



UNIWERSYTET
MIKOŁAJA KOPERNIKA
W TORUNIU

Wydział Fizyki, Astronomii
i Informatyki Stosowanej

Tutorial programowania dla urządzeń ubieralnych

Marek Przybyłowski

12.02.2021



Spis treści

- [1. Tworzenie projektu w Android Studio](#)
- [2. Uruchamianie emulatora](#)
- [3. Parowanie telefonu z emulatorem urządzenia ubieralnego](#)
- [4. Definiowanie własnego layoutu](#)
- [5. Wysyłanie i odbieranie wiadomości za pomocą Wear](#)
- [6. Koniec](#)



1. Tworzenie projektu w Android Studio





1. Klikamy kolejno **File > New > New Project**.
2. Przechodzimy do zakładki Wear OS i wybieramy Blank Activity i klikamy **Next**.
3. Na następnym ekranie ustawiamy nazwę projektu – **Wearable App** i minimalną wersję SDK – **API 28 – Android 9.0 (Pie)** i zaznaczamy **Pair with my empty phone app** (utworzona zostanie aplikacja dla urządzeń ubieralnych i pustych dla smartfonów).
4. Klikamy **Finish** i czekamy, aż **Android Studio** utworzy projekt.

5. Przechodzimy do katalogu z utworzonym projektem (**C://.../WearableApp/**), a następnie do katalogu **app**.
6. Otwieramy plik **build.gradle** i modyfikujemy wersję kompilacyjną i docelową SDK na 28 w sekcji **dependencies**.

```
dependencies {  
    implementation 'androidx.wear:wear:1.0.0'  
    implementation 'com.google.android.support:wearable:2.8.1'  
    compileOnly 'com.google.android.wearable:wearable:2.8.1'  
}
```



7. Następnie przechodzimy do katalogu **src/main** i modyfikujemy plik **AndroidManifest.xml**. W tagu **<application>** znajduje się tag **<meta-data>**, w którym należy zmienić **android name** oraz **android value**.

```
<application>
```

```
...
```

```
  <meta-data
```

```
    android:name="com.google.android.wearable.standalone"
```

```
    android:value="true" />
```

```
...
```

```
</application>
```



8. W pliku **AndroidManifest.xml** czy tag **<user-feature>** przyjmuje następującą wartość:

```
<manifest>
```

```
...
```

```
  <uses-feature android:name="android.hardware.type.watch" />
```

```
...
```

```
</manifest>
```



2. Uruchamianie emulatora





1. W Android Studio wybieramy zakładkę **Tools-> AVD Manager**.
2. Następnie dodajemy emulator klikając **Create Virtual Device**. Przechodzimy do zakładki **Wear OS** i wybieramy **Android Wear Square**. Teraz wystarczy już tylko pobrać i uruchomić emulator.



3. Parowanie telefonu z emulatorem urządzenia ubieralnego



1. Na telefonie uruchamiamy **debugowanie usb - Ustawienia->Opcje programistyczne->Debugowanie USB**. Jeśli opcje programistycznie nie są włączone to aktywujemy w następujący sposób: **Ustawienia->Informacje o telefonie->Numer kompilacji (klikamy ok.7 razy)**.
2. Pobieramy aplikację **Wear OS app** na telefon.
3. W **Wear OS app** powinien wyświetlić się komunikat o dostępnych do parowania urządzeniach ubieralnych. Przechodzimy przez cały proces i na koniec klikamy **Pair with Emulator**.
4. Przechodzimy do **ustawień**, a następnie ustawień urządzenie i dodajemy konto Google. Na koniec, jeśli zostaniemy poproszeni o hasło blokady ekranu i hasło konta Google to podajemy je.

4. Definiowanie własnego layoutu





1. Aktualizujemy plik activity_main.xml.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_square" />
</LinearLayout>
```

Biblioteka **Wear UI** zawiera komponenty zarówno dla urządzeń z kwadratowymi (**BoxInsetLayout**) jak i okrągłymi ekranami (**Curved Layout**).

BoxInsetLayout – komponentu można używać do obu rodzajów ekranów przy czym uzyskiwany jest taki sam efekt.

Curve Layout – pionowa lista elementów dla urządzeń z okrągłymi ekranami.

Aby skompilować projekt w Android Studio należy upewnić się, że repozytorium Google jest zainstalowane w menedżerze **Android SDK**. Ponadto dodajemy następujące zależności w pliku **build.gradle**:

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:wear:26.0.0'  
}
```



<androidx.wear.widget.BoxInsetLayout

```
xmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:app="http://schemas.android.com/apk/res-auto"  
android:layout_height="match_parent"  
android:layout_width="match_parent"  
android:padding="15dp">
```

<FrameLayout

```
android:layout_width="match_parent"  
android:layout_height="match_parent"  
android:padding="5dp"  
app:layout_boxedEdges="all">
```

<TextView

```
android:gravity="center"  
android:layout_height="wrap_content"  
android:layout_width="match_parent"  
android:text="@string/sometext"  
android:textColor="@color/black" />
```

<ImageButton

```
android:background="@null"  
android:layout_gravity="bottom|left"  
android:layout_height="50dp"  
android:layout_width="50dp"  
android:src="@drawable/ok" />
```

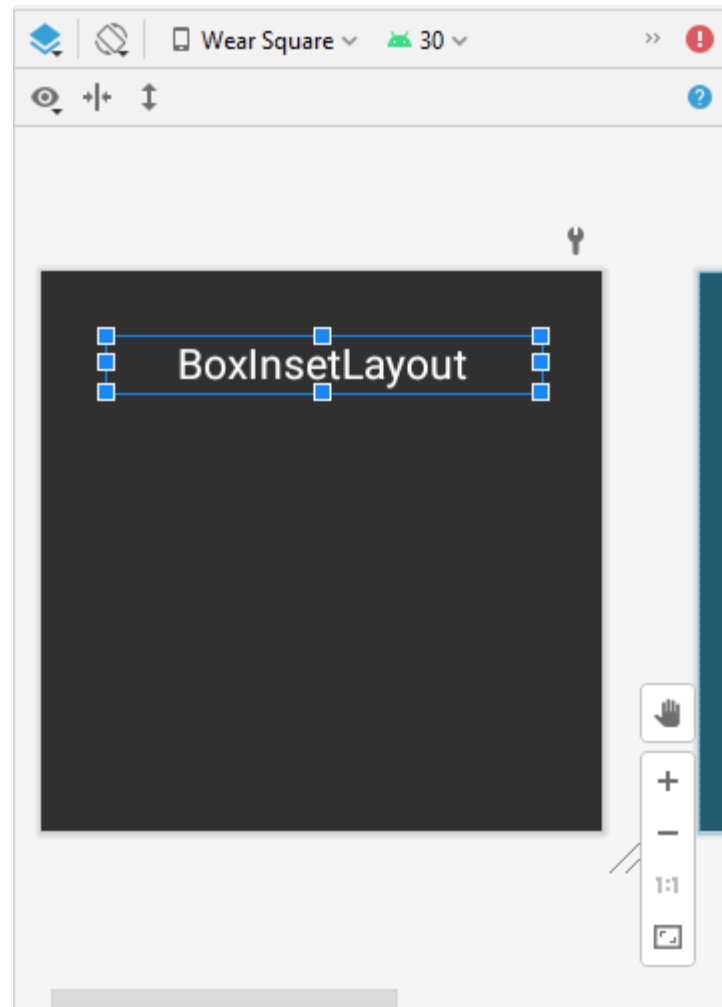
<ImageButton

```
android:background="@null"  
android:layout_gravity="bottom|right"  
android:layout_height="50dp"  
android:layout_width="50dp"  
android:src="@drawable/cancel" />
```

```
</FrameLayout>
```

```
</androidx.wear.widget.BoxInsetLayout>
```

Przykładowe użycie
komponentu **BoxInsetLayout**.





Tworzenie layoutu jednocześnie dla urządzeń z okrągłymi i kwadratowymi ekranami umożliwia modyfikacja plików **dimens.xml** i **dimens.xml** z katalogu **values** i **values-round**. Poniżej znajduje się kod z pliku **dimens.xml**, który umieszcza w danym miejscu nagłówki oraz listę.

```
<dimen name="header_start_padding">36dp</dimen>  
<dimen name="header_end_padding">22dp</dimen>  
<dimen name="list_start_padding">36dp</dimen>  
<dimen name="list_end_padding">22dp</dimen>
```



Analogiczne dla ekranu okrągłego. W tym przypadku należy zmienić widok z **Android** na **Project**.

```
<dimen name="header_start_padding">36dp</dimen>  
<dimen name="header_end_padding">22dp</dimen>  
<dimen name="list_start_padding">36dp</dimen>  
<dimen name="list_end_padding">22dp</dimen>
```

Teraz utworzymy przykładowy layout dalej modyfikując plik **activity_main.xml**:

```
<FrameLayout
...>
<androidx.wear.widget.RoundedDrawable
    android:id="@+id/androidbtn"
    android:src="@drawable/ic_android"
.../>
<ImageButton
    android:id="@+id/lovebtn"
    android:src="/sources/android.png"
    android:paddingTop="5dp"
    android:paddingBottom="5dp"
    android:layout_gravity="bottom"
.../>
</FrameLayout>
```



Część paska została ucięta, dlatego użyjemy atrybutu **fitsSystemWindows**, aby był widoczny cały pasek na dole ekranu.

```
<ImageButton  
  android:id="@+id/lovebtn"  
  android:src="@drawable/ic_favourite"  
  android:paddingTop="5dp"  
  android:paddingBottom="5dp"  
  android:fitsSystemWindows="true"  
  .../>
```



Górne i dolne wartości **padding** są nadpisywane, aby wszystko pasowało do okna systemowego, aby tego uniknąć należy skorzystać z **InsetDrawable**.

```
<inset  
  xmlns:android="http://schemas.android.com/apk/res/android"  
  android:drawable="@drawable/ic_favourite"  
  android:insetTop="5dp"  
  android:insetBottom="5dp" />
```

Usuwamy **padding** z **activity_main.xml**.

```
<ImageButton  
  android:id="@+id/lovebtn"  
  android:src="@drawable/inset_favourite"  
  
  android:fitsSystemWindows="true"  
  .../>
```



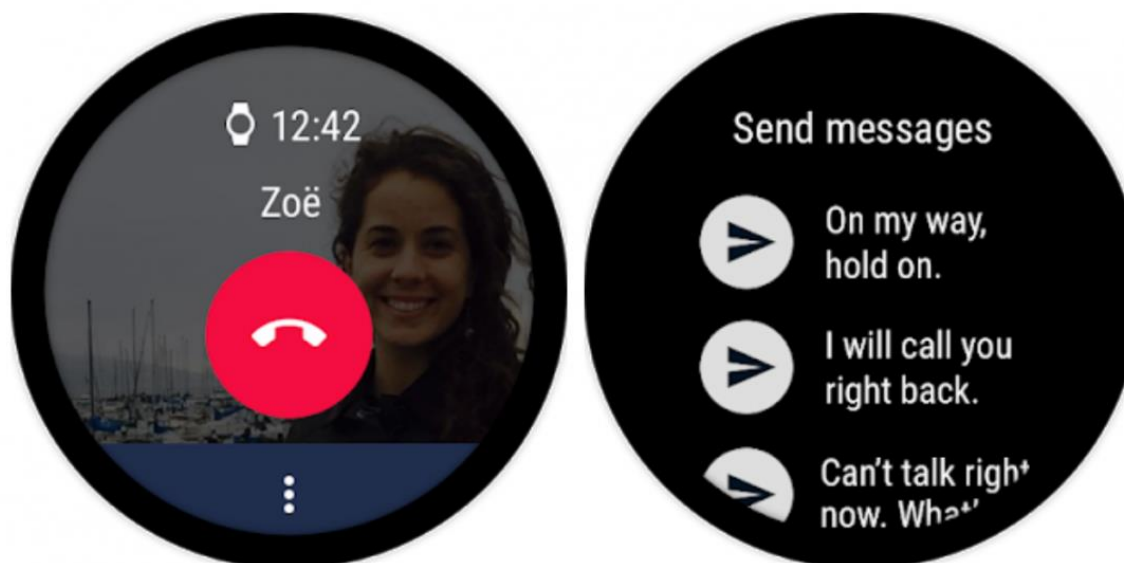


Dodanym paskiem można również zarządzać z poziomu klasy
MainActivity.java:

```
private int chinSize;
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // find the outermost element
    final View container = findViewById(R.id.outer_container);
    // attach a View.OnApplyWindowInsetsListener
    container.setOnApplyWindowInsetsListener(new View.OnApplyWindowInsetsListener()
    {
        @Override
        public WindowInsets onApplyWindowInsets(View v, WindowInsets insets) {
            chinSize = insets.getSystemWindowInsetBottom();
            // The following line is important for inner elements which react to insets
            v.onApplyWindowInsets(insets);
            return insets;
        }
    });
}
```



5. Wysyłanie i odbieranie wiadomości za pomocą Wear





1. Tworzymy projekt tak jak [slajdzie 4](#).

Do wysyłania i odbierania wiadomości użyjemy aktywności **WearableListenerService** i komponentu **ListView** do wyświetlania przez **MessageApi.MessageListener**.

2. Zaczynamy od aplikacji dla telefonów (w widoku *Projektu* katalog mobile). W AndroidManifest.xml zamieniamy tag **<meta-data>**:

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


3. Przechodzimy do klasy **MainActivity**. Pierwszą rzeczą, którą musimy zrobić, to połączenie **GoogleApiClient** i zaimplementowanie interfejsu **GoogleApiClient.ConnectionCallbacks**:

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

4. Kiedy **GoogleApiClient** zakończy połączenie, zostanie wywołana metoda **onConnected**. W takim przypadku wyślemy wiadomość za pośrednictwem interfejsu **Message API** do usługi **WearableListenerService** (która zostanie zaimplementowana w dalszych krokach) działającej na urządzeniu Wear, aby rozpocząć naszą aktywność związaną z zegarkiem.

```
@Override  
  
public void onConnected(Bundle bundle) {  
  
    sendMessage( START_ACTIVITY, "" );  
  
}
```

5. **START_ACTIVITY** to ciąg znaków, oznaczający ścieżkę wiadomości

```
private static final String START_ACTIVITY = "/start_activity";
```

6. Oprócz naszej początkowej wiadomości, chcemy również, aby wiadomość była wysyłana po naciśnięciu naszego przycisku „**wyślij**”. Odbywa się to za pomocą prostego **OnClickListener**, który używa innej ścieżki **value / message** i wysyła tekst obecny w **EditText** na dole ekranu.

```
1. mSendButton.setOnClickListener( new View.OnClickListener() {  
2.     @Override  
3.     public void onClick( View view ) {  
4.         String text = mEditText.getText().toString();  
5.         if ( !TextUtils.isEmpty( text ) ) {  
6.             mAdapter.add( text );  
7.             mAdapter.notifyDataSetChanged();  
8.             sendMessage( WEAR_MESSAGE_PATH, text );  
9.         }  
10.    }  
11. }  
12. });
```

7. Następnie implementujemy metodę **sendMessage()**. Aby wysłać wiadomość do urządzenia ubieralnego, musimy najpierw skorzystać z **Node API**, aby uzyskać listę węzłów podłączonych do urządzenia. Gdy mamy tę listę, wysyłamy wiadomość do każdego węzła za pomocą **MessageAPI** z odniesieniami do **GoogleApiClient**, identyfikatorem węzła, ścieżką używaną do określenia typu wysyłanej wiadomości oraz ładunkiem wiadomości w postaci tablicy bajtów. Zwracany jest **MessageApi.SendMessageResult**, którego można użyć do określenia, czy komunikat został pomyślnie wysłany do bieżącego węzła. Po wysłaniu wiadomości czyścimy widok **EditText**, aby umożliwić użytkownikowi wprowadzenie większej ilości tekstu.



```
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();
}
```



8. W metodzie **onDestroy()**, musimy odłączyć się od **GoogleApiClient**.

```
@Override  
  
protected void onDestroy() {  
  
    super.onDestroy();  
  
    mApiClient.disconnect();  
  
}
```



9. Klasa **MainActivity** urządzenia mobilnego jest już gotowa. Przechodzimy do katalogu **wear** i otwieramy **MainActivity.java**. Teraz musimy utworzyć w osobnym pliku klasę **WearableListenerService**, czyli usługę implementującą trzy interfejsy komunikacyjne dla urządzenia ubieralnego. Dla naszych celów musimy tylko przeciążamy metodę **onMessageReceived()**. W tej metodzie sprawdzamy ścieżkę, która została wysłana przez **MessageApi**, aby określić typ wiadomości.



```
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 );

    }

}
```




10. Gdy skończymy wdrażać naszą usługę, musimy dodać ją do pliku **AndroidManifest.xml** zużycia.

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

11. Jak wspomniano wcześniej, istnieją dwa sposoby otrzymania wiadomości na Android Wear. Drugim sposobem jest zaimplementowanie **MessageApi.MessageListener** w naszym Wear **MainActivity**. Dodajemy metodę **onConnected()**.

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

13. Następnie po prostu przeciążamy metodę **onMessageReceived()** z interfejsu **MessageListener**.

```
@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();  
  
            }  
  
        }  
  
    } );  
  
}
```



Dziękuję za uwagę!