**ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO**

INSTITUTO POLITÉCNICO DA GUARDA

# *ANDROID WEAR*: TRANSMISSÃO DE MENSAGENS ENTRE *SMARTPHONE* E *SMARTWATCH*

---

WORKSHOP

| | |
|---|---|
| **Curso** | **Mestrado em Computação Móvel** |
| **Unidade Curricular** | Seminário |
| **Ano Letivo** | 2016/2017 |
| **Docente** | Doutor Carlos Carreto |
| **Coordenador da área disciplinar** | Doutor José Carlos Fonseca |
| **Data** | 21/01/2017 |
| **Alunos** | Anabela Tavares Nº1011109 |

# Índice

# 1. Introdução

Ao longo dos anos, as tecnologias não pararam de evoluir, e a sua influência na vida humana não ficou despercebida. As novas tecnologias fazem já parte do dia-a-dia dos indivíduos, por isso, e de uma forma geral, pode-se verificar que torna a vida de qualquer um mais facilitada e proporciona muitas melhorias no seu quotidiano.

O *Android wear* foi uma destas tecnologias que vieram trazer algo novo ao quotidiano sendo que estes proporcionam que fiquemos conectados ao *smartphone* e receber notificações em tempo real. O sistema não foi desenvolvido para substituir as funções realizadas no *smartphone*, foi sim projetado para complementar e facilitar algumas tarefas, principalmente avisos e notificações rápidas e importantes. A interação com o utilizador é o essencial nestes dispositivos e, portanto é maioritariamente feita com gestos e comandos de voz, no entanto também podemos utilizar o *touch* para pequenos ajustes.

Este workshop vai abordar o *android wear* de uma forma simples, em que se obtenha os conhecimentos base para a configuração de um projeto para o *smartwatch*.

# 2. Plataforma *Android wear*

O *Android wear* é o sistema operacional da Google voltado para os *wearables*, ou dispositivos vestíveis, num termo em Português. Este tem uma interface completamente nova e traz informações para o utilizador baseadas na localização e nas ativadas diárias.

Diferentemente do que acontece com os *smartphone*s, que ficam maior parte do tempo no bolso, o *smartwatch* vai fazer parte do nosso visual, então o design é uma parte muito importante. Sendo assim, temos *smartwatch's* para todos os gostos, redondos ou quadrados, dando a oportunidade de o utilizador escolher o visual que preferir. A Figura 2 apresenta uns exemplos de visuais, o Motorola 360 de tela redonda e o *Sony Xperia Smart Devices* que apresenta tela quadrada.


*Figura 1 – Vários tipos de Smartwatch*

Então, mas como é que estes dispositivos se conectam com o *Smartphone*?

Fácil, através de uma comunicação sem fios que neste caso é o *Bluetooth* (*Bluetooth Low Energy* (BLE)). A partir do momento em que é efetuado o emparelhamento com o *smartphone*, o sistema começa a enviar automaticamente uma serie de mensagens, juntamente com parâmetros tais como comandos de voz e gestos para efetuar a configuração do *smartwatch*.

Quando a configuração estiver pronta entre o *smartphone* e *smartwatch*, as mensagens podem ser trocadas entre os dispositivos para acionar ações tanto no *smartphone* como no *smartwatch*. Através destas ações podemos ativar várias APIs (Figura 2) que são ativadas quando se trata de comunicar entre o *smartphone* e o *smartwatch*:

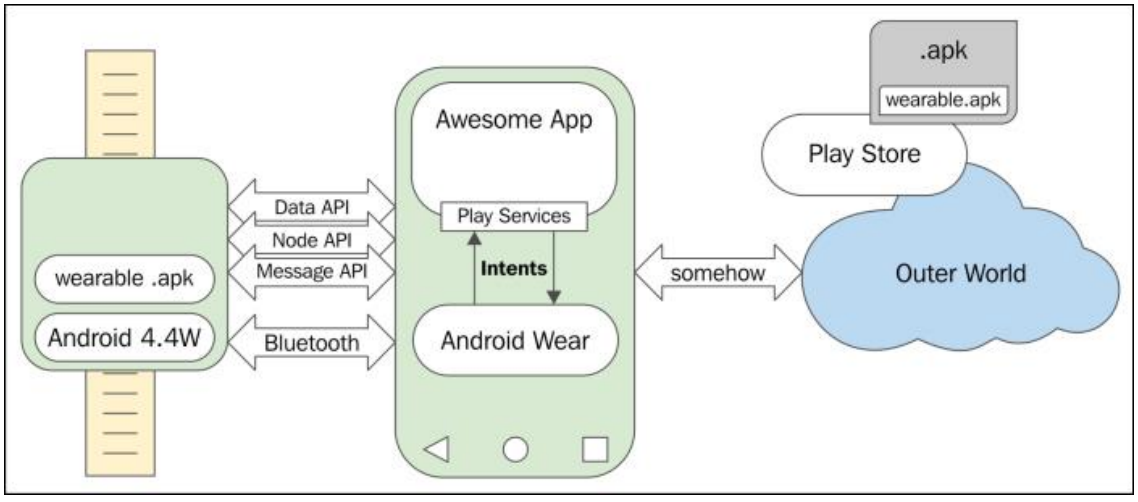


*Figura 2 – Arquitetura de um smartwatch.*

Cada nó (API) têm “uma responsabilidade”, podendo um nó lidar com várias funções. As várias API’s podem trabalhar ao mesmo, por exemplo, um nó pode trabalhar a da câmara do *smartphone*, enquanto outro nó pode acompanhar o GPS do *smartphone*.

# 3. Configuração do Ambiente de Desenvolvimento do *Android wear*

Como já referido antes, o *android wear* é um sistema bastante vasto na interação com o utilizador, podendo tornar a vida do mesmo simples e eficaz. Além de receber notificações, dar “ordens” ao *smartphone* podemos fazer mais uma série de coisas interessantes, tais como:

- Ver informações em um piscar de olhos;
- Fazer perguntas e realizar tarefas diárias;
- Viagens;
- Acompanhar o nosso exercício físico;
- Ouvir músicas e, controla-las.
- …

Estas “ordens” são as pré-definidas pelo o sistema operativo. No entanto, no *android wear* podem ser instaladas aplicações desenvolvidas por nós, assim como no *android*, basta apenas colocar “mãos à obra”. Sendo assim, o tema proposto é enviar mensagens do *smartphone* para o *smartwatch*, isto é, com o nosso *smartphone* vamos enviar mensagens e estas vão ser sincronizados com o *wear*able, ou seja, o *smartwatch* vai receber as mensagens.

## 3.1. Configuração do Ambiente

Para desenvolvermos os nosso aplicativos *wear* precisamos de um *smartphone* ou *tablet android* com a versão 4.3 ou superior e o *Google Play Services 5* ou superior instalado. Também devemos ter no *smartphone* o aplicativo “*Android wear*”, que permite a conexão com o *smartwatch*. A Figura 3 mostra este aplicativo no Google Play.

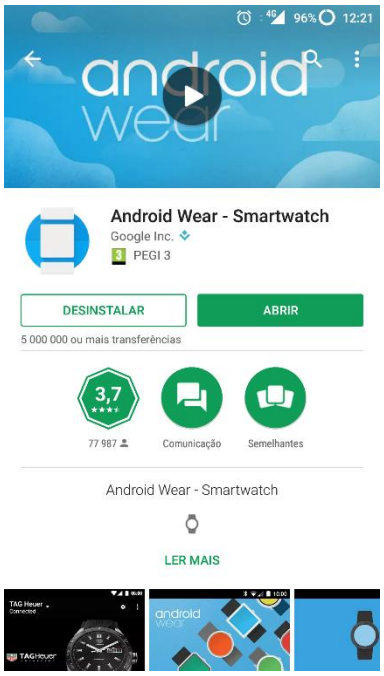


*Figura 3 – Aplicativo do Android wear no Google Play.*

Para realizar os testes é necessário ter um *smartwatch*, mas como ainda não é assim tao comum no quotidiano, faremos teste no emulado do *android Studio*. Para isso basta seguir os próximos passos.

1. Ter o aplicativo "*Android wear*" instalado no *smartphone*. (Figura 3)
2. Abrir o emulador do *android Studio*. (Figura 4)

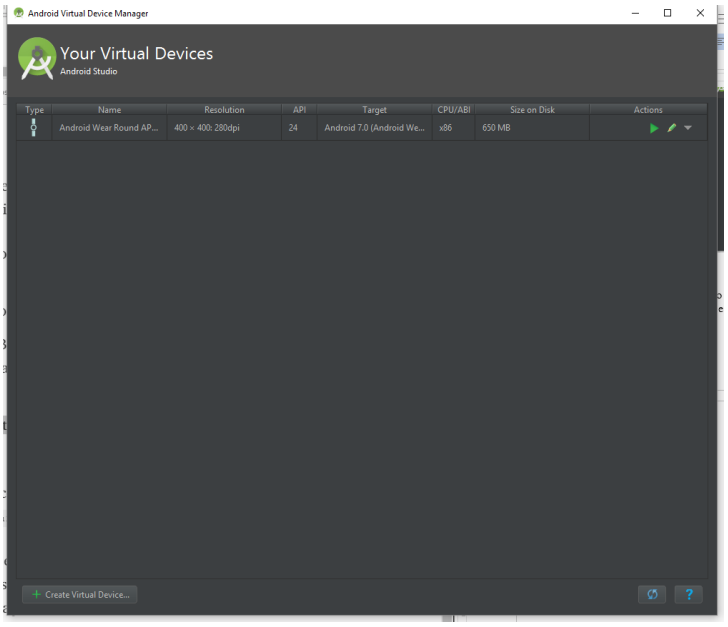


*Figura 4 – Emulador do Android Studio.*

3. Conectar o cabo USB no *smartphone*, abrir o terminal de comados do *android* Studio e digitar: **`adb -d forward tcp:5601 tcp:5601.`** (Figura 5)

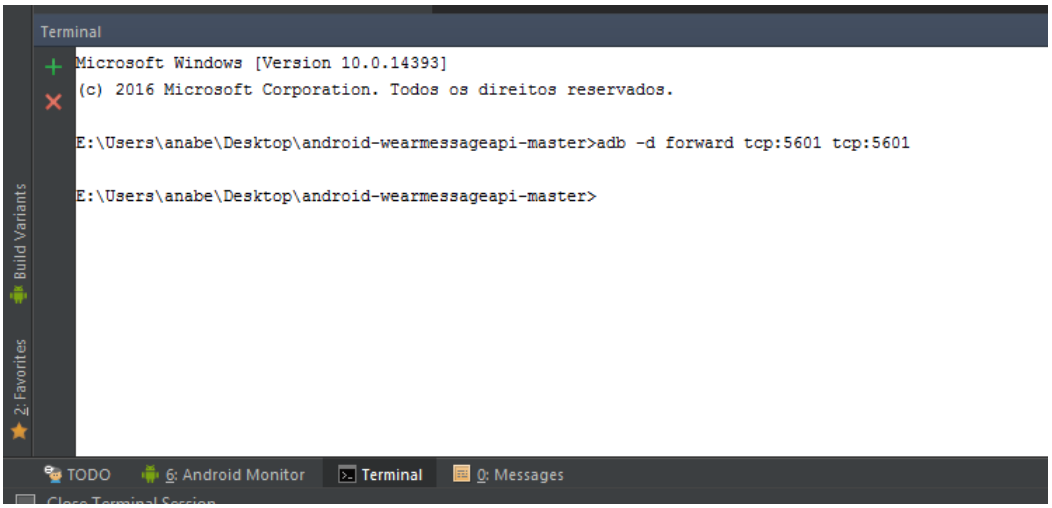


*Figura 5 – Terminal do Android Studio.*

4. Abrir o aplicativo *android wear* e selecione para se conectar com o emulador. (Figura 6)


*Figura 6 - Ligação ao Emulador*

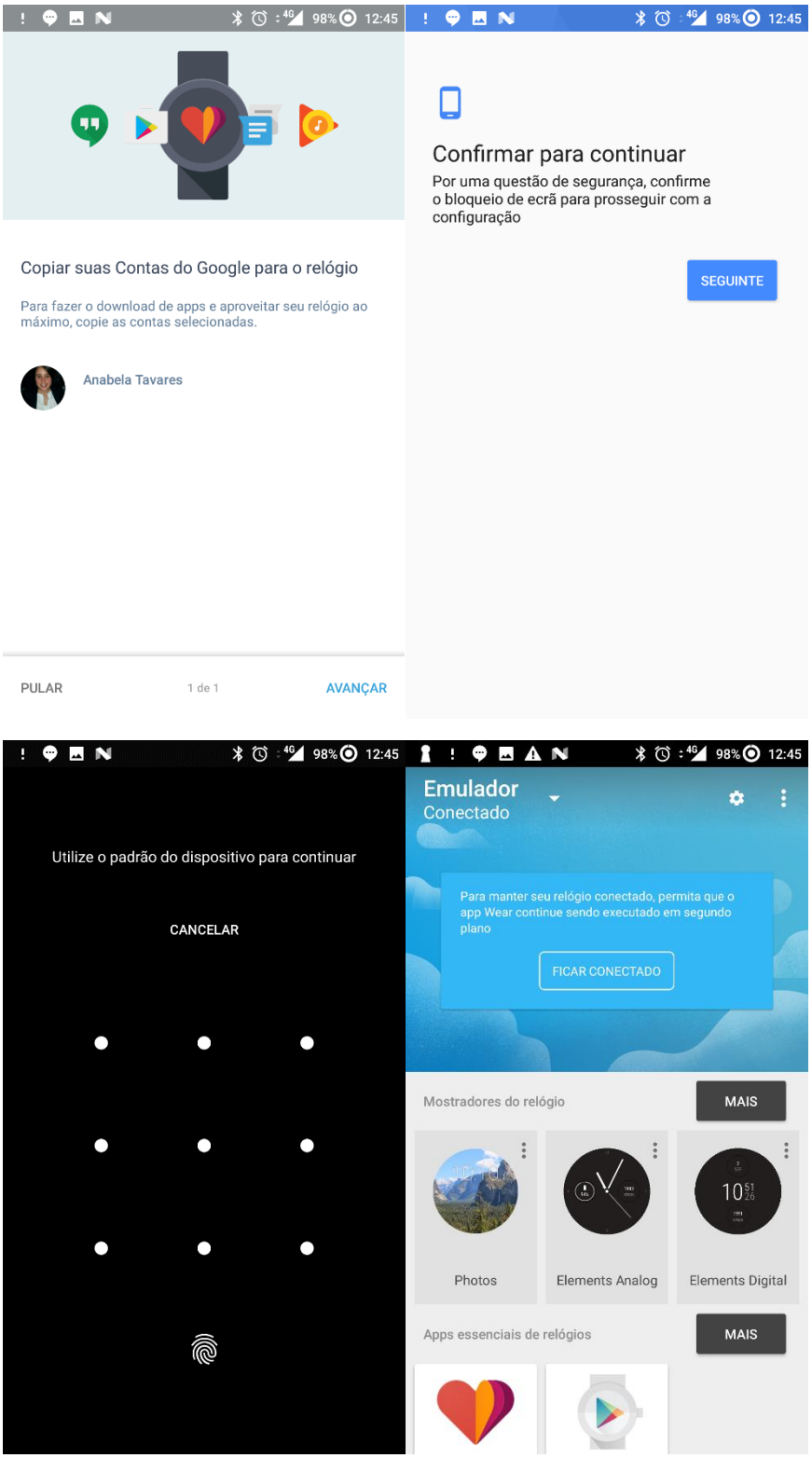


*Figura 7 – Passos para configurar o smartphone para ligação ao emulador.*

Obs: sempre que desconectarmos o *smartphone* da USB, temos que refazer o comando **`adb -d forward tcp:5601 tcp:5601.`**

## 4. Desenvolvimento no *Android wear*

Um aplicativo que é executado em um *smartwatch* normalmente utiliza algumas das funcionalidades do *smartphone* que está emparelhado, ou seja, vai ser necessário criar dois aplicativos *android* diferentes. Um que é executado no *wear*able e outro que é executado no *smartphone*. Estes dois aplicativos vão comunicar entre si através do Bluetooth.

*E estes dispositivos simplesmente comunicam entre o Bluetooth?*

Não, a google para os dispositivos *wear* apresenta *Wear*able Message API que fornece acesso à camada de dados entre os dois dispositivos. As mensagens enviadas através do *smartphone* e são disponibilizadas no *wear*able. O seguinte o diagrama exemplifica como e que uma simples mensagem é transferida do *smartphone* para o *wear*able (Figura 8).

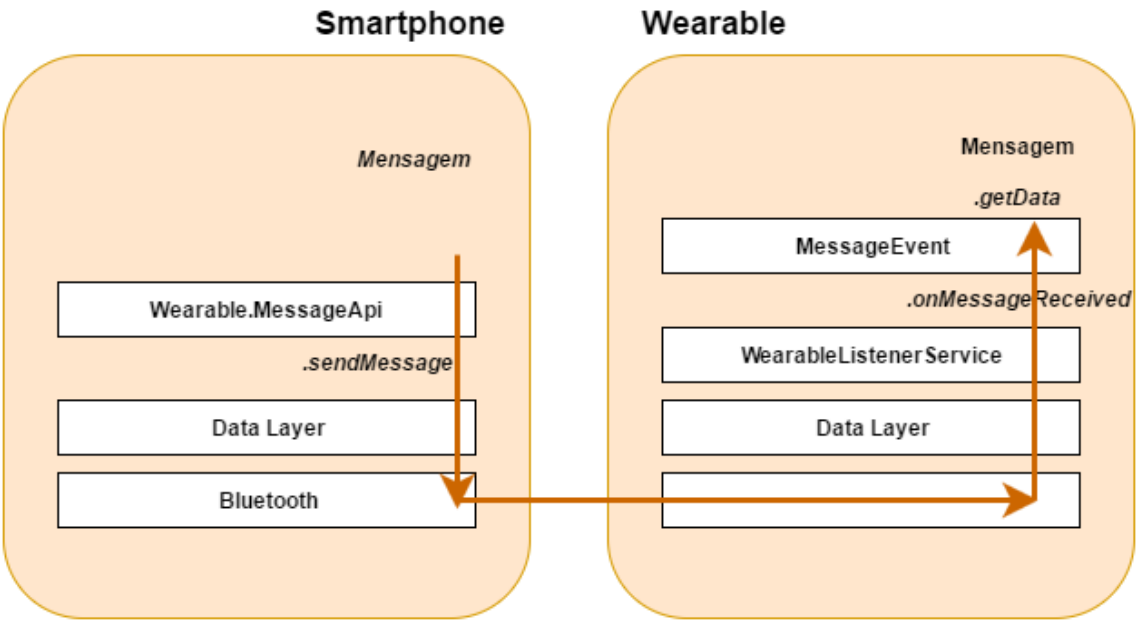


*Figura 8 - Diagrama de uma mensagem a ser transferida.*

Neste exemplo, o *smartphone* envia uma mensagem para o *wear*able usando o método sendMessage do *Wear*able.MessageApi. No *wear*able, um *Wear*ableListenerService monitoriza a camada de dados e retorna o valor do

onMessageReceived quando a mensagem chega. O Listener Service, em seguida, vai executar a tarefa que foi especificada no aplicativo.

## 4.1. Primeira aplicação para *Android wear*

1) Primeiramente criamos um projeto no *Android* Studio (Figura 9). O novo projeto deverá ter duas aplicações, uma para o *smartphone* e outra para o *wear*. Estas duas aplicações deverão ter o mesmo nome para que a camada de dados funcione (Figura 10).

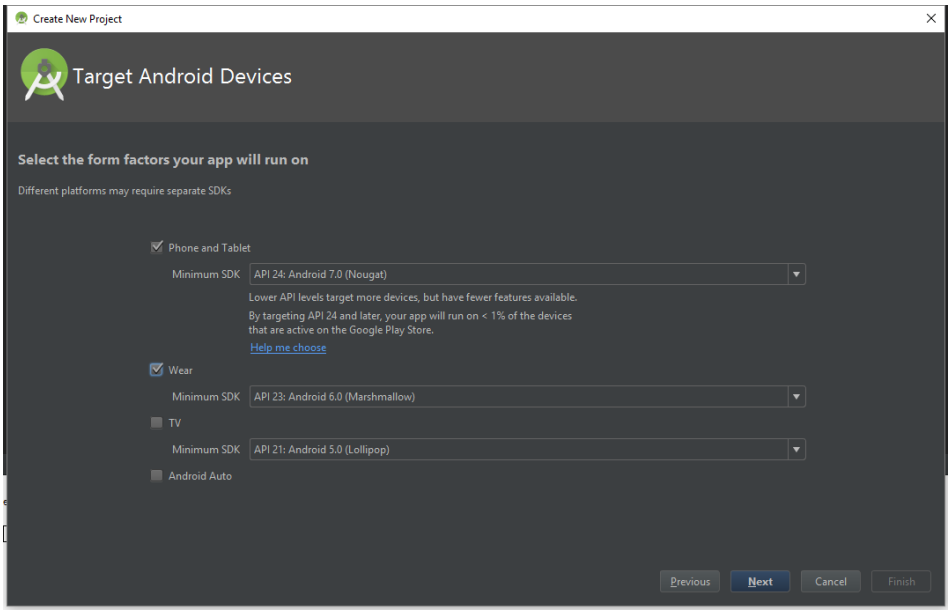


*Figura 9 – Criação de um novo projeto.*

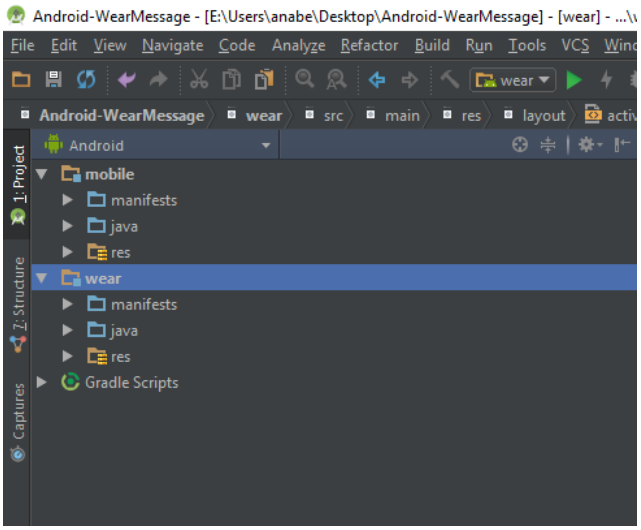


*Figura 10 – Mesmo projeto, duas aplicações.*

2) Adicionar Metadata para conseguir trabalhar com os serviços da google

```
<application>
...
  <meta-data android:name="com.google.android.gms.version"
   android:value="@integer/google_play_services_version" />
</application>
```

3) Criar um remetente para a mensagem

Para enviar uma menagem, temos que implementar na main activity do *smartphone* o *Google Play Services client* para conseguir conectar à *Wear*able API.

```
public class MainActivity extends Activity implements
GoogleApiClient.ConnectionCallbacks {

private static final String START_ACTIVITY = "/start_activity";
private static final String WEAR_MESSAGE_PATH = "/message";
private GoogleApiClient mApiClient;
private ArrayAdapter<String> mAdapter;

private ListView mListView;
private EditText mEditText;
private Button mSendButton;


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

            init();
            initGoogleApiClient();
        }

        private void initGoogleApiClient() {
            mApiClient = new GoogleApiClient.Builder( this )
                    .addApi( Wearable.API )
                    .build();

            mApiClient.connect();
        }
}
```

Adicionar métodos de retorno para a camada de dados.

```
public class MainActivity extends Activity implements
GoogleApiClient.ConnectionCallbacks {

        @Override
        public void onConnected(@Nullable Bundle bundle) {

        }
```

```
        @Override
        public void onConnectionSuspended(int i) {

        }
}
```

Definir uma classe que implementa um método que envia a nossa mensagem para todos os nós atualmente conectados à camada de dados.

```
public class MainActivity extends Activity implements
GoogleApiClient.ConnectionCallbacks {

        private void init() {
            mListView = (ListView) findViewById( R.id.list_view );
            mEditText = (EditText) findViewById( R.id.input );
            mSendButton = (Button) findViewById( R.id.btn_send );

            mAdapter = new ArrayAdapter<String>( this,
        android.R.layout.simple_list_item_1 );
            mListView.setAdapter( mAdapter );

            mSendButton.setOnClickListener( new
        View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    String text = mEditText.getText().toString();
                    if (!TextUtils.isEmpty(text)) {
                        mAdapter.add(text);
                        mAdapter.notifyDataSetChanged();

                        sendMessage(WEAR_MESSAGE_PATH, text);
                    }
                }
            });
        }


        private void sendMessage( final String path, final String
        text ) {
            new Thread( new Runnable() {
                @Override
                public void run() {
                    NodeApi.GetConnectedNodesResult nodes =
        Wearable.NodeApi.getConnectedNodes( mApiClient ).await();
                    for(Node node : nodes.getNodes()) {
                        MessageApi.SendMessageResult result =
        Wearable.MessageApi.sendMessage(
                                mApiClient, node.getId(), path,
        text.getBytes() ).await();
                    }

                    runOnUiThread( new Runnable() {
                        @Override
                        public void run() {
                            mEditText.setText( "" );
                        }
                    });
                }
            }).start();
}
```

4) Implementar um Listener para a mensagem

Para receber uma mensagem, temos que criar uma class com o nome "*Wear*MessageListenerService" e, estender à plataforma *Wear*ableListenerService para que o *smartwatch* consiga receber a mensagem.

```
public class WearMessageListenerService extends WearableListenerService {
    private static final String START_ACTIVITY = "/start_activity";

    @Override
    public void onMessageReceived(MessageEvent messageEvent) {
        if( messageEvent.getPath().equalsIgnoreCase( START_ACTIVITY ) ) {
            Intent intent = new Intent( this, MainActivity.class );
            intent.addFlags( Intent.FLAG_ACTIVITY_NEW_TASK );
            startActivity( intent );
        } else {
            super.onMessageReceived(messageEvent);
        }
    }
}
```

5) Adicionar a class *Wear*MessageListenerService ao ficheiro Manifest.

```
<service android:name=".WearMessageListenerService">
<intent-filter>
    <action android:name="com.google.android.gms.wearable.BIND_LISTENER" />
</intent-filter>
</service>
```

6) Mostrar as mensagens recebidas

Para mostrar as mensagens recebida no *wear*able vamos implementar na main activity a MessageApi.MessageListener e GoogleApiClient.ConnectionCallbacks para que consiga receber a mensagem sem erros.

```
public class MainActivity extends Activity implements
MessageApi.MessageListener, GoogleApiClient.ConnectionCallbacks {

    private static final String WEAR_MESSAGE_PATH = "/message";
    private GoogleApiClient mApiClient;
    private ArrayAdapter<String> mAdapter;
    private ListView mListView;
    private EditText editTextKeyb;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mListView = (ListView)findViewById(R.id.list);

        mAdapter = new ArrayAdapter<String>( this, R.layout.list_item );
        mListView.setAdapter( mAdapter );
        editTextKeyb = (EditText)findViewById(R.id.editTextCursor);

getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        initGoogleApiClient();
    }
```

```
    private void initGoogleApiClient() {
        mApiClient = new GoogleApiClient.Builder( this )
                .addApi( Wearable.API )
                .addConnectionCallbacks( this )
                .build();

        if( mApiClient != null && !( mApiClient.isConnected() ||
mApiClient.isConnecting() ) )
            mApiClient.connect();
    }

    @Override
    protected void onResume() {
        super.onResume();
        if( mApiClient != null && !( mApiClient.isConnected() ||
mApiClient.isConnecting() ) )
            mApiClient.connect();
    }

    @Override
    protected void onStart() {
        super.onStart();
    }

    @Override
    public void onMessageReceived( final MessageEvent messageEvent ) {
        runOnUiThread( new Runnable() {
            @Override
            public void run() {
                if( messageEvent.getPath().equalsIgnoreCase(
WEAR_MESSAGE_PATH ) ) {
                    mAdapter.add(new String(messageEvent.getData()));
                    mAdapter.notifyDataSetChanged();
                }
            }
        });
    }

    @Override
    public void onConnected(Bundle bundle) {
        Wearable.MessageApi.addListener( mApiClient, this );
    }

    @Override
    protected void onStop() {
        if ( mApiClient != null ) {
            Wearable.MessageApi.removeListener( mApiClient, this );
            if ( mApiClient.isConnected() ) {
                mApiClient.disconnect();
            }
        }
        super.onStop();
    }

    @Override
    protected void onDestroy() {
        if( mApiClient != null )
            mApiClient.unregisterConnectionCallbacks( this );
        super.onDestroy();
    }

    @Override
    public void onConnectionSuspended(int i) {

    }
}
```

## 5. Conclusão

O desenvolvimento de aplicações para dispositivos móveis através da plataforma *Android*, tem sido um pontapé de saída para muitos desenvolvedores e, sendo que há cada vez mais pessoas com *smartphone* torna-se em algo cada vez mais estimulante de fazer. O *android wear* é algo ainda "novo", mas com muito para explorar, o que ainda torna mais fascinante o mundo do *android*.

Com a realização deste workshop deu para ver o que uma "tecnologia" pequena e poderosa pode mudar no dia-a-dia do consumidor. Sendo que é uma API ainda em vasto desenvolvimento, poderemos esperar grandes feitos com esta tecnologia.