

Použití vývoj aplikací pro Motorola Moto 360

Bc. Jakub Hromada

Diplomová práce
2015



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2014/2015

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jakub Hromada**

Osobní číslo: **A13542**

Studijní program: **N3902 Inženýrská informatika**

Studijní obor: **Informační technologie**

Forma studia: **kombinovaná**

Téma práce: **Použití a vývoj aplikací pro Motorola Moto 360**

Téma anglicky: **The Use and Development of Applications for Motorola Moto 360**

Zásady pro vypracování:

1. Vytvořte literární rešerši na téma tvorby aplikací pro mobilní zařízení Motorola Moto 360 se zaměřením se na operační systém Android
2. Navrhněte vzorovou aplikaci pro Moto 360
3. Naprogramujte aplikaci a otestujte ji
4. Udělejte rozbor časové náročnosti pro nasazení na další smartwatch zařízení
5. Popište programování v Android pro toto zařízení a srovnejte ho s programováním v ostatních programovacích jazycích

Rozsah diplomové práce:
Rozsah příloh:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. RUIZ, David Cuartielles. **Professional android wearables**. Indianapolis: Wrox, February 17, 2015, pages cm. ISBN 11-189-8685-7.
2. SMITH, Dave a Jeff FRIESEN. **Android Recipes**, 3rd Edition. New York City: Apress, 2014. ISBN 978-1-4302-6322-7.
3. SCHWARZ, Ronan, Phil DUTSON, James STEELE a Nelson TO. **The Android Developer's Cookbook**, 2nd Edition. Boston: Addison-Wesley, 2013. ISBN 978-0-321-89753-4.
4. HELLMAN, Erik. **Android programming: pushing the limits**. Chichester, West Sussex: Wiley, 2014. ISBN 978-111-8717-370.
5. CALVO, Andres. **Beginning Android Wearables**. New York City: Apress, 2015. ISBN 978-1-4842-0518-1.
6. ZAPATA, Belen Cruz. **Android Studio application development: create visually appealing applications using the new IntelliJ IDE Android Studio**. New Edition. Birmingham, UK: Packt Pub, 2013. ISBN 978-178-3285-273.
7. MACLEAN, Satya Komatineni and Dave. **Expert Android**. 2013. vyd. New York: Apress, 2013. ISBN 978-143-0249-504.
8. JACKSON, Wallace. **Pro Android UI**. New edition. New York City: Apress, 2014, xxvi, 552 pages. ISBN 14-302-4986-2.

Vedoucí diplomové práce: **Ing. Milan Navrátil, Ph.D.**
Ústav elektroniky a měření
Datum zadání diplomové práce: **6. února 2015**
Termín odevzdání diplomové práce: **15. května 2015**

Ve Zlíně dne 6. února 2015




doc. Mgr. Milan Adámek, Ph.D.
Adámek


doc. Mgr. Roman Jašek, Ph.D.
Jašek

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

ABSTRAKT

Cílem diplomové práce je zmapování tvorby aplikací pro mobilní zařízení Motorola Moto 360 a obecně SmartWatch, za pomoci android API a Google API a přidruženého pomocného zařízení. Další část popisuje časovou náročnost pro nasazení na další SmartWatch zařízení a možností jiných technologiích pro daný vývoj SmartWatch. Praktická část prezentuje vzorovou aplikaci, která demonstruje použití dané technologie

Klíčová slova: Android, Java, Wear

ABSTRACT

The aim of this paper is to describe the development of applications for mobile devices Motorola Moto 360 and generally SmartWatch, using Android API and the Google API and related auxiliary equipment. The next section describes the time required to deploy other SmartWatch devices and the availability of other technologies for the development SmartWatch. The practical part is presenting a sample project, to show how to use the technology

Keywords: Android, Java, Wear

Děkuji za podporu své rodině, která mě podporovala při mém studiu na Univerzitě Tomáše Bati. Děkuji svému vedoucímu diplomové práce Ing. Milanu Navrátilovi, Ph.D.

OBSAH

ÚVOD	9
I TEORETICKÁ ČÁST	10
1 TVORBA APLIKACI PRO CHYTRÉ HODINKY	11
1.1 ÚVOD	11
1.2 MOTOROLA MOTO 360.....	11
1.3 POŽADAVKY PRO VÝVOJ CHYTRÝCH HODINEK	11
1.4 POPIS APLIKACÍ BĚŽÍCÍCH NA CHYTRÝCH HODINKÁCH	11
1.5 NASTAVENÍ ANDROID HODINEK PRO VÝVOJ	12
1.6 VYTVOŘENÍ PROJEKTU	14
1.7 TVOŘENÍ VLASTNÍHO ROZLOŽENÍ	17
1.7.1 Tvoření vlastních notifikací	17
1.7.2 Tvoření vlastních kontejnerů za pomoci UI knihovny.....	18
1.7.3 Tvoření vlastního kontejneru zvláště pro kulaté a čtvercové android hodinky.....	20
1.8 POSÍLÁNÍ A SYNCHRONIZACE DATA.....	21
1.8.1 Požadavky	22
1.8.2 Komponenty	22
1.8.2.1 Přístup na datovou vrstvu android wear	23
1.8.2.2 Synchronizace datových položek.....	24
1.8.2.3 Synchronizace pomocí Data mapy	24
1.8.2.4 Naslouchání změn v datech	26
1.8.2.5 Zasílání velkých dat	28
1.8.2.6 Posílání a přijímání zpráv	30
II PRAKTICKÁ ČÁST	33
2 VÝVOJOVÁ APLIKACE	34
2.1 STRUKTURA PROJEKTU	34
2.2 DATACORE	35
2.2.1 Třídy	35
2.2.2 Volání HTTP metod	36
2.2.3 Servisy	41
2.3 MOBILE	44
2.3.1 Synchronizace dat	44
2.3.2 Zasílání zpráv	47
2.4 SHARED CORE	48
2.4.1 Třída Constant	48
2.4.2 Třída Utils	48
2.5 WEAR.....	49
2.5.1 Vrácení dat z googleAPI	49
2.5.1.1 AsyncTask	49
2.5.1.2 Naslouchání dat z WearableListenerService	51
2.5.2 Notifikace	52
ZÁVĚR	54
SEZNAM POUŽITÉ LITERATURY	55

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	56
SEZNAM OBRÁZKŮ	57
SEZNAM PŘÍLOH.....	58

ÚVOD

Náplní mé diplomové práce bylo zmapování možností vývoje pro zařízení Motorola Moto 360 a obecně chytrých hodinek za pomoci android api a google api.

Byly popsány jednotlivé body, které jsou potřebné pro spuštění vývojové aplikace na chytrých hodinkách, dále vývojové api, které obsahuje prostředky pro oznamování důležitých zpráv jak na chytrých hodinkách, tak na spárovaném zařízení, vlastní kontejnery pro chytré hodinky, které usnadní vývoj aplikace, možností hlasového vstupu, synchronizace dat, posílání a přijímání zpráv z chytrých hodinek a spárovaného zařízení. Dále jsou popsány možnosti ladění dané aplikace na chytrých hodinkách a spárovaného zařízení, vystavování dané výsledné aplikace.

Vývojové prostředí bylo vybráno android studio, kvůli lepší integritě programových prostředků pro chytré hodinky a taky kvůli lepším programovým nástrojům. Jazyk, ve kterém byl psán výsledný zdrojový kód, byl zvolen java.

Vzorová aplikace, která byla naprogramovaná, využívá synchronizaci dat a zaslání zpráv za pomoci google play services, notifikací, běžících procesů v pozadí, programové komponenty a kontejnery komponent poskytnutých googlem. Dále daná vzorová aplikace využívá službu portálu UTB, aby aplikace měla dynamickou datovou vrstvu.

I. TEORETICKÁ ČÁST

1 TVORBA APLIKACI PRO CHYTRÉ HODINKY

1.1 Úvod

Tvorba aplikací pro chytré hodinky využívá, již vytvořenou filozofii tvorby aplikací pro mobilní telefony, využívá se v ní tvoření kontejnerů daných obrazovek za pomoci xml, oživení obrazovek za pomoci aktivit, rozdělení obrazů pomocí fragmentů.

1.2 Motorola Moto 360

Dané zařízení disponuje operačním systémem android 5, nemá usb port, čili komunikace se spárovaným zařízením je za pomoci bluetooth technologie. Minimální operační systém běžící na spárovaném zařízením musí být 4.3 .

1.3 Požadavky pro vývoj chytrých hodinek

Podle oficiální stránek androidu je doporučován vývoj chytrých hodinek v android studiu, kvůli snadnějšímu vystavování, nastavování projektů, a importů knihoven. Minimální verze android studia potřebná pro vývoj je 0.8 nebo novější, gradle 0.12 nebo novější, SDK tools 23.0.0 nebo novější, SDK Android api 4.4.W2 (API 20) nebo novější, dané balíky jsou součástí sdk manageru.[1][2]

1.4 Popis aplikací běžících na chytrých hodinkách

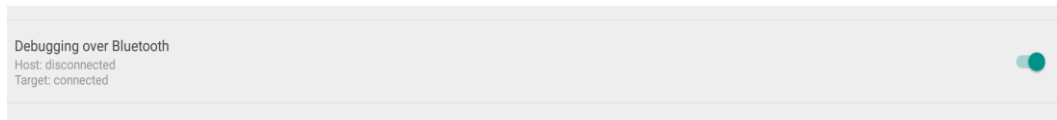
Dané aplikace běží přímo na chytrých hodinkách, mohou přistupovat na standardní hardwarové prostředky jako sensory a GPU. Aplikace běžících na hodinkách v porovnání. Pokud daná aplikace běží v popředí a uživatel nevyužívá danou aplikaci, tak se zařízení uspí. Po probuzení se zobrazí domovská obrazovka, místo aplikace, která byla v popředí. Dané aplikace jsou relativně malé co do velikosti využívání a funkcionality v porovnání s aplikacemi na mobilních zařízeních. Obsah daných aplikací by měl tvořit jen malou podmnožinu informací korespondující s spárovaných zařízením. Obecně aplikace, které běží na chytrých hodinkách, by měli přenechávat operační logiku spárovanému zařízením, čili operace jako dotazování na internetové služby, nebo těžké operace jako složité výpočty, poté dané výsledky by se měly odeslat dané aplikaci na smartwatch. Aplikace na chytrých hodinkách uživatel přímo neinstaluje, ale daná aplikace je součástí aplikace na mobilním zařízením, čili se automaticky nainstaluje spolu s ní. To neplatí pro vývoj dané aplikace, ta se instaluje zvlášť na chytré hodinky.[2][3]

1.5 Nastavení android hodinek pro vývoj

Pro vývoj je potřeba nastavit[3][4]:

1. Instalace aplikace android wear, která je dostupná na google store, na mobilní zařízení
2. Provedeme jednotlivé kroky, podle android wearu, na spárování android hodinek.
3. Necháme otevřenou aplikaci na mobilním zařízení
4. Musíme zapnout ladění na android hodinkách
 - a. Přejdeme na nastavení (settings) v hodinkách a na položku o hodinkách (about)
 - b. Klikneme sedmkrát na build number
 - c. Potáhneme z pravá do leva pro navrácení zpět
 - d. Úplně dolů se objevila nová možnost developer options, klikneme
 - e. Najdeme položku ADB debugging a zapneme
5. Připojíme zařízení pomocí USB, a můžeme instalovat přímo vývojové aplikace na android hodinky
6. Pokud dané android hodinky nemají USB port jako Motorola Moto 360, která má kontakt pouze přes bluetooth je potřeba dodatečného nastavení
 - a. Zapneme ladění v daném mobilním zařízení přes nastavení a možnosti vývojáře
 - i. Pokud mobilní zařízení nebo tablet nemá danou možnost, přejdeme na možnost "o telefonu", nebo "o tabletu". Po klikáme sedmkrát na číslo sestavení, poté by se mělo objevit možnost "možnosti vývojáře" v nastavení
 - b. Zapneme ladění na android hodinkách, přejdeme do nastavení a na „možnosti vývojáře“ a zapneme ladění přes bluetooth
 - i. Pokud mobilní zařízení nebo tablet nemá danou možnost, přejdeme na možnost "o telefonu", nebo "o tabletu". Po klikáme sedmkrát na číslo sestavení, poté by se mělo objevit možnost "možnosti vývojáře" v nastavení

- c. Nastavíme připojení android hodinky – mobilní zařízení. Na mobilním zařízení otevřeme aplikaci android wear, nahoře vpravo klikneme na menu a vybereme nastavení
- d. Zapneme ladění přes bluetooth



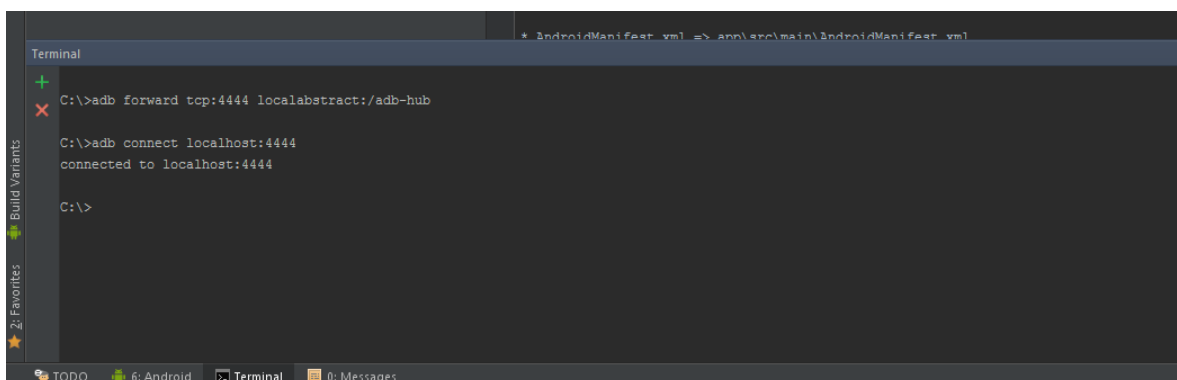
Obr. 1 Výsledný stav po zapnutí ladění přes bluetooth

- e. Připojíme mobilní zařízení do počítače přes USB a spustíme v konzoli následující příkaz :

```
adb forward tcp : 4444 localabstract:/adb-hub
```

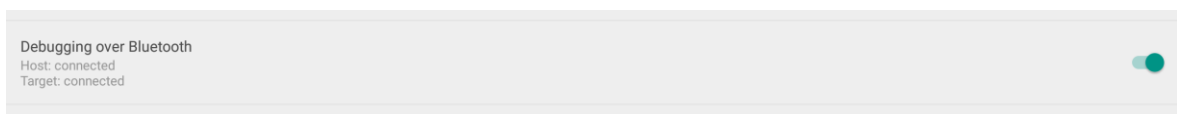
```
adb connect localhost:4444
```

Následující příkaz můžeme spustit v klasickém příkazovém řádku podle operačního systému nebo přímo v android studiu



Obr. 2 Spuštění příkazu na propojení mobilního zařízení a android hodinek

Port můžeme použít libovolný, ale musíme mít k němu přístup. Pokud daný příkaz adb nejde spustit, je potřeba specifikovat přímou cestu k němu. Nachází se v sdk androidu : *Android\sdk\platform-tools*. Umístění je závislé na operačním systémem.

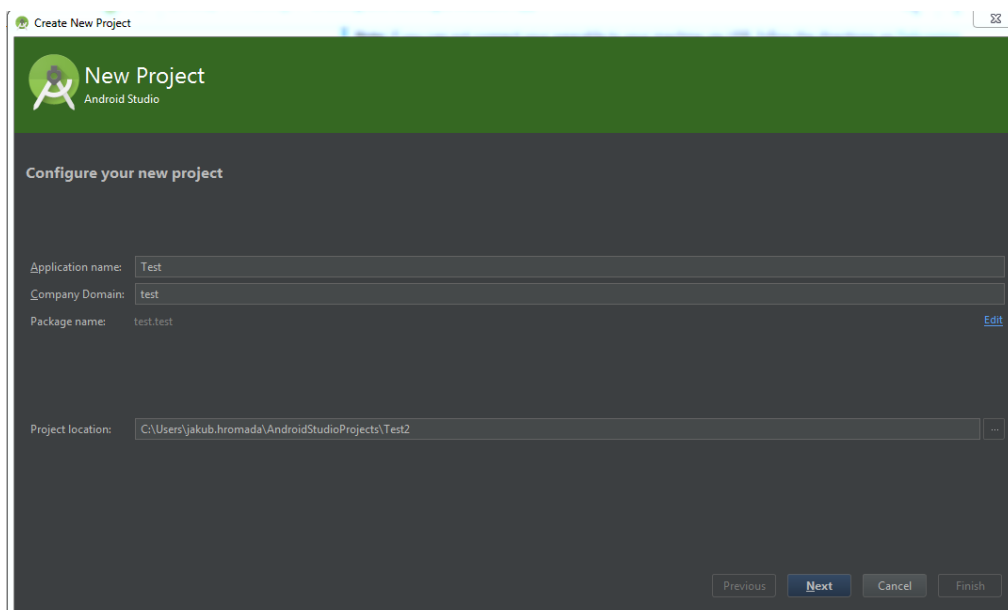


Obr. 3 Výsledné propojení mobilního zařízení a zařízení android hodinky

1.6 Vytvoření projektu

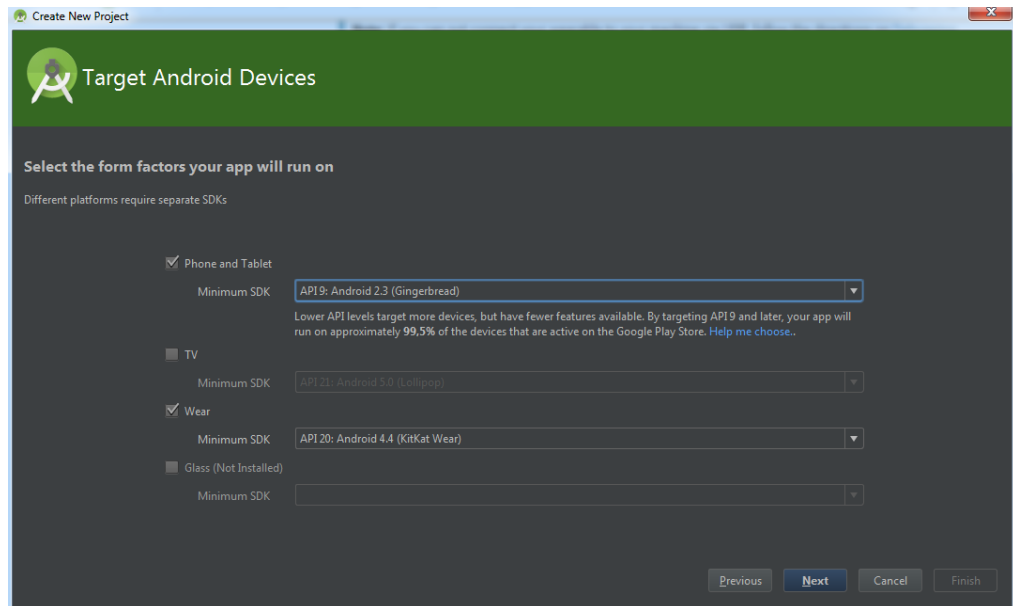
Začít s vývojem je potřeba vytvořit projekt, který obsahuje modul pro android hodinky a aplikační modul který je nahráván na mobilní zařízení. V android studiu klikneme na *File – New Project* a projdeme pomocníkem pro vytvoření projektu[6][7].

1. V konfiguračním okně vašeho okna, napíšeme jméno aplikace a jméno balíku



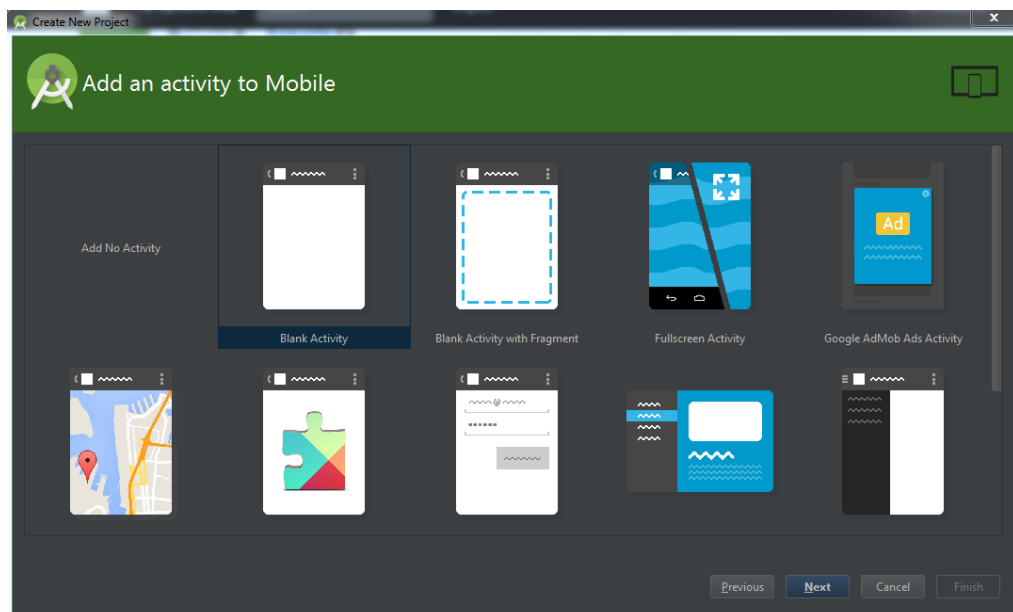
Obr. 4 Okno nového projektu

2. V dalším okně vybereme API, který bude projekt využívat pro vývoj android wearu je potřeba minimálně API 20 (Android 4.4 Kitkat). Pro výběr API pro mobilní zařízení, záleží co naše aplikace bude pokrývat, pokud to má být aplikace jenom pro komunikaci s hodinkami tak zvolíme API minimálně 18, poběží to jen na mobilních zařízeních které podporují komunikaci s android wear, nebo pokud to má být aplikace která bude mít význam i na starších zařízeních tak zvolíme API 9, jen musíme mít na paměti ,že v API 9 chybí spousta funkcí a musí se různě obcházet, oproti API 14, tudíž vývoj je složitější



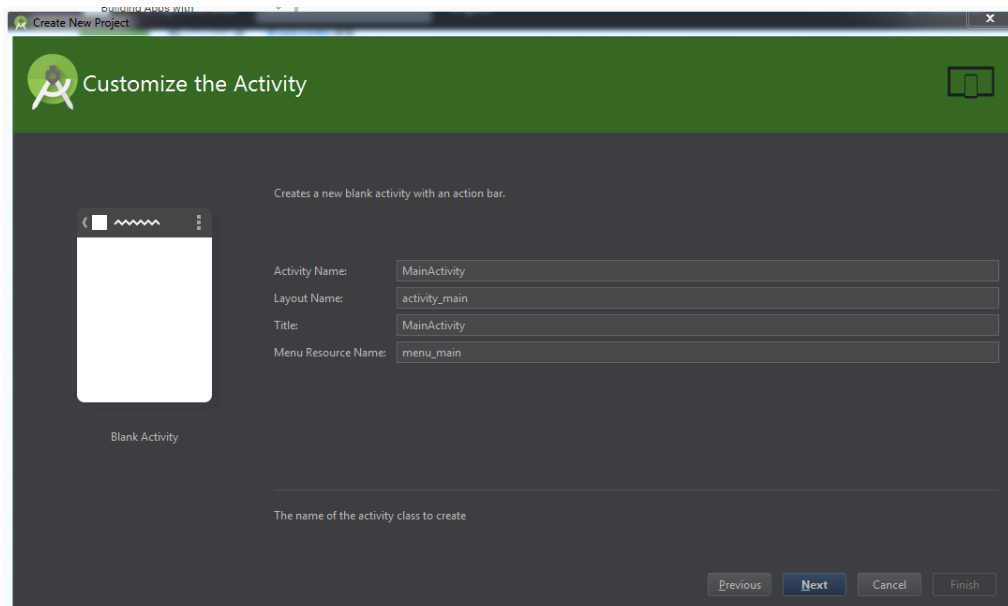
Obr. 5 Výběr API, které bude projekt využívat

3. Dále vybíráme druh aktivity pro mobilní zařízení, je zde před definovaných několik šablon, které vygeneruje android studio. Zvolíme blank Activity



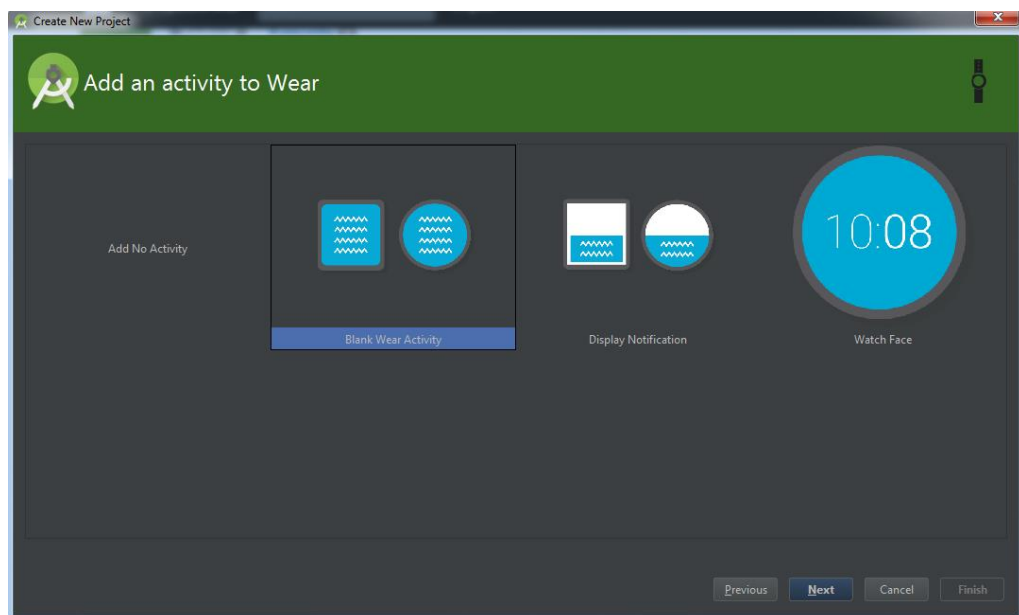
Obr. 6 Výběr šablon aktivit pro mobilní zařízení

4. Dále pojmenováváme aktivitu, layout a titulek. Necháme výchozí



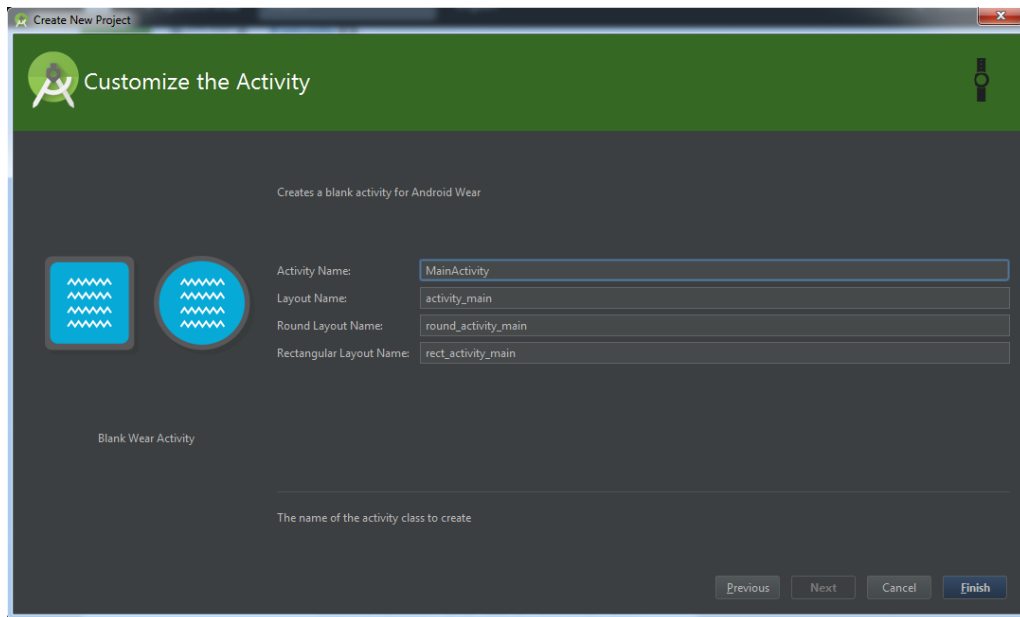
Obr. 7 Okno se zvolením názvu aktivity, třídy a layoutu

5. Dále vybíráme šablonu pro android wear, google zde před definoval nějaké výchozí šablony, vybereme blank wear acitivity



Obr. 8 Výběr aktivity pro android wear

6. Dále se zobrazí okno, kde pojmenujeme danou aktivitu pro android wear, necháme výchozí, a dokončíme průvodce



Obr. 9 Zvolení pojmenování aktivity a třídy pro android wear

Když je průvodce kompletní, android studio vytvoří nový projekt s dvěma modulami, mobile a wear,

1.7 Tvoření vlastního rozložení

Tvoření kontejneru pro android Wear je to samé jako pro mobilní zařízení, kromě toho že se musí tvořit stylově, aby příslušný uživatel rychle porozuměl způsobu fungování aplikace.[5]

1.7.1 Tvoření vlastních notifikací

Obecně by se mělo postupovat tak, že při tvoření notifikací na mobilním zařízení, by se mělo automaticky synchronizovat do android hodinek. Tímto způsobem stačí vytvořit notifikaci jednou a objeví se na mnoho druhů zařízení (ne jenom na android hodinkách, ale taky v autě, nebo v TV). Bez toho aniž bychom, programovali zvlášť pro jednotlivá zařízení.[8]

Postup tvoření vlastních notifikací:[4][5]

1. Vytvoříme rozložení a nastavíme dané rozložení pro danou aktivitu

```
public void onCreate(Bundle bundle)
{
    setContentView(R.layout.notification_activity_test);
}
```

2. Nadefinujeme vlastnosti dané aktivity v android manifestu, pro povolení dané aktivity na zobrazení se v daném obsahu android hodinek. Je potřeba nadefinovat vlastnosti dané aktivity jako exportovaná, vkládána a nesmí sdílet prostředky s jiné aktivity

```
<activity android:name="com.test.MyTestDisplayActivity"
    android:exported="true"
    android:allowEmbedded="true"
    android:taskAffinity=""
    android:theme="@android:style/Theme.DeviceDefault.Light" />??????????
```

3. Vytvoříme pendingIntent pro danou aktivitu kterou chceme zobrazit

```
Intent notificationIntent = new Intent(this, MyTestDisplayActivity.class);
PendingIntent notificationPendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, PendingIntent.FLAG_UPDATE_CURRENT);
```

4. Sestavíme notifikaci a zavoláme *setDisplayIntent()* poskytnuté *PendingIntent* . Systém používá *PendingIntent* na spuštění aktivity, když uživatel stiskne danou zobrazenou notifikaci.

5. Spustíme danou notifikaci *notify()*

1.7.2 Tvoření vlastních kontejnerů za pomoci UI knihovny

UI knihovna pro android wear je automaticky zahrnuta, když je vytvořen projekt pro android wear ve android studiu.[4][6]

```
dependencies {  
  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
  
    compile 'com.google.android.support:wearable:+'  
  
    compile 'com.google.android.gms:play-services-wearable:+'  
  
}
```

Daná knihovna je nápomocná při sestavování UIs pro android wear.

Základní třídy poskytované android API :[5]

- *BoxInsetLayout* – Rámový kontejner, který detekuje tvar hodinek (hrnaté kulaté)
- *CardFragment* – Fragment, který prezentuje obsah v rozšiřitelné, vertikálně posouvací kartě
- *CircledImageView* – Kontejner pro zobrazení obrázků, vytvarovaná na kruh
- *ConfirmationActivity* – Aktivita která zobrazuje dynamické potvrzení m poté, co uživatel dodělal akci
- *DelayedConfirmationView* – Kontejner který poskytuje kruhové stopky, typicky používané pro potvrzení operace, při daném zpožděním
- *DismissOverlayView* – Kontejner pro implementaci dlouhého stisknutí na ukončení aplikace
- *DotsPageIndicator* – Indikátor ve formě teček, který ukazuje, na jaké stránce se daný uživatel nachází, typicky používaný v *GridViewPager*
- *GridViewPager* – Stránkovací kontejner, který poskytuje uživateli stránkování ve vertikálním směru a horizontálním směru
- *GridPagerAdapter* – Adapter zodpovědný za manipulaci dat v *GridViewPager*
- *FragmentGridPagerAdapter* – Implementace *GridPagerAdapter*, který prezentuje stránky jako fragmenty
- *WatchViewStub* – Třída, která upraví specifický kontejner, podle tvaru android hodinek

- *WearableListView* – Alternativa *listView* , která je optimalizovaná pro snadné užití malá displejů android hodinek. Zobrazí vertikálně seznam položek a automaticky přiblíží na nejbližší položku, když uživatel přestane posouvat

1.7.3 Tvoření vlastního kontejneru zvlášť pro kulaté a čtvercové android hodinky

Třída *WatchViewStub*, která je ve Wearable UI knihovně, nám poskytuje definici pro kulaté a čtvercové obrazovky. Daná třída detekuje tvar obrazovky za běhu a načte podle toho korespondující kontejner.[7]

Postup definice:

1. Přidání *WatchViewStub* třídy jako hlavní element kontejneru aktivity
2. Vytvoření specifického kontejneru v XML pro kulaté displeje s *rectLayout* atributem
3. Vytvoření specifického kontejneru v XML pro kulaté displeje s *roundLayoutem* atributem

Definování kontejneru aktivity v XML:

```
<android.support.wearable.view.WatchViewStub
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/watch_view_stub_test"
android:layout_width="match_parent"
android:layout_height="match_parent"
app:rectLayout="@layout/rect_test_activity_wear"
app:roundLayout="@layout/round_test_activity_wear">
</android.support.wearable.view.WatchViewStub>
```

Nastavení kontejneru v dané aktivitě:

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

Poté vytvoříme různé kontejnery, které definují vzhled na daných zařízeních. Vytvoříme soubor `res/layout/rect_activity_wear.xml` a `res/layout/round_activity_wear.xml`. Nadefinujeme danou strukturu. Definice je ta samá, co na mobilních zařízeních.

Kontejnery, které jsme nadefinovali, pro čtvercové a kulaté displeje, nejsou načteny dokud `WatchViewStub` nedetekuje tvar daného displeje, proto musíme nadefinovat listener, ve kterém pak můžeme manipulovat s danými komponentami[3]

```
@Override  
  
protected void onCreate(Bundle savedInstanceState) {  
  
super.onCreate(savedInstanceState);  
  
setContentView(R.layout.activity_wear_test);  
  
WatchViewStub stub = (WatchViewStub) findViewById(R.id.watch_view_stub_test);  
stub.setOnLayoutInflatedListener(new WatchViewStub.OnLayoutInflatedListener() {  
  
@Override  
  
public void onLayoutInflated(WatchViewStub stub) {  
  
TextView tv = (TextView) stub.findViewById(R.id.text);      ...  
  
}  
  
});  
  
}
```

1.8 Posílání a synchronizace data

API datové vrstva android hodinek, která je součástí Google play services, poskytuje komunikační kanál pro mobilní zařízení a android hodinek. API se skládá z množiny dato-

vých objektů, které napomáhají synchronizaci dat mezi mobilním zařízením a android hodinek[4]

1.8.1 Požadavky

Pro přístup k dané API je potřeba Android 4.3 (API level 18) a vyšší na mobilním zařízení a nejnovější verzi google play services.[1]

1.8.2 Komponenty

- Data items – Poskytuje uložiště s automatickou synchronizací s mobilním zařízením[2]
- Messages – Daná třída posílá zprávy a je užitečná pro vzdálené volání procedur, jak ovládání hudby z android hodinek, nebo spuštění obsahu na android hodinkách z mobilního zařízení. Zprávy jsou užitečné pro jednosměrnou komunikaci nebo pro obousměrnou komunikaci – požadavek / odpověď model. Pokud mobilní zařízení a hodinky jsou spárovány, systém jednotlivé požadavky vkládá do fronty zpráv a vrací úspěšnou odpověď. Pokud dané zařízení nejsou propojeny, vrátí chybový kód. Úspěšný výsledný kód neindikuje, že zpráva byla doručena úspěšně, protože zařízení se mohlo odpojit po přijetí úspěšného kódu.[2]
- Asset – Objekt pro posílání binárních dat, jako obrázky. Připojíme celky do data uložiště a systém automaticky se postará o přenos, bere v potaz šířku pásma bluetooth a podle toho cachuje velké obrázky, aby se vyhnul přeposlání.[2]
- *WearableListenerService* – Zděděním dané třídy můžeme naslouchat změn datové vrstvy. Daná třída musí být implementovaná jako services. Systém řídí životní cyklus, podle potřeby připojuje a odpojuje danou servisu[2]
- Datalistener – Podobné jako *WearableListenerService*, rozdíl je v tom že se používá, když je daná aktivita v popředí, čili naslouchá, jen když je daná aktivita aktivní.[2]
- Channel – Využívá se pro přenos velkých dat jako hudba, videa z mobilního zařízení do android hodinek. Přenáší datové soubory mezi dvěma nebo více zařízení, bez automatické synchronizace, jako je to mu u Assetu. Channel API šetří místo na disku, oproti DataApi třídě, která tvoří kopii assetu na lokální zařízení před synchronizací[2]

Android Wear podporují, připojení více android hodinek na mobilní zařízení. Například, když uživatel uloží poznámku na mobilní zařízení, automaticky se objeví na obou spárovaných zařízeních[3]

1.8.2.1 Přístup na datovou vrstvu android wear

Pro zavolání datové vrstvy API, musíme vytvořit instanci *GoogleApiClient*, hlavní bod pro každou google play service APIs.[8]

GoogleApiClient poskytuje rozhraní které usnadňuje tvoření daného klienta.

```
GoogleApiClient mGoogleApiClient = new GoogleApiClient.Builder(this)
.addConnectionCallbacks(new ConnectionCallbacks() {

@Override

public void onConnected(Bundle connectionHint) {

Log.d(TAG, "onConnected: " + connectionHint);

}

@Override

public void onConnectionSuspended(int cause) {

Log.d(TAG, "onConnectionSuspended: " + cause);

}

})

.addOnConnectionFailedListener(new OnConnectionFailedListener() {

@Override

public void onConnectionFailed(ConnectionResult result) {

Log.d(TAG, "onConnectionFailed: " + result);

}

})

.addApi(Wearable.API)

.build();
```

1.8.2.2 Synchronizace datových položek

DataItem definuje, datové rozhraní, které systém používá na synchronizaci android mobilních zařízeních a android hodinek. *DataItem* se skládá z následujících položek :[6]

- Payload – Bytové pole, které se může nastavit na jakoukoliv posloupnost dat, je nutné napsat vlastní serializaci a deserializaci. Velikost je omezena na 100 Kb.
- Path – Unikátní řetězec který musí začínat dopředním lomítkem, např „./path/to/data“. Pokud tvoříme hierarchickou datovou strukturu v dané aplikaci, schéma cest

Neimplementujeme *DataItem* přímo, místo toho:

1. Vytvoříme objekt *PutDataRequest*, kterému nastavíme cestu, která unikátně identifikuje daný objekt
2. Zavoláme metodu *setData()* na nastavení bytového pole
3. Zavoláme metodu *DataApi.putDataItem()*, která vyzve systém na vytvoření *dataItemu*
4. Když voláme dané *DataItems* , systém vrací objekty, které implementují *DataItem* rozhraní

Je doporučováno, místo pracování s holými byte, přes *setData()*, se doporučuje používání data mapy, která rozšiřuje metody *DataItem*, na snadné pracování s datovými typy.

1.8.2.3 Synchronizace pomocí Data mapy

Je-li to možné, doporučuje se používat *DataMap* třídu. Daný postup nám ulehčí práci s datovými položkami ve formě android *bundle*, takže serializaci a deserializaci za nás provede systém. Manipulace s daty je ve formě klíče – hodnoty.[7]

Použití data mapy:[7]

1. Vytvoření *PutDataMapRequest* instance a nastavení cesty k datové položce
2. Zavolání metody *PutDataMapRequest.getDataMap()*, k vrácení objektu datové mapy, kde můžeme nastavovat hodnoty ve formě klíče – hodnoty.

3. Za pomoci metody *put..()*, kterou voláme ve formě např. *putString()* vložíme pár do dané mapy.
4. Zavolání metody *PutDataMapRequest.asPutDataRequest()*, k vrácení objektu *PutDataRequest*
5. Zavolání metody *DataApi.putDataItem()*, kde systém vytvoří danou datovou položku

Pokud jsou mobilní zařízení a android hodinky odpojeny, data jsou vložena do vyrovnávací paměti a po obnovení připojení jsou synchronizovány mezi dané zařízení[6]

Příklad tvoření data mapy:[6]

```
public class MainActivity extends Activity implements      DataApi.DataListener,
GoogleApiClient.ConnectionCallbacks,      GoogleA-
piClient.OnConnectionFailedListener {

private static final String COUNT_KEY = "com.test..key.count";

private GoogleApiClient mGoogleApiClient;

private int countRequest = 0;

// Create a data map and put data in it

private void increaseCounter() {

PutDataMapRequest putDataMapReq = PutDataMapRequest.create("/countRequest");

putDataMapReq.getDataMap().putInt(COUNT_KEY, countRequest ++);

PutDataRequest putDataReq = putDataMapReq.asPutDataRequest();

PendingResult<DataApi.DataItemResult> pendingResult =

Wearable.DataApi.putDataItem(mGoogleApiClient, putDataReq);

}

}
```

Při každém zavolání dané funkce *increaseCounter* , se vloží hodnota do *DataMapy* a automaticky se zavolají dané listenery.[6]

1.8.2.4 Naslouchání změn v datech

Pokud změníme hodnoty v datech, tak můžeme implementovat *DataApi.DataListener*, pomocí kterého může obsluhovat změny v datech na daném zařízení.[5]

```
public class MainActivity extends Activity implements DataApi.DataListener, GoogleA-
piClient.ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener {

    private static final String COUNT_KEY = " com.test..key.count";

    private GoogleApiClient mGoogleApiClient;

    private int count = 0;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        mGoogleApiClient = new GoogleApiClient.Builder(this).addApi(Wearable.API)
        .addConnectionCallbacks(this)

        .addOnConnectionFailedListener(this).build();

    }

    @Override    protected void onResume() {

        super.onStart();

        mGoogleApiClient.connect();

    }

    @Override

    public void onConnected(Bundle bundle) {

        Wearable.DataApi.addListener(mGoogleApiClient, this);

    }

    @Override

    protected void onPause() {

        super.onPause();
```

```
Wearable.DataApi.removeListener(mGoogleApiClient, this);

mGoogleApiClient.disconnect();

}

@Override

public void onDataChanged(DataEventBuffer dataEvents) {

for (DataEvent event : dataEvents) {

if (event.getType() == DataEvent.TYPE_CHANGED) {

// DataItem changed

DataItem item = event.getDataItem();

if (item.getUri().getPath().compareTo("/count") == 0) {

DataMap dataMap = DataMapItem.fromDataItem(item).getDataMap();

updateCount(dataMap.getInt(COUNT_KEY));

}

} else if (event.getType() == DataEvent.TYPE_DELETED) {

// DataItem deleted

}

}

}

// Our method to update the count

private void updateCount(int c) {

...

}

... }

}
```

Důležité je vytvořit *GoogleApiClient*, zaregistrujeme jeho interface. V metodě *onResume()*, která se volá, když se daná aktivita ožíví, čili hned po vytvoření, nebo po přejítí do popředí, v dané metodě připojíme daného klienta na danou službu. Po úspěšném připojení, čili v metodě *onConnected()*, zaregistrujeme metodu, ve které budeme naslouchat změny

v datové vrstvě, čili v metodě *onDataChange()*. V ní podle typu změny, jestli byla dána položka upravena, nebo smazána, vykonáme danou akci. Pokud daná aktivita přejde do pozadí tak v metodě *onPause()*, odstraníme daný listener a odpojíme *GoogleApiClient*. [8]

1.8.2.5 Zasilání velkých dat

Pro posílání velkých celků binárních data přes bluetooth, jako obrázky, je implementované rozhraní v *PutDataRequest*. Je to Asset. Třída *PutDataRequest*, disponuje metodou *putAsset*, které předáme objekt typu Asset, který je vytvořen z klasického *ByteArrayOutputStream*. [2]

Metoda pro vytvoření Assetu : [2]

```
private static Asset createAssetFromBitmap(Bitmap bitmap) {  
    final ByteArrayOutputStream byteStream = new ByteArrayOutputStream();  
    bitmap.compress(Bitmap.CompressFormat.PNG, 100, byteStream);  
    return Asset.createFromBytes(byteStream.toByteArray());  
}
```

Metoda pro vložení do datové vrstvy: [2]

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.image);  
Asset asset = createAssetFromBitmap(bitmap);  
PutDataRequest request = PutDataRequest.create("/image");  
request.putAsset("profileImage", asset);  
Wearable.DataApi.putDataItem(mGoogleApiClient, request);
```

Assety se automaticky cachují, kvůli prevenci znovu poslání přes bluetooth. Běžný postup je, že mobilní zařízení stáhne daný obrázek, ořeže ho na velikost vhodnou pro android hodinky a pošle daná obrázek přes Assety. Velikost daných dat v assetu není nijak omezeno.

Pro načtení daného obrázku z datové vrstvy: [2]

@Override

```
public void onDataChanged(DataEventBuffer dataEvents) {  
    for (DataEvent event : dataEvents) {
```

```
if (event.getType() == DataEvent.TYPE_CHANGED &&
event.getDataItem().getUri().getPath().equals("/image")) {

    DataMapItem dataMapItem = DataMapItem.fromDataItem(event.getDataItem());
    Asset profileAsset = dataMapItem.getDataMap().getAsset("profileImage");
    Bitmap bitmap = loadBitmapFromAsset(profileAsset);

    // Do something with the bitmap

        }

    }

}

public Bitmap loadBitmapFromAsset(Asset asset) {
    if (asset == null) {
        throw new IllegalArgumentException("Asset must be non-null");
    }

    ConnectionResult result = mGoogleApiClient.blockingConnect(TIMEOUT_MS, Ti-
meUnit.MILLISECONDS);

    if (!result.isSuccess()) {    return null;    }

    // convert asset into a file descriptor and block until it's ready

    InputStream assetInputStream = Wearable.DataApi.getFdForAsset( mGoogleApiClient,
asset).await().getInputStream();

    mGoogleApiClient.disconnect();

    if (assetInputStream == null) {

        Log.w(TAG, "Requested an unknown Asset.");

        return null;    } // decode the stream into a bitmap    return BitmapFacto-
ry.decodeStream(assetInputStream);

}
```

1.8.2.6 Posílání a přijímání zpráv

Pro posílání a přijímání zpráv se používá *MessageApi* a jsou důležité dvě proměnné, podle kterých se tvoří daná třída. Obsah zprávy a unikátní identifikátor, který identifikuje akci dané zprávy. Mezi zprávami není žádná synchronizace, jako je tomu u *DataItem*. Zprávy jsou využity k jednosměrné komunikaci, toho se využívá, např. když chceme spustit nějakou aktivitu na android hodinkách z mobilního zařízení, nebo opačně. Je ale potřeba získat *nodeID*, to je unikátní identifikátor, který identifikuje dané spárované zařízení. Pomocí daného identifikátor pošleme jednotlivým zařízením zprávu.[3]

Získání všech *nodeID*: [3]

```
public static Collection<String> getNodes(GoogleApiClient client) {  
    Collection<String> results= new HashSet<String>();  
    NodeApi.GetConnectedNodesResult nodes =  
        Wearable.NodeApi.getConnectedNodes(client).await();  
    for (Node node : nodes.getNodes()) {  
        results.add(node.getId());  
    }  
    return results;  
}
```

Ukázka zasílání zpráv: [3]

```
private void sendMessage(String constant,byte [] body)  
{  
    Log.v(TAG, "sendMessage from sendService");  
    GoogleApiClient googleApiClient = new GoogleApiClient.Builder(this)  
        .addApi(Wearable.API)  
        .build();  
  
    // It's OK to use blockingConnect() here as we are running in an
```

```
// IntentService that executes work on a separate (background) thread.

ConnectionResult connectionResult = googleApiClient.blockingConnect(
    Constant.GOOGLE_API_CLIENT_TIMEOUT_S, TimeUnit.SECONDS);

if (connectionResult.isSuccess() && googleApiClient.isConnected()) {

    // Loop through all nodes and send a clear notification message
    Iterator<String> itr = Utils.getNodes(googleApiClient).iterator();
    while (itr.hasNext()) {
        Wearable.MessageApi.sendMessage(
            googleApiClient, itr.next(), constant, body);
    }
    googleApiClient.disconnect();
    }
}

```

Připojíme *GoogleApiClient* a pomocí funkce *GetNodes(googleApiClient)* si vrátíme jednotlivé uzly a pošleme zprávu. [3]

Pro přijímání zpráv se musí vytvořit *WearableListenerService*, ve kterém implementujeme danou metodu *onMessageReceived*

@Override

```
public void onMessageReceived(MessageEvent messageEvent) {
    Log.v(TAG, "onMessageReceived: " + messageEvent);
    if (Constant.CLEAR_NOTIFICATIONS_PATH.equals(messageEvent.getPath())) {
        // Request for this device to clear its notifications
        SendService.clearNotification(this);
    } else if (Constant.START_NAVIGATION_PATH.equals(messageEvent.getPath())) {

```

```
// Request for this device to start Maps walking navigation to  
// specific tourist attraction  
String attractionQuery = new String(messageEvent.getData());  
Uri uri = Uri.parse(Constant.MAPS_NAVIGATION_INTENT_URI +  
Uri.encode(attractionQuery));  
Intent intent = new Intent(Intent.ACTION_VIEW, uri);  
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);  
startActivity(intent);  
}else if(Constant.DEADLINES_REQUEST.equals(messageEvent.getPath())){  
SendService.triggerCheckDeadlines(this);  
}  
}
```

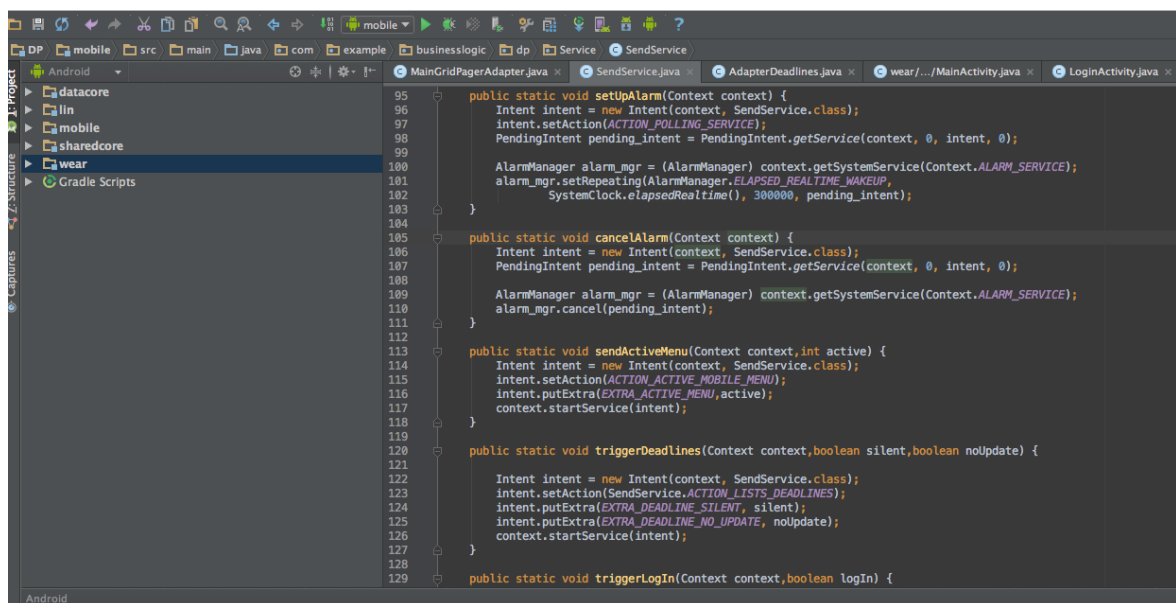
V dané metodě je důležitý parametr `messageEvent`, který obsahuje unikátní cestu, podle které provedeme specifickou akci, a jako další obsahuje data do kterých se ukládá posloupnost bajtů, čili obsah dané zprávy[3]

II. PRAKTICKÁ ČÁST

2 VÝVOJOVÁ APLIKACE

Pro ukázkou vývoje pro android hodinky bylo zvoleno téma aplikace, registrace zkouškových termínů pomocí portálu UTB. Využívá se veřejně dostupná REST služba UTB - <http://stag-ws.utb.cz/ws/help/> . Aplikace jak na mobilním zařízení tak na android hodinkách umí zaregistrovat daného studenta na daný termín. Na mobilním zařízení běží služba, která kontroluje každých 5 min, jestli nepřibyl nový termín a notifikuje jak android hodinky tak mobilní zařízení o novém termínu

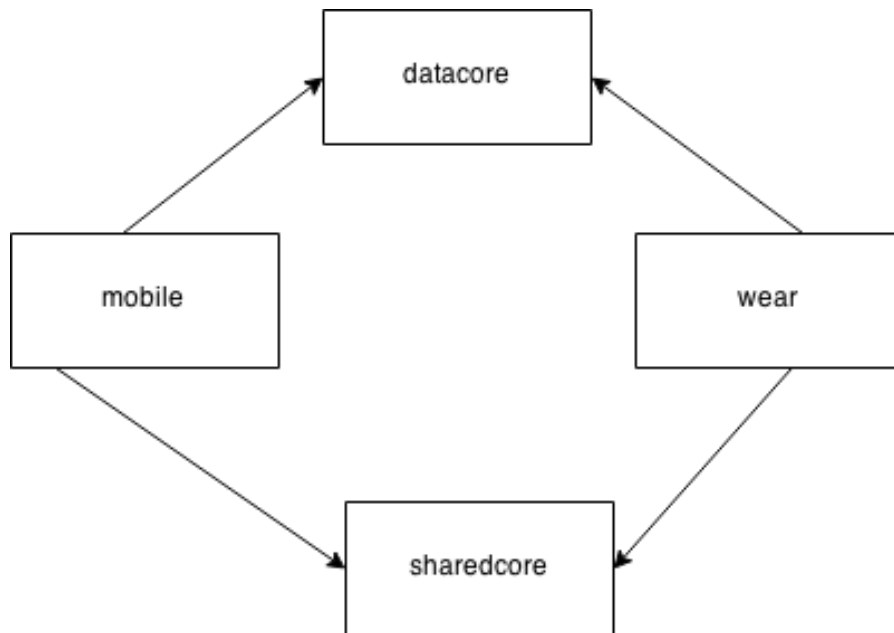
2.1 Struktura projektu



Obr. 10 Struktura ukázkového projektu

Projekt se skládá z:

1. Modul datacore – Daný modul je zodpovědný za komunikaci s danou službou UTB a obsluhou daných odpovědí z dané služby, taky se stará o parsování daných odpovědí a převodu na daný objekt.
2. Modul lin – Knihovna, která poskytuje podporu vysouvacího bočního menu
3. Modul mobile – Daný modul, je aplikace na mobilní zařízení. Zde voláme dané služby z datacore a posíláme na android hodinky
4. Modul sharedcore – V daném modulu jsou metody a konstanty které používají jak modul mobile tak modul wear
5. Modul wear – Daný modul, je aplikace na android hodinky.



Obr. 11 Jednotlivé závislosti mezi moduly

Modul mobile, má závislost na datacore a sharedcore. Modul wear má závislost na datacore a na sharedcore, závislost na modul datacore je jen z důvodu přístupu k objektům které se tvoří z odpovědí dané služby.

2.2 Datacore

2.2.1 Třídy

Daný modul, se skládá z jednotlivých tříd objektů reprezentující volanou odpověď ze služby.

Ukázka dané třídy:

```
public class Deadline {  
  
    public static final String PARA_DEADLINE_ID = "termIdno";  
  
    @SerializedName("termIdno")  
  
    public int DeadlineID;  
  
    @SerializedName("ucitel")
```

```
public Data.Teacher Teacher;

@SerializedName("predmet")

public String Subject;

public static String createJsonList(List<Deadline> deadlines)
{
    Deadline [] _deadlines = deadlines.toArray(new Deadline[deadlines.size()]);
    return new Gson().toJson(_deadlines,Deadline[].class);
}

public static List<Deadline> parse(String result)
{
    List<Deadline> deadlinesList = null;

    try
    {
        Deadline[] deadlines = new Gson().fromJson(result, Deadline[].class);
        deadlinesList = Arrays.asList(deadlines);
    }catch (Exception e)
    {
        e.printStackTrace();
    }

    return deadlinesList;
}
```

Na parsování odpovědí z dané služby, používám knihovnu Gson, která umí mapovat objekty a přiřazovat danou hodnotu proměnné podle mapování.

2.2.2 Volání HTTP metod

Na volání daných služeb se využívá AsyncTask

```
private class HttpRequest extends AsyncTask<Input, Integer, ResultHttpService> {
```

```
@Override
protected ResultHttpService doInBackground(Input... input) {
    StringBuilder builder = null;
    Integer response = null;
    Map<String, List<String>> headers = null;
    Input _input = input[0];
    try {
        // init url
        Utils.Print.Object("service: " + _input.Service + " URL : " + _input.Url);
        java.net.URL url = new URL(_input.Url);
        //set property for connection
        System.setProperty("http.keepAlive", "false");
        HttpURLConnection httpCon = (HttpURLConnection) url.openConnection();
        httpCon.setConnectTimeout(10000);
        // set request method
        httpCon.setRequestMethod(_input.TypeRequest.toString());
        //set cookies if need it
        java.net.CookieManager msCookieManager = (java.net.CookieManager)CookieHandler.getDefault();
        if(msCookieManager != null) {
            List<HttpCookie> cookies = msCookieManager.getCookieStore().getCookies();
            if (_input.SetCookie && cookies.size() > 0) {
                Utils.Print.Object("Setting cookie :");
                for (HttpCookie cookie : cookies)
                    Utils.Print.Object("cookie name : " + cookie.getName() + " value : " +
                    cookie.getValue());
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
        httpCon.setRequestProperty("Cookie", TextUtils.join(",", cookies));
    }
}

// if is post request set body
if (_input.TypeRequest == TypeRequest.POST) {
    httpCon.setRequestProperty("Content-Type", "application/json");
    String body = _input.Body;
    Utils.Print.Object("Body :" + body);
    if (body != null) {
        byte[] outputInBytes = body.getBytes("UTF-8");
        OutputStream os = httpCon.getOutputStream();
        os.write(outputInBytes);
        os.close();
    }
}

// if we set additional headers print it
if(_input.Headers.size() > 0)
{
    Utils.Print.ObjectRed("service: " + _input.Service + " Hedears :");
    for (String key : _input.Headers.keySet())
    {
        String value = _input.Headers.get(key);
        httpCon.setRequestProperty(key,value);
        Utils.Print.ObjectRed("Key : " + key + " Value: " + value );
    }
}
}
```

```
//get response code

response = httpCon.getResponseCode();

Utils.Print.ObjectRed("service: " + _input.Service + " Response :"+ response);

// if we need to manipulate with response headers then return it to service

if(_input.GetResponseHeaders && response == HttpStatus.SC_OK)

{

    headers = httpCon.getHeaderFields();

    if(_input.FilterHeadrs != null) {

        for (String filterHeadr : _input.FilterHeadrs) {

            Utils.Print.ObjectRed("service: " + _input.Service + " Key :"+ filterHeadr);

            List<String> value = headers.get(filterHeadr);

            Utils.Print.ObjectRed("Value: ");

            Utils.Print.ObjectsRed(value);

        }

    }

}

// get reponse

BufferedReader reader = new BufferedReader(new InputStreamReader(httpCon.getInputStream()));

builder = new StringBuilder();

for (String line = null; (line = reader.readLine()) != null; ) {

    builder.append(line).append("\n");

}

Utils.Print.Object("service: " + _input.Service + " JSON :"+ builder.toString());

} catch (Exception e) {
```

```
e.printStackTrace();

if(response != null)

{

    return new ResultHttpService(builder == null ? null : builder.toString(),response,_input.Service,headers);

}else

{

    return null;

}

}

return new ResultHttpService(builder.toString(), response,_input.Service,headers);

}

protected void onProgressUpdate(Integer... params) {

    //Update a progress bar here, or ignore it, it's up to you

}

protected void onPostExecute(ResultHttpService result) {

    super.onPostExecute(result);

    if(iResult != null && !Sync)

    {

        iResult.Result(result);

    }

}
```

V metodě *doInBackground()*, spustíme pomocí *HttpURLConnection*, požadavek na danou službu a v metodě *onPostExecute()* vrátíme výsledný výsledek, i s kódem odpovědi.

2.2.3 Servisy

Obsluhy, které volají daný Async task a volají dané parsování odpovědi ze služby, jsou servisy. Je to třída, která má metody podle URL, kterou voláme z dané služby UTB, a má obsluhu odpovědi z AsyncTask.

```
public class SDeadline extends BaseService implements IResult {  
  
    public static final String GET_DEADLINES_STUDENT = "services/rest/terminy/getTerminyProStudenta";  
  
    public ISDeadline isDeadline;  
  
    public SDeadline(Context context) {  
  
        super(context);  
  
    }  
  
    public List<Deadline> GetDeadLines(boolean sync)  
  
    {  
  
        super.setSetJsonFormatOutput(true);  
  
        super.StudentAutentication.refreshProperties(this.Context);  
  
        RefreshCookieManager(false);  
  
  
        Input input = new Input();  
  
        input.SetCookie = true;  
  
        input.Url = super.getPrefixUrl() + GET_DEADLINES_STUDENT + "?" +  
  
            Autentication.PARA_PERSONAL_NUMBER + "=" + su-  
  
        per.StudentAutentication.PersonalNumber +  
  
            super.getEndfixUrl();  
  
  
        input.TypeRequest = TypeRequest.GET;  
  
        input.Service = Service.DEADLINES_STUDENT;
```

```
BaseHttpService httpService = new BaseHttpService(this);

if(sync)

{

    try {

        ResultHttpService resultHttpService = httpService.startExecuteSync(input);

        if(resultHttpService != null && resultHttpService.resultCode == HttpStatus.SC_OK)

            {

                return WrapperDeadline.parse(resultHttpService.result);

            }

        } catch (ExecutionException e) {

            e.printStackTrace();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }else {

        httpService.startExecute(input);

    }

    return null;

}

@Override

public void Result(ResultHttpService result) {

    if(result == null){

        isDeadline.Error("fatal error");

        return;

    }
```

```
if(result.resultCode != HttpStatus.SC_OK)
{
    String message = this.GetError(result.result);

    if (result.resultCode == HttpStatus.SC_UNAUTHORIZED) {
        if (message == null) {
            message = "Pro danou akci se musíte přihlásit";
        }
        isDeadline.UnAuthorized(message);
    } else {
        if (message == null) {
            message = "Fatal error";
        }
        isDeadline.Error(message);
    }
    return;
}

switch (result.service)
{
    case DEADLINES_STUDENT:
        List<Deadline> deadlines = WrapperDeadline.parse(result.result);
        this.isDeadline.DeadlinesStudent(deadlines);
        break;
}
}
}
```

Daná servisa se stará i přihlašování a odhlašování a kontrolu cookies, jelikož pokud se dotazujeme na službu UTB tak se autentizujeme cookiem.

2.3 Mobile

Modul mobile, se stará o veškeré volání datové vrstvy a přenášení dat do wear modulu. Využívá IntenService a GoogleApi pro posílání zpráv a synchronizaci data mezi daným mobilním zařízením a android hodinkami.

2.3.1 Synchronizace dat

Po každém přihlášení studenta do aplikace se synchronizují data mezi mobilním zařízením a android hodinkami. Využívá se *IntentService*, která pokud je spuštěna, běží na pozadí a vykoná tělo metody *onHandleIntent* a skončí

```
public class SendService extends IntentService {  
  
    private static final String TAG = SendService.class.getSimpleName();  
  
    public static final String ACTION_LISTS_DEADLINES = "list_deadlines";  
  
    public static void triggerDeadlines(Context context) {  
  
        if(!Utils.isMyServiceRunning(context, SendService.class.getName())) {  
  
            Intent intent = new Intent(context, SendService.class);  
  
            intent.setAction(SendService.ACTION_LISTS_DEADLINES);  
  
            context.startService(intent);  
  
        }  
  
    }  
  
    public SendService() {  
  
        super(TAG);  
  
    }  
  
    @Override  
  
    protected void onHandleIntent(Intent intent) {  
  
        String action = intent != null ? intent.getAction() : null;
```

```
        Log.e(TAG, "On handleIntent : " + action);
    if (ACTION_LISTS_DEADLINES.equals(action)){
        refreshDeadlines();
    }

    private void refreshDeadlines(){
        SDeadline sDeadline = new SDeadline(this);
        List<Deadline> deadlines = sDeadline.GetDeadLines(true);
        sendDataToWearable(deadlines);
    }

    private void sendDataToWearable(List<Deadline> deadlines) {
        if(deadlines == null || deadlines.size() == 0 ){
            sendMessage(Constant.DEADLINE_ERROR,null);
            return;
        }
        Log.d(TAG,"Sending deadlines to weareable");
        GoogleApiClient googleApiClient = new GoogleApiClient.Builder(this)
            .addApi(Wearable.API)
            .build();

        // It's OK to use blockingConnect() here as we are running in an
        // IntentService that executes work on a separate (background) thread.
        ConnectionResult connectionResult = googleApiClient.blockingConnect(
            Constant.GOOGLE_API_CLIENT_TIMEOUT_S, TimeUnit.SECONDS);

        ArrayList<DataMap> deadlinesData = new ArrayList<>(deadlines.size());
        for (Deadline deadline : deadlines) {
            DataMap deadlineData = new DataMap();
```

```
deadlineData.putBoolean(Constant.Deadline.CAN_SIGN, deadli-
ne.CanSignInOut);

deadlineData.putBoolean(Constant.Deadline.SIGN_IN, deadline.SignIn);
deadlineData.putString(Constant.Deadline.CHAMBER, deadline.Chamber);
deadlineData.putString(Constant.Deadline.SUBJECT, deadline.Subject);
}

if (connectionResult.isSuccess() && googleApiClient.isConnected()) {

    PutDataMapRequest dataMap = PutDataMap-
Request.create(Constant.DEADLINES_PATH);

    data-
Map.getDataMap().putDataMapArrayList(Constant.EXTRA_ARRAY_LIST_DEADLINES,
deadlinesData);

    dataMap.getDataMap().putLong(Constant.EXTRA_TIMESTAMP, new Da-
te().getTime());

    dataMap.getDataMap().putBoolean(Constant.EXTRA_DEADLINE_SILENT, si-
lent);

    dataMap.getDataMap().putBoolean(Constant.EXTRA_DEADLINE_NO_UPDATE,
noUpdate);

    PutDataRequest request = dataMap.asPutDataRequest();

    // Send the data over

    DataApi.DataItemResult result =

        Wearable.DataApi.putDataItem(googleApiClient, request).await();

    if (!result.getStatus().isSuccess()) {

        Log.e(TAG, String.format("Error sending data using DataApi (error code =
%d)",

            result.getStatus().getStatusCode()));

    }

} else {
```

```
        Log.e(TAG, String.format(Constant.GOOGLE_API_CLIENT_ERROR_MSG,
            connectionResult.getErrorCode()));
    }
    googleApiClient.disconnect();
}
}
```

Metoda *triggerDeadlines* spustí danou *IntentService*, která poběží na pozadí. V metodě *onHandleIntent(Intent intent)*, se spustí metoda *refreshDeadlines*, která zavolá službu na datacore, vrátí si dané termíny a spustí vkládání do datové vrstvy GoogleAPI v metodě *sendDataToWearable*. V těle dané metody je klasické připojení na GoogleAPI a vložení daných záznamu do třídy *DataMap*

2.3.2 Zasílání zpráv

Zasílání zpráv v mobilní aplikaci využívám třeba na synchronizaci menu mezi mobilním zařízením a android hodinkami. Pokaždé když kliknu na menu v mobilní aplikaci tak se vybere v menu v android hodinkách. Opět využívám *IntentService*.

Zde je, metoda na zasílání zprav do android hodinek

```
private void sendMessage(String constant,byte [] body)
{
    Log.v(TAG, "sendMessage from sendService");
    GoogleApiClient googleApiClient = new GoogleApiClient.Builder(this)
        .addApi(Wearable.API)
        .build();
    // It's OK to use blockingConnect() here as we are running in an
    // IntentService that executes work on a separate (background) thread.
    ConnectionResult connectionResult = googleApiClient.blockingConnect(
        Constant.GOOGLE_API_CLIENT_TIMEOUT_S, TimeUnit.SECONDS);
}
```

```
if (connectionResult.isSuccess() && googleApiClient.isConnected()) {  
    // Loop through all nodes and send a clear notification message  
    Iterator<String> itr = Utils.getNodes(googleApiClient).iterator();  
    while (itr.hasNext()) {  
        Wearable.MessageApi.sendMessage(  
            googleApiClient, itr.next(), constant, body);  
    }  
    googleApiClient.disconnect();  
}  
}
```

2.4 Sharedcore

Daný modul obsahuje pouze třídu Utils a třídu Constant.

2.4.1 Třída Constant

V dané třídě jsou konstanty, které používají jak modul mobile tak modul wear. Jsou tam cesty, klíče, které se používají při odesílání, přijímání zpráv, synchronizace dat jak na mobilním zařízení tak na android hodinkách.

```
public class Constant {  
    public static class Deadline{  
        public static final String SUBJECT = "subject";  
        public static final String DATE = "date";  
        public static final String DEADLINE_ID = "deadlineID";  
    }  
}
```

2.4.2 Třída Utils

Daná třída obsahuje metody, které používá jak modul mobile, tak modul wear.

Hlavní metoda:

```
public static Collection<String> getNodes(GoogleApiClient client) {  
  
    Collection<String> results= new HashSet<String>();  
  
    NodeApi.GetConnectedNodesResult nodes =  
  
        Wearable.NodeApi.getConnectedNodes(client).await();  
  
    for (Node node : nodes.getNodes()) {  
  
        results.add(node.getId());  
  
    }  
  
    return results;  
  
}
```

Daná metoda vrací nodeID připojených zařízení, které se používá při posílání zpráv

2.5 Wear

Modul pro aplikaci na android hodinky. Daná aplikace umí zobrazit seznam termínu a zaregistrovat se na příslušný termín. Taky notifikuje, pokud přibyl nějaký nový termín. Skládá se z AsyncTasku které čtou z datové vrstvy googleAPI daná data a zobrazují je, poté pomocí DataMap synchronizuje vybraný termín.

2.5.1 Vrácení dat z googleAPI

2.5.1.1 AsyncTask

Pomocí třídy asyncTask si načteme data z googleAPI datové vrstvy. Při spuštění aplikace se spustí daný asyncTask

@Override

```
protected ArrayList<Deadline> doInBackground(Uri... params) {  
  
    // Connect to Play Services and the Wearable API  
  
    GoogleApiClient googleApiClient = new GoogleApiClient.Builder(mContext)  
  
        .addApi(Wearable.API)  
  
        .build();
```

```
ConnectionResult connectionResult = googleApiClient.blockingConnect(
    Constant.GOOGLE_API_CLIENT_TIMEOUT_S, TimeUnit.SECONDS);

if (!connectionResult.isSuccess() || !googleApiClient.isConnected()) {
    Log.e(TAG, String.format(Constant.GOOGLE_API_CLIENT_ERROR_MSG,
        connectionResult.getErrorCode()));
    return null;
}

ArrayList<Deadline> deadlines = new ArrayList<>();

Uri deadlinesUri = params[0];

DataApi.DataItemResult dataItemResult =
    Wearable.DataApi.getDataItem(googleApiClient, deadlinesUri).await();

if (dataItemResult.getStatus().isSuccess() && dataItemResult.getDataItem() !=
null) {
    DataMapItem dataMapItem = DataMapI-
tem.fromDataItem(dataItemResult.getDataItem());

    List<DataMap> deadlinesData = dataMapI-
tem.getDataMap().getDataMapArrayList(Constant.EXTRA_ARRAY_LIST_DEADLINES);

    // Loop through each attraction, adding them to the list
    for (DataMap deadlineData : deadlinesData) {
        Deadline deadline = new Deadline();

        deadline.Chamber = deadlineDa-
ta.getString(Constant.Deadline.CHAMBER);

        deadline.CanSignInOut = deadlineDa-
ta.getBoolean(Constant.Deadline.CAN_SIGN);

        deadlines.add(deadline);
    }
}
```

```

    }

    googleApiClient.disconnect();

    return deadlines;
}

```

Čtení z googleApi musíme provádět asynchronně proto asyncTask.

2.5.1.2 Naslouchání dat z WearableListenerService

Daná třída naslouchá změn v datové vrstvě googleApiClient. Čili cokoliv se změní, v daných datových objektech se poté zavolá daná metoda.

```

public class ListenerService extends WearableListenerService {

    private static final String TAG = ListenerService.class.getSimpleName();

    @Override

    public void onDataChanged(DataEventBuffer dataEvents) {

        Log.d(TAG, "onDataChanged: " + dataEvents);

        final List<DataEvent> events = FreezableUtils.freezeIterable(dataEvents);

        for (DataEvent event : events) {

            if (event.getType() == DataEvent.TYPE_CHANGED

                && event.getDataItem() != null

                &&

                Constant.DEADLINES_PATH.equals(event.getDataItem().getUri().getPath())) {

                DataMapItem          dataMapItem          =          DataMapI-
                tem.fromDataItem(event.getDataItem());

                Log.d(TAG, "onDataChanged Uri : " + dataMapItem.getUri());

                int [] deadlineIDs = new int [deadlinesData.size()];

                for (int i = 0 ; i < deadlineIDs.length; i++){

                    deadlineIDs[i]          =          deadlinesDa-
                    ta.get(0).getInt(Constant.Deadline.DEADLINE_ID);

                }

```

```
        UtilityService.triggerNewDeadlinesNotification(this,deadlineIDs);  
    }  
}  
}  
}
```

2.5.2 Notifikace

Spouštění notifikací, na android hodinách, musíme spustit lokálně na zařízení k tomu, slouží třída *Notification.Builder*.

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)  
.setContentTitle("Počet nových termínů " + deadlinesID.length)  
setSmallIcon(R.drawable.ic_deadline)  
.setLocalOnly(true)  
.setDefaults(Notification.DEFAULT_ALL)  
.setContentIntent(updateIntent);  
  
NotificationCompat.WearableExtender wearableOptions = new NotificationCom-  
pat.WearableExtender();  
  
    for (int deadlineID : deadlinesID) {  
        for (DataMap dataDeadline : deadlinesData) {  
            if (deadlineID == dataDeadli-  
ne.getInt(Constant.Deadline.DEADLINE_ID)) {  
                Notification deadlinesNotif = new Notification.Builder(this)  
.setContentTitle(dataDeadline.getString(Constant.Deadline.SUBJECT)  
.setContentText(dataDeadline.getString(Constant.Deadline.CHAMBER)).build();  
                wearableOptions.addPage(deadlinesNotif);  
                break;  
            }  
        }  
    }
```

```
    }  
  }  
  
  builder.extend(wearableOptions);  
  
  Notification notification = builder.build();  
  
  ((NotificationManager) getSystemService(NOTIFICATION_SERVICE))  
    .notify(Constant.WEAR_NOTIFICATION_ID, notification);
```

Daný notificační builder vytvoří několik stránek podle toho kolik je nových temínů.

ZÁVĚR

Těžištěm práce bylo, zmapování programování android wearu za pomoci android API a google API v jazyce Java. Dále byl za úkol vytvořit testovací aplikaci pro dané zařízení, jako ukázkou vývoje v dané technologii. Bylo použito opravdové zařízení Motorola MOTO 360, jako vývojový prostředek. Celkově android hodinky, mají podle mě velký potenciál, kvůli rychlosti vyřízení jednoduchý úkonu. Bohužel hodinky mají jednu obrovskou nevýhodu a to je výdrž baterie. Dané hodinky vydrží nanejvýš den při plném využití.

SEZNAM POUŽITÉ LITERATURY

- [1] RUIZ, David Cuartielles. Professional android wearables. Indianapolis: Wrox, February 17, 2015, pages cm. ISBN 11-189-8685-7.
- [2] SMITH, Dave a Jeff FRIESEN. Android Recipes, 3rd Edition. New York City: Apress, 2014. ISBN 978-1-4302-6322-7.
- [3] SCHWARZ, Ronan, Phil DUTSON, Jamse STEELE a Nelson TO. The Android Developer's Cookbook, 2nd Edition. Boston: Addison-Wesley, 2013. ISBN 978-0-321-8953-4.
- [4] HELLMAN, Erik. Android programming: pushing the limits. Chichester, West Sussex: Wiley, 2014. ISBN 978-111-8717-370
- [5] CALVO, Andres. Beginning Android Wearables. New York City: Apress, 2015. ISBN 978-1-4842-0518-1.
- [6] ZAPATA, Bélen Cruz. Android Studio application development: create visually appealing applications using the new IntelliJ IDE Android Studio. New Edition. Birmingham, UK: Pack Pub, 2013. ISBN 978-178-3285-273.
- [7] MACLEAN, Satya Komatineni and Dave. Expert Android. 2013. vyd. New York: Aúress, 2013. ISBN 978-143-0249-504.
- [8] JACKSON, Wallace. Pro Android UI. New edition. New York City: Apress, 2014, xxvi, 552 pages. ISBN 14-302-4986-2

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API Application Programming Interface

UTB Univerzita Tomáše Bati

SEZNAM OBRÁZKŮ

Obr. 1 Výsledný stav po zapnutí ladění přes bluetooth	13
Obr. 2 Spuštění příkazu na propojení mobilního zařízení a android hodinek	13
Obr. 3 Výsledné propojení mobilního zařízení a zařízení android hodinky	13
Obr. 4 Okno nového projektu	14
Obr. 5 Výběr API, které bude projekt využívat	15
Obr. 6 Výběr šablon aktivit pro mobilní zařízení	15
Obr. 7 Okno se zvolením názvu aktivity, třídy a layoutu	16
Obr. 8 Výběr aktivity pro android wear	16
Obr. 9 Zvolení pojmenování aktivity a třídy pro android wear	17
Obr. 10 Struktura ukázkového projektu.....	34
Obr. 11 Jednotlivé závislosti mezi modulami.....	35

SEZNAM PŘÍLOH

P1: nosič CD-ROM