aplicação IoT poderia avisar a pessoa idosa quando ela tiver esquecido itens em casa, tais como chaves, cartões de acesso, carteiras, entre outros. Este aplicativo deveria rastrear objetos que foram registrados por meio de *beacons*. Dessa forma, uma aplicação IoT, que considere a interação entre objetos, teria capacidade de identificar aqueles objetos que estivessem fora/ além de seu raio de alcance.

A definição desse cenário de uso foi fundamental para que fossem identificadas as atribuições que a aplicação deveria realizar. Assim, na próxima seção são descritas as tarefas (procedimentos a serem disponibilizados aos usuários) para interação com a aplicação. Conforme já mencionado, a aplicação *"Perdi?"* foi assim denominada para refletir a situação nos momentos de esquecimento, que as pessoas idosas frequentemente enfrentam e, podem vir a indagar se perderam o objeto, ou qual é o objeto que lhes faz falta.

2.5 Tarefas propostas

As principais funcionalidades da aplicação *"Perdi?"* no *smartphone* são: visualizar, adicionar objetos para monitoramento e configurar notificações. No *smartwatch*, as funções são: visualizar, ativar e desativar notificações. Para que um determinado usuário interaja com a aplicação e execute as funcionalidades nesses dispositivos, foi proposto um conjunto de tarefas, visando atender cenários hipotéticos de uso.

As tarefas propostas para o *smartphone*, de maneira simplificada, são:

(**T1-smph.**) - Conectar o *smartwatch* ao *smartphone*;

(**T2-smph.**) - Visualizar a tela inicial da aplicação;

(**T3-smph.**) - Adicionar um novo objeto para monitoramento;

(**T4-smph.**) - Configurar as notificações.

Para o *smartwatch*, as tarefas propostas são:

(**T1-smwt.**) - Visualizar a tela inicial;

(**T2-smwt.**) - Ativar uma notificação;

(**T3-smwt.**) - Visualizar a tela de notificação;

(**T4-smwt.**) - Desativar a notificação.

A partir de definidos os cenários e as tarefas, foi possível criar os protótipos, e a aplicação pode ser construída e implementada. Nas próximas subseções, são descritas as telas de interação, desenvolvidas na aplicação *"Perdi?"*, que correspondem às interfaces que viabilizam as tarefas propostas.

2.5.1 Telas para interação com "Perdi?" no Smartphone

Nesta subseção são apresentadas as telas referentes às tarefas mencionadas anteriormente, bem como é descrito o funcionamento esperado, mas sem detalhamento técnico (a ser descrito no Capítulo 3).

*Conexão dos dispositivos (**T1-smph.**)*

A Figura 2 mostra a tela com a representação de todos os relógios conectados ao *smartphone*. Nela é possível escolher qual relógio será utilizado para o monitoramento (no caso, existem 3 opções: Apple Watch 2, "Meu relógio Gear 2" e CEE Watch Model 3355). Para o exemplo, tem-se que o segundo (o "Meu relógio Gear 2") é o *smartphone* do usuário.

Figura 2 – *Smartwatches* disponíveis para conexão

Fonte: Elaborada pelo autor.

Uma vez que o relógio tenha sido escolhido, a tela para visualizar os objetos monitorados será exibida e a tarefa proposta poderá ser executada.

*Visualizar monitoramentos (**T2-smph.**)*

Essa tela tem o objetivo de informar os objetos cadastrados no sistema, bem como editar essas informações. A Figura 3 mostra todos os objetos monitorados pelo sistema, além das opções de adicionar e desativar, excluir um monitoramento ou configurar a aplicação.

Ao desativar um monitoramento, espera-se que o sinal do objeto pare de ser observado pelo *smartphone*, podendo ser ativado a qualquer momento. Se desativado, o *smartwatch* não receberá notificações por parte do *smartphone*. Ao excluir um monitoramento, espera-se que o *smartphone* não seja mais capaz de monitorar o objeto, ou seja, o *beacon* deve ser alocado

Figura 3 – Objetos monitorados

Fonte: Elaborada pelo autor.

para outro lugar para que possa ser reutilizado. Por fim, é possível acessar a tela de adição de monitoramento, que é descrita a seguir.

*Adicionar monitoramento (**T3-smph.**)*

Com intuito de adicionar um novo monitoramento, é necessário adicionar o nome do objeto e escolher o sensor que será acoplado ao mesmo, como mostra a Figura 4. Após completar as informações, o objeto é adicionado.

*Configurações (**T4-smph.**)*

A última tarefa do cenário proposto é a de mudança nas configurações da aplicação. Nessa tela, é possível modificar o alcance dos *beacons* e desativar o recebimento de todas as notificações, como pode ser visto na Figura 5.

2.5.2 Tela para interação com "Perdi?" no Smartwatch

Todas as tarefas relacionadas ao *smartwatch* são realizadas em apenas uma tela, visto que as interações com esse dispositivo foram limitadas devido ao tamanho limitado da tela. A Figura 6 apresenta a tela de interação com o *smartwatch* (**T1-smwt.**), na qual é possível ativar (**T2-smwt.**)

Figura 4 – Adicionar monitoramento

Fonte: Elaborada pelo autor.

ou desativar (**T4-smwt.**) as notificações dos objetos que estão sendo observados/monitorados (**T3-smwt.**).

No próximo Capítulo serão apresentadas as tecnologias empregadas e como foram utilizadas para o desenvolvimento dos programas que compõem a aplicação *"Perdi?"*. Além das tecnologias, a lógica de verificação se um *beacon* está fora de sua zona segura é discutida.

Figura 5 – Configurações da aplicação

Fonte: Elaborada pelo autor.

Figura 6 – Tela do *Smartwatch*

Fonte: Elaborada pelo autor.

CAPÍTULO

3

MANUAL TÉCNICO

Neste capítulo apresentamos uma descrição sobre como foi desenvolvida a aplicação *"Perdi?"*, as tecnologias empregadas, os detalhes técnicos sobre os *beacons*, organização do projeto e as possíveis melhorias.

3.1 Tecnologias empregadas

A aplicação trabalha com a comunicação entre dispositivos (*smartphone* e *smartwatch*), que é realizada por meio do canal *bluetooth* dos aparelhos. Além dos dois dispositivos, há a presença de *beacons* acoplados a objetos. Para o reconhecimento dos *beacons* foi utilizado o conceito de Programação Reativa, por meio da biblioteca *RxBeacon*[1], que é um projeto de Software Livre.

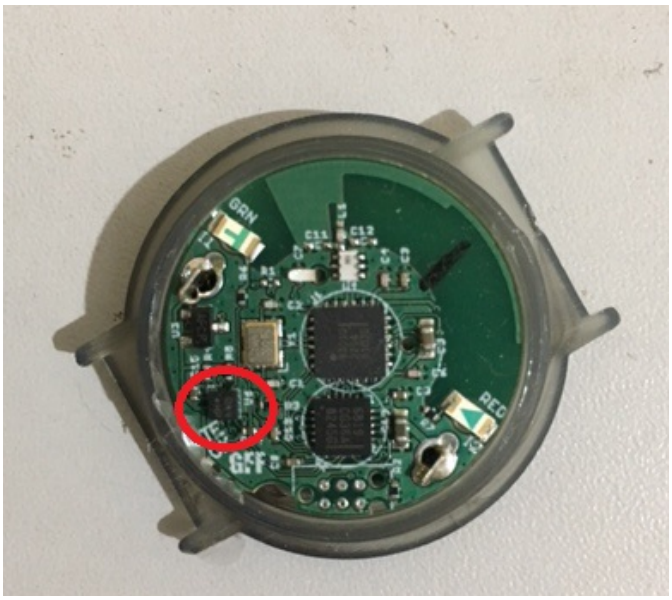
Conforme visto no Capítulo 1, na Figura 1 foi representado o esquema da arquitetura da aplicação *"Perdi?"*. O componente **Central**, o responsável pela Comunicação entre as demais partes, viabiliza a interação entre as "coisas": objetos pessoas com "*Beacons*" e *Smartwatches*. Em um *Smartphone* são alocados os módulos centrais da aplicação *"Perdi?"*.

3.2 *Bluetooth beacons*

Os *Bluetooth Beacons*, ou apenas *Beacons*, são dispositivos de *hardware* da classe *Bluetooth Low Energy* (BLE) capazes de transmitir a sua identificação por meio de *broadcast*. Esse tipo de dispositivo é popularmente utilizado para localização *indoor* pois possuem melhor desempenho, que é obtido pelo menor número de ruídos. A Figura 7 mostra uma foto do *Beacon* utilizado na aplicação *"Perdi?"*.

[1] <https://github.com/florent37/RxBeacon>

Figura 7 – Foto do *Beacon* utilizado



O uso de *beacons* em aplicações que possuem funcionalidade associada à proximidade entre dispositivos ou qualquer outro objeto instrumentado com *beacons* também é comum, uma vez que é possível calcular, com certa precisão, a distância entre o emissor e receptor, no nosso caso, *beacon* e o *smartphone*.

Para calcular a distância entre emissor e receptor alguns dados são necessários, os quais são providos pelo *beacon*, tais como: a indicação de intensidade do sinal recebido (do inglês, *Received Signal Strength Indication* – RSSI) e a força do sinal (do inglês, *Measured Power* – MP).

O RSSI é a força do sinal esperado, num cenário ideal, para 1 metro de distância. Já a força do sinal é a do sinal que é recebido de um *beacon* num dado instante. A partir dessas duas informações, foi possível aproximar a distância entre emissor e receptor utilizando a seguinte equação:

$$\text{Distância} = 10^{((MP-RSSI)/10*N)} \tag{3.1}$$

onde N é uma constante que deve ser utilizada de acordo com o ambiente de uso. Na aplicação em questão, o valor utilizado foi 4 por apresentar melhores resultados com base em testes empíricos já executados.

3.3 Implementação

Toda a implementação foi realizada exclusivamente para plataforma *Android*, utilizando a linguagem de programação Java 8[2].

De modo resumido, a principal finalidade da aplicação *"Perdi?"* no *Smartphone* é parear com um *smartwatch* e notificá-lo caso o *beacon* acoplado em algum objeto esteja fora de alcance pré-definido. Portanto, o controle de acionamento para essa notificação ocorre em tempo-real, de execução da aplicação. Esse tipo de controle requer estratégias e tecnologias especializadas para que os dispositivos envolvidos atendam ao fluxo de execução com interrupções (ou seja, possibilitem intervenções assíncronas) e que sincronizem para obtenção das informações significativas para o cenário definido.

3.3.1 Programação reativa

A Programação Reativa consiste em programar a partir de fluxos (*streams*) de dados assíncronos que são dados sem uma ordem ou regra de recebimento. Nesse contexto, trabalhar com *streams* de dados assíncronos pode se tornar extremamente difícil, envolvendo sincronização e criação de *threads* e/ou *callbacks*. Portanto, é interessante utilizar a Programação Reativa.

Na aplicação *"Perdi?"*, as *streams* de dados são representadas pelos dados dos *beacons*, que chegam assincronamente até o dispositivo e pelo fluxo de dados entre *smartwatch* e *smartphone*. Para a captura e uso da *stream* dos *Beacons* utilizou-se a biblioteca *RxBeacon*. Para a criação das *streams* entre os dispositivos utilizou-se a biblioteca *RxWear*. Ambas baseadas em *RxJava2*, um *framework* famoso e popular para Programação Reativa.

3.3.2 RxBeacon

O Código 1 mostra como utilizar a biblioteca *RxBeacon* para encontrar um *Beacon* específico. Para executar essa tarefa, uma *thread* separada é instanciada, já que a *thread* principal é utilizada apenas para UI e cálculos pequenos.

Código 1 – Identificação de um *Beacon* conhecido *smartwatch*

```
1
2     reativeBeacons.observe()
3         scribeOn(Schedulers.io())
4             .observeOn(AndroidSchedulers.mainThread())
```

[2] https://www.java.com/en/download/faq/java8.xml

```
5           .filter(beacon -> (beacon.macAddress.address.equals(macSctring)))
6
```

Uma vez que as *threads* foram corretamente configuradas, há o filtro dos *beacons* que serão ouvidos/observados pela aplicação. Na aplicação *"Perdi?"* em específico, o filtro ocorre pelo endereço MAC dos sensores conhecidos.

3.3.3 RxWear

A comunicação com o *smartwatch* foi criada utilizando a biblioteca *RxWear*, que possibilita a sincronização entre *wearables* e *smartphones* Android por meio de Programação Reativa.

Com o *RxWear*, é possível listar todos os dispositivos pareados ao *smartphone*, caso estejam cadastrados no sistema *Wear OS*, da Google. O Código 2 exemplifica o funcionamento do envio de dados do *smartphone* para o *smartwatch*.

Com a *RxWear* já instanciada, basta utilizar a função *sendDataMapToAllRemoteNodes()* para enviar um dado. Neste caso, foi definido que todos os *smartwatches* que escutam em "/nome do relógio", irão receber o dado (*boolean*).

Código 2 – Código para envio de dados ao *smartwatch*

```
1       rxWear.message()
2           .sendDataMapToAllRemoteNodes("/" + this.smartWatch)
3           .putBoolean("notification", true)
4           .toObservable()
5
```

O procedimento para receber o dado no *smartwatch* é similar ao de envio, porém a função *listen()* é utilizada, como pode ser verificado no Código 3.

Código 3 – Código para recebimento de dados no *smartwatch*

```
1       rxWear.message()
2           .listen("/" + deviceName, MessageApi.FILTER_LITERAL)
3           .compose(MessageEventGetDataMap.noFilter())
4           .doOnNext(item ->{vibrate();})
5
```

3.4 Aplicação *"Perdi?"* no *smartwatch*

De acordo com o Código 3, toda vez que uma notificação (um *boolean*) chega, o método *vibrate* (Código 4) é acionado fazendo com que o *smartwatch* vibre.

Código 4 – Código para vibração

```
1            public void vibrate(){
2                Vibrator vibrator = (Vibrator) getSystemService(
   VIBRATOR_SERVICE);
3                long[] vibrationPattern = {0, 500, 50, 300};
4                final int indexInPatternToRepeat = -1;
5                vibrator.vibrate(vibrationPattern, indexInPatternToRepeat);
6            }
7
```

O *smartphone* poderia enviar qualquer tipo de dado, desde que serializável, para o *smartwatch* (*strings*, *integer*, etc). Porém, foi utilizado *boolean* por questão semântica. Assim, faz sentido enviar *true* quando a ação é executada com sucesso, apesar de redundante.

3.5 Aplicação *"Perdi?"* no *smartphone*

Diferentemente da aplicação do *smartwatch*, que é responsável apenas por receber um dado, o *smartphone* possui mais recursos funcionais, e compõem um conjunto de interações mais sofisticado. A ideia dessa aplicação é conseguir encontrar um grupo de *beacons*, previamente cadastrados e acoplados em objetos, em um alcance pré-definido e ajustável. Assim que um dos sensores fica fora do raio de alcance, uma notificação é enviada para o *smartwatch*.

Para exibição de listagens (*smartwatches* pareados ao *smartphone*, objetos observados e sensores cadastrados) são utilizadas *RecyclerViews*. Uma *RecyclerView* é um componente do *framework Android* capaz de exibir itens dinamicamente em uma lista. Esses componentes são extremamente eficientes e de fácil implementação. Normalmente esse componente faz uso de um adaptador, que é responsável por fazer o *bind* entre item e a posição que esse deve ser exibido, além de definir o layout de cada um dos itens.

O Código 5 é o adaptador utilizado na listagem de *smartwatches* pareados ao *smartphone*, onde é possível ver claramente a estrutura básica que todo adaptador de *RecyclerView* possui.

Código 5 – Adaptador de uma *RecyclerView*

```
1
2 public class SmartwatchRecyclerViewAdapter extends RecyclerView.Adapter<
    SmartwatchRecyclerViewAdapter.NumberViewHolder> {
3     private ArrayList<String> connectedDevices;
4     private ListenItemClick myClickListenner;
5
6     /*Construtor da classe, recebe como parâmetro a quantidade de itens*/
7     public SmartwatchRecyclerViewAdapter(ArrayList<String> connectedDevices,
    ListenItemClick myClickListenner){
8         this.connectedDevices = connectedDevices;
9         this.myClickListenner = myClickListenner;
10    }
11
12    /*Infla o layout*/
13    @Override
14    public NumberViewHolder onCreateViewHolder(ViewGroup parent, int viewType)
    {
15        Context context = parent.getContext();
16        int layoutId = R.layout.smartwatch_recyclerview_layout;
17        LayoutInflater inflater = LayoutInflater.from(context);
18        boolean attachImmediately = false;
19
20        View view = inflater.inflate(layoutId, parent, attachImmediately);
21        NumberViewHolder viewHolder = new NumberViewHolder(view);
22
23        return viewHolder;
24    }
25
26     /*Faz o ligamento entre view <-> posição na lista*/
27    @Override
28    public void onBindViewHolder(NumberViewHolder holder, int position) {
29        holder.bind(position);
30    }
31
32    /*Quantidade de itens na lista*/
33    @Override
34    public int getItemCount() {
```

```
35          return connectedDevices.size();
36      }
37
38      /*View de cada item propriamente dita*/
39      class NumberViewHolder extends RecyclerView.ViewHolder implements View.
    OnClickListener {
40          TextView deviceName = null;
41
42          public NumberViewHolder(View itemView) {
43              super(itemView);
44              deviceName = itemView.findViewById(R.id.textViewTeste);
45              itemView.setOnClickListener(this);
46          }
47
48          public void bind(int listIndex){
49              deviceName.setText(connectedDevices.get(listIndex));
50          }
51
52          @Override
53          public void onClick(View v) {
54              int clickPosition = getAdapterPosition();
55              myClickListenner.onItemClick(clickPosition);
56          }
57      }
58
59      /*Limpa a recyclerview*/
60      public void clear() {
61          int size = connectedDevices.size();
62          connectedDevices.clear();
63          notifyItemRangeRemoved(0, size);
64      }
65
66      /*Adiciona toque na recyclerview*/
67      public interface ListenItemClick {
68          public void onItemClick(int clickItem);
69      }
70  }
```

Para encontrar os *beacons*, filtrá-los e identificar se estão dentro ou fora do alcance, utilizou-se a biblioteca *RxBeacon*, detalhada na subseção 3.3.2. A lógica se resume em verificar a existência de algum sinal de *beacon* conhecido, caso exista, e então a distância entre o objeto e o *smartphone* é calculada. Se a distância for maior do que a definida, o *smartwatch* é avisado, caso contrário, a verificação continua. As próximas subseções contêm detalhes de implementação das principais funcionalidades da aplicação.

3.5.1 Envio de notificações

Uma notificação no contexto da aplicação *"Perdi?"* é definida por uma interação bem sucedida entre *smartphone* e *smartwatch*. As notificações são criadas toda vez que um objeto observado encontra-se fora do raio "seguro", ou seja, do raio de alcance pré-definido. O Código 6 é responsável por encontrar os *beacons* cadastrados, verificar a distância e criar notificações.

Código 6 – Código para verificar se um objeto está no raio de interesse

```
1
2  void findBeacons(String macSctring){
3      RxPaperBook settingsBook = RxPaperBook.with("settings");
4      Single<Object> savedSettings = settingsBook.read("settings").
       onErrorReturnItem(new Settings(1, true));
5
6      Disposable listItemsDisposable =  savedSettings.flatMapObservable(settings
       -> {
7                  Settings mySettings = (Settings) settings;
8
9                  RxPaperBook book = RxPaperBook.with("monitored_things");
10                 Single<Object> cache = book.read("monitored_things").
       onErrorReturnItem(Collections.emptyList());
11
12                 return cache.subscribeOn(Schedulers.computation())
13                         .observeOn(AndroidSchedulers.mainThread())
14                         .doOnSuccess(listItems -> {
15                             monitoredThings = (ArrayList<TrackedThing>)
       listItems;
16                             setupRecyclerView(this, (ArrayList<TrackedThing>)
       listItems);
17                             this.labelToMac(monitoredThings);
18                         })
```

```
19                      .flatMapObservable(item ->
20                              reactiveBeacons.observe()
21                                      .subscribeOn(Schedulers.io()) //Faz o
    trabalho numa thread separada
22                                      .observeOn(AndroidSchedulers.mainThread
    ()) //Observa na thread principal
23                                      .filter(beacon -> (beacon.macAddress.
    address.equals(macSctring) ))
24                                      .flatMap(beaconData -> {
25                                          for(TrackedThing auxThing:
    monitoredThings){
26                                              if(beaconData.macAddress.
    address.equals(auxThing.getBeaconMac())){
27                              double distance = rssiToMeters(beaconData);
28
29                              if(!mySettings.getHasNotification()){
30                                  distance = -1;
31                              }
32
33                              if(distance> (mySettings.getRange() + 1) &&
    distance != -1 && auxThing.isAvailable()){
34                              return rxWear.message().
    sendDataMapToAllRemoteNodes("/" + this.smartWatch)
35                              .putBoolean("notification", true)
36                              .toObservable()
37                              .doOnComplete(() -> Log.d("SUCCESS", "enviei ;)
    "));
38                                      }
39                                  }
40                              }
41                              return Observable.just(false);
42                          })
43                      );
44              }
45      ).doOnError(error -> Log.d("ERROR", error.toString())).ignoreElements().
    onErrorComplete().subscribe();
46
47      subscription.add(listItemsDisposable);
```

```
48              }
49
```

Como pode ser verificado no Código 6, recupera-se as configurações salvas da aplicação a partir da memória *cache*. Com as configurações carregadas, a lista de objetos observados é exibida na tela. A listagem de coisas observadas também é mantida na memória *cache*. Já a listagem de objetos é utilizada para inflar uma *RecyclerView* utilizando o método *setupRecyclerView*. Esse método configura a exibição e *Listenners* da lista.

Com a utilização da biblioteca *RxBeacon*, instanciada como *reactiveBeacons* no código, uma busca por *beacons* próximos é executada. Durante a busca, apenas *beacons* com endereço MAC conhecido são aceitos. Essa filtragem acontece por meio da função *filter* disponível nativamente no padrão *Rx*.

Quando um *beacon* de interesse é encontrado, existe a transformação do RSSI para metros, utilizando a fórmula que está disponível na seção 3.2. Se tudo ocorrer de forma correta, a comparação entre raio de interesse definido e distância do *beacon* é realizada.

3.5.2 Adição de objetos observados

O Código 7 mostra a implementação de objetos a serem observados/monitorados pela aplicação *"Perdi?"*, além de os cadastrar.

Código 7 – Código para adicionar um objeto no raio de interesse

```
1
2     Disposable addDisposable = book.read("available_sensors")
3                 .subscribeOn(Schedulers.io())
4                 .observeOn(AndroidSchedulers.mainThread())
5                 .doOnSuccess(item -> {
6                     arrayString = (ArrayList<TrackedThing>) item;
7                     //Log.d("HelpMe", "addMonitoring:  " + Integer.toString(
    arrayString.size()));
8                     setView(view, arrayString);
9                 })
10                .doOnError(error -> {
11                    Log.d("HelpMe", "addMonitoring:  " + error.toString());
12                    TrackedThing auxTf = new TrackedThing(" ", "Sensor 1", true
    );
```

```
13                      arrayString.add(auxTf);
14
15                      TrackedThing auxTf2 = new TrackedThing(" ", "Sensor 2",
    true);
16                      auxTf2.setName("Sensor 2");
17                      arrayString.add(auxTf2);
18
19                      book.write("available_sensors", arrayString)
20                              .subscribeOn(Schedulers.io())
21                              .observeOn(AndroidSchedulers.mainThread())
22                              .doOnComplete(() -> {
23                                  setView(view, arrayString);
24                              }).doOnError(error2 -> Log.d("HelpMe", "
    addMonitoring:  " + error2.toString())).subscribe();
25                  }).ignoreElement().onErrorComplete()
26                  .subscribe();
27
```

3.5.3 Configurações

O Código 8 mostra a implementação da aplicação, mediando as notificações que devem ser comunicadas entre os dispositivos da aplicação *"Perdi?"*.

Código 8 – Código para verificar se há notificação referente a qualquer objeto que esteja fora do raio de interesse

```
1           public class Settings {
2           private boolean hasNotification;
3           private int range;
4
5           public Settings(int range, boolean hasNotification){
6               this.range = range;
7               this.hasNotification = hasNotification;
8           }
9
10          public Boolean getHasNotification() {
11              return hasNotification;
12          }
```

```
13
14          public void setHasNotification(Boolean hasNotification) {
15              this.hasNotification = hasNotification;
16          }
17
18          public int getRange() {
19              return range;
20          }
21
22          public void setRange(int range) {
23              this.range = range;
24          }
25
26      }
27
```

3.5.4 Navegação entre telas

A aplicação *"Perdi?"* foi construída utilizando o modelo de *Single Activity*, ou seja, existe apenas uma *Activity* em toda a aplicação. Essa *Activity* funciona como *container* para todos os *Fragments* utilizados. Esse modelo foi escolhido pois a criação de *Activities* é um processo computacionalmente pesado e como a aplicação *"Perdi?"* possui muitas operações de entrada/saída, (do inglês, *input/output* (I/O)) é importante economizar em processamento.

Para navegar entre os *Fragments*, foi criada uma classe *Utils* em que estão disponíveis todos os modos de navegação. Os métodos disponíveis são:

1. **`navigateToFragment:`** é utilizado para navegar entre dois *Fragments* sem que nenhum dado seja passado entre as telas;

2. **`navigateToFragmentWithData:`** é utilizado para navegar entre dois *Fragments* passando como parâmetro uma listagem de dados (sensores disponíveis, por exemplo);

3. **`navigateToFragmentWithString:`** é utilizado para navegar entre dois *Fragments* passando como parâmetro uma *String*.

3.6 Requisitos mínimos

Para execução da aplicação *"Perdi?"*, é requerido obrigatoriamente: *smartphone*, *beacon* e *smartwatch*.

O *smartphone* deve possuir:

- Android 5.0 (*Lollipop*) ou superior - que é o sistema operacional do dispositivo;
- Conexão *bluetooth* para que seja possível conectar os dispositivos com o *beacon*;
- *Wear OS* instalado no *smartphone*;

Os itens acima são necessários para o funcionamento adequado da aplicação, sendo que a falta deste resultará em erros.

3.7 Licença

Essa aplicação faz uso da licença "GNU General Public License v3.0", que permite que a aplicação *"Perdi?"* seja distribuída e modificada como *software* livre sobre as condições impostas pela licença. A licença pode ser encontrada, juntamente com o código fonte do projeto, no seguinte repositório do GitHub: <https://github.com/abe2602/Beacon_Application>.

CAPÍTULO

4

CONSIDERAÇÕES FINAIS

A Internet das Coisas é um novo paradigma que está crescendo significantemente e tem o potencial de fornecer suporte e assistência às pessoas idosas. A aplicação descrita neste manual técnico tem o objetivo de auxiliar esses usuários de forma a evitar o esquecimento de seus pertences e proporcionar autonomia e independência no seu cotidiano.

Neste manual, foi descrito um detalhamento dos aspectos técnicos utilizados na implementação da aplicação *"Perdi?"*. É apresentado o funcionamento da aplicação *"Perdi?"*, desde o seu uso, sua modelagem, *design* e configuração, até o desenvolvimento da aplicação para plataforma Android.

Com esse manual, espera-se que o usuário compreenda os procedimentos para instalação e utilização da aplicação *"Perdi?"*. Para os desenvolvedores, espera-se que seja possível o entendimento sobre a estrutura da aplicação e a possibilidade de novas contribuições para o mesmo. Desse modo, qualquer pessoa pode contribuir com novos desenvolvimentos e aprimoramentos para a aplicação.

"Perdi?" consiste em uma aplicação de código livre, cujo código fonte está hospedado no repositório do GitHub: <https://github.com/abe2602/Beacon_Application>. Portanto, para contribuir basta entrar em contato com algum dos autores. Em caso de dúvidas sobre o projeto, entre em contato com seus autores.

REFERÊNCIAS

ASHTON, K. *et al.* That “internet of things” thing. **RFID journal**, v. 22, n. 7, p. 97–114, 2009. Citado na página 7.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787 – 2805, 2010. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128610001568>. Citado na página 7.

DOMINGO, M. C. An overview of the internet of things for people with disabilities. **Journal of Network and Computer Applications**, v. 35, n. 2, p. 584 – 596, 2012. ISSN 1084-8045. Simulation and Testbeds. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804511002025>. Citado na página 8.

FINN, K.; JOHNSON, J. A usability study of websites for older travelers. In: ____. **Universal Access in Human-Computer Interaction. User and Context Diversity: 7th International Conference, UAHCI 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part II**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 59–67. ISBN 978-3-642-39191-0. Disponível em: <http://dx.doi.org/10.1007/978-3-642-39191-0_7>. Citado 2 vezes nas páginas 7 e 8.

HAYFLICK, L. **How and why we become older (in Portuguese)**. Rio de Janeiro−RJ: Campus, 1996. ISBN 9788535200508. Citado 2 vezes nas páginas 7 e 8.

IBGE. **Instituto Brasileiro de Geografia e Estatística**. 2015. Disponível em: <http://censo2010.ibge.gov.br/resultados>. Acesso em: 20 set. 2019. Citado na página 7.

MALONE, L. A.; BASTIAN, A. J. Age-related forgetting in locomotor adaptation. **Neurobiology of Learning and Memory**, v. 128, p. 1 – 6, 2016. ISSN 1074-7427. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1074742715002051>. Citado na página 13.

Statista. **Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)**. 2019. Disponível em: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide>. Acesso em: 20 set. 2019. Citado na página 7.

TAIVALSAARI, A.; MIKKONEN, T. A roadmap to the programmable world: Software challenges in the iot era. **IEEE Software**, v. 34, n. 1, p. 72–80, Jan 2017. ISSN 0740-7459. Citado na página 7.

WHO. **World Health Organization**: Number of people over 60 years set to double by 2050: major societal changes required. 2015. Disponível em: <https://www.who.int/mediacentre/news/releases/2015/older-persons-day/en/>. Acesso em: 20 set. 2019. Citado na página 7.