Použití a vývoj aplikací pro Motorola Moto 360

Bc. Jakub Hromada

Diplomová práce
2015

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. Indianopolis: 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

L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

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

Diplomová práce se zabývá možnostmi tvorby aplikací pro mobilní zařízení Motorola Moto 360 a obecně SmartWatch, za pomocí Android API a Google API a přidruženého pomocného zařízení. Je zde taky popsána časová náročnost pro nasazení na další SmartWatch zařízení a možnosti jiných technologií pro daný vývoj SmartWatch. V praktické části je popsána vzorová aplikace, která demonstruje použití dané technologie.

Klíčová slova: Android, Java, Wear

ABSTRACT

This thesis deals with the possibilities of creating applications for mobile devices Motoroloa Moto 360 and generally SmartWatch, using Android API and the Google API and the associated auxiliary device. There is also described timeconsuming proces to deploy the next SmartWatch device and the possibility of other technologies for the development of the SmartWatch. The practival part describes a sample application that demonstrates how to use the technology.

Keywords: Android, Java, Wear

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

OBSAH

ÚVOD 9
I TEORETICKÁ ČÁST 10
1 TVORBA APLIKACE 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 pomocí UI knihovny 18
1.7.3 Tvoření vlastního kontejneru zvlášť pro kulaté a čtvercové android hodinky 20
1.8 Posílání a synchronizace data 22
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 25
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 Modul Datacore 35
2.2.1 Třídy 35
2.2.2 Volání HTTP metod 36
2.2.3 Pomocné služby na volání AsyncTasku 38
2.3 Mobile 41
2.3.1 Synchronizace dat 41
2.3.2 Zasílání zpráv 44
2.4 Modul Sharedcore 45
2.4.1 Třída Constant 45
2.4.2 Třída Utils 46
2.5 Modul Wear 46
2.5.1 Vrácení dat z Google API 46
2.5.1.1 AsyncTask 46
2.5.1.2 Naslouchání dat z WearableListenerService 48
2.5.2 Notifikace 48

2.6 ROZBOR ČASOVÉ NÁROČNOSTI NA NASAZENÍ NA DALŠÍ SMARTWATCH ZAŘÍZENÍ 50
2.6.1 Přizpůsobení aplikace na tvar displeje dalšího smartwatch zařízení 50
2.6.1.1 Přizpůsobení tvaru displeje testovací aplikace 50
2.6.2 Synchronizace dat a posílání zpráv 51
2.6.3 Využití senzorů 51
2.6.4 Shrnutí 52
2.7 SROVNÁNÍ S OSTATNÍMI PROGRAMOVACÍMI JAZYKY 52
2.7.1 HTML 5 javascript 52
2.7.2 XAMARIN 53
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 pomocí 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 programových nástrojů. Jazyk, ve kterém byl psán výsledný zdrojová kód, byl zvolen jazyk Java.

Vzorová aplikace, která byla naprogramovaná, využívá synchronizaci dat a zasílání zpráv za pomocí Google Mobile Services, notifikací, běžících procesů v pozadí, programové komponenty a kontejnery tříd pro práci s UI. Veškeré tyto položky jsou k dispozici v Android SDK. Vývojová aplikace využívá REST API portálu UTB, aby demonstrovala ukázku práci s dynamickou datovou vrstvou. Dále byl implementován pomocný server, který se stará o příchozí notifikace, jako je např. nový termín. Za Framework byl zvolen Google App Engine a programovací jazyk Python. Spolu s pomocným serverem, byl taky implementován testovací server. Který byl použit, na otestování vytvořené vývojové aplikace. Framework a programovací jazyk byl taky zvolen Google App Engine a Python.

V další části popisuji časovou náročnost migrace vývojové testovací aplikace, na další Smartwatch zařízení. Jedná se o analyzování obtížnosti migrace na odlišné Smartwatch zařízení, a pospání postupů které napomáhají k rychlejší migraci vytvořené testovací aplikace, pomocí kontejnerů a metod které poskytuje Android Wear API.

Poslední část popisuje porovnání programování pro android s jiným programovacími jazyky. Je možné programovat pro android Wear, v jiném programovacím jazyce než je jazyk Java. A cílem bylo popsat výhody, nevýhody ostatních platforem

I. TEORETICKÁ ČÁST

1 TVORBA APLIKACE 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 pomocí XML, oživení obrazovek za pomocí aktivit, rozdělení obrazů pomocí fragmentů a další klasické metody Android SDK.

1.2 Motorola Moto 360

Zařízení Motorola Moto 360 disponuje operačním systémem Android 5, nemá USB port, takže komunikace se spárovaným zařízením je za pomocí Bluetooth technologie. Disponuje krokoměrem, mikrofonem a optickým snímačem tepové frekvence Minimální verze operačního systému Android běžící na spárovaném zařízením musí být 4.3.

1.3 Požadavky pro vývoj chytrých hodinek

Podle oficiální stránek dokumentace Android vývoje, je doporučován vývoj chytrých hodinek v prostředí Android studiu, kvůli snadnějšímu vystavování, nastavování modulu, verzováním a importů pomocných knihoven. Minimální verze vývojového prostředí Android studia potřebná pro vývoj aplikací Smartwatch zařízení 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. Veškerá instalace daných balíčků je součástí SDK Manageru [1], [2].

1.4 Popis aplikací běžících na chytrých hodinkách

Aplikace běžící přímo na chytrých hodinkách, mohou přistupovat na standartní hardwarové prostředky jako sensory a GPU. 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. Aplikace jsou relativně malé co do velikosti využívání a funkcionality v porovnání s aplikacemi na mobilních zařízeních. Obsah 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 zařízení, čili operace jako dotazování na internetové služby, nebo těžké operace jako složité výpočty. Po vyřízení požadavků, se výsledná data na mobilním zařízení odešlou, spárují s aplikaci na Smartwatch zařízení. Aplikace na chytrých hodinkách uživatel přímo

neinstaluje, ale daná aplikace je součástí aplikace na mobilní zařízení, 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]:

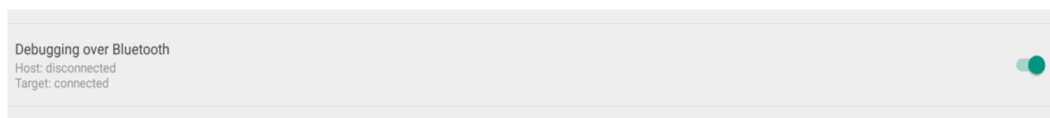
1. Instalace aplikace Android Wear, která je dostupná na Google Store na mobilním zařízení
2. Provedeme jednotlivé kroky, podle nainstalované Android Wear aplikace na spárování android hodinek.
3. Necháme otevřenou aplikaci na mobilním zařízením
4. Musíme zapnout ladění na android hodinkách
    a. Přejdeme na nastavení (*settings*) v hodinkách a na položku „o hodinkách" (*about*)
    b. Klikneme sedm krát 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 zařízením přes *„nastavení a možnosti vývojáře"*
        i. Pokud mobilní zařízení nebo tablet nemá danou možnost, přejdeme na možnost „*o telefonu*", nebo „*o tabletu*". Po klikáme sedm krát 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 sedm krát na číslo sestavení, poté by se mělo objevit možnost „*možnosti vývojáře*“ v nastavení

c. Nastavíme připojení android hodinky – mobilní zařízení. Na mobilním zařízením otevřeme aplikaci Android Wear, nahoře vpravo klikneme na menu a vybereme nastavení

d. Zapneme ladění přes Bluetooth



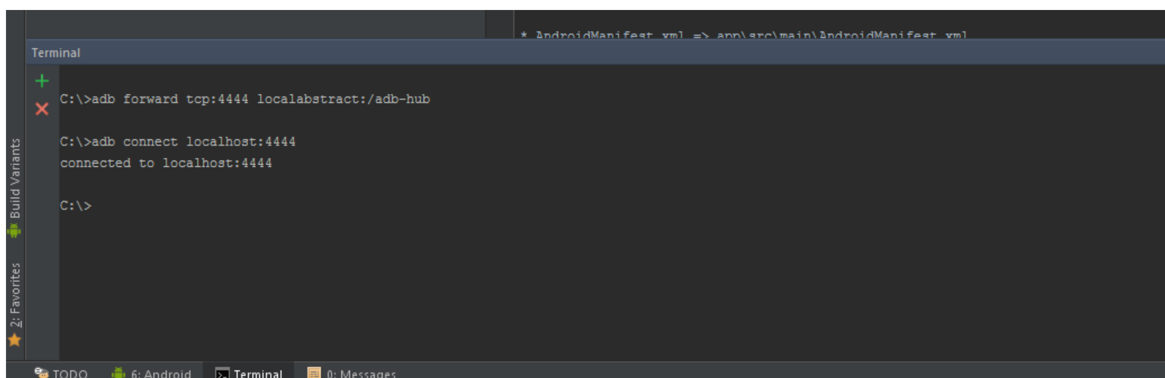
Obr. 1 Výsledný stav po zapnutí ladění přes Bluetooth

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

adb forward tcp : 4444 localabstract:/adb-hub

adb connect localhost:4444

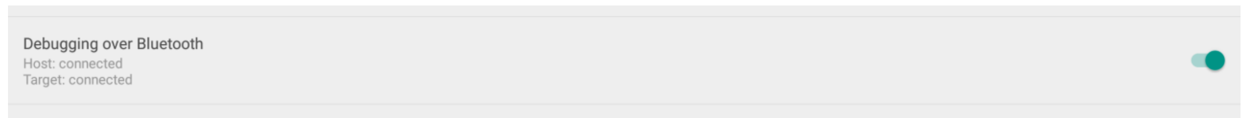
Následující příkaz můžeme spustit v klasickém příkazovém řádku podle operačního systému nebo přímo v android studiu



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. Na-

chází se v SDK Androidu: *Android\sdk\platform-tools*. Umístění je závislé na operačním systému.

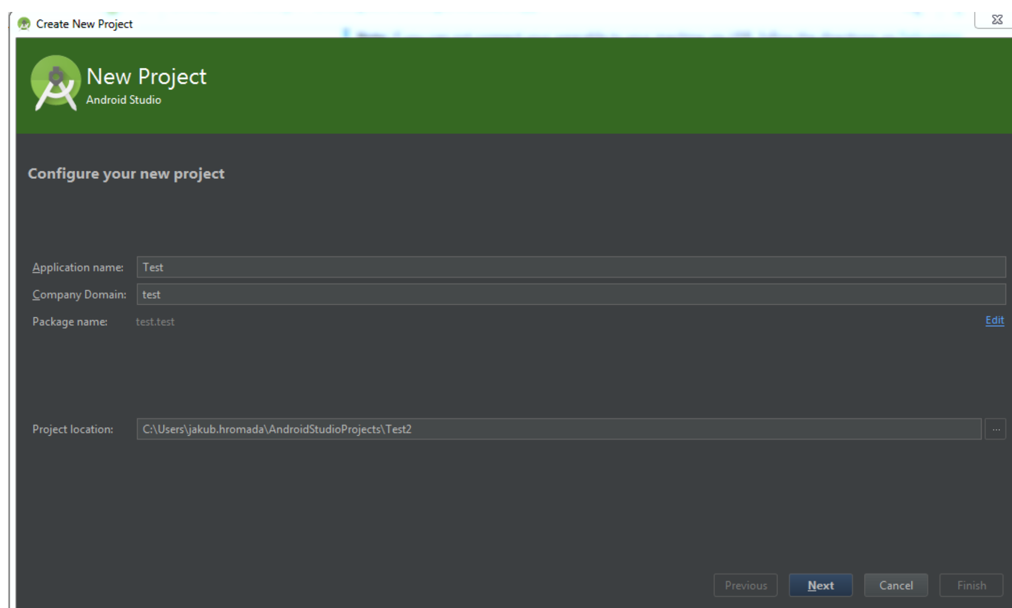


Obr. 3 Výsledné propojení mobilního zařízení a zařízení android hodinky

1.6 Vytvoření projektu

Pro to, abychom mohli začít s vývojem, je potřeba vytvořit projekt, který obsahuje modul pro android hodinky a aplikační modul, který je nahráván na mobilní zařízení. Ve vývojovém prostředí Android studiu klikneme na *File – New Project* a projdeme pomocníkem pro vytvoření projektu[6], [7].

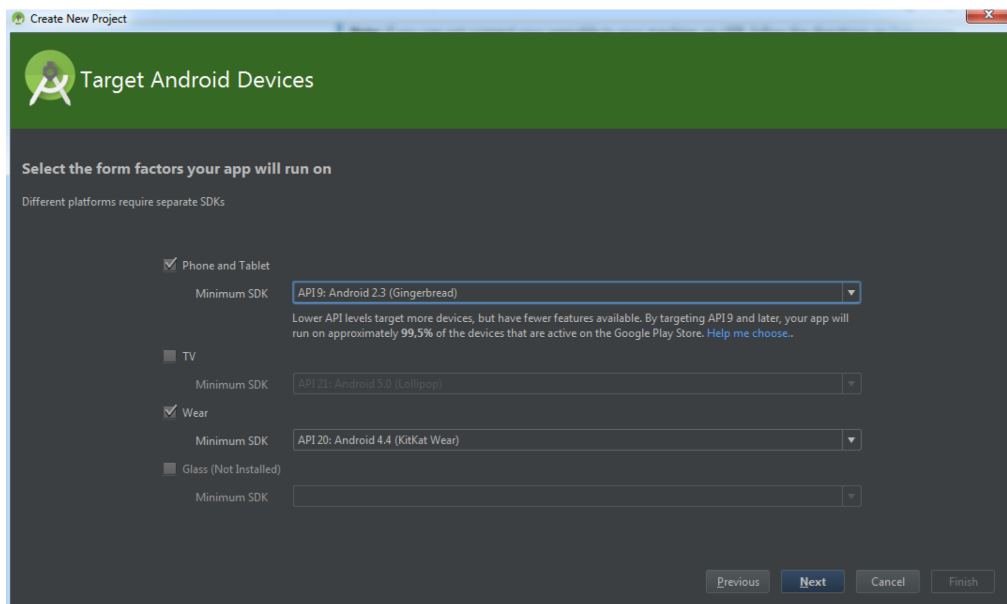
1. V konfiguračním okně vašeho okna, napíšeme jméno aplikace a jméno balíku



Obr. 4 Okno nového projektu

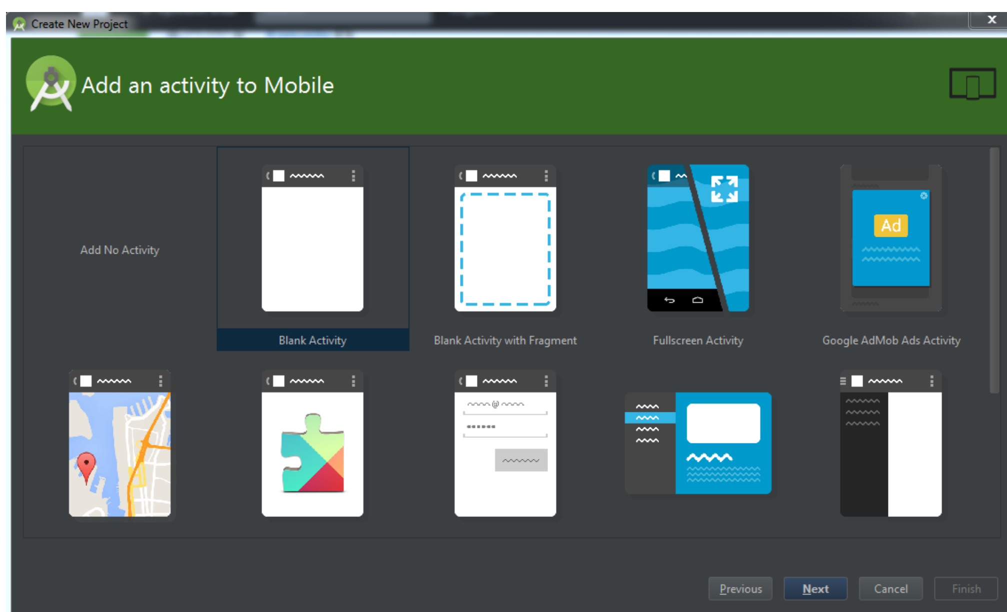
2. V dalším okně musíme zatrhnout „*Phone and Tablet*“ a „*Wear*“ pro vývoj aplikace na hodinky. Dále je nutné vybrat verzi vývojového API, který bude projekt využívat pro vývoj. Pro vývoj Android Wear je potřeba vybrat minimálně API 20 (Android 4.4 Kitkat). Výběr API pro mobilní zařízení. Zde záleží 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 pamětí, že v API 9 chybí spousta funkcí a musí se různě obcházet, oproti API 14, tudíž vývoj je složitější



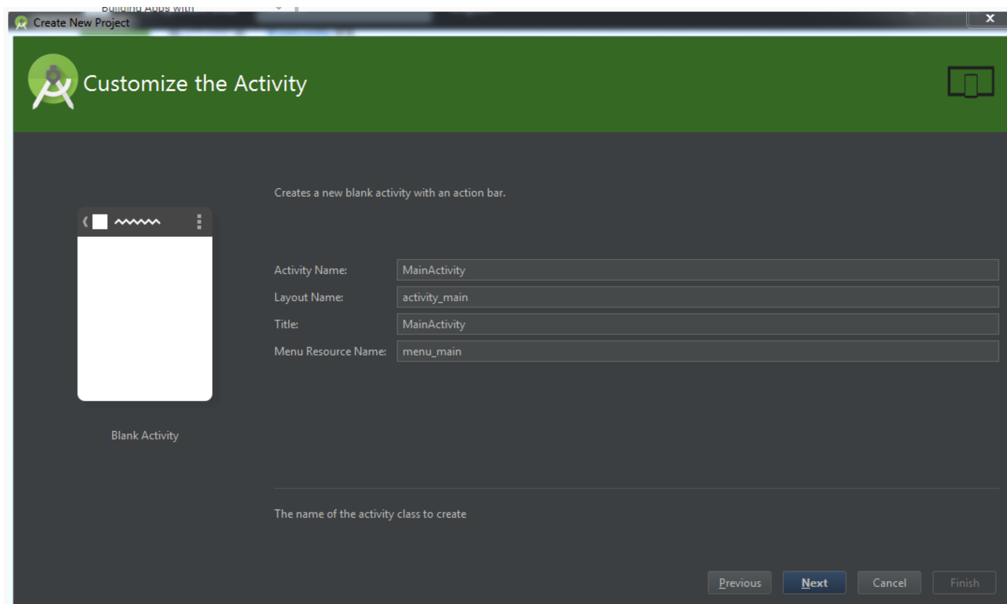
Obr. 5 Výběr API, které bude projekt využívat

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



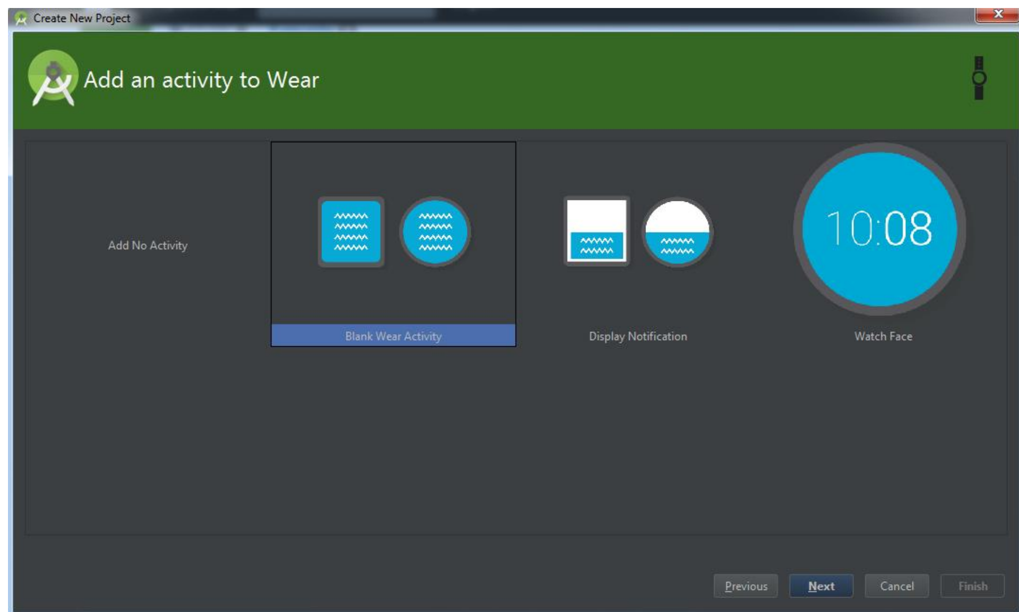
Obr. 6 Výběr šablon aktivit pro mobilní zařízení

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



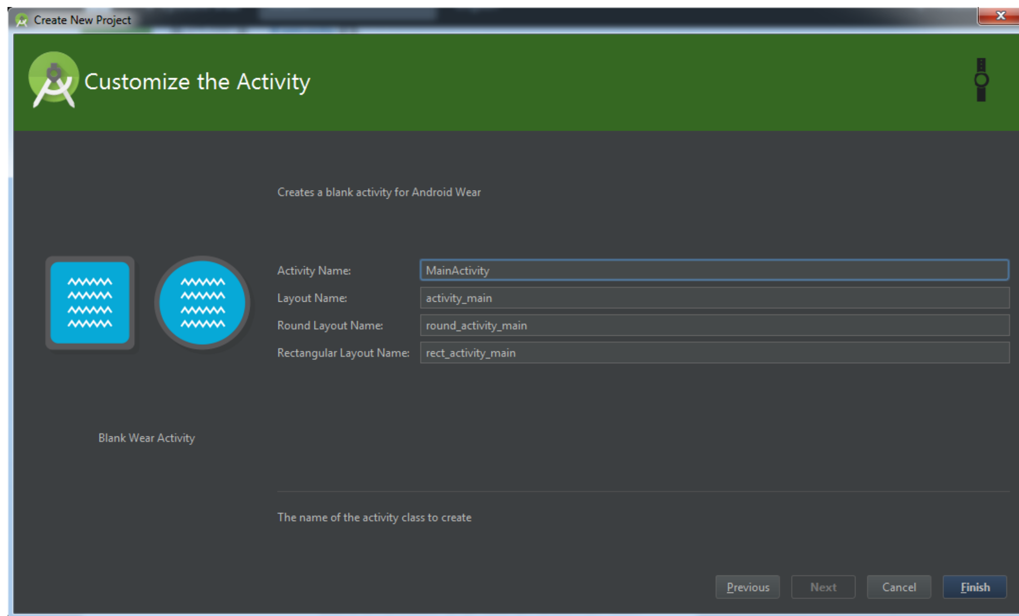
Obr. 7 Okno se zvolením názvu aktivity, třídy a layoutu

5. Dále vybíráme šablonu pro Android Wear. Google zde před definoval nějaké výchozí šablony, vybereme *Blank Wear Activity*



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 tvoření kontejneru pro mobilní zařízení. Hlavní věc, na kterou se musí myslet při vývoji, že daná výsledná aplikace musí být intuitivní, aby příslušný uživatel rychle porozuměl způsobu fungování aplikace [5].

1.7.1 Tvoření vlastních notifikací

Obecný princip je takový, že při přijetí notifikace na mobilním zařízením, by se měla automaticky synchronizovat daná notifikace 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)
{
// nastavení vzhledu dané aktivity
setContentView(R.layout.notification_activity_test);
}
```

2. Nadefinujeme vlastnosti dané aktivity v android manifestu. Pro povolení sdílení notifikací na Android hodinkách, se musí nastavit daná aktivita jako exportovaná, vkládána a nesmí sdílet prostředky s jiné aktivity

```
<activity android:name="com.test.MyTestDisplayActivity"
android:exported="true" <!-- exportovaná -->
android:allowEmbedded="true"    <!-- vkládána -->
 android:taskAffinity=""     <!-- ne sdílení prostředků -->
 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 pomocí UI knihovny

UI knihovna pro Android Wear je automaticky zahrnuta do *build.gradle*, 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 (hranaté kulaté)
- *CardFragment* – Fragment, který prezentuje obsah v rozšířitelné, vertikálně posouvací kartě
- *CircledImageView* – Kontejner pro zobrazení obrázků, vytvarovaná na kruh
- *CinfirmationActivitz* – 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. Přídání *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"  <!—vzhled pro hranaté displeje -->
  app:roundLayout="@layout/round_test_activity_wear">  <!—vzhled pro kulaté displeje
--> </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 vytvořenými XML layouty [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
// event listener pro načtení daného xml vzhledu a přiřažení jednotlivých manipulujících
//prvků
public void onLayoutInflated(WatchViewStub stub) {
TextView tv = (TextView) stub.findViewById(R.id.text);            ...
 }
});
}
```

1.8 Posílání a synchronizace data

API datové vrstvy android hodinek, která je součástí Google Mobile Services, poskytuje komunikační kanál pro mobilní zařízení a android hodinek. API se skládá z množiny datových objektů, které napomáhají synchronizaci dat mezi mobilním zařízením a android hodinek [4].

1.8.1 Požadavky

Pro přístup k dané API je potřeba Andoid 4.3 (API level 18) a vyšší na mobilním zařízení a nejnovější verzi Google Mobile 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 spouštění 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 synchronizaci[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 zařízení[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 Mobile Service APIs.[8]

GoogleApiClient poskytuje rozhraní, které usnadňuje tvoření daného klienta.

```
// vytvoření instance daného clienta

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

@Override

public void onConnected(Bundle connectionHint) {

// zde můžeme zahájit komunikaci

}

@Override

public void onConnectionSuspended(int cause) {

// pokud se ztratí spojení , musíme zajistit integritu funkcí jednotlivých prvků, které komunikují s danou službou

}

})

.addOnConnectionFailedListener(new OnConnectionFailedListener() {
@Override

public void onConnectionFailed(ConnectionResult result) {

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

}
```

```
})
.addApi(Wearable.API)
.build();
```

GoogleApiClient poskytuje 3 *callbacky*, první *onConnected* je pro úspěšné připojení k službě, ve které můžeme začít s komunikací s danou službou, druhý *onConnectionSuspended* je obsluha pro ztracení spojení, např třeba při dočasném odpojení přenosu a třetí poslední *onConnectionFailed* , pokud při pokusu se připojit dojde k chybě.

1.8.2.2 Synchronizace datových položek

DataItem definuje, datové rozhraní, které systém používá na synchronizace 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. Zavoláni metody *PutDataMapRequest.getDataMap()*, k vrácení objektu datové mapy, kde můžeme nastavovat hodnoty ve formě klíče – hodnoty.
3. Za pomocí metody *put(), kterou* voláme ve formě např. *putString()* vložíme pár do dané mapy.
4. Zavolání metody *PutDataMapRequest.asPutDataRequest()*, k vrácení objektu *Put-DataRequest*
5. Zavolání metody *DataApi.putDataItem()*, kde systém vytvoří danou datovou polož-ku

Pokud jsou mobilní zařízení a android hodinky odpojeny, data jsou vložena do vyrovnáva-cí 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. *DataMapa* funguje jako na principu klíč – hodnota. Daná struktura umí uložit všechny primitivní typy a taky sama sebe, to znamená vložení další *DataMapy* [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) {
…
 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);
}
protected void 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 *GoogleApiClienta* , zaregistrujeme jeho interface. V metodě *onResume()*, která se volá, když se daná aktivita oživí, čili hned po vytvoření, nebo po přejití do popředí. V dané metodě připojíme *GoogleApiClienta* 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 *GoogleApiClienta* [8].

1.8.2.5 Zasílání velkých dat

Pro posílání velkých celků binárních data přes bluetooth , jako obrázky, je implementované rozhrání 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 automatický cachuji, což je prevence znovu přeposílání data 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);
…
 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 třida. 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 komunikace, 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();
    ConnectionResult connectionResult = googleApiClient.blockingConnect(
        Constant.GOOGLE_API_CLIENT_TIMEOUT_S, TimeUnit.SECONDS);
```

```
    if (connectionResult.isSuccess() && googleApiClient.isConnected()) {
        Iterator<String> itr = Utils.getNodes(googleApiClient).iterator();
        while (itr.hasNext()) {
            Wearable.MessageApi.sendMessage(
                    googleApiClient, itr.next(), constant, body);
        }
        googleApiClient.disconnect();
    }
}
```

Připojíme *GoogleApiClienta* a pomocí funkce *GetNodes*(*googleApiClient*) si vrátíme jednotlivé a 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())) {
        SendService.clearNotification(this);
    } else if (Constant.START_NAVIGATION_PATH.equals(messageEvent.getPath())) {
        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);
```

```
    }
  }
```

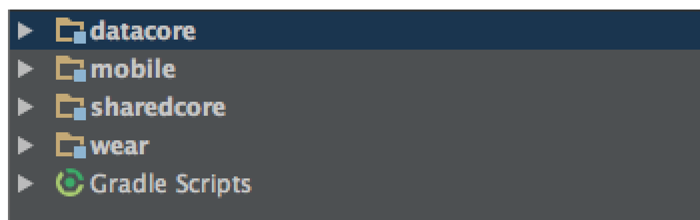
V dané metodě je důležitý parametr *messageEvent*, který obsahuje unikátní cestu, podle které provedeme specifickou akci, a jako další obsahuje data, do kterých se ukládá posloupnost bajtů, čili obsah dané zprávy[3]

II. PRAKTICKÁ ČÁST

2 VÝVOJOVÁ APLIKACE

Pro ukázku vývoje pro android hodinky bylo zvoleno téma aplikace, registrace zkouškových termínu pomocí portálu UTB. Využívá se veřejně dostupná REST služba UTB - *http://stag-ws.utb.cz/ws/help/* . Aplikace běží jak na mobilním zařízením tak na android hodinkách umí zaregistrovat daného studenta na daný termín. Zajišťuje autentizaci uživatele. Dále byla vytvořena pomocná služba, která běží na Google App Enginu, která kontroluje, jestli nepřibyl nový termín, a další pomocná služba která poskytuje testovací data a funkce na otestování dané aplikace, taky běžící na Google App Enginu.

2.1 Struktura projektu



Obr. 10 Struktura ukázkového projektu

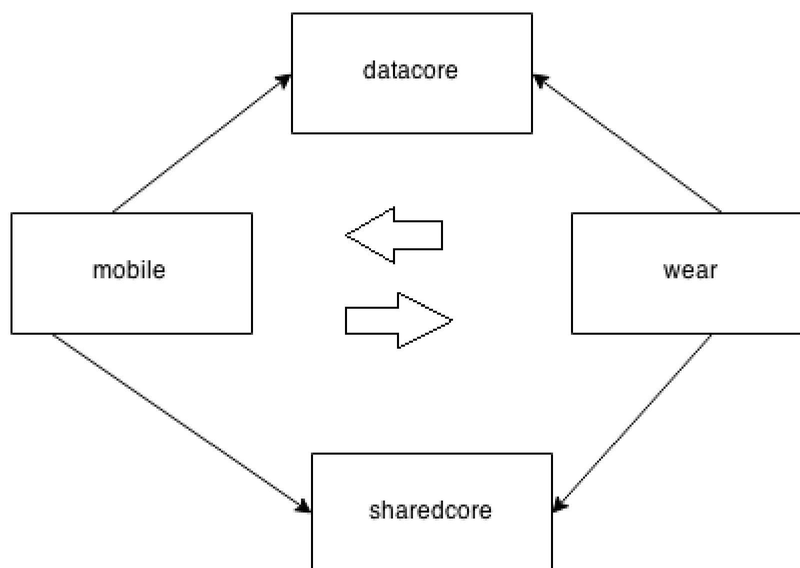
Projekt se skládá z:

1. **Modul datacore** – Daný modul je zodpovědný za komunikaci s danou službou UTB a obsluhou daných odpovědí z dané služby, taky se stará parsování daných odpovědí a převodu na daný objekt. A poskytuje interface, pomocí kterých aplikační vrstva komunikuje s danou datovou vrstvou. Díky interface, je možné lehce přepínat mezi ostrým serverem a testovacím serverem

2. **Modul mobile** – Daný modul, je aplikace na mobilní zařízení. Zde voláme dané služby z datacore a synchronizujeme vrácené data na hodinky a zpět na mobilní zařízení. Daný modul i přijímá zprávy z modulu wear. Jako jestli je uživatel přihlášený. Taky se stará o přijetí notifikaci ze vzdálené služby a její předání aplikaci na Android hodinkách

3. **Modul sharedcore** – V daném modulu jsou metody a konstanty které používají jak modul mobile tak modul wear. Dané konstanty slouží k tomu jako identifikace cest kde jsou uložená synchronizovaná data mezi modulem wear a modulem mobile.

4. **Modul wear** – Daný modul, je aplikaci na android hodinky. Je tam základní logika fungování aplikace, příslušné layouty, obrázky pro danou aplikaci, a naslouchání změn z mobilní aplikace



Obr. 11 Jednotlivé závislosti mezi modulami

Modul *mobile*, má závislost na modulu *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 tvoři z odpovědí dané služby. Další závislost je mezi modulem *mobile* a modulem *wear*, ale daná závislost není přes reference ale přes Google Mobile services.

2.2 Modul Datacore

2.2.1 Třídy

Daný modul, se skládá ze tříd, které se využívají ke komunikování s danou UTB službou.

Ukázka dané třídy:

```
public class Deadline {

   public static final String PARA_DEADLINE_ID = "termIdno";

   @SerializedName("termIdno")

   public int DeadlineID;
```

```
public static String createJsonList(List<Deadline> deadlines)
{
    Deadline [] _deadlines = deadlines.toArray(new Deadline[deadlines.size()]);
        // vytvoření JSON pomocí GSON
    return new Gson().toJson(_deadlines,Deadline[].class);
}
public static List<Deadline> parse(String result)
{
        ...
        // parsování pomocí GSON
            Deadline[] deadlines  = new Gson().fromJson(result, Deadline[].class);
            deadlinesList = Arrays.asList(deadlines);            ...
        return deadlinesList;
}
```

Každá třída obsahuje jednotlivé proměnné, do kterých se mapuje hodnota z dané služby. Ke snadnějšímu mapování využívám knihovnu *Gson*, která podle pomocného attributu *@SerializedName* přidělí danou hodnotu. Třída obsahuje statickou metodu *parse* kde se pomocí *Gson*, vytvoří daný objekt prezentovaný danou třídou z výsledného řetězce, který přišel ze služby. Taky obsahuje statickou metodu *createJson*, která vytvoří řetězec ve formátu *JSON* , který prezentuje danou třídu *deadline*.

2.2.2 Volání HTTP metod

Na volání daných služeb se využívá *AsyncTask*. Jelikož volání metod dané služby není vyřízeno hned, je nutné daný požadavek vyřídit na pozadí, aby neblokovalo hlavní UI vlákno. A poté naslouchat v daném *callbacku onPost* o výsledné odpovědi.

Ukázka obecné implementace třídy, která obsluhuje dané požadavky a odpovědi :

```
private class HttpRequest extends AsyncTask<Input, Integer, ResultHttpService> {
        protected ResultHttpService doInBackground(Input... input) {
```

```
try {
...
    java.net.URL url = new URL(_input.Url);
    HttpURLConnection httpCon = (HttpURLConnection) url.openConnection();
//nastavení druhu metody požadavku
    httpCon.setRequestMethod(_input.TypeRequest.toString());
   httpCon.setRequestProperty("Cookie", TextUtils.join(",", cookies));
   ...
          // zapsání těla požadavku
          byte[] outputInBytes = body.getBytes("UTF-8");
          OutputStream os = httpCon.getOutputStream();
          os.write(outputInBytes);
          os.close();
// vrácení odpovědi
    response = httpCon.getResponseCode()
    }
// zpracujeme výsledný obsah
    BufferedReader reader = new BufferedReader(new InputStreamRea-
der(httpCon.getInputStream()));
    builder = new StringBuilder();
    for (String line = null; (line = reader.readLine()) != null; ) {
       builder.append(line).append("\n");
    }
return new ResultHttpService(builder.toString(), response,_input.Service,headers);
}
```

```
    protected void onPostExecute(ResultHttpService result) {
        // zde předáme výsledek do vyšších vrstev projektu
    }
```

V metodě *doInBackground()*, spustíme požadavek na metodu služby testovacího nebo ostrého serveru pomocí *HttpURLConnection*. Dané požadavky musí být autentizované. K autentizaci požadavku se využívá autentizační cookie, pomocí třídy *CookieManager*, která ukládá permanentně cookie. U testovacího serveru autentizace neprobíhá, pouze simulovaná. Poté co se vrátí response z *HttpURLConnection* načteme daný *stream* dat do *BufferReader* a sestavíme výslednou odpověď ve formě *stringu*. V metodě *onPostExecute()* předáme výsledek daného požadavku pomocí callbacku *Result*.

2.2.3 Pomocné služby na volání AsyncTasku

Obsluhy, které volají daný *AsyncTtask HTTP* požadavku a volají dané parsování odpovědí ze služby, jsou servisy – pomocné třídy které ulehčují logiku zpracování dotazu. Je to třída, která má metody podle URL, kterou voláme z dané služby UTB nebo z testovacího serveru a má obsluhu odpovědí z *AsyncTasku*.

Ukázka kódu:

```
// třída implementuje interface
public class DeadlineUTB extends BaseService implements IDeadlineRequest,IResult {
// url požadavku
   public static final String GET_DEADLINES_STUDENT = "servi-
ces/rest/terminy/getTerminyProStudenta";
// inject odpovědi z modulu mobile
   public IDeadlineResponse iDeadlineResponse;
// nastavení vstupu do async tasku
private void setPropertiesGetDeadlines()
   {
        ...
      input.Init();
```

```
        input.SetCookie = true;
          // vytvoření volací URL
        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;
    }
// volani požadavku
    public void GetDeadLines() {
        setPropertiesGetDeadlines();
        BaseHttpService httpService = new BaseHttpService(this);
        httpService.startExcecute(input);
    }
          // asynchroní vrácení dotazu
    public void Result(ResultHttpService result) {
        if(result == null){
          // fatalní chyba
        }
        if(result.resultCode != HttpURLConnection.HTTP_OK)
        {
          // odpověď přišla ale s chybou
        }
        DeadlineResponse deadlineResponse = new DeadlineResponse();
```

```
// typ odpovědi
 switch (result.service)
      {
        case DEADLINES_STUDENT:
            List<Deadline> deadlines = WrapperDeadline.parse(result.result);
         // parsování a předání do vyších vrstev
            break;
```

Každá veřejná metoda dané třídy řídí vstup do daného *asyncTasku* pomocí třídy *Input*, zajišťuje předání autentizační *cookie*, řídí formát URL požadavku. UTB služba má dva druhy výstupních formátů. JSON a XML. V metodě *Result* je vrácená odpověď z dané služby, kde se rozhoduje podle kódu odpovědi (jestli daná odpověď byla úspěšná, čili je vrácena 200, nebo neúspěšná cokoliv kromě 200). Poté výsledný *String* ze služby je zpracován pomocí metody *parse* a je vytvořený výsledný objekt a předán dál do mobilního modulu. Každá taková třída má duplikát, kvůli testovacímu serveru, implementuje ty samé volací metody ale s jiným tělem. V aplikaci jde přepnout mezi testovacím a ostrým provozem u přihlašovací aktivity.

Ukázka interface:

```
public interface IDeadlineRequest {
   /*Request*/
   void GetDeadLines();
   List<Deadline> GetDeadLinesSync();
```

V tomhle případě jde o volání vrácení termínu. Kterou třídu použít pro volání daných požadavků podle zvolené služby rozhoduje třída *ApiCreateHelper*.

Ukázka *ApiCreateHelper*:

```
public IDeadlineRequest getInstance(Context context,IDeadlineResponse iDeadlineResponse)
   {
         if(api.compareTo(UTB_API) == 0)
```

```
        {
            return new DeadlineUTB(context,iDeadlineResponse);
        }else{
            return new DeadlineUTBTest(context,iDeadlineResponse);
        }
    }
```

2.3 Mobile

Modul mobile, se stará o veškeré volání datové vrstvy a přenášení dat do *wear* modulu. Také je zodpovědný za vyřízení zpráv a posílání zpráv do android hodinek. Využívá *IntenService* a *GoogleApi* pro posílání zpráv a synchronizaci data mezi daným mobilním zařízením a android hodinkami. Stará se taky o přijetí notifikací ze vzdálené služby a jejím následném předání do android hodinek.

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 nastartována, běží v danou chvíli na pozadí a vykoná tělo metody *onHandleIntent* a tím skončí. Veškerá komunikace a synchronizace musí být implementovaná asynchronně, pomocí callbacku, nebo volaná synchronně ale ve vlákně na pozadí.

```
public class SendService extends IntentService {
    public static final String ACTION_LISTS_DEADLINES = "list_deadlines";
    public static void triggerDeadlines(Context context) {
…
            Intent intent = new Intent(context, SendService.class);
            intent.setAction(SendService.ACTION_LISTS_DEADLINES);
            context.startService(intent);
..
    }
```

```
    // vykonání akce na pozadí
    protected void onHandleIntent(Intent intent) {
        String action = intent != null ? intent.getAction() : null;
if (ACTION_LISTS_DEADLINES.equals(action)){
            refreshDeadlines();
        }
// zavolání synchroně dotaz na službu
private void refreshDeadlines){
        SDeadline sDeadline = new SDeadline(this);
        List<Deadline> deadlines = sDeadline.GetDeadLines(true);
        sendDataToWearable(deadlines);
    }
// pošleme data do android hodinek
    private void sendDataToWearable(List<Deadline> deadlines) {
// vytvoříme google api clienta
        GoogleApiClient googleApiClient = new GoogleApiClient.Builder(this)
                .addApi(Wearable.API)
                .build();.
// pokusíme se připojit
        ConnectionResult connectionResult = googleApiClient.blockingConnect(
                Constant.GOOGLE_API_CLIENT_TIMEOUT_S, TimeUnit.SECONDS);
        ArrayList<DataMap> deadlinesData = new ArrayList<>(deadlines.size());
// vložíme data do dataMapy
        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);
}
// pokud jsme připojení odešleme data
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();
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()));
}
}
```

```
googleApiClient.disconnect();
  }
}
```

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

2.3.2 Zasílání zpráv

Zasílání zpráv mezi mobilní aplikací a aplikací na Android hodinkách, se využívá na zjišťování různých stavů, jako jestli je přihlášen, provedení akce odhlášení, přihlášení, jestli jsem provedl registraci na termín. Na zasílání zprav z mobilního zařízení do zařízení android hodinek, se využívá opět *IntentService*, a volání zpráv z android hodinek, se využívá asynchronního volání.

Zde je, metoda na zasílání zprav do android hodinek :

```
private void sendMessage(String constant,byte [] body)
  {
       // instance API klienta
     GoogleApiClient googleApiClient = new GoogleApiClient.Builder(this)
          .addApi(Wearable.API)
          .build();
// připojení k Google mobile service
     ConnectionResult connectionResult = googleApiClient.blockingConnect(
          Constant.GOOGLE_API_CLIENT_TIMEOUT_S, TimeUnit.SECONDS);
     if (connectionResult.isSuccess() && googleApiClient.isConnected()) {


       // vracení připojených uzlů
```

```
Iterator<String> itr = Utils.getNodes(googleApiClient).iterator();
        while (itr.hasNext()) {
            Wearable.MessageApi.sendMessage(
                    googleApiClient, itr.next(), constant, body);
        }
        googleApiClient.disconnect();
    }
}
```

Princip je, že nejprve se ověří, jestli bylo úspěšné připojení na GMS službu a poté můžeme pomocí *MessageApi*, poslat zprávu do android hodinek. Dané metodě *MessageApi.sendMessage* , se musí předat cesta, která určuje akci na hodinka a *bytové pole*, do kterého můžeme předat obsah zprávy. Protože je parametr *byte Array*, můžeme do něho předat cokoliv, když to správně převedeme na posloupnost *byte*. A po přijetí to zase správně převedeme na typ, před zakódováním do *byte pole*.

2.4 Modul Sharedcore

Daný modul obsahuje pouze třídu Utils a třídu Constant. Je to pomocný modul na sdílení cest a metod mezi mobilní aplikací a aplikací hodinek.

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 zařízením 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 zařízeních, které se používá při posílání zpráv.

2.5 Modul 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 GMS příslušná data a zobrazují je. Poté pomocí *DataMap* se synchronizuje vybraný termín mezi mobilem a hodinkami.

2.5.1 Vrácení dat z Google API

2.5.1.1 AsyncTask

Pomocí třídy *AsyncTask* si načteme data z Google API datové vrstvy. Při spuštění aplikace se spustí daný AsyncTask

```
@Override
    protected ArrayList<Deadline> doInBackground(Uri... params) {
        GoogleApiClient googleApiClient = new GoogleApiClient.Builder(mContext)
```

```
                .addApi(Wearable.API)
                .build();
// připojení na GMS službu synchronně
 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;
            }
         //  vraceni DataItemu z GMS
            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);
               for (DataMap deadlineData : deadlinesData) {
                  Deadline deadline = new Deadline();
                  deadline.Chamber = deadlineDa-
ta.getString(Constant.Deadline.CHAMBER);
               }
            }
            googleApiClient.disconnect();
            return deadlines;
```

```
        }
```

Čtení z googleApi musíme provádět asynchronně proto asyncTask.

2.5.1.2 Naslouchání dat z WearableListenerService

Daná třída naslouchá změny v datové vrstvě *GoogleApiClient*. Čili cokoliv se změní, v daných datových objektech se poté zavolá daná metoda.

```
public class ListenerService extends WearableListenerService {
   public void onDataChanged(DataEventBuffer dataEvents) {
        // vracení kolekce změn
      final List<DataEvent> events = FreezableUtils.freezeIterable(dataEvents);
        // iterace
      for (DataEvent event : events) {
        // podle konstanty DataEvent.TYPE_CHANGED zjistíme že nastala změna
        if (event.getType() == DataEvent.TYPE_CHANGED
              && event.getDataItem() != null
              &&
Costant.DEADLINES_PATH.equals(event.getDataItem().getUri().getPath())) {
//provedeme patřičnou akci
        }
…
```

Po nastání změny si vrátíme kolekci *eventu*, v ní je uložena změna. Pokud v průběhu iterace příslušný *event* má nastavený *getType()* na *DataEvent.TYPE_CHANGED*, tak víme že nastala změna, daná položka obsahuje cestu k daný datům, takže si můžeme vrátit příslušná data a provést konkrétní akci

2.5.2 Notifikace

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

Ukázka kódu:

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
.setContentTitle("Počet nových termínu " + 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 notifikační *builder* vytvoří několik stránek podle toho kolik je nových termínu. A pokud, klikneme na konkrétní stránku, přejdeme na detail daného termínu.

2.6 Rozbor časové náročnosti na nasazení na další smartwatch zařízení

Systém android wear, disponuje mnoho pomocných komponent, které usnadňují přizpůsobení pro různá smartwatch zařízení. Velmi nápomocné je i android studio, které poskytuje mnoho layoutu daných zařízení, a můžeme tak nasimulovat jak bude vypadat naše aplikace na jiném smartwatch zařízení.

2.6.1 Přizpůsobení aplikace na tvar displeje dalšího smartwatch zařízení

Všechny aplikace na smartwatch zařízeních mají určité uživatelské rozhraní, přes které daný uživatel ovládá danou aplikaci. Vzhled uživatelského rozhrání se mění podle tvaru displeje daného smartwatch zařízení. V základu jsou dva druhy kulatý a hranatý. U hranatého tvaru, můžeme obsah roztáhnout na okraj daného displeje, s určitou mezerou mezi okrajem a obsahem. U kulatého displeje, musíme vystředit daný obsah na střed. Tudíž přizpůsobení aplikace tvaru displeje zabírá největší část časové náročnosti na nasazení odlišného smartwatch zařízení.

Podle komplexnosti daného uživatelského rozhrání se odvíjí časová náročnost. Čím je složitější uživatelské rozhraní dané aplikace (různé animace, přenesení textu …) tím je přenesení na jiný tvar displeje složitější.

2.6.1.1 Přizpůsobení tvaru displeje testovací aplikace

U testovací aplikace jsem využil systémovou komponentu *BoxInsetLayout* , která hlídá odsazení od krajů displeje, podle tvaru displeje. Tudíž, přizpůsobení displeji vyřeší za mě.

Obr. 12 Přizpůsobení aplikace displeji pomocí BoxInsetLayout

Ukázka kódu:

```
<android.support.wearable.view.BoxInsetLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content">

    <FrameLayout
        android:id="@+id/deadline_frame"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        app:layout_box="all">

        <android.support.wearable.view.WearableListView
            android:id="@+id/deadline_wearable_list"
            android:layout_height="match_parent"
            android:layout_width="match_parent">
        </android.support.wearable.view.WearableListView>

    </FrameLayout>
</android.support.wearable.view.BoxInsetLayout>
```

2.6.2 Synchronizace dat a posílání zpráv

Synchronizace dat a posílání zpráv funguje nezávisle na druhu smartwatch zařízení, ovšem jedná-li se pouze a jenom o zařízení na kterém běží systém android wear. Čili nasazení dané testovací aplikace na jiné smartwatch zařízení by se daná logika zpracování dat a zpráv zůstala neměnná.

2.6.3 Využití senzorů

Každé smartwatch zařízení disponuje základními senzory a dodatečnými senzory které můžou a nemusí mít dané druhy smartwatch zařízení. Je důležité, aby každé volání api

daného senzoru, bylo ošetřeno, jestli dané zařízení disponuje danými metodami. Pokud ne hrozí pád aplikace.

Ukázka kódu:

mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

Sensor mHeartRateSensor = mSensorMa-nager.getDefaultSensor(Sensor.TYPE_HEART_RATE);

Kontrola na null mHeartRateSensor

2.6.4 Shrnutí

Časová náročnost je tedy minimální co se týče přenosu testovací aplikace na jiné smartwatch zařízení. Pokud disponujeme různými zařízeními, tak danou aplikaci můžeme ihned přenést pomocí android studia na dané zařízení. Pokud správně využívá systémové komponenty android wearu, měla by aplikace na odlišném zařízení fungovat.

2.7 Srovnání s ostatními programovacími jazyky

Programování pro android wear a celkově pro mobilní platformu android se vesměs programuje v javě, kontrétně v JDK 1.7, JDK 1.8 ještě není implementovaná v SDK. Samozřejmě je možnost programovat dané aplikace v jiném programovacím jazyce než je java.

2.7.1 HTML 5 javascript

Jelikož některé Smartwatch disponuji webovým prohlížečem, tak je možné programovat přes HTML 5 Javascript. Bohužel, tímto způsobem jsme značně omezení jak optimalizačně protože jsme, vazba na webový prohlížeč daného zařízení, tak z hlediska přístupu k hardwarovému API daného zařízení. Ve výsledku tedy nevyužijeme plný potenciál daného zařízení, navíc jsme ochuzeni o systémové komponenty, které poskytuje daný android SDK. Využijeme jen klasické HTML5 API a programovací jazyk Javascript., které poskytuje daný webový engine. A jsme závislí na webovém prohlížeči a WI-FI. Ale na druhou stranu obrovská výhoda plyne z toho, že jedna webová aplikace může být portována na zařízení, které disponuje internetovým připojením a webovým enginem, čili to zahrnuje desktop, mobilní zařízení, čtečka knížek, chytrá televize, ….

Existuje řešení, které odstraňuje závislost na webovém prohlížeči a na připojení k internetu. Projekt se jmenuje CocoonJS. Kompiluje se v cloudu a využívá HTML5 canvas k programování aplikace pro android wear a taky mobilní platformu jak android tak

iOS. Napsaný kód se nahraje na daný cloud zkompiluje se a vratí se dané apk které se může ručně nahrát na dané zařízení. Můžeme přistupovat k hardwarovému api daného zařízení. Bohužel nemůžeme využít DOM tagu jako <div>,<table>. Až vyšším android 4+ se využívá komponenta WebView které komunikuje s HTML canvasem a je možné využívat dané tagy. Bohužel zde je omezení, že nevyužijeme plný potencionál vývojového prostředí, musíme portovat dané APK ručně.

2.7.2 XAMARIN

Projekt xamarin zahrnuje vývoj mobilních zařízení andrid, iOS, windows phone a také android wearu. Vývojovým jazykem je C#. Poskytuje plný vývoj a přístup k android SDK. Výhoda je že daný backend aplikace je možné převést na úplně jinou platformu jako je třeba iOS. Horší je to s frotendem (UI aplikace), tam už je portace na jinou platformu složitější. Rozdíly mezi vývojem android wearu pomocí javy nebo c# není důležitý, zde jsou spíše subjektivní důvody, oba jsou plnohodnotnými objektovými programovacími jazky. Xamarin má největší nevýhodu a to že není zadarmo, má verzi která je zdarma ale nedisponuje plnohodnotnými nástroji jako disponuje android studio.

ZÁVĚR

Cílem práce bylo, zmapovaní programovacích možností Android Wear za pomocí Android API, Google API a Android Wear API v jazyce Java. Dané vývojové API poskytuje mnoho pomocných komponent a funkcí, které usnadňují vývoj aplikací na android hodinky. Jako vývojové prostředí bylo využito Android studio, které je dostupné zdarma. Byla naprogramována a odladěna testovací aplikace pro Android hodinky, jako ukázka vývoje v dané technologii. K vývoji testovací aplikace bylo použito zařízení Motorola MOTO 360, jako vývojový prostředek. Vyvinutá aplikace slouží jako prostředek na zápis na zkouškové termíny UTB, byla použita ostrá verze veřejně dostupné služby UTB. Výsledná aplikace autentizuje uživatele a vrátí příslušné termíny. Na každý termín se lze přihlásit, pokud splňuje požadavky. Byli využity základní systémové komponenty dostupné v Android Wear SDK, jako ukázka práce s nimi. Bylo nutné implementovat dva pomocné servery. První napomáhal k notifikaci nových termínu pro přihlášeného uživatele, druhý pomocný server sloužil k otestování testovací aplikace na android hodinkách, kvůli nedostatku testovacích dat z ostré služby UTB. Aplikace byla odladěna a otestována jak ze strany ostré služby UTB, tak ze strany testovacího serveru. Bylo porovnáno a popsáno programování pro android s jinými programovacími jazyky. V dnešní době vzniká trend, kdy jeden programovací jazyk vládne všem a výslednou vyvinutou aplikaci je možné přenést na jednotlivé typy mobilního zařízení, Android, iOS, Windows Phone, včetně android hodinek. Jeden z takových vývojových prostředků je Xamarin, nebo HTML5 a Javascript. Byla popsána časová náročnosti přenesení výsledné testovací aplikace na jiný druh Smartwatch zařízení. Časová náročnost je, minimální co se týče přenesení na odlišné android hodinky. Celkově android hodinky, mají podle mě velký potenciál, kvůli rychlosti vyřízení jednoduchý úkonu, jako je vyřízení oznámení, které přišlo ze spárovaného zařízení. Co se týče programování a dostupného API z android SDK, má Google precizně zdokumentované dané rozhraní, včetně mnoha příkladu a názorného použití. Bohužel hodinky mají jednu obrovskou nevýhodu a to je výdrž baterie. Hodinky vydrží nanejvýš den při plném využití. A při vývoji musí být neustále na nabíječce, protože ladění je hodně energicky náročné.

SEZNAM POUŽITÉ LITERATURY

[1] RUIZ, David Cuartielles. Professional android wearables. Indianopolis: Wrox, Frebruary 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

HTML HyperText Markup Language

DOM Document Object Mode

UI User Interface

REST Representational State Transfer

SDK Software development kit

XML Extensible Markup Language

HTTP Hypertext Transfer Protocol

URL Uniform Resource Locator

ADB Android Debug Bridge

GMS Google Mobile Services

WIFI Wireless Ethernet Compatibility

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 14

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

Obr. 12 Přizpůsobení aplikace displeji pomocí BoxInsetLayout 50

SEZNAM PŘÍLOH

P1: nosič CD-ROM