Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Pilotní projekt Android wearable zařízení v prostředí ZČU

Plzeň, 2016 Jiří Plaček

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne ....

Jiří Plaček

Abstrakt

Tato bakalářská práce popisuje vývoj aplikace pro platformu Android wearable. První část práce se věnuju popisu zařízení, na kterých může platforma Android běžet, především wearable zařízení. Další část se zabývá možnostmi tvorby pro wearable zařízení a návrhem aplikace vhodné do prostředí ZČU. Aplikace bude využívat webové služby IS/STAG k získání dat zobrazovaných uživateli. Poslední část je pak věnována tvorbě a testování této aplikaci a navrhnutí možných rozšíření.

Abstract

The purpose of this work is to describe development on Android platform, especially Android wearable platform. First part is describing devices, which are capable to run Android system. Next part is describing how is possible to create wearable applications and design new wearable app usable in UWB campus. Application will use web services of IS/STAG to retrieve data. Last part is dedicated to testing of this app and suggesting new extensions options.

Obsah

1. Úvod

V současné době se mobilní telefony staly součástí každodenního života téměř každého člověka. „Chytré“[1] telefony jsou dnes rozšířeným zařízením a hlavně mladí lidé, například studenti, používají tato zařízení každý den. Pomocí těchto zařízení lze dnes ověřit, jaké bude počasí, koupit elektronickou jízdenku do tramvaje nebo objednat letenky na dovolenou. Lze jím nahradit také fotoaparát a kapesní konzole, protože mobilní telefony dnes mohou zastat mnoho činností. Tomuto trendu se také přizpůsobili výrobci hardwaru a softwaru a mobilní telefony je možné pomocí různých aplikací obohatit o téměř jakoukoli funkci.

V poslední době se ovšem do popředí dostává trend takzvaných wearable (nositelné) zařízení. Do této kategorie patří například fitness náramky nebo hodinky. Tato zařízení mohou sbírat data o uživateli (například při cvičení) a poté je synchronizovat s aplikací v jeho mobilním telefonu, nebo se mohou snažit minimalizovat potřebu vytahovat telefon na cestách díky zobrazovaní notifikací na jejich displeji (například hodinky), který má uživatel více po ruce. S rychlým rozšiřováním těchto zařízení začíná vznikat potřeba vytvářet pro ně nové aplikace nebo optimalizovat a rozšiřovat stávající mobilní aplikace pro spolupráci s hodinkami.

Wearable zařízení (a hlavně hodinky) jsou ideálním nástrojem, jak studentům (a potažmo lektorům) poskytnout rychlý přehled o jejich časovém rozvrhu bez nutnosti kontrolovat celý rozvrh online nebo v aplikaci v jejich mobilním telefonu. Protože žádná podobná aplikace pro weareable zařízení prozatím v prostředí ZČU neexistuje, klade si tato práce za cíl navrhnout a realizovat aplikaci, která vhodným způsobem poslouží studentům a vyučujícím, a mimo jiné poskytne přehled o jejich časovém rozvrhu týkajícím se ZČU.

1.1. Cíle práce

Cílem této práce je prozkoumat typy zařízení, na kterých může běžet platforma Android, zejména wearable zařízení. Dále by práce měla poskytnout náhled do možností tvorby aplikací pro tato zařízení a popsat, jak na tato zařízení programovat a co lze například vytvořit.

V práci bude také navržena aplikace vhodná do prostřední ZČU, která bude využitelná na Android wearable zařízeních. Tato aplikace bude využívat webové služby IS STAG a bude poskytovat přehled o časovém rozvrhu studentů či vyučujících.

Za poslední a hlavní cíl si tato práce klade vytvoření této aplikace a její otestování v praktických případech použití.

[1] Chytrý telefon (nebo také smartphone) je telefon, který obsahuje pokročilý operační systém (OS) kombinující možnosti klasického desktopového OS s novými funkcemi užitečnými pro mobilní telefon a aplikační rozhraní umožňující instalaci či úpravu programů.

2. Zařízení na platformě Android

Systém Android běží na velkém množství zařízení. Počínaje mobilními telefony přes tablety a televize až k herním konzolím. Tato kapitola stručně popisuje jednotlivé typy zařízení, na kterých může tento systém běžet.

2.1. Mobilní telefony

Pravděpodobně nejrozšířenějším zařízením, na kterém běží systém Android, je mobilní telefon. Dle statistických údajů více jak osmdesát procent mobilních telefonů obsahuje systém Android [1]. Chytré telefony s tímto systémem mohou přistupovat k internetu nebo využívat velké množství aplikací (jako například poznámky, počasí, kompas a dále) a her. Většina zařízení nabízí možnost využívat, kromě mobilního připojení, také Bluetooth, WiFi nebo GPS. Díky GPS lze využít tato zařízení jako navigaci nebo používat aplikace vhodné pro konkrétní situace a místa.

Největší nevýhodou těchto zařízení, z pohledu vývojáře, jsou obrovské rozdíly mezi jednotlivými modely. Jeden mobilní telefon může obsahovat jednojádrový procesor na nízké frekvenci (1 GHz), bez GPS, s 256MB paměti RAM, nízkým rozlišením displeje (800 x 400 pixelů, 233 ppi) a velmi starou verzí systému (verze 2.3.5)[2, 3] a v druhém lze nalézt osmijádrový[2] procesor na vysoké frekvenci (2,1 GHz), s GPS, 3GB pamětí RAM, vysokým rozlišením displeje (2560 x 1440 pixelů, 577 ppi) a aktuální verzí systému (verze 6) [4]. Toto roztříštění trhu klade na vývojáře spoustu nároků při tvorbě aplikací a vývojář musí určit a otestovat, na kterých zařízeních jeho aplikace může vůbec fungovat. Tento problém alespoň částečně řeší verze API. Při vývoji nových aplikací je možné nastavit minimální možnou verzi, která je nutná pro správný běh aplikace. Pokud zařízení neobsahuje verzi systému s tímto API, není aplikaci možné nainstalovat.

2.2. Tablety

Stejně jako na mobilních telefonech, je možné systém Android využívat i na tabletech. Tablety nabízejí podobné služby jako mobilní telefony, ovšem s výhodou větších displejů a vyššího výkonu. Tablety jsou vhodné především tam, kde se využije jejich velikost. Například v případě, že primárním účelem zařízení bude přehrávání filmů, čtení knih a časopisů nebo využívání zařízení pro zapisování poznámek, vytváření či prohlížení různých grafů a tabulek, je vhodnou volbou tablet.

Z technického hlediska je situace podobná mobilním telefonům. I zde je velká roztříštěnost výkonu a dostupných funkcí u jednotlivých tabletů [5, 6].

[2] Procesor obsahuje osm jader, ovšem vždy běží pouze čtyři. Při potřebě výkonu běží jádra s vyšší frekvencí a při šetření energie běží čtyři jádra na nižší frekvenci.

2.3. Televize

V posledních letech se tento systém také začíná dostávat do dalších zařízení, jako jsou například televize. Výrobci jako například Sony nebo Sharp již nyní[3] nabízí televize, které jsou Androidem vybaveny, a uživatel nepotřebuje žádné další přídavné zařízení. Tyto chytré televize uživateli zařídí přístup k jeho datům z Google služeb jako například Google Filmy nebo Google Hudba. Stejně tak je v nich dostupný obchod, kde lze nové filmy kupovat nebo půjčovat na omezenou dobu. Postupně také přibývají využitelné aplikace, které nabídnou nové funkce (obr. 2.1 [sony.com]). V případě, že vybraná televize (popřípadě ovladač nebo bezdrátově připojený telefon) je vybavena mikrofonem, je možné televizi začít ovládat hlasem. To značně rozšiřuje možnosti těchto chytrých televizí a lze tedy například rychle hledat filmy nebo zkontrolovat počasí bez nutnosti použít počítač nebo telefon.

Obr. 2.1 Android TV společnosti Sony

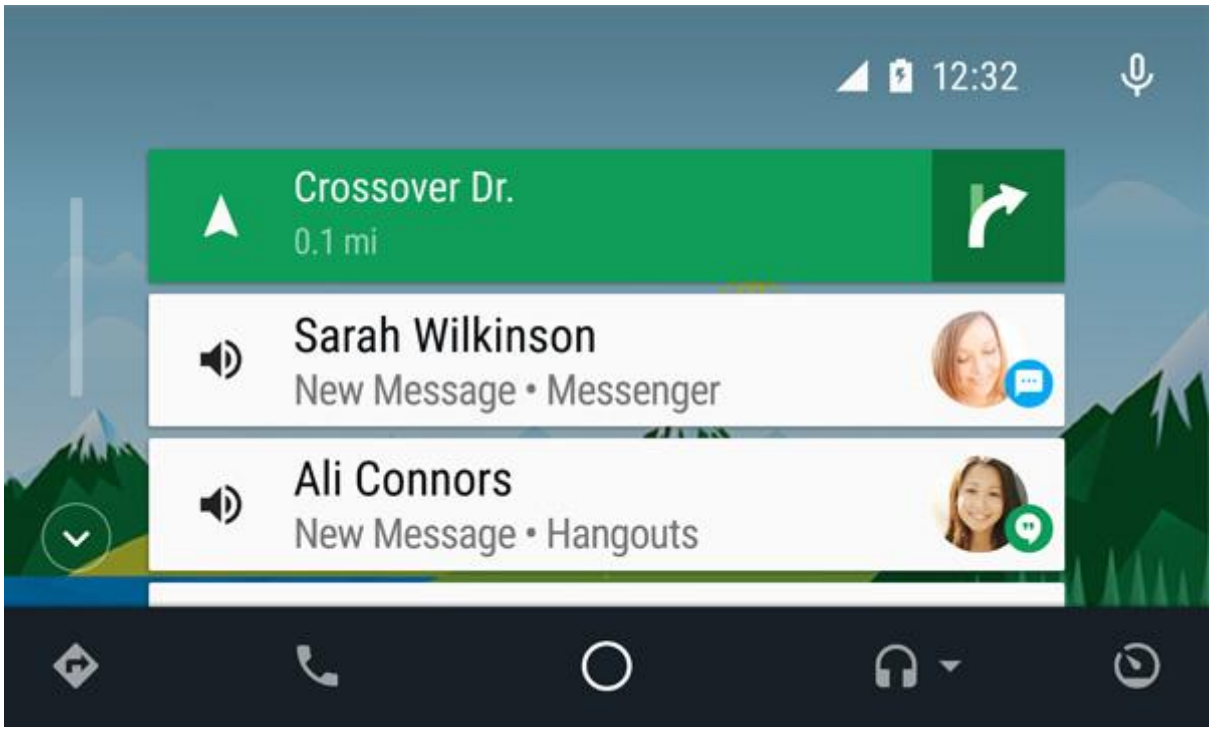
Všechny výše uvedené služby je také možné získat ke starším (nebo prostě jen ne-androidím) televizím díky set-top boxům, které obsahují Android také a lze je k televizi připojit stejně jednoduše jako jiné set-top boxy [7].

2.4. Automobily

Na začátku roku 2015 vydala společnost Google také aplikaci Android Auto [8], která umožňuje propojení automobilu s telefonem se systémem Android. Aby bylo možné tuto aplikaci využívat, musí uživatel vlastnit Android mobilní telefon, který připojí ke

[3] Rok 2015

kompatibilnímu automobilu [9]. Android Auto poté promítá upravené rozhraní mobilního telefonu a umožní uživateli využívat Google Hudbu a další aplikace. Aktuálně[4] lze využívat aplikace Google Mapy, Google Hudba, Google Play Now a další aplikace, které mají upravené rozhraní tak, aby je bylo možné v Android automobilech použít. Telefon také automaticky přesměruje příchozí hovory a zprávy na displej vozidla nebo do jeho reproduktorů. Po připojení mobilního telefonu a zapnutí aplikace Android Auto se na displeji telefonu zobrazí pouze název aplikace a od té chvíle je možné telefon ovládat pouze skrze promítané rozhraní na displeji automobilu (obr. 2.2)[10]. Nevýhodou tedy může být to, že mobilní telefon během připojení nelze k ničemu dalšímu využít. Z důvodu bezpečnosti je to ovšem správný krok.



Obr. 2.2 Přehled zobrazený vozidlem při použití Android Auto

V případě starších automobilů je také možnost, jak tyto chytré funkce začít využívat. Na trhu se začala objevovat autorádia podporující Android Auto a přinášející všechny funkce, které lze najít v kompatibilních automobilech [11].

2.5. Herní konzole

V roce 2013 se na internetu objevil projekt Ouya. Vývojáři se rozhodli vytvořit herní konzoli využívající systém Android. Bohužel tento projekt nedopadl příliš dobře. Zařízení není schopné cokoli přehrát a je vytvořeno pouze pro hraní her, kterých také není příliš mnoho. Celkově projekt příliš neprorazil a i přestože toto zařízení podpořilo na crowdfundingové kampani přes šedesát tisíc lidí s částkou vyšší než osm a půl milionu dolarů [12], zůstane toto zařízení prozatím spíše jako slepá ulička technologického vývoje [13].

[4] Prosinec 2015

2.6. Wearable zařízení

Poslední kategorií jsou wearable zařízení. V posledních letech se tato zařízení začala rozšiřovat a dnes lze najít velké množství různých zařízení kompatibilních se systémem Android.

Lze zde najít například fitness náramky. Existují jednoduché náramky bez displeje, které celý den monitorují počet kroků, spálené kalorie či délku spánku a všechny tyto informace synchronizují s mobilním telefonem [14]. Další variantou jsou pak náramky nesoucí displej zobrazující všechny tyto údaje v reálném čase a v případě příchozího hovoru například i zobrazení jména volajícího.

Jako další je možné zmínit například čepici SmartCap [15]. Tato čepice sbírá stejná data jako výše zmíněné náramky a také je synchronizuje do aplikace v mobilním telefonu.

Ovšem největší (a pro tuto práci nejdůležitější) skupinou jsou Android Wear, představující chytré hodinky. Díky nim nyní může uživatel kontrolovat upozornění na hovory či nové zprávy bez toho, aby mobilní telefon vytáhl z kapsy. Chytré hodinky slouží převážně jako přídavné zařízení upozorňující uživatele na nové a důležité informace, které přišly na jeho mobilní telefon. Umožňují ovládání aplikací nainstalovaných na mobilním telefonu, jako například ovládání hudby nebo ovládání navigace na cestách. Většina hodinek také obsahuje senzory monitorující pohyb, srdeční rytmus nebo barometrický tlak. Hodinky lze tedy také využívat na sledování tělesných funkcí během cvičení, běhání nebo každodenním pohybu.

Android Wear zařízení jsou kompatibilní také s operačním systémem iOS používaným v zařízeních Apple. Firma Apple ale samozřejmě nabízí vlastní alternativu Apple Watch. Tato zařízení jsou obdobou Android Wear.

3. Možnosti tvorby aplikací pro platformu Android Wear

Při tvorbě aplikací pro Android Wear se lze ubírat dvěma směry. Jako první je zde možnost vytvořit aplikaci pro mobilní telefon (popřípadě použít již dříve vytvořenou) a rozšířit ji o notifikace přizpůsobené pro Android Wear. Druhou možností je pak vytvoření nové aplikace přímo pro Android Wear.

3.1. Prvky uživatelského rozhraní

Android Wear přinesly nové specifické prvky uživatelského rozhraní, využitelné pouze pro ně.

3.1.1. *Watch Face*

Reprezentuje výchozí vzhled hodinek (obr 3.1 [developer.android.com]). Zobrazuje čas a může určovat pozadí výchozí obrazovky. Také určuje kolik karet z *Context Streamu* (kap. 3.1.2) se zobrazuje ve spodní části obrazovky.

Obr 3.1 *Watch Face*

- **Design a vytváření *Watch Face***

Android Wear jsou dostupné v různých velikostech a tvarech. Při tvorbě *Watch Face* je tedy důležité zvážit, jak bude vypadat vytvářený *Watch Face* na kulatých i čtvercových hodinkách. Vývojář by měl vytvářet vzhled, který bude na obou zařízeních vypadat podobně a bude sdílet stejné UI prvky. Přesto je ale možné na různých tvarech hodinek vytvářet odlišný vzhled.

- **Gesta**

V rámci *Watch Face* je možné využívat pouze jednoduché klepnutí. Ostatní gesta jsou rezervovaná pro systém nebo zakázaná (tab. 3.1 [developer.android.com]).

Stav	Dostupné			Rezervováno pro systém		
Gesto	*Klepnutí*	*Nahoru*	*Vlevo*	*Dolů*	*Dlouhé klepnutí*	*Vpravo*
Užití	Například změna vzhledu nebo spuštění aktivity	Pohyb v seznamu notifikací	Otevření *Cue Card*	Rychlé nastavení	Výběr *Watch Face*	Rezervováno pro systém

Tab. 3.1 Přehled gest

Pro jednodušší akce, například změna barvy pozadí, lze použít celý displej. Pro složitější akce může vývojář použít pro různé části displeje různé akce (obr. 3.2 [developer.android.com]). Po klepnutí na jednu z částí se pak tato část přesune do středu displeje a zobrazí podrobnější informace (obr. 3.3 [developer.android.com]). Klepnutím na okraj kruhu se pak *Watch Face* vrátí zpět do výchozího stavu.

K vlastnímu *Watch Face* je také možné vytvořit doprovodnou aplikaci pro mobilní telefon, kde lze umožnit nastavení různých vlastností, například pozadí, designu hodin a tak podobně.

Obr. 3.2 Výchozí stav

Obr. 3.3 Detail na teploty

3.1.1. *Home Screen*

Jinými slovy také výchozí obrazovka. Na pozadí je zobrazena fotka dle první karty v pozadí nebo styl závisející na *Watch Face*. Jsou zobrazeny indikátory stavu (například stav baterie, připojení a tak dále) a v některých případech počet nepřečtených položek (záleží na *Watch Face*). Na spodní straně obrazovky je zobrazena část první karty (popřípadě více karet, také záleží na *Watch Face*) z *Context Streamu*.

3.1.2. *Context Stream*

Context stream je vertikální seznam karet. Tyto karty obsahují užitečné informace a nejčastěji reprezentují notifikace přijaté z mobilního telefonu. Každou kartu lze rozšířit o další funkce, které lze použít po posunutí karty vlevo:

- **Stránka (*Page*)**
  Poskytuje dodatečné informace k zobrazené kartě. Například v případě, že vybraná karta je karta počasí zobrazující aktuální teplotu, může stránka zobrazovat předpověď počasí pro několik následujících dní.
- **Tlačítko (*Action Button*)**
  Tlačítko provádějící specifickou akci. Akce může být provedena přímo na wearable zařízení, nebo v mobilním telefonu.

3.1.3. *Cue Card*

Zobrazuje seznam doporučených hlasových příkazů. Je vyvolána klepnutím na pozadí výchozí obrazovky nebo vyslovením „OK Google“. Po zadání nebo vybrání příkazu je provedena akce, kterou může obsloužit mobilní telefon nebo hodinky samotné. Na hodinkách se také může spustit plnohodnotná aplikace, popřípadě pouze zobrazit potvrzující animace.

3.2. Zásady správného designu

Protože zařízení Android Wear je odlišné od klasických mobilních telefonů, musí vývojář dbát na správný návrh vytvářené aplikace. Z toho důvodu je zde několik základních designových pravidel.

- **Pravidlo 5 sekund**
  Protože hodinky jsou zařízením vhodným při používání během jiných aktivit, neměla by aplikace uživatele při těchto aktivitách zastavit. Obsloužení aplikace uživatelem by tedy nemělo trvat déle než pět vteřin.
- **Velikost prvků uživ. rozhraní**
  Při používání mobilního telefonu je k dispozici velký displej a většinou i čas k přesnějším gestům. Hodinky mají mnohem menší displej a jsou využívány hlavně během jiných aktivit, proto by aplikace měla být dle toho navržena a designové prvky by měly být dostatečné velké.
- **Využívání senzorů**
  Nejlepší využití najdou hodinky tam, kde je potřeba zobrazit správný obsah ve správnou chvíli. Aplikace by měla využívat senzory polohy, fyzické aktivity nebo například času k zobrazení relevantních a vhodných údajů.
- **Zbytečné vibrování**
  Uživatel má hodinky na rukou téměř pořád. Aplikace by tedy neměla využívat vibrace tak často, jako aplikace na mobilním telefonu a vibrovat by měla pouze při důležitých upozorněních.

3.3. Interaktivní a *Ambient* mód

Hodinky používají dva hlavní módy. V interaktivním módu (obr 3.4 [developer.android.com]) lze využít celou paletu barev a vizuálních efektů. Druhý režim (tzv. *Ambient mode* - obr. 3.5 [developer.android.com]) šetří energii zařízení a proto může omezit barvy a použité efekty. V tomto režimu je *Watch Face* omezen na černou a šedou barvu, i přesto lze ale použít barvy až pro pět procent pixelů na displeji. Hodinky také aktualizují displej pouze jednou za minutu, a proto nezobrazují sekundy, ale pouze hodiny a minuty. Při přechodu do tohoto režimu je *Watch Face* upozorněn a jeho design by se měl umět tomuto přechodu přizpůsobit.

Obr. 3.4 Interaktivní mód

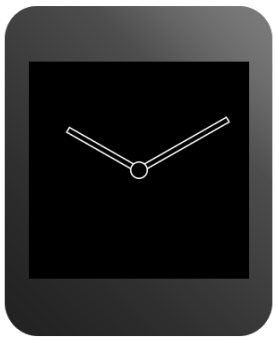
Obr. 3.5 Ambient mód

- **Speciální displeje**

Některé displeje mohou v *Ambient* módu využívat omezené množství barev (*low-bit* mód). V tomto módu je možné na displeji vykreslit pouze černou, bílou, modrou, červenou, zelenou, žlutou, modrozelenou (*cyan*) a magentu. Při použití tohoto módu by na pozadí měla být použita černá nebo bílá barva (pro OLED displeje pouze černá) a pixelů, které nejsou součástí pozadí, musí být méně než deset procent. Některé displeje také nemusí vykreslovat vůbec žádnou barvu, v těchto případech by pozadí mělo být také černé nebo bílé.

Dalším případem jsou displeje OLED, které stále trpí vypalováním[5] obrazu. Hodinky v *Ambient* módu sami posouvají pravidelně obsah displeje o několik pixelů, aby tomuto efektu předešly. Devadesát pět procent displeje by ale stejně mělo zůstat v černé barvě a vyplněné bloky pixelů by měly být nahrazeny pouze obrysy (obr. 3.6 [developer.android.com]).

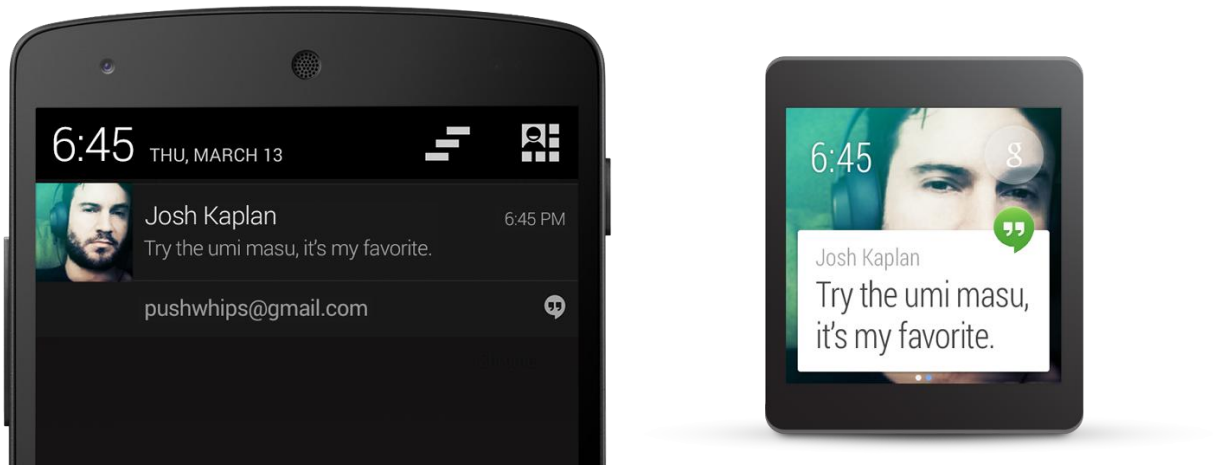
[5] Pokud zůstane na obrazovce příliš dlouho stejný obraz, může se takzvaně vypálit. V tom případě je poté na pozadí obrazovky stále vidět tento obraz i když už dávno není zobrazen.



Obr. 3.6 Předcházení vypálení displeje

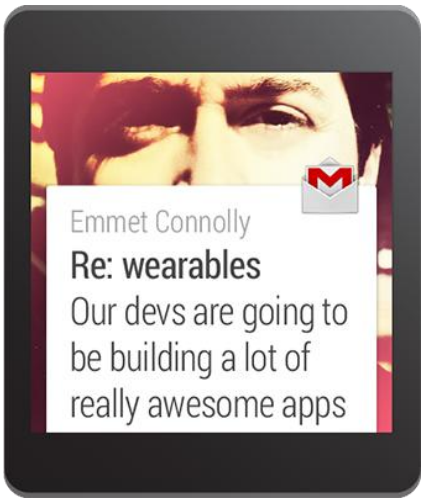
3.4. Rozšíření aplikace o notifikace

Po propojení hodinek a mobilního telefonu, začne mobilní telefon automaticky sdílet notifikace s hodinkami (obr 3.7 [developer.android.com]). Na hodinkách se tyto notifikace zobrazují jako karty v C*ontext Streamu*. Těmto notifikacím lze ovšem přidat další funkce rozšiřující využitelnost wearable zařízení.



Obr. 3.7 Mobilní telefon a notifikace na hodinkách

Notifikaci je možné vytvořit v rozšířeném režimu, kdy tato notifikace může zobrazit delší text. Tento režim je v originále pojmenován jako „*Big View*“ (obr. 3.8 [developer.android.com]).

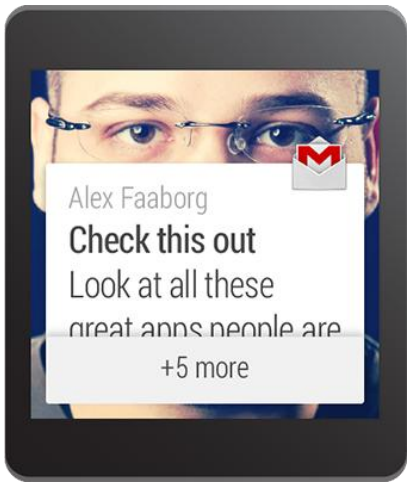


Obr. 3.8 „*Big View*“

Každá notifikace k sobě může mít přidané stránky nebo tlačítka. Stránky se zobrazují za vybranou notifikací, na její pravé straně. Jak bylo popsáno dříve, tyto stránky slouží k zobrazení dodatečných informací.

Za stránkami mohou být k dispozici tlačítka. Stisknutím tlačítka se vyvolá vývojářem definovaná akce, kterou provede zařízení samotné nebo spárované mobilní zařízení. Je také možné využít hlasový vstup. Po stisknutí tlačítka se pořídí hlasový záznam a ten se odešle do spárovaného mobilního zařízení. Tlačítka se ve výchozím nastavení zobrazují také na mobilním zařízení. Je ale možné tato tlačítka vytvářet tak, aby byla zobrazována pouze na Android Wear.

Na rozdíl od mobilních zařízení, notifikace od stejné aplikace na Android Wear nemohou být spojeny do jedné notifikaci zobrazující pouze shrnutí. Pokud by Android Wear zobrazovaly například notifikaci „3 nové zprávy“, uživatel nemá možnost zjistit jakékoli podrobnosti o těchto zprávách, aniž by musel otevřít aplikaci na mobilním zařízení. Z tohoto důvodu je na Android Wear možné notifikace spojovat do skupin. Po spojení notifikací se zobrazí první notifikace a informace o tom, kolik dalších notifikací v této skupině je (obr. 3.9 [developer.android.com]).



Obr. 3.9 Ukázka seskupení notifikací

3.4.1. Odpověď hlasem

Notifikace upozorňující například na emaily nebo textové zprávy často obsahují tlačítko „Odpovědět“. Toto tlačítko umožní uživateli napsat odpověď bez toho, aby musel otevřít příslušnou aplikaci. Android Wear zařízení ovšem neobsahují klávesnici a proto je jedinou možností použít hlasový vstup nebo předdefinované možnosti. Uživatel tedy může po stisknutí tlačítka nadiktovat svoji odpověď, nebo vybrat až z pěti možných předdefinovaných možností. Aplikace na mobilním zařízení poté může získat textový přepis nadiktované odpovědi.

3.5. Získání lokace

Některá Android Wear zařízení obsahují GPS senzor, ale jiná bohužel ne. V případě, že zařízení GPS senzor nemá, je možné využít GPS senzor mobilního telefonu, který je se zařízením spárován. Programátor se nemusí starat o to, z kterého zařízení bude poloha získána, protože systém sám vybere nejefektivnější metodu. Vytvářená aplikace by ale měla umět obejít bez těchto dat, pokud je zařízení odpojeno od mobilního zařízení a nemá vlastní GPS přijímač.

Aktuálně je oficiálně doporučeno při získávání lokace využívat API služeb Google Play. Toto API poskytuje informace o tom, zda je polohu možné získat, či nikoli. Poté, co se zařízení připojí ke službám Google Play, je možné začít získávat lokalizační data. Od této chvíle může aplikace v pravidelných intervalech žádat o aktualizaci těchto dat.

Může se stát, že z nějakého důvodu Android Wear zařízení přijde o GPS signál nebo o spojení s mobilním zařízením a nebude tedy možné získat nové aktualizace. V těchto situacích je možné získat alespoň poslední známou polohu.

V případě, že není možné polohu detekovat pomocí GPS, je možné využít alternativní řešení:

- **Volba uživatelem**
  Uživateli bude nabídnuta volba místa, kde se nachází. V navrhované aplikaci by uživatel například mohl vybrat volbu „ZČU“ při vstupu do areálu ZČU. Při odchodu pak naopak vybere jinou možnost.
- **Detekce dle WiFi**
  Zařízení umožňuje získat seznam dostupných WiFi sítí v okolí (včetně například sítě, ke které je zařízení aktuálně připojeno). V případě navrhované aplikace by například bylo možné využít sítě *eduroam*. Pokud by byla tato síť v okolí (či aktuálně využívaná), znamená to, že se uživatel s velkou pravděpodobností nachází v areálu ZČU.

3.6. Sdílení dat

Služby Google Play poskytují datovou vrstvu umožňující komunikaci a sdílení dat mezi Android Wear a mobilními zařízeními. Tato vrstva obsahuje několik objektů, které můžou zařízení mezi sebou sdílet a synchronizovat (podrobnější popis a ukázky implementace lze nalézt v kapitole 4.3.8).

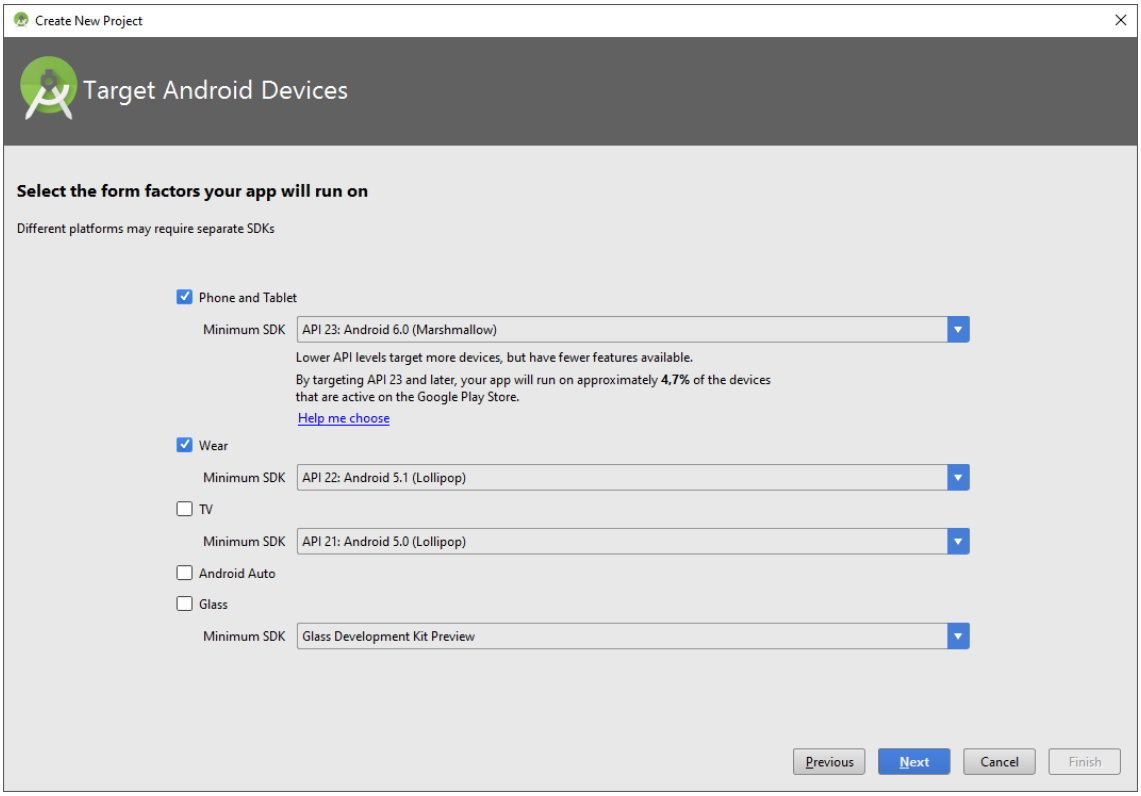
- ***Asset***
  Objekt vhodný pro odesílání binárních bloků dat. Slouží ke sdílení například obrázků.
- ***DataItem***
  Poskytuje datové úložiště s automatickým synchronizováním mezi Android Wear zařízením a mobilním zařízením. Tento objekt je naklonován do každého zařízení a obsahuje malý blok dat a přiložené *assety*.
- ***Messages***
  Slouží k odesílání menších bloků dat. Je vhodný například pro jednostranné zprávy nebo pro zprávy typu požadavek/odpověď. Poskytuje také informaci o tom, zda bylo doručení zprávy úspěšné. Odeslané zprávy jsou soukromé a jsou tedy dostupné pouze pro stejnou aplikaci.
- ***Channel***
  Objekt ideální pro přenos větších dat bez nutnosti pravidelné synchronizace. Využívá se například pro přenos filmů, hudby nebo streamovaných dat a hlasového vstupu z mikrofonu.

4. Tvorba aplikací pro platformu Android Wear

Tato kapitola se pokusí dříve obsažené informace popsat z hlediska programátora. Bude zde uvedena například ukázka vytváření notifikací, či získání lokace. Kapitola se věnuje pouze platformě Android Wear a předpokládá se, že čtenář je seznámen se základy programování na zařízení Android. Pro seznámení se s programováním aplikací na Android a Android Wear mohu doporučit například knihu „Vývoj aplikací pro Android“ [16].

4.1. Vytvoření projektu

Pro správné fungování funkcí, které obsahují komunikaci mezi mobilním zařízením Android a zařízením Android Wear je nutné, aby aplikace pro obě zařízení byly součástí stejného projektu. V Android Studiu toho docílíme vytvořením projektu obsahující dva moduly (obr. 4.1).



Obr. 4.1 Projekt v Android Studiu obsahující dva moduly

Aplikace vytvářené v tomto projektu poté sdílejí stejný název aplikace a je možné mezi nimi komunikovat ať již pomocí notifikací, nebo jinou dostupnou síťovou komunikací.

4.2. Notifikace

V dřívějších kapitolách byl popsán teoretický způsob vytváření a využití rozšířených notifikací. Notifikace se vytvářejí obdobně jako na zařízení Android, ovšem je možné je rozšířit o další funkce dostupné pouze na platformě Android Wear.

Standardní notifikace se vytvářejí stejně jako na mobilní zařízení Android. Pro vytvoření jednoduché notifikace lze využít třídu `NotificationCompat.Builder`. Vytvoříme její instanci a nastavíme jí titulek a text, který bude notifikace obsahovat (obr. 4.2).

```
NotificationCompat.Builder notificationBuilder =
new NotificationCompat.Builder(this)
      .setSmallIcon(R.drawable.icon)
      .setContentTitle("Notifikace")
      .setContentText("Ahoj z notifikace");
```

Po vytvoření notifikace je ještě nutné zařídit její odeslání do zařízení. Toho dosáhneme využitím třídy `NotificationManagerCompat`.

```
int idNotifikace = 1;
NotificationManagerCompat notificationManager =
      NotificationManagerCompat.from(this);
notificationManager.notify(idNotifikace,
      notificationBuilder.build());
```

Jak je uvedeno v prvním řádku kódu, je vhodné nastavit notifikaci její číselnou identifikaci. Druhý řádek kódu vytvoří instanci této třídy a třetí řádek zařídí zobrazení notifikace na zařízení (jak na mobilním, tak na zařízení Android Wear).

Obr 4.2 Jednoduchá notifikace

Pro zobrazení notifikace je nutné přiřadit ji alespoň ikonu metodou `setSmallIcon(int icon)`. Notifikaci lze tedy vytvořit i bez jakéhokoli textu, ale bez použití této metody se notifikace na zařízení nezobrazí.

Notifikacím je možné přidávat další prvky, jako tlačítka, výše zmíněný *Big View*, stránky nebo nastavit jiné pozadí. Další ukázka kódu zobrazí notifikaci, ke které jsou přidány dvě stránky a tlačítko. Tyto stránky jsou zobrazeny pouze na zařízení Android Wear.

Notifikace má také na zařízení Android Wear skrytou ikonku, nastavené vlastní pozadí a z notifikace lze tlačítkem otevřít novou aktivitu v mobilním zařízení.

Jako první je znovu vytvořena číselná identifikaci notifikace. Navíc se také připraví `ArrayList`, do kterého budou ukládány vytvářené stránky a připraví se nový `Intent`, který zobrazí novou aktivitu po klepnutí na tlačítko „*Open on phone*“.

```
int idNotifikace = 1;
Intent viewIntent = new Intent(this, NotifikaceAktivity.class);
PendingIntent viewPendingIntent =
      PendingIntent.getActivity(this, 0, viewIntent, 0);
List extras = new ArrayList();
```

Dalším krokem je vytvoření první stránky (obr. 4.3) pomocí třídy `BigTextStyle`. Na zařízení Android Wear nebude znát rozdíl mezi použitím klasické notifikace a notifikace z této třídy, ovšem na mobilním zařízení by takto vytvořená notifikace zobrazila nezkrácená (text běžné notifikace je zkracován). V kódu je nejdříve vytvořen styl a ten je poté nastaven nové instanci notifikace. Na závěr je tato notifikace přidána do dříve vytvořeného `ArrayListu`.

```
NotificationCompat.BigTextStyle extraPageStyle =
      new NotificationCompat.BigTextStyle();
extraPageStyle.setBigContentTitle("BigTextStyle")
      .bigText("Dlouhý text stránky 2, který zkouší jak moc dlouhý
            může  být. Takhle přesně vypadá použití BigTextStylu v
            praxi.");
Notification extraPageNotification =
      new NotificationCompat.Builder(this)
      .setStyle(extraPageStyle)
      .build();
extras.add(extraPageNotification);
```

Obr. 4.3 První vytvořená stránka

Druhá vytvářená stránka bude vytvořena třídou `InboxStyle`. Notifikace (nebo stránky) vytvářené touto třídou zobrazují text po řádkách. Do notifikace se přidávají řádky textu metodou `addLine(CharSequence cs)`. Takto přidaný text dostane jeden řádek v notifikaci (obr. 4.4).

```
NotificationCompat.InboxStyle extraPageStyle2 =
      new NotificationCompat.InboxStyle();
extraPageStyle2.setBigContentTitle("InboxStyle")
      .addLine("Řádek 1 - Dlouhý text, který se zobrazí jen z
            části")
      .addLine("Řádek 2")
      .addLine("Řádek 3 - Krátký text")
      .addLine("Řádek 4")
      .addLine("Řádek 5")
      .addLine("Řádek 6")
      .addLine("Řádek 7")
      .addLine("Řádek 8");
extraPageNotification = new NotificationCompat.Builder(this)
      .setStyle(extraPageStyle2)
      .build();
extras.add(extraPageNotification);
```

Obr. 4.4 Druhá vytvořená stránka

Posledním krokem před vytvořením samotné notifikace je připravení *„wearable-only“* akcí[6]. Příprava proběhne s využitím třídy `WearableExtender`. Na počátku je vytvořena nová instanci této třídy. Této instanci je pak nastaveno pozadí, nezobrazení ikonky a jsou jí předány vytvořené stránky.

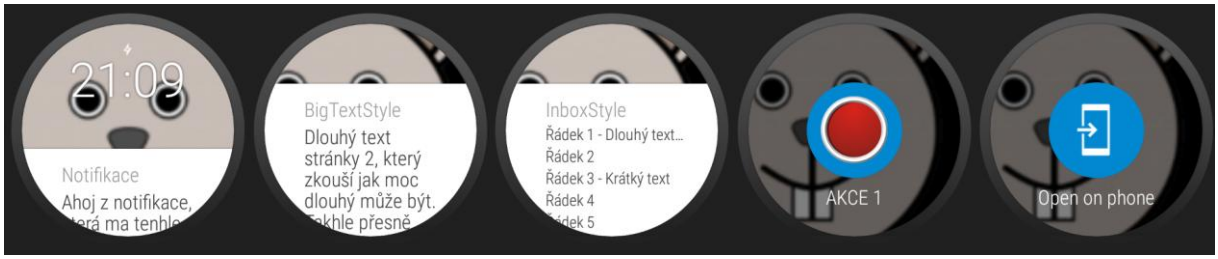
[6] Akce dostupné pouze na zařízení Android Wear

```
NotificationCompat.WearableExtender wearableExtender =
      new NotificationCompat.WearableExtender()
      .setHintHideIcon(true)
      .addPages(extras)
      .setBackground(BitmapFactory.decodeResource(
            getApplicationContext().getResources(),
            R.drawable.my_icon));
```

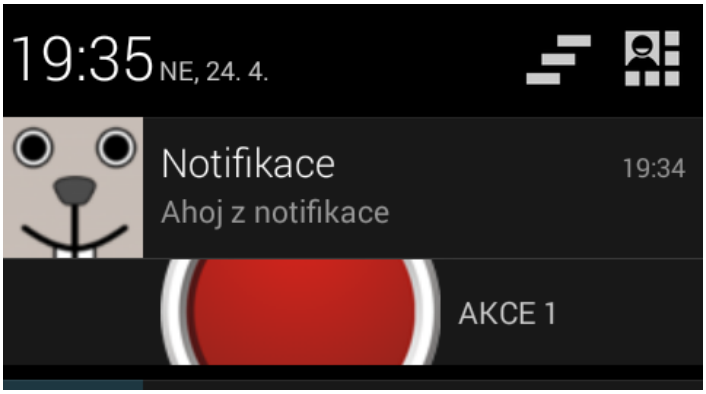
Poslední část kódu vytvoří samotnou notifikaci. Část je shodná s kódem popsaným výše. Navíc je zde nastavení aktivity, která se má zobrazit po klepnutí na tlačítko *„Open on phone“* (metoda `setContentIntent(PendingIntent intent)`), přidání nového vlastního tlačítka (metoda `addAction(int icon, CharSequence title, PendingIntent intent)`) a rozšíření o *„wearable-only“* akce. Tlačítku je pro zjednodušení kódu nastaveno také otevření nové aktivity, stejně jako systémovému tlačítku zmíněnému dříve.

```
NotificationCompat.Builder notificationBuilder =
      new NotificationCompat.Builder(this)
      .setSmallIcon(R.drawable.my_icon)
      .setContentTitle("Notifikace")
      .setContentText("Ahoj z notifikace, která ma tenhle text
            nastavený standardním způsobem.")
      .setContentIntent(viewPendingIntent)
      .addAction(R.drawable.button_icon, "AKCE 1",
            viewPendingIntent)
      .extend(wearableExtender);

NotificationManagerCompat notificationManager =
      NotificationManagerCompat.from(this);
notificationManager.notify(idNotifikace,
      notificationBuilder.build());
```

Na obrázkách (obr. 4.5 a 4.6) je vidět výsledná notifikace, která se zobrazí po vytvoření notifikace.



Obr. 4.5 Notifikace zobrazená na hodinkách



Obr. 4.6 Notifikace zobrazená na mobilním zařízení

4.3. Možnosti uživatelského rozhraní

Při vytváření aplikací na platformu Android Wear je dostupná knihovna Wearable UI, která obsahuje nové prvky uživatelského rozhraní využitelné pro tuto platformu. Tyto prvky lze pak použít při vytváření vlastních aplikací a jejich vzhledu či funkcionality.

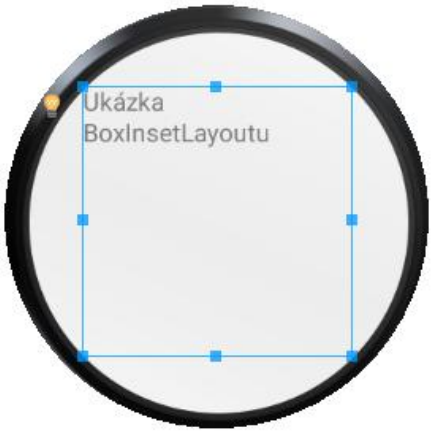
4.3.1. Tvar displeje - *BoxInsetLayout*

Pokud uživatel nechce vytvářet odlišné layouty pro kulaté a čtvercové displeje, může využít nový typ layoutu, který hlídá tvar hodinek. V případě, že je aplikace spuštěná na hodinkách s kulatým displejem, vytvoří vnořený čtverec tak, aby prvky layoutu nebyly zobrazeny mimo obrazovku. V případě spuštění na hodinkách se čtvercovým displejem se pak tento layout chová jako standardní `FrameLayout`.

Prvkům uvnitř tohoto layoutu je možné nastavit, které z hranic mají hlídat. Na obrázku 4.7 lze vidět nastavení pro hlídání všech hranic. Tuto vlastnost lze nastavit vlastností `app:layout_box`. Případ dodržení všech hranic je realizován následujícím kódem.

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

      <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_box="all"
            android:background="#000000">
            .....TextView.....
      </RelativeLayout>
</android.support.wearable.view.BoxInsetLayout>
```

Na obrázku (obr 4.8) je pak také vidět vzhled při nastavení `app:layout_box="left | top"`.



Obr. 4.7 Nastavení "`all`"

Obr. 4.8 Nastavení "`left | top`"

4.3.2. Potvrzování akcí – *ConfirmationAcitivity*, *DelayedConfirmationView*

Pokud je potřeba uživateli zobrazit potvrzení nebo nabídnout možnost zrušit akci před jejím vykonáním, je důležité dát mu to najevo zřetelně a nejlépe přes značnou část obrazovky.

Pro zobrazení potvrzení je možné využít `ConfirmationActivity`. Před jejím použitím je nutné ji přidat do manifestu aplikace.

```
<manifest>
      <application>
      ...
      <activity
      android:name =
            "android.support.wearable.activity
            .ConfirmationActivity">
      </activity>
      </application>
</manifest>
```

Pro použití je pak možné zavolat `Intent` otevírající tuto aktivitu.

```
Intent intent = new Intent(this, ConfirmationActivity.class);
intent.putExtra(ConfirmationActivity.EXTRA_ANIMATION_TYPE,
      ConfirmationActivity.SUCCESS_ANIMATION);
intent.putExtra(ConfirmationActivity.EXTRA_MESSAGE,
      "Smazáno!");
startActivity(intent);
```

Aktivita zobrazí animaci a poté se ihned ukončí. Pomocí `Extras` lze aktivitě předat typ zobrazované animace a zprávu, která se zobrazí. Na výběr jsou tři typy animace (obr. 4.9):

- **`SUCCES_ANIMATION`** Animace potvrzení
- **`OPEN_ON_PHONE_ANIMATION`** Animace otevření na mobilním zařízení

- **FAILURE_ANIMATION** Animace selhání

Obr. 4.9 Animace potvrzování

`DelayedConfirmationView` pak umožňuje dát uživateli čas k případnému přerušení před skutečným vykonáním akce. Tento vizuální prvek dědí od `CircleImageView` a je tedy možné ho téměř shodně používat. Navíc mu lze ale nadefinovat kruhový indikátor stavu, který lze programově spustit. Využití je vhodné například u tlačítka pro mazání nebo odeslání zprávy.

```
<android.support.wearable.view.DelayedConfirmationView
      android:id="@+id/delayed_confirm"
      android:layout_width="40dp"
      android:layout_height="40dp"
      android:src="@drawable/delete"
      app:circle_border_color="#FF0000"
      app:circle_border_width="4dp"
      app:circle_radius="20dp">
</android.support.wearable.view.DelayedConfirmationView>
```

V kódu lze vidět nadefinování vlastností prvku. Kromě standardních vlastností je určena barva, šířka a poloměr kruhového indikátoru stavu. Namísto vlastnosti `android:src` je možné využít vlastnost `android:background`. V případě použití první vlastnosti, bude obrázek zobrazený nad vykreslovaným kruhem a je nutné hlídat si nastavené velikosti (aby byl kruh nakonec vůbec vidět). V případě, že je použita druhá vlastnost, bude zobrazený obrázek jako spodní vrstva a kruh se bude vykreslovat v popředí (obr. 4.10).

Samotné animaci je možné nastavit délku průběhu a poté ji spustit.

```
delayedConfirmationView.setTotalTimeMs(2000);
delayedConfirmationView.start();
```

Poté, co je implementováno rozhraní poskytující potřebné metody - rozhraní `DelayedConfirmationView.DelayedConfirmationListener`, je nutné také implementovat metody `onTimerFinished(View view)` a `onTimerSelected(View view)`, které jsou volány, pokud se animace dokončí bez reakce uživatele (obr. 4.11),

nebo pokud uživatel v průběhu animace prvek stiskl (obr. 4.12). Protože událost `onTimerSelected` je zavolána i v případě, že bylo s prvkem interagováno mimo animaci, je vhodné prvek umístit například do samostatné aktivity, která se zobrazí až při zavolání akce (například při stisku tlačítka „Zobraz animaci“). Tento příklad ale pro jednoduchost zobrazuje prvek ve stejné aktivitě jako tlačítko akci vyvolávající.

Obr. 4.10 Použití vlastností `src` a `background`

```
public class MainActivity extends WearableActivity implements
      DelayedConfirmationView.DelayedConfirmationListener
{
      DelayedConfirmationView  mDelayedView;

      @Override
      protected void onCreate(Bundle savedInstanceState)
      {
            ...
            mDelayedView = (DelayedConfirmationView)
                  findViewById(R.id.delayed_confirm);
            mDelayedView.setListener(this);
      }

      //pokud uživatel nereagoval
      @Override
      public void onTimerFinished(View view)
      {
            Intent intent = new Intent(this,
                  ConfirmationActivity.class);
            intent.putExtra(
                  ConfirmationActivity.EXTRA_ANIMATION_TYPE,
                  ConfirmationActivity.SUCCESS_ANIMATION);
            intent.putExtra(ConfirmationActivity.EXTRA_MESSAGE,
                  "Smazáno!");
            startActivity(intent);
            mDelayedView.reset();
      }
```

```
        //pokud uživatel s prvkem interagoval
        @Override
        public void onTimerSelected(View view)
        {
                Intent intent = new Intent(this,
                        ConfirmationActivity.class);
                intent.putExtra(
                        ConfirmationActivity.EXTRA_ANIMATION_TYPE,
                        ConfirmationActivity.FAILURE_ANIMATION);
                intent.putExtra(ConfirmationActivity.EXTRA_MESSAGE,
                        "Přerušeno!");
                startActivity(intent);
                mDelayedView.reset();
        }
}
```

V uvedeném kódu lze vidět, že při vytvoření aktivity se v layoutu nalezne prvek `DelayedConfirmationView` a tomu je jako listener nastavena třída implementující rozhraní `DelayedConfirmationView.DelayedConfirmationListener`, které poskytuje metody reagující na události (v našem případě stejná třída). Tyto metody jsou pak dále implementovány a v případě jejich zavolání zobrazí výše popsanou `ConfirmationActivity`. Na konci obou metod je také zavolána metoda `reset`, která zajistí zmizení vykresleného kruhu.

Obr. 4.11 Průběh v případě, že uživatel nereaguje

Obr. 4.12 Průběh v případě, že uživatel s prvkem interaguje

4.3.3. Přechod mezi dvěma *Drawable* – *CrossfadeDrawable*

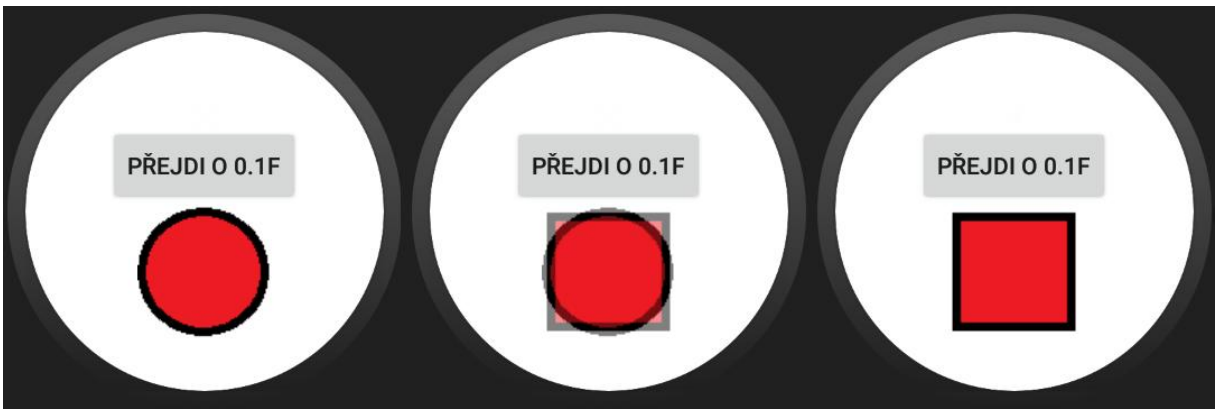
Pokud je potřeba přejít mezi dvěma `Drawable`, přineslo SDK možnost na platformě Android Wear využít novou třídu `CrossfadeDrawable`. Tato třída umožňuje zvolit počáteční a cílový `Drawable` a poté mezi nimi nastavit přechodový stav od 0 do 1 (0 – původní `Drawable`, 1 – cílové `Drawable`).

```
CrossfadeDrawable crossfadeDrawable = new CrossfadeDrawable();
crossfadeDrawable.setBase(getResources().getDrawable(
      R.drawable.kruh, null));
crossfadeDrawable.setFading(
      getResources().getDrawable(R.drawable.ctverec, null));

ImageView imageView = (ImageView)findViewById(R.id.imageView);
imageView.setImageDrawable(crossfadeDrawable);
```

Po vytvoření instance třídy je v kódu nastaven počáteční stav metodou `setBase(Drawable d)`, které se jako parametry předá objekt `Drawable`. Totéž proběhne pro cílový stav – metoda `setFade(Drawable d)`.

```
public void onClick(View v)
{
      progress = progress + 0.1f;
      crossfadeDrawable.setProgress(progress);
}
```

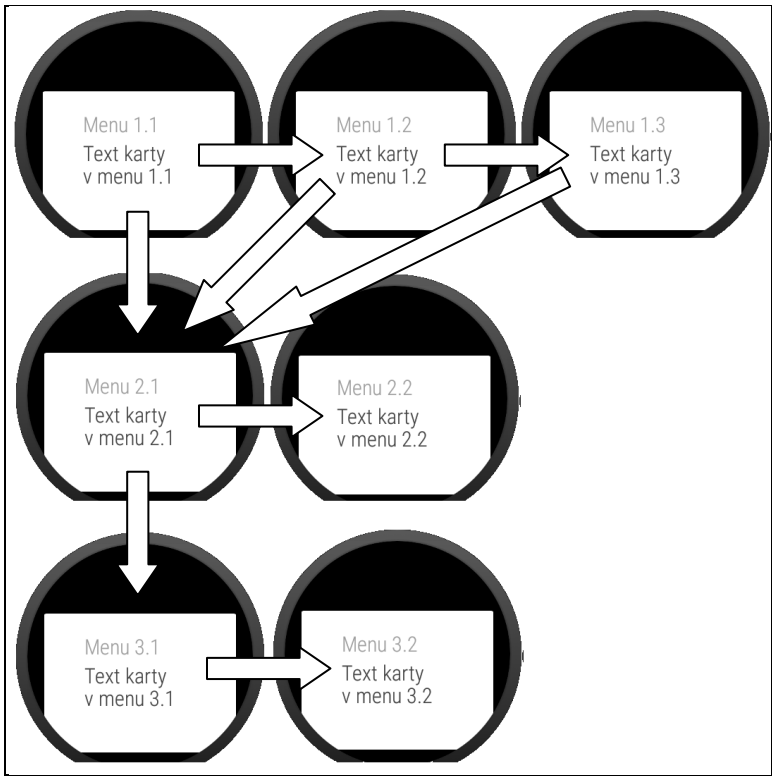
Ovládání přechodu probíhá voláním metody `setProgress(float p)`. V kódu výše je vidět přechod po desetinách, tedy trvající deset kroků (obr. 4.13).



Obr. 4.13 Přechod mezi `Drawable` na začátku, po pěti stisknutích tlačítka a na konci

4.3.4. Matice (mříž) prvků – *GridViewPager*, *FragmentGridPagerAdapter*

Pro displeje o malé velikosti, jakými disponují hodinky Android Wear, je vhodné řešit například menu maticí prvků (obr 4.14).



Obr. 4.14 Menu jako matice (mříž)

Mezi prvky takto vytvořené matice lze v řádku libovolně přecházet pomocí běžných gest. V případě přechodů mezi řádky je přechod vždy na první sloupec nového řádku. Řešení tedy není vhodné, pokud by měly být realizovány přechody v jednom sloupci skrze více řádků.

```
<android.support.wearable.view.GridViewPager
      android:id="@+id/pager"
      android:layout_width="match_parent"
      android:layout_height="match_parent"/>
```

K vytváření těchto menu slouží `GridViewPager`. Tento prvek je vkládán do layoutu aktivity a typicky je zobrazen přes celý displej. Není potřeba nastavovat další vlastnosti, protože samotné poskytování dat obstarává další třída, `GridPagerAdapter`.

```
GridViewPager pager = (GridViewPager)findViewById(R.id.pager);
pager.setAdapter(new PickerAdapter(getFragmentManager()));
```

Při vytvoření aktivity (metoda `onCreate(Bundle b)`) je z layoutu získán vložený `GridViewPager` a metodou `setAdapter(GridPagerAdapter GPA)` je nastaven poskytoval dat. Jako parametr použitý `PickerAdapter` je vlastní třída, která dědí od třídy `FragmentGridPagerAdapter` a poskytuje zobrazovaná data.

```
public class PickerAdapter extends FragmentGridPagerAdapter
{
      List<List<String>> menu = new ArrayList<List<String>>();

      public PickerAdapter(FragmentManager fm)
```

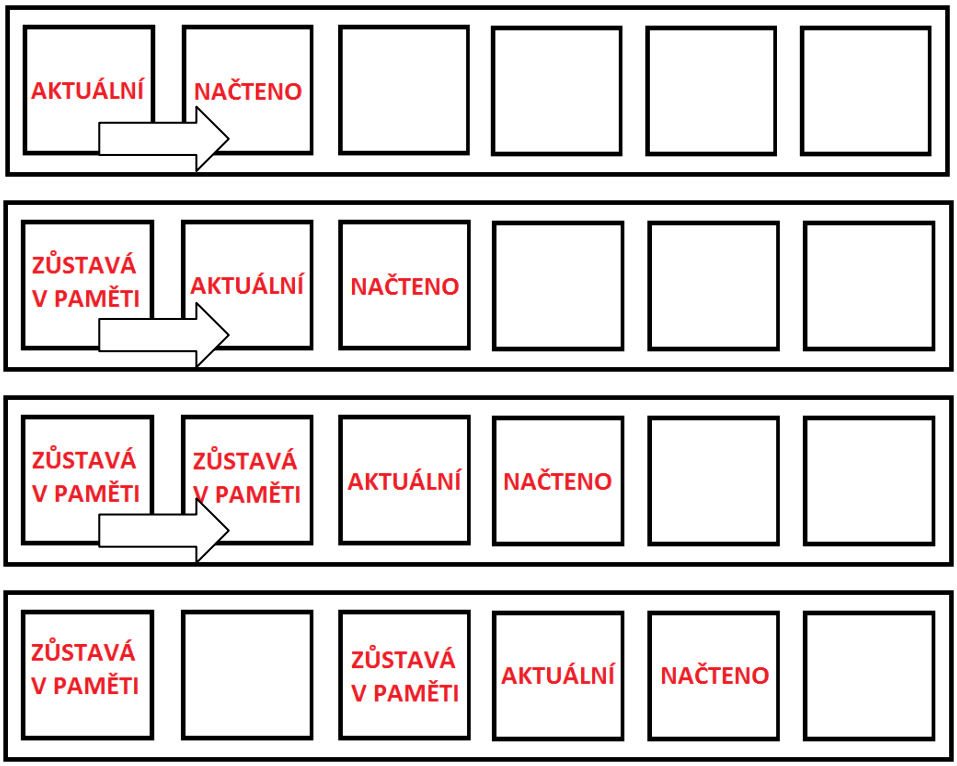
```
	{
		super(fm);
		menu.add(new ArrayList<String>());
		menu.add(new ArrayList<String>());
		menu.add(new ArrayList<String>());

		menu.get(0).add("Menu 1.1");
		...
	}

	@Override
	public Fragment getFragment(int row, int col)
	{
		return CardFragment.create(menu.get(row).get(col), "Text
			karty v menu " + (row + 1) + "." + (col + 1));
	}

	@Override
	public int getRowCount()
	{
		return menu.size();
	}

	@Override
	public int getColumnCount(int row)
	{
		return menu.get(row).size();
	}
}
```

Třída `PickerAdapter`, která dědí od třídy `FragmentGridPagerAdapter` musí implementovat tři metody (všechny je lze vidět v kódu) a je nutné jí vytvořit konstruktor volající kontruktor jejího rodiče. Jak lze vidět v kódu, menu lze realizovat například pomocí ArrayListů reprezentujících sloupce řádků vnořených do jednoho ArrayListu reprezentujícího řádky.

Metoda `getFragment(int row, int col)` poskytuje data, která se zobrazí v prvku `GridViewPager`, který je vložen v layoutu aktivity. Ve vloženém kódu metoda vrací jednoduchý `CardFragment`, který má výchozí vzhled a lze mu nastavit nadpis a text uvnitř karty. Metoda ovšem může vracet libovolnou třídu, která dědí od třídy `Fragment` a je tedy možné zobrazit vlastní a komplexnější vzhledy. Ukázka použití vlastního fragmentu je k dispozici níže v kapitole o tvorbě aplikace (kap. 6).

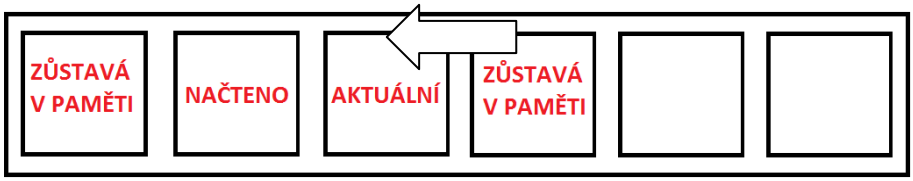
Metody `getRowCount()` a `getColumnCount(int row)` je pak nutné implementovat tak, aby vracely počet řádků a počet sloupců v aktuálním řádku. V kódu je zobrazena ukázka, která vrátí správné hodnoty v případě použití ArrayListů.

`GridViewPager` je optimalizován pro rychlé zobrazení na Android Wear a proto nenačítá každý fragment při přechodu, ale fragmenty přednačítá dříve než je nutné je zobrazit. Při prvním otevřením je načten první fragment (0 - řádek, 0 - sloupec) a jeho okolí – fragment (0, 1) a fragment (0, 2). Pokud uživatel začne přecházet doprava v aktuálním řádku, načte se při každém přechodu budoucí fragment a v paměti zůstane první fragment řádku, první fragment následujícího řádku a fragment, ze kterého se přecházelo (obr. 4.15).



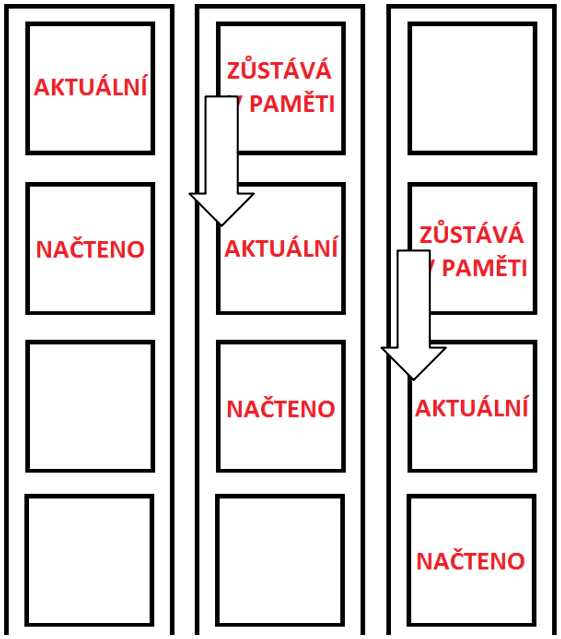
Obr. 4.15 Přechod doprava

Pokud uživatel poté přejde zpět doleva, zobrazí se neaktualizovaný fragment a načte se další po směru přesunu (obr. 4.16).



Obr 4.16 Po přechodu doleva

Jak již bylo zmíněno výše, po celou dobu pohybu v řádku je stále v paměti uložen první fragment aktuálního řádku a první fragment následujícího řádku. Pohyb v řádcích pak funguje obdobným způsobem, při přechodu na nový řádek se načte následující fragment ve směru pohybu a fragment vpravo v řádku, do kterého se přešlo. V paměti pak zůstává poslední navštívený fragment. Na obrázku (obr 4.17) lze vidět tyto přechody bez načtení prvního fragmentu vpravo od aktuálního (ten se ale také načítá).



Obr. 4.17 Přechody mezi řádky

Při používání fragmentů, ve kterých se dynamicky mění zobrazované údaje, je tedy nutné zajistit jejich správné zobrazení. Toho lze docílit použitím metody `notifyDataSetChanged()`.

4.3.5. Indikace aktuálního sloupce – *DotsPageIndicator*

Pokud každý řádek obsahuje různý počet sloupců, je vhodné uživatele o počtu sloupců v aktuálním řádku informovat. Popřípadě mu také sdělit, ve kterém sloupci se aktuálně nachází. Tomuto účelu slouží prvek `DotsPageIndicator`. Prvek je vložen do layoutu stránky na pozici, ve které by se měl zobrazovat.

```
<android.support.wearable.view.DotsPageIndicator
      android:id="@+id/dots"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
```

Po vložení je ještě nutné prvku přiřadit `GridPageAdapter`, který bude vracet počet sloupců a další hodnoty potřebné pro správnou indikaci.

```
DotsPageIndicator dots = (DotsPageIndicator)findViewById(R.id.dots);
dots.setPager(gridPagerAdapter);
dots.setDotColor(Color.BLACK);
dots.setDotRadius(5);
dots.setDotColorSelected(Color.RED);
dots.setDotRadiusSelected(10);
dots.setDotFadeWhenIdle(false);
```

Prvku lze také nastavit různé vzhledové vlastnosti jako je barva či velikost teček indikujících počet sloupců. V případě použití kódu výše by prvek na displeji hodinek vypadal stejně jako na obrázku (obr. 4.18). Poslední řádek kódu ještě navíc nastaví, aby

byly tečky vidět po celou dobu. Ve výchozím nastavení tečky po určité době bez interakce zmizí.

Obr. 4.18 Ukázka `DotsPageIndicator`

4.3.6. List prvků – *WearableListView*, *WearableListView*.Adapter

Poslední zde popsaný prvek reprezentuje seznam položek. `WearableListView` je alternativou k `ListView` a je optimalizován pro použití na menším displeji, kterými disponují zařízení Android Wear. Zobrazuje seznam položek a v momentě, kdy uživatel přestane seznamem rolovat, vycentruje nejbližší položku do středu displeje.

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

Prvek lze, stejně jako `GridViewPager` popsaný výše, přidat do layoutu aktivity. Nutnými vlastnostmi jsou pouze šířka (`layout_width`) a výška (`layout_height`). Prvku je pak v kódu aktivity (například ihned po vytvoření v metodě `onCreate(Bundle b)`) nastaven `WearableListView.Adapter`, který prvku předává data (včetně informace o tom, jak je zobrazit).

```
WearableListView wearableListView =
      (WearableListView)findViewById(R.id.listView);
wearableListView.setAdapter(new ListAdapter(this));
```

Přiřazení adaptéru je také podobné přiřazování adaptéru pro `GridViewPager`. Třída `ListAdapter` je nová třída, která musí dědit od `WearableListView.Adapter` a musí implementovat metody `onCreateViewHolder(ViewGroup parent, int viewType)`, `onBindHoldViewHolder(WearableListView.ViewHolder holde, int position)` a `getItemCount()`.

```
public class ListAdapter extends WearableListView.Adapter
{
      List<String> menu = new ArrayList<>();
      Context context;

      public ListAdapter(Context context)
      {
            this.context = context;
            menu.add("Položka 1");
            //Další položky
     }
... Kód dále v textu...
}
```

Jako globální proměnné jsou v kódu vytvořeny `List` pro uchování položek seznamu a `Context` pro uchování kontextu. Při vytvoření instance se pak kontext uloží a seznam se naplní textovými položkami, které se později budou zobrazovat.

```
@Override
public WearableListView.ViewHolder onCreateViewHolder(ViewGroup
      parent, int viewType)
{
      return new WearableListView.ViewHolder(
            new ListItemView(context));
}

@Override
public void onBindViewHolder(WearableListView.ViewHolder holder, int
      position)
{
      ListItemView itemHolder = (ListItemView) holder.itemView;
      TextView view = (TextView)itemHolder.itemView;
      view.setText(menu.get(position));
}

@Override
public int getItemCount()
{
      return menu.size();
}
```

Další část kódu zobrazuje implementaci tří povinných metod. Metoda `onCreateViewHolder(ViewGroup parent, int viewType)` vytváří objekt `WearableListView.ViewHolder`, který udržuje informace o pohledu. Hlavním účelem této třídy je udržet v proměnných prvky pohledu získávaných metodou `findViewById(int id)`, která může být potencionálně drahá a její časté volání může průchod seznamem zpomalit. Při tvorbě a navrácení nové instance této třídy je jako

parametr předána třída `ListItemView`, ze které objekt vznikne a která bude popsána níže, po popisu dalších dvou metod.

Metoda `onBindViewHolder(WearableListView.ViewHolder h, int position)` zajišťuje změnu dat ve vykresleném seznamu. Jako parametr je jí předán objekt vytvořený v předchozí metodě. Metoda z tohoto objektu získá v proměnné uložená vizuální pole a může do nich uložit data dle pozice. V tomto případě tedy do textového pole `itemView` uloží v každé pozici text z proměnné `menu`.

Poslední implementovaná metoda `getItemCount()` pouze vrací počet položek v seznamu.

```
public class ListItemView extends RelativeLayout implements
      WearableListView.OnCenterProximityListener
{
      ImageView imageView;
      TextView textView;

      public ListItemView(Context context)
      {
            super(context);
            View.inflate(context, R.layout.list_item, this);
            imageView = (ImageView)findViewById(R.id.imageView);
            textView = (TextView) findViewById(R.id.textView);
      }

      @Override
      public void onCenterPosition(boolean b)
      {
            imageView.animate().scaleX(1f).scaleY(1f);
            textView.animate().scaleX(1f).scaleY(1f);
      }

      @Override
      public void onNonCenterPosition(boolean b)
      {
            imageView.animate().scaleX(0.6f).scaleY(0.6f);
            textView.animate().scaleX(0.6f).scaleY(0.6f);
      }
}
```

Poslední popsanou třídou bude již výše zmíněný `ListItemView` použitý při vytváření nové instance třídy `WearableListView.ViewHolder`. Tato třída zajistí získání vizuálních prvků z předaného layoutu a jejich uložení do proměnných. Zároveň tato třída implementuje rozhraní `WearableListView.OnCenterProximityListener`, které po implementaci metod `onCenterPosition(boolean b)` a `onNonCenterPosition(boolean b)` umožňuje měnit vlastnosti vycentrovaného prvku seznamu a vlastnosti ostatních nevycentrovaných prvků. V kontruktoru třídy je

nejdříve získán `View` z XML layoutu, který definuje vzhled jednotlivých položek seznamu (obr. 4.19).

```
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent">

      <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Text"
            android:id="@+id/textView"
            android:layout_gravity="center_vertical"
            android:layout_centerVertical="true"
            android:layout_centerHorizontal="true"/>

      <ImageView
            android:layout_width="30dp"
            android:layout_height="30dp"
            android:id="@+id/imageView"
            android:background="@drawable/icon"
            android:layout_centerVertical="true"
            android:layout_toStartOf="@+id/textView"/>
</RelativeLayout>
```

Po načtení layoutu se do proměnných uloží prvky, které by měly být později měněny (zde je uložen také `ImageView`, i když se v ukázkovém kódu nemění). Po konstruktoru jsou pak implementovány metody, které umožní změnu prvků při jejich procházení. Když se prvek seznamu dostane do pozice uprostřed displeje, je zvětšen a poté, co se přesune mimo tuto pozici, je zmenšen. Celkovým výsledkem implementace je poté ukázkový seznam na obrázku (obr 4.20).

Obr. 4.19 Vzhled jedné položky

Obr. 4.20 Vytvořený seznam

4.3.7. Získávání polohy

Během vytváření aplikací pro platformu Android Wear je vhodné využít například získávání polohy a zobrazovat upozornění jenom v případě, že se uživatel vyskytuje v určité oblasti. Získávání polohy je téměř shodné se získáváním polohy na mobilních zařízeních. Služby Google Play, které poskytují API pro získávání lokace sami rozhodnou o tom, zda bude poloha získána z GPS v Android Wear zařízení nebo ze senzoru v mobilním zařízení. Zařízení Android Wear ale nejsou vždy vybavena senzory GPS a proto je důležité zajistit funkčnost i v případě, že se spojení mezi mobilním zařízením a hodinkami Android Wear přeruší.

```
public void initGoogleApiClient()
{
      googleApiClient = new GoogleApiClient.Builder(this)
            .addApi(LocationServices.API)
            .addApi(Wearable.API)
            .addConnectionCallbacks(this)
            .build();
    googleApiClient.connect();
}
```

Před samotným získáním polohy je nutné se připojit ke službám Google Play. V uvedeném kódu lze vidět vytvoření nové instance třídy `GoogleApiClient`. Této instanci je určeno, které API chce vytvářená aplikace využít (v tomto případě API pro získávání lokace a `Wearable.API`, které poskytuje služby datové vrstvy). Řádek `addConnectionCallbacks(this)` pak určuje, že třída která volá `googleApiClient.connect()` také bude implementovat rozhraní `GoogleApiClient.ConnectionCallbacks` a metody tohoto rozhraní `onConnected(Bundle b)` a `onConnectionSuspended(int i)`. Poslední řádek poté zajistí připojení k službám Google Play.

```
@Override
public void onConnected(Bundle bundle)
{
      button.setEnabled(true);
      LocationRequest locationRequest = LocationRequest.create()
            .setPriority(
                  LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY)
            .setInterval(1000);
      try
      {
            LocationServices.FusedLocationApi
                  .requestLocationUpdates(googleApiClient,
                        locationRequest, this);
      catch (SecurityException e)
      {
            e.printStackTrace();
```

```
        //Pravděpodobně není oprávnění pro získání polohy
        }
}

@Override
public void onConnectionSuspended(int i)
{
        button.setEnabled(false);
        //vypnutí UI prvků, které používají služby
}
```

Pokud připojení ke službám Google Play bylo úspěšné, je zavolána metoda `onConnected(Bundle b)`. V uvedeném kódu tato metoda povolí stisknutí tlačítka pro získání poslední polohy (kód níže) a zaregistruje získávání aktualizací o změně polohy. Pro toto zaregistrování je nutné vytvořit nový `LocationRequest`. Při vytvoření je instanci nastavena priorita a interval získávání polohy. Priorita, nastavená metodou `setPriority(int p)`, může být nastavena na čtyři různé úrovně (nastavuje se pomocí konstant třídy `LocationRequest`):

- **`PRIORITY_NO_POWER`**
  Klient nebude sám žádat o nové aktualizace polohy. Funguje jako pasivní posluchač a aktualizace získává z pozic vyžádaných ostatními aplikacemi.
- **`PRIORITY_LOW_POWER`**
  Je požadována nízká přesnost (může být až kolem 10km)
- **`PRIORITY_HIGH_ACCURACY`**
  Je požadovaná nejvyšší možná přesnost.
- **`PRIORITY_BALANCED_POWER_ACCURACY`**
  Přesnost přibližně kolem 100m. Často využívá pouze hrubý odhad polohy.

Nastavení intervalu metodou `setInterval(long l)` určuje, jak často by klient chtěl získávat aktualizace polohy v milisekundách. Nastavením intervalu ovšem není zaručeno, že klient aktualizace polohy opravdu dostane. Nemusí být doručena žádná aktualizace nebo může být doručena aktualizace později. Protože je také možné, že aktualizace budou rychlejší než nastavený interval (například) pokud aktualizace získává více aplikací), je možné metodou `setFastestIntervat(long l)` nastavit, jak nejčastěji klient aktualizace bude získávat.

Pokud bude vytvářená aplikace požadovat pouze omezený počet aktualizací, je možné nastavit počet, který bude požadován metodou `setNumUpdates(int numUpdates)`.

Po vytvoření požadavku je možné nastavit získávání polohy. K tomu je v kódu použita metoda `requestLocationUpdate(GoogleApiClient g, LocationRequest l, LocationListener ll)`, které je, kromě již popsaných vytvořených instancí, předána také třída implementující rozhraní `LocationListener`.

```
@Override
public void onLocationChanged(Location location)
{
	textView.setText("Poloha: " +
		String.valueOf(location.getLatitude())
		+ ", " + String.valueOf(location.getLongitude()));
}
```

V tomto příkladu implementuje rozhraní `LocationListener` hlavní třída obsahující všechen dosud uvedený kód, a proto také implementuje i metodu `onLocationChanged(Location location)`. Tato metoda při obdržení změny aktualizuje textové pole v uživatelském rozhraní aplikace. Pokud by aplikace požadovala pouze poslední získanou polohu, je možné ji získat metodou `LocationServices.FusedLocationApi.getLastLocation(GoogleApiClient g)`, která vrátí instanci třídy `Location`.

Protože není vždy nutné získávat polohu (například když aplikace není aktivní), je vhodné přepsat metodu `onPause()` a zrušit získávání polohy.

```
@Override
protected void onPause()
{
	super.onPause();
	if (googleApiClient.isConnected())
	{
	LocationServices.FusedLocationApi.removeLocationUpdates(
		googleApiClient, this);
	}
	googleApiClient.disconnect();
}
```

Pokud navíc používané zařízení Android Wear nemá vlastní GPS senzor, je nutné, aby aplikace počítala s možností odpojení od mobilního zařízení poskytujícího polohu. Odpojení zařízení je možné sledovat třídou, která dědí od třídy `WearableListenerService`. Poté, co jej třída začne dědit, je možné přepsat metodu `onPeerDisconnected(Peer p)`.

```
@Override
public void onPeerDisconnected(Node peer)
{
	//zastavení získávání polohy nebo vypnutí funkcí, které polohu
	využívají
}
```

Pro používání této třídy jako služby je ještě nutné ji přidat do manifestu aplikace mezi tagy `<application>` a `</application>`.

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

4.3.8. Komunikace mezi zařízením Android Wear a mobilním zařízením

Jak bylo již popsáno v kapitole 3.5, ke komunikaci mezi zařízeními lze použít tři přístupy. V této kapitole budou všechny tři přístupy popsány a názorně předvedeny.

- **Messages**

Pro odeslání menších bloku dat (například zpráv) je možné využít `MessageApi`. Pro odeslání zprávy je nutné, aby aplikace obsahovala instanci třídy `GoogleApiClient`, která je připojená ke službám Google Play (viz. kapitola 4.3.7). Po připojení je pak možné po kliknutí na tlačítko nechat odeslat zprávu.

```
public void sendMessage()
{
      new Thread(new Runnable()
      {
            @Override
            public void run()
            {
                  String nodeId = "-";

                  NodeApi.GetConnectedNodesResult result =
                        Wearable.NodeApi
                        .getConnectedNodes(googleApiClient)
                        .await();

                  List<Node> nodes = result.getNodes();

                  if (nodes.size() > 0)
                  {
                        nodeId = nodes.get(0).getId();

                        Wearable.MessageApi
                              .sendMessage(googleApiClient,
                              nodeId, "/WEAR", new String("Zprava z
                              hodinek").getBytes()).await();

                        runOnUiThread(new Runnable()
                        {
                              @Override
                              public void run()
                              {
                                    textView.setText("Odesláno");
```

```
                        }
                    });
                }
            }
        }).start();
}
```

V metodě je nutné vytvořit nové vlákno (popřípadě použít například `AsyncTask`), aby odeslání neblokovalo hlavní vlákno aplikace. Před samotným odesláním zprávy je nutné získat identifikaci zařízení, kterému má být zpráva odeslána. Tuto identifikaci lze získat metodou `getConnectedNodes(GoogleApiClient g)`, která vrátí seznam připojených zařízení. Protože těchto zařízení může být více (například více hodinek připojených k jednomu mobilnímu zařízení), odešle se zpráva pro jednoduchost pouze pro jedno z těchto zařízení. Zároveň musí být zkontrolováno, že alespoň jedno zařízení připojeno je. Samotné odeslání zprávy pak realizuje metoda `sendMessage(GoogleApiClient g, String nodeId, String path, byte[] zprava)`. Po odeslání zprávy je v kódu ještě nastavení textu pro textové pole v aktivitě (obr 4.21).

Obr. 4.21 Po odeslání zprávy

Pro přijmutí zprávy je nutné v programu vytvořit třídu dědící od `WearableListenerService`. Tuto třídu je také nutné přidat do manifestu (viz. kapitola 4.3.7). V této třídě lze přepsat metodu `onMessageReceived(MessageEvent m)`, která je zavolána v případě přijetí zprávy. Po přijetí zprávy pak lze například odeslat zprávu zpět, nebo pouze zobrazit indikaci o přijetí (obr 4.22).

```
@Override
public void onMessageReceived(MessageEvent messageEvent)
{
      Toast.makeText(getApplicationContext(), new
            String(messageEvent.getData()),
            Toast.LENGTH_SHORT).show();
}
```

Obr. 4.22 Informace o příchodu zprávy

- **Channel**

Další možností sdílení dat je využití `ChannelApi`. Toto řešení je vhodné pro odesílání souborů nebo větších bloků dat. Protože většina použitého kódu je shodná s odesílám zpráv popsaném v předcházejícím bloku, reprezentuje následující kód část, která je obsažena v metodě `run()` nově vytvořeného vlákna, v bloku podmínky `nodes.size() > 0`.

```
final PendingResult<ChannelApi.OpenChannelResult> result =
      Wearable.ChannelApi.openChannel(googleApiClient, nodeId,
      "ROZVRH");
result.setResultCallback(new
      ResultCallback<ChannelApi.OpenChannelResult>()
{
      @Override
      public void onResult(ChannelApi.OpenChannelResult
            openChannelResult)
      {
            if(openChannelResult.getStatus().isSuccess())
            {
                  File folder = getDir("soubory", 0);
                  File f = new File(folder, "soubor");
                  openChannelResult.getChannel()
                        .sendFile(googleApiClient, Uri.fromFile(f));
            }
            else
            {
                  //Zpráva nebyla odeslána
            }
      }
});
```

V kódu výše lze vidět postup, pokud namísto prostého zavolání `await()` (jak tomu bylo v předchozím bloku o zprávách), chce vývojář například znát výsledek akce a reagovat na něj. Nejdříve je tedy otevřen kanál metodou `openChannel(GoogleApiClient g, String nodeId, String path)` a výsledek akce je uložen do proměnné `result`. Této instanci je pak nastaven nový `ResultCallback`, který v metodě `onResult(OpenChannelResult o)` zkontroluje výsledek akce a v případě, že byl kanál v pořádku otevřen, odešle soubor do připojeného zařízení.

Na straně přijímajícího zařízení je nutné implementovat metodu `onChannelOpened(Channel ch)`, která je zavolána (metoda musí být v třídě dědící od `WearableListenerService`), když se jiné zařízení pokusí otevřít kanál.

```
@Override
public void onChannelOpened(Channel channel)
{
      File folder = getDir("soubory", 0);
      File f = new File(folder, "soubor");
      channel.receiveFile(googleApiClient, Uri.fromFile(f), false);
}
```

V této metodě je pak zavolána metoda `receiveFile(GoogleApiClient g, String Uri, boolean append)`, která přijatý soubor uloží do umístění předaného jako druhý parametr. Třetí parametr určuje, zda bude soubor přepsán, nebo se bude pokračovat v zapisování na konec souboru. Je také možné přepsat metodu `onInputClosed(Channel ch, int reason, int errorCode)`, která je zavolána po dokončení přenosu souboru.

Pokud je nutné sledovat, zda došlo k uzavření kanálu, je možné přepsat také metodu `onChannelClosed(Channel ch, int reason, int code)`.

- **DataItems**

Poslední zde popsanou možností je využití třídy `DataApi`. Při použití tohoto řešení jsou všechny vložené objekty kopírované mezi všechna připojená zařízení.

```
File folder = getDir("soubory", 0);
File f = new File(folder, "soubor");

PutDataRequest putRequest = PutDataRequest.create("/soubor");
Asset asset = Asset.createFromUri(f);
putRequest.putAsset("test", asset);
putRequest.setUrgent();

Wearable.DataApi.putDataItem(googleApiClient, putRequest);
```

Nejprve je v kódu vytvořen nový požadavek metodou `create(String path)`, které je jako parametr předána cesta – ta slouží například k identifikaci vložených souborů. Po

vytvoření požadavku je připravena nová instance třídy `Asset`, do které je uložen sdílený soubor. Před samotným odesláním požadavku je pak ještě nastaven příznak urgentnosti, aby data byla předaná do ostatních zařízení co nejrychleji. Poslední řádek kódu pak realizuje samotné odeslání požadavku.

```
@Override
public void onDataChanged(DataEventBuffer dataEventBuffer)
{
	for(DataEvent event : dataEventBuffer)
	{
		Map<String, DataItemAsset> assets =
			event.getDataItem().getAssets();
		String data = new String(event.getDataItem().getData());
	}
}
```

Změnu sdílených dat je možné sledovat metodou `onDataChanged(DataEventBuffer d)` (ve stejné třídě jako metody pro sledování zpráv a kanálů). Protože se může stát, že dorazí více požadavků najednou, je nutné je zpracovat všechny. V kódu je pak také vidět získání assetů a popisu (dat) z jednoho požadavku.

5. Webové služby IS STAG

Aplikace, která je součástí této práce, bude využívat webové služby informačního systému studijní agendy (dále pouze IS STAG) [17]. Tyto webové služby poskytují jednoduchý přístup k informacím o studentovi (nebo také předmětu a podobně), jako například získání rozvrhu nebo termínu zkoušek.

5.1. Použité standardy

S webovými službami nad informačním systémem IS STAG lze komunikovat pomocí dvou následujících standardů:

- **SOAP**
  Webové služby SOAP jsou vhodnější pro komunikaci typu stroj - stroj. Specifikace je psána tak, aby bylo možné vytvářet strojově klienty. Protokol SOAP je postaven na XML zprávách a proto není vhodný pro komunikaci člověk-stroj, kde je potřeba data jednoduše rozdělovat a třídit.
- **REST**
  Základem REST jsou URL adresy. Každá dostupná služba má svoji URL adresu a uživateli pouze stačí tuto adresu zavolat (popřípadě do adresy přidat parametry, ať už povinné nebo volitelné). Webové služby poté navrátí data v jednom z dostupných formátů. Výběr formátu se provádí skrze parametr poskytnutý v adrese.

5.2. Dostupné formáty

V případě využívání REST je možné zvolit, v jakém formátu služba žádaná data vrátí. IS/STAG podporuje převod do několika formátů a výběr formátu probíhá skrze parametr *outputFormat* předávaný v adrese.

- **XLS**
  Formát využívaný programem Microsoft Excel. V případě, že uživatel chce později využít tento program, je vhodné získat data v tomto formátu.
- **CSV**
  Jednoduchý textový formát vhodný pro ukládání tabulkových dat. Jednotlivé záznamy jsou odděleny čárkou.
- **JSON**
  Velmi rozšířený datový formát umožňující zápis dat jako objektů. Je nezávislý na platformě a lze jej vcelku snadno číst i strojově rozdělovat. V aplikaci se bude využívat především tento formát.
- **ICS**
  Formát pro ukládání kalendářových dat. Služby samotné nevrací data v tomto formátu, ovšem u služeb, kde to má smysl (například harmonogram roku, rozvrhy a podobné), existuje alternativa služby končící slovem „ICAL“, která vrací data v tomto formátu.

5.2.1. Kódování výstupu

Pokud chce uživatel získat data ve formátu CSV, může se mu hodit nastavení požadovaného kódování. Proto lze parametrem *outputFormatEncoding* službě předat kódování, které je vyžadováno. Výchozím kódováním je *windows-1250*.

5.3. Ověření uživatele

Některé služby obsahují údaje, které nejsou dostupné veřejnosti, ale pouze oprávněným uživatelům. Takové služby lze volat pouze přes HTTPS a při jejich volání je nutné ověřit uživatele. Ověření uživatele probíhá standardním mechanismem HTTP BASIC a lze jej provést třemi metodami.

- **Standardní (uživ. jméno a heslo)**
  Klient poskytne své uživatelské jméno a heslo dle specifikace zakódované do HTTP hlavičky. Využitelné hlavně pro klienty využívající webový prohlížeč.
- **Ticket**
  Po úspěšném přihlášení skrze HTTP BASIC je uživateli vygenerován ticket (dlouhý jedinečný řetězec), který má omezenou platnost a lze ho po dobu této platnosti (prodlužuje se s použitím při zavolání služby) použít pro budoucí ověření.
- **Cookies**
  Stejně jako v případě předchozí možnosti, server při navrácení ticketu vrací také cookie, která mimo jiné obsahuje výše uvedený ticket, a kterou lze použít pro další ověřování.

5.4. Příklad využitelných služby REST

Navrhovaná aplikace bude využívat webových služeb REST. Pro získání dat tedy bude stačit zavolat URL adresu a webové služby vrátí požadovaná data (viz kód pod tímto odstavcem), která se poté zpracují a zobrazí ve vhodné formě.

```
<ns1:getRozvrhByStudentResponse xmlns:ns1="http://stag-ws.zcu.cz/">
      ...
      <roakIdno>341523</roakIdno>
      <nazev>Programátorské strategie</nazev>
      <katedra>KIV</katedra>
      <predmet>PRO</predmet>
      <statut>B</statut>
      <ucitIdno>17239</ucitIdno>
      <ucitel>
            <ucitIdno>17239</ucitIdno>
            <jmeno>Ivana</jmeno>
            <prijmeni>Kolingerová</prijmeni>
            <titulPred>Prof. Dr. Ing.</titulPred>
            <platnost>A</platnost>
            <zamestnanec>A</zamestnanec>
      </ucitel>
      <rok>2015</rok>
```

```
      <budova>UP</budova>
      <mistnost>115</mistnost>
      <kapacitaMistnosti>60</kapacitaMistnosti>
      <planObsazeni>60</planObsazeni>
      <obsazeni>36</obsazeni>
      <typAkce>Přednáška</typAkce>
      <typAkceZkr>Př</typAkceZkr>
      <semestr>ZS</semestr>
      <platnost>A</platnost>
      <den>Pondělí</den>
      <denZkr>Po</denZkr>
      ...
</ns1:getRozvrhByStudentResponse>
```

5.4.1. Příklad adresy

Jako příklad uvádím adresu pro získání rozvrhu uživatele.

https://stag-ws.zcu.cz/ws/services/rest/rozvrhy/getRozvrhByStudent?osCislo=A12B0141P

Celou adresu lze rozdělit na části:

- https://stag-ws.zcu.cz/ws/services/ - Základní URL webových služeb
- rest/rozvrhy/ - Adresa konkrétní služby
- getRozvrhByStudent - Název metody, která data navrátí
- ?osCislo=A12B0141P - Parametr předaný metodě

Základní URL zůstává vždy stejná. Mění se pouze volané služby, metody a zadávané parametry.

6. Aplikace Rozvrh

Jedním z cílů této práce je vytvoření aplikace pro zařízení (hodinky) Android Wear využitelné v prostředí ZČU. Vytvořená aplikace využívá webové služby IS STAG a má následující vlastnosti a funkce:

- Umožňuje stažení rozvrhu přímo ze zařízení, pokud má zařízení přístup k internetu
- Pokud není internet dostupný, je možné rozvrh stáhnout na mobilním zařízení a odeslat ho do Android Wear
- Může uchovávat několik stažených rozvrhů
- Rozvrh nastavený jako hlavní lze zobrazit z menu jedním kliknutím
- Obsahuje správu rozvrhů
- Správa rozvrhů nabízí stažení nového rozvrhu
- U stažených rozvrhů lze rozvrh prohlížet, aktualizovat, smazat nebo nastavit jako hlavní
- Vždy zobrazuje rozvrh pro aktuální týden (nebo pokud je víkend, týden příští)

Aplikace vyžaduje zařízení Android Wear s verzí systému Android nejméně 5.1. Pro správné rozložení prvků aplikace je také požadavkem, aby hustota displeje cílového zařízení byla *hdpi* (přibližně 240dpi). Také je doporučeno používat zařízení s rozlišením alespoň 320x320 pixelů. Mobilní verze aplikace, umožňující stažení a odeslání rozvrhu do hodinek, vyžaduje verzi systému Android alespoň 4.4.

Vytvořená aplikace obsahuje dva moduly, modul *wear* a modul *mobile*. Jednotlivé moduly reprezentují část aplikace, která se spustí na daném zařízení. Protože hlavním modulem obsahujícím většinu funkčnosti je modul *wear*, bude v následujících podkapitolách popsán tento modul (a bude na něj odkazováno jako na aplikaci). Modul *mobile*, který slouží k odeslání dat z mobilního zařízení do zařízení Android Wear bude popsán v kapitole 6.11.

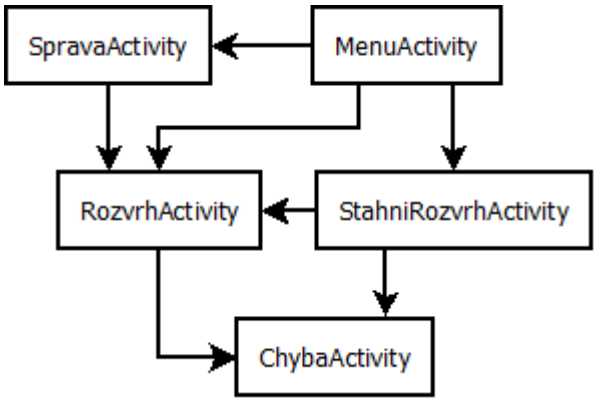
Dle dokumentace [18] emulátor Android Wear neumožňuje hlasový vstup. Protože testování (kap. 7) probíhalo pouze na emulátoru, neumožňuje nyní aplikace získávat osobní číslo hlasovým vstupem. Přestože se mi podařilo úspěšně použít mikrofon k diktování a obdržení textového přepisu (v kódu aplikace jsou tyto metody zachovány), nebylo diktování spolehlivé. Nová verze Android Wear 2.0 [19] navíc umožní použití dalších metod pro uživatelský vstup (například gesta na obrazovce) a vytvořené aplikaci je v případě použití na reálném zařízení možné odeslat rozvrh z mobilního zařízení, věnoval jsem svoji pozornost ostatním částem aplikace.

6.1. Struktura aplikace

Aplikace je rozdělená do několika balíků tak, aby pří případných úpravách, či opravách aplikace bylo snadné najít požadovaný soubor s kódem (příloha A).

- **aktivity**
  Obsahuje balíky popsané níže a třídy reprezentující aktivity.
- **aktivity.menu, aktivity.rozvrh, aktivity.sprava**
  Obsahuje aktivity jednotlivých částí aplikace. K těmto aktivitám obsahuje také adaptéry (`FragmentGridPagerAdapter`) a fragmenty těchto adaptérů.
- **utils**
  Obsahuje třídy využívané ostatními částmi programu a třídu reprezentující službu aplikace.

Aplikace obsahuje celkem pět aktivit. Tyto aktivity jsou mezi sebou provázány (obr. 6.1) a budou popsány v dalších kapitolách.



Obr. 6.1 Diagram aktivit

6.2. AndroidManifest.xml

V každé aplikaci (popřípadě každém modulu aplikace) vyvíjené pro Android je obsažen soubor *AndroidManifest.xml*. Tento soubor obsahuje důležité informace o vytvářené aplikaci (např. balík (*package*) aplikace, seznam aktivit a dalších prvků, vyžadovaná oprávnění a tak dále).

6.2.1. Požadovaná oprávnění

Aplikace *Rozvrh* (oba její moduly) při instalaci vyžaduje dvě oprávnění.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
      android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Prvním vyžadovaným oprávněním je oprávnění k přístupu k internetu a jeho používání. Druhým oprávněním je pak aplikaci poskytnuta možnost kontrolovat stav připojení k internetu.

6.2.2. Seznam aktivit a služeb

Vytvořená aplikace obsahuje pět aktivit, které budou popsány v dalších kapitolách. Všechny tyto aktivity jsou v souboru AndroidManifest.xml uvedeny. Hlavní aktivita je pak určena tagem `<intent-filter>`, který udává možnosti aktivity.

```
<intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Součástí aplikace je také jedna služba, která naslouchá příchozím připojením z mobilního zařízení, které může odeslat nový rozvrh.

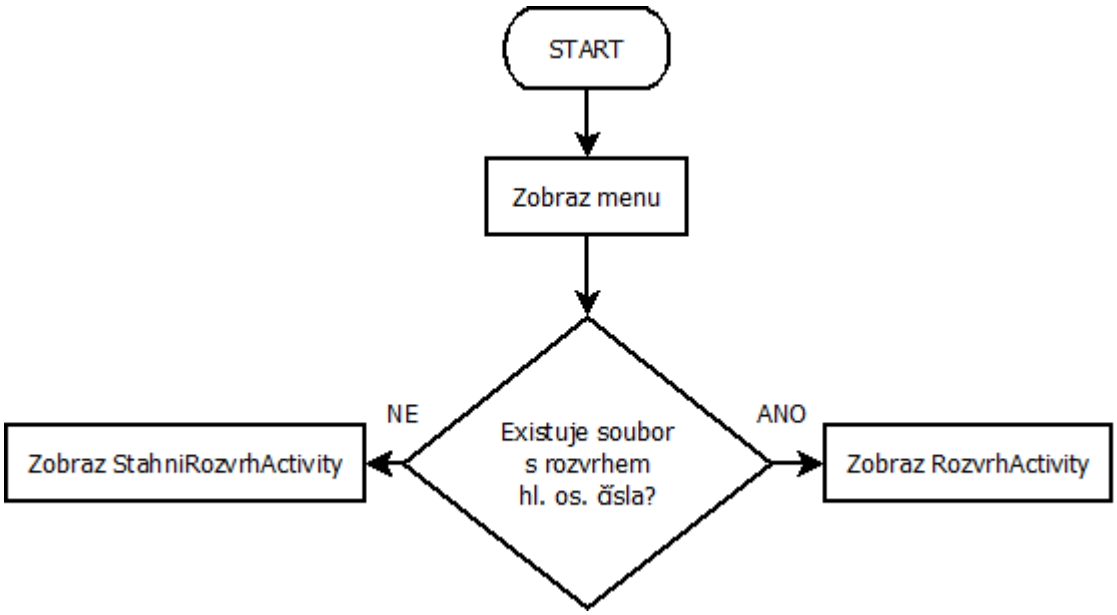
```
<service android:name=".utils.WearListenerService" >
      <intent-filter>
            <action android:name=
                  "com.google.android.gms.wearable.BIND_LISTENER" />
      </intent-filter>
</service>
```

6.3. Hlavní menu

Po spuštění aplikace je zobrazeno hlavní menu reprezentováno aktivitou `MenuActivity`. Tato aktivita je pak závislá na dalších dvou třídách. Všechny tři třídy budou popsány nyní.

6.3.1. Aktivita *MenuActivity*

Po spuštění aplikace je spuštěna aktivita `MenuActivity`. Aktivita po spuštění zobrazí hlavní menu aplikace. Ihned po jeho zobrazení aktivita zkontroluje, zda existuje soubor s rozvrhem pro hlavní osobní číslo. Toto osobní číslo je uloženo v nastavení aplikace. Nastavení lze získat a upravit třídou `SharedPreferences` [20]. Pokud soubor existuje, je zobrazena aktivita `RozvrhActivity`, zobrazující rozvrh pro hlavní osobní číslo. Pokud soubor neexistuje, je spuštěna aktivita `StahniRozvrhActivity`, která nabízí možnost stáhnout nový rozvrh. Diagram aktivity je možné vidět na obrázku 6.2.



Obr. 6.2 Diagram `MenuActivity`

Samotné menu je realizováno jako matice prvků (kap. 4.3.4) a využívá `FragmentGridPagerAdapter`.

6.3.2. *MenuPickerAdapter*

`MenuPickerAdapter` je třídě dědící od `FragmentGridPagerAdapter`, který byl popsán výše v kapitole 4.3.4. Při vytvoření nové instance této třídy jsou jí předány položky menu, které se budou zobrazovat. Implementovaná metoda `getFragment(int row, int col)` pak vrací nový fragment. V kapitole 4.3.4 byla popsána možnost navrácení nového fragmentu reprezentovaného třídou `CardFragment`.

```
public Fragment getFragment(int row, int col)
{
      return MenuFragment.newInstance(nadpisy.get(row));
}
```

Vytvářená aplikace ovšem nezobrazuje karty, ale jiné layouty a proto je fragment reprezentován další třídou s názvem `MenuFragment`. Tato třída je volána statickou metodou `newInstance(String text)`. Důvod tohoto volání je popsán v další kapitole.

6.3.3. *MenuFragment*

Tato třída reprezentuje jednotlivé položky menu. Třída dědí od třídy `Fragment` a implementuje metodu `onCreateView(LayoutInflater in, ViewGroup v, Bundle b)`. Tato metoda je volána při zobrazení a je v ní nastaven vzhled celého fragmentu.

Vzhled každé položky menu vychází ze stejného layoutu, který je získán z XML souboru metodou `inflater.inflate(R.layout.fragment_menu, container, false)`. Metoda vrací novou instanci třídy `View`, která obsahuje celý layout a je v ní tedy možno měnit jednotlivé prvky vzhledu. V tomto případě je z fragmentu získáno tlačítko a jeho text je změněn na text předaný fragmentu. Po změně textu ještě následuje nastavení nové akce po kliknutí, aby se po stisku tlačítka zobrazila správná část programu.

```
public static MenuFragment newInstance(String text)
{
      MenuFragment sp = new MenuFragment();
      Bundle b = new Bundle();
      b.putString("Rozvrh", text);
      sp.setArguments(b);

      return sp;
}
```

Samotná třída je z adaptéru volána statickou metodou. Tato statická metoda vytvoří novou instanci třídy a nastaví jí text, který je potřeba pro zobrazení na tlačítku a nastavení správné akce po kliknutí. Předaný text je pak v metodě `onCreate(Bundle b)` získán a nastaven do globální proměnné třídy. Důvodem pro využití statické třídy je znovu vytváření fragmentů systémem. Systém může fragment vytvořit znovu, a pokud se tak stane, volá konstruktor bez parametrů. V případě zavolání tohoto konstruktoru, nejsou předány žádné parametry, které jsou potřeba například pro nastavení textu tlačítka. Použitím statické metody a předáním parametrů pomocí třídy `Bundle` je dosaženo toho, že parametry budou dostupné v této třídě i při znovuvytvoření fragmentu skrze konstruktor bez parametrů.

6.3.4. Vzhled menu

Vzhled menu je definován layoutem menu a fragmentem popsaným v minulé třídě. Layout menu obsahuje pouze `GridViewPager`, kterému je v kódu nastaven jeho adaptér. Fragment pak vychází z jednoduchého layoutu, který obsahuje jedno tlačítko (obr 6.3).

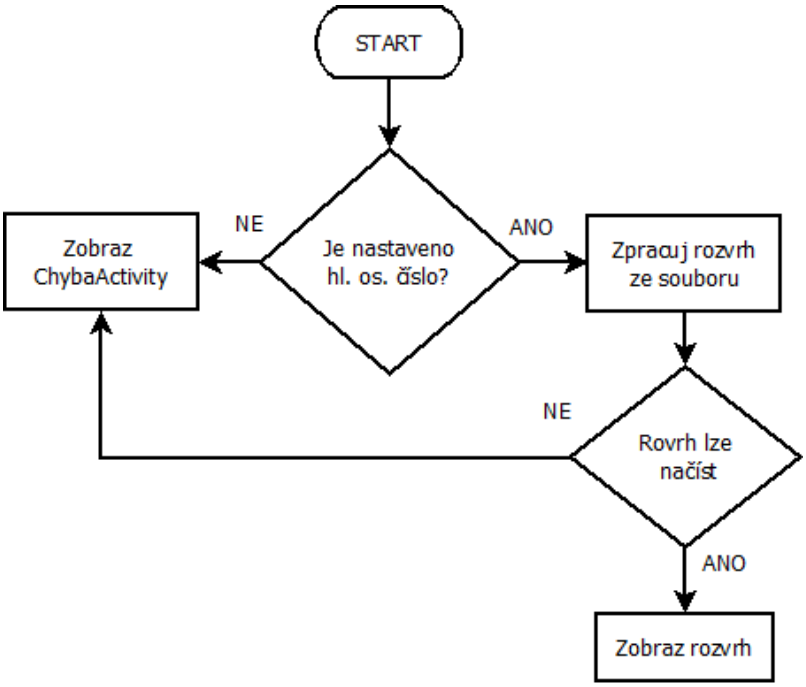
Obr. 6.3 Vzhled jednoho fragmentu

6.4. Rozvrh

Rozvrh je zobrazen, stejně jako menu, jako matice prvků. Je tedy také tvořen třemi třídami popsanými dále.

6.4.1. Aktivita *RozvrhActivity*

Po startu aktivity zkontrolováno nastavení hlavního rozvrhu. Pokud je nastaven, proběhne jeho načtení ze souboru a pokud i to proběhlo v pořádku, je rozvrh zobrazen. Průběh těchto akcí lze vidět na diagramu (obr. 6.4).



Obr. 6.4 Diagram akcí po spuštění aktivity

Po nastavení adaptéru pro zobrazení rozvrhu je jako výchozí zobrazen aktuální den v týdnu (pokud je víkend, je zobrazeno pondělí následujícího týdne). Všechny stažené rozvrhy obsahují data celého roku. Protože existují předměty, které mají rozvrhovou akci například pouze jednou za semestr nebo pouze v určité týdny, je načtený rozvrh vždy zpracován a zobrazen pouze pro aktuální týden. Zpracovaný rozvrh je pak předán adaptéru pro zobrazení.

6.4.2. *RozvrhPickerAdapter*

Stejně jako menu, je rozvrh tvořen z matice prvků, a proto je nutné mít adaptér pro poskytnutí dat. V tomto případě je ovšem vracen jak `CardFragment`, reprezentující kartu s textem, tak vlastní fragment. Karta je vracena v případě, že je zobrazen první sloupec rozvrhu. Tento sloupec reprezentuje dny v týdnu (obr. 6.5).

Obr. 6.5 Karta se dnem v týdnu

Karta obsahuje kromě dne v týdnu také počet předmětů, které vpravo od karty následují. Tyto počty se spočtou při vytvoření instance třídy a uloží se do pole, ze kterého jsou pak získávány.

Pokud má být zobrazen předmět, je vytvořena nová instance třídy `RozvrhFragment`.

6.4.3. *RozvrhFragment*

Tato třída reprezentuje jednotlivé fragmenty předmětů v zobrazeném rozvrhu. Třída opět obsahuje statickou metodu `newInstance(Predmet p)`, která vytvoří novou instanci, předá jí parametry a navrátí jí. Stejně jako fragment menu, i zde je přepsána metoda `onCreateView(LayoutInflater i, ViewGroup v, Bundle b)`. V této metodě je získán layout z XML a do prvků layoutu jsou nastaveny údaje o zobrazovaném předmětu.

6.4.4. Vzhled rozvrhu

V aktivitě rozvrhu můžou být zobrazeny dva prvky. První možností je zobrazení jednoduché karty (obr. 6.5). Druhá možnost je pak zobrazení vlastního fragmentu (obr 6.6).

Obr. 6.6 Vzhled fragmentu s předmětem

Fragment vychází z layoutu, který je složen převážně z textových polí. Jako oddělovací přímka je použit prvek `View`.

```
<View
      android:layout_width="match_parent"
      android:layout_height="1dp"
      android:background="@android:color/darker_gray"/>
```

Levá strana fragmentu pak obsahuje textové pole, které má jako pozadí nastavenou barvu dle typu rozvrhové akce.

6.5. Správa předmětu

Správa předmětu je menu s volbami akcí pro jednotlivé rozvrhy. V tomto menu je možné jednotlivé rozvrhy mazat, aktualizovat nebo vybrat jako hlavní. Je zde také možnost stáhnout nový rozvrh. Menu správy je také vytvořeno jako matice prvků.

6.5.1. Aktivita *SpravaActivity*

Oproti ostatním dříve popsaným aktivitám, neprovádí tato aktivita při spuštění žádné zvláštní akce. Aktivita rovnou vytvoří novou instanci adaptéru a předá mu seznam předmětů.

6.5.2. *SpravaPickerAdapter*

Adaptér pro zobrazení prvků správy pouze vrací fragmenty obsahující prvky správy.

6.5.3. *SpravaFragment*

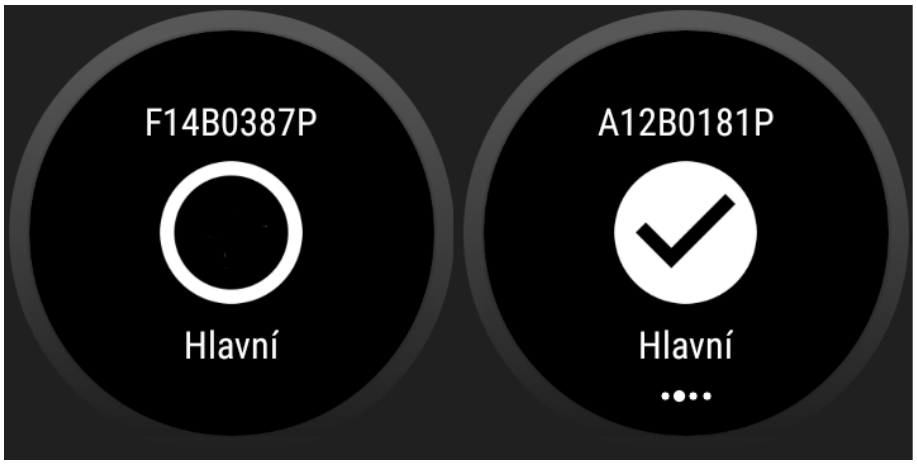
Fragment správy je opět vytvářen skrze statickou metodu `newInstance(int row, int col, String rozvrh)`. Třídě je předána pozice zobrazovaného fragmentu a osobní číslo rozvrhu. Při vytváření fragmentu je nejdříve vždy nastaveno osobní číslo zobrazené v horní části fragmentu. Další nastavení fragmentu je rozhodováno dle pozice. Pokud je zobrazen první řádek, obsahuje tento řádek pouze jeden prvek s tlačítkem pro přidání nového rozvrhu (obr. 6.7).

Obr. 6.7 Tlačítko pro přidání nového rozvrhu

Pokud jsou již staženy alespoň nějaké rozvrhy, jsou zobrazeny v dalších řádkách. Každý další řádek obsahuje několik akcí, které lze s jednotlivými rozvrhy provádět. Vzhled jednotlivých akcí vychází ze stejného souboru s layoutem. Mění se vždy pouze pozadí tlačítka, akce po jeho stisku a text pod tlačítkem.

- **Přidat**
  Akce „Přidat“ otevře aktivitu pro stažení rozvrhu (`StahniRozvrhActivity`).
- **Zobrazit**
  Po stisku tlačítka v tomto fragment je zobrazena aktivita `Rozvrh`, která zobrazí vybraný rozvrh.

- **Hlavní**
  Tlačítko hlavní může mít dvě různá pozadí (obr. 6.8). Pozadí tlačítka je nastaveno podle hlavního rozvrhu nastaveného v nastavení aplikace. Pokud je rozvrh nastaven jako hlavní, je tlačítko „zaškrtnuto“. V opačné situaci je tlačítko „nezaškrtnuto“. Po stisku tlačítka je pak rozvrh nastaven jako hlavní a pozadí tlačítka se změní.



Obr. 6.8 Dva různé stavy tlačítka

- **Aktualizovat**
  Po stisku tlačítka „Aktualizovat“ je vybraný rozvrh aktualizován. Pokud je aktualizování dokončeno v pořádku, zobrazí se `Toast` se zprávou „OK“. Pokud není, je zobrazena aktivita s chybou (kap. 6.7). Samotné stažení je realizováno stejně jako v aktivitě `StahniRozvrhActivity` (kap. 6.6).
- **Smazat**
  Stiskem tohoto tlačítko je docíleno smazání souboru s rozvrhem. Po jeho smazání je celá aktivita spuštěna znovu s aktualizovaným seznam rozvrhů. Pokud byl odstraněný rozvrh nastaven jako hlavní, je také odstraněn z nastavení.

6.6. Aktivita *StahniRozvrhActivity*

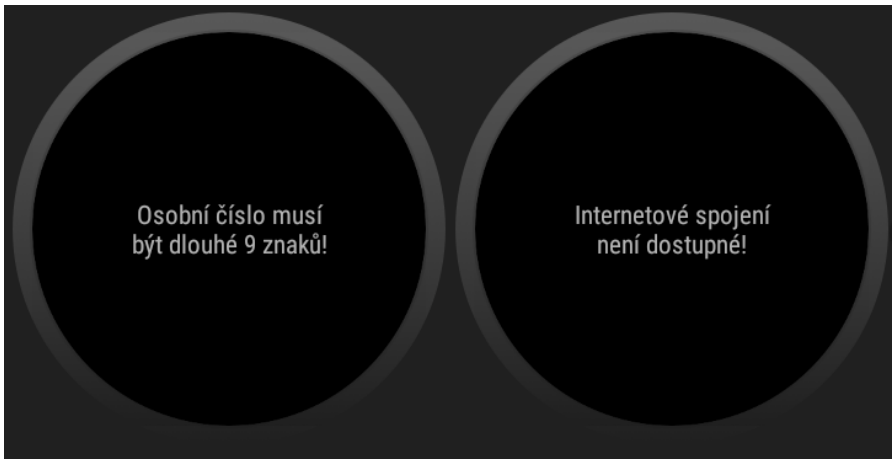
Pokud je aplikace spuštěna poprvé, nebo je ve správě stisknuto tlačítko „Přidat“, je uživateli zobrazeno jednoduché uživatelské rozhraní obsahující jedno textové pole a tlačítko (obr. 6.9). Do textového pole uživatel může zadat osobní číslo, pro které chce stáhnout a uložit rozvrh. Po zadání čísla je zkontrolována jeho délka (musí být dlouhé devět znaků). Pokud je číslo v pořádku, je zkontrolováno připojení k internetu. Tato kontrola je realizována metodou `testPripojeni()`. Metoda jako výsledek vrací, zda je připojení dostupné nebo zda dostupné není.

```
boolean testPripojeni()
{
	ConnectivityManager cm = (ConnectivityManager)
		getSystemService(Context.CONNECTIVITY_SERVICE);
	NetworkInfo netInfo = cm.getActiveNetworkInfo();
	if(netInfo != null)
	{
		return netInfo.isConnected();
	}
	return false;
}
```

Kontrola připojení probíhá třídami `ConnectionManager` a `NetworkInfo`. Pokud je připojení dostupné vrátí metoda `isConnected()` hodnotu `true`.

Obr. 6.9 Aktivita `StahniRozvrhActivity`

Pokud některá z těchto kontrol selže (délka osobního čísla, dostupnost internetové připojení), zobrazí se aktivita s chybou (obr. 6.10).



Obr. 6.10 Ukázka chybových hlášení

Pokud bylo osobní číslo i internetové spojení v pořádku, spustí se stahování rozvrhu. Protože stahování dat může blokovat hlavní vlákno aplikace, je ke stahování využita třída `AsyncTask`. Třídy, které od této třídy dědí, jsou vhodné pro spouštění úloh na

pozadí a zároveň nabízí možnost interagovat s hlavním vláknem uživatelského rozhraní.

```
class LoginTask extends AsyncTask<String, Void, List<Predmet>>
{
	protected List<Predmet> doInBackground(String... str)
	{
		//úloha v pozadí (stažení rozvrhu)
		return rozvrh;
	}

	protected void onPostExecute(List<Predmet> rozvrh)
	{
		//po dokončení úlohy (po stažení)
	}
}
```

Aplikace vytvoří vlastní třídu a implementuje v ní metody `doInBackground(String... str)` a `onPostExecute(List<Predmet> rozvrh)`. První zmíněná metoda provede úlohu ve vlastním vlákně (tedy stažení rozvrhu) a vrátí výsledek. Výsledek může být seznam předmětů nebo hodnota `null` (v případě chyby při stahování).

```
String target = Konstanty.STAG_ROZVRH
	+ osobniCislo
	+ "&outputFormatEncoding=windows-1250";

URL url = new URL(target);
URLConnection connection = url.openConnection();
connection.connect();

BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream(), "UTF-8"));
```

Pro stažení rozvrhu z webových služeb STAG je využito třídy `URLConnection`. Po připojení k webovým službám je získán `InputStreamReader` a z něho je přečten celý rozvrh.

Druhá zmíněná metoda je již volána hlavním vláknem uživatelského rozhraní a může tedy měnit jeho prvky. Jako parametr je jí předaná hodnota vrácena první metodou. Proběhne tedy kontrola, zda nebyla navrácena hodnota `null`, a pokud nebyla, uloží seznam předmětů (rozvrh) do souboru – pokud by byla navrácena, zobrazí se aktivita s chybou (viz. výše). Soubor se ukládá do složky aplikace, do podsložky definované konstantou ve třídě `Konstanty`. Po uložení souboru je uloženo toto osobní číslo jako hlavní. Po uložení souboru a nastavení, je spuštěna aktivita `RozvrhActivity`.

6.7. Aktivita *ChybaActivity*

Tato aktivita je volána v případě nutnosti zobrazení delších chybových hlášek. Zobrazená chybová hláška je aktivitě předána pomocí `Extras`. Jako druhá možnost se jevilo použití třídy `Toast`. Toto řešení jsem ale zavrhl z důvodu malé možnosti úpravy zobrazení (různá barva pozadí a písma, velikost textu) a délky zobrazení. Takto se uživatel o chybě dozví a musí aktivitu sám pomocí gesta zrušit. Díky tomuto řešení je tedy jasné, že se uživatel o chybě dozvěděl.

6.8. Třída *WearListenerService*

Třída `WearListenerService` slouží ke komunikaci s mobilní verzí aplikace. V případě, že hodinky, na kterých je aplikace nainstalována, nemají přístup k internetu, je možné odeslat rozvrh z mobilního zařízení. Tato třída je definována v manifestu jako služba a dědí od `WearableListenerService` . Díky dědění je možné tedy přepsat metodu `onChannelOpened(Channel channel)` zavolanou při otevření kanálu a `onInputClosed(Channel ch, int reason, int errorCode)` zavolanou po dokončení přenosu. Popis přenosu souborů mezi zařízeními, včetně ukázek kódu, je dostupný v kapitole 4.3.8. Třída po otevření kanálu uloží přenesený soubor s rozvrhem a po dokončení přenosu tento rozvrh nastaví jako hlavní. Následně také otevře aktivitu `MenuActivity` a přenesený rozvrh zobrazí – aplikace se tedy chová jako po novém spuštění.

6.9. Třída *Konstanty*

V této třídě jsou uloženy všechny v aplikaci využívané konstanty. Není zde obsaženo příliš mnoho konstant, ovšem pro případ budoucího rozšiřování aplikace je tato třída vhodná. Aktuálně jsou zde uloženy například adresy webových služeb STAG nebo název souboru s uloženým nastavení pro třídu `SharedPreferences`.

6.10. Třída *Predmet*

Pro uložení informací o jednotlivých předmětech rozvrhu byla vytvořena třída `Predmet`. Krom standardních částí (proměnné, konstruktory, getry a setry) obsahuje třída dvě statické metody. Tyto třídy slouží pro převod dne týdne reprezentovaného číslem na text s názvem dne.

Třída také implementuje rozhranní `Parcelable`. Důvodem k tomuto kroku je předávání instance této třídy mezi třídami `RozvrhPickerAdapter` a `RozvrhFragment`, které slouží k zobrazení rozvrhu. Třída tedy také obsahuje povinné metody `describeContents()` a `writeToParcel(Parcel desc, int flags)`. První zmínění metoda vrací pouze nulu. Druhá zmíněná metoda zapíše všechny informace (proměnné) do objektu `Parcel`, ze kterého lze pak v konstruktoru `Predmet(Parcel in)` vše zase přečíst a uložit do proměnných. Při přenosu instance této třídy je využito statické pole, které zařídí zavolání správného konstruktoru.

```
public static final Parcelable.Creator<Predmet> CREATOR = new
      Parcelable.Creator<Predmet>()
{
      public Predmet createFromParcel(Parcel in)
      {
            return new Predmet(in);
      }

      public Predmet[] newArray(int size)
      {
            return new Predmet[size];
      }
};
```

6.11. Modul *mobile*

Mobilní verze aplikace (modul *mobile*) slouží ke stažení rozvrhu na mobilním zařízení a jeho odeslání do hodinek. Aplikace obsahuje jednoduché uživatelské rozhraní (obr. 6.11) podobné tomu v aktivitě `StahniRozvrhActivity`.

Po zadání osobního čísla a stisku tlačítka jsou provedeny kontroly délky osobního čísla a dostupnosti připojení k internetu (stejně jako při stahování rozvrhu v kap. 6.6). Po stažení a uložení rozvrhu jsou získána připojená zařízení, ke každému je otevřen kanál (kap. 4.3.8) a rozvrh je odeslán.

Kromě hlavní třídy `MainActivity` je v modulu obsažena také třída `Predmet`, která je shodná s třídou v modulu *wear* a třída `Konstanty` obsahující konstanty využívané v tomto modulu.

Obr. 6.11 Vzhled mobilní verze

7. Testování aplikace Rozvrh

Vytvářená aplikace byla průběžně testována během celého vývoje. Přesto je vhodné aplikaci otestovat po implementaci všech jejích funkcí. Pro otestování základní funkcionality uživatelského rozhraní modulu *wear* byla vytvořena sada testů pro otestování knihovnou Espresso [21].

7.1. Espresso

Espesso je knihovna umožňující automatické testování uživatelského rozhraní aplikací pro Android. Před použitím samotné knihovny bylo nutné projekt (aplikaci) připravit. Pokud Android Studio samo nevytvořilo v modulu *wear* složku *androidTest*, je nutné ji vytvořit ručně. Složka byla vytvořena v adresáři *src* (*PROJEKT*/wear/src). V této složce následují podsložky reprezentující název balíku aplikace, celá cesta tedy bude:

PROJEKT/wear/src/androidTest/java/com/example/sk1x1/rozvrh

V této složce jsou následně vytvářeny třídy obsahující testovací metody. Před samotným vytvářením je ještě nutné přidat závislosti do souboru *build.gradle* (soubor modulu *wear*). Do tohoto souboru musí být vloženy následující řádky:

```
android
{
      ...
      defaultConfig
      {
            ...
            testInstrumentationRunner
                  "android.support.test.runner.AndroidJUnitRunner"
      }
...
}
dependencies
{
      ...
      testCompile 'junit:junit:4.12'
      compile 'com.android.support:support-annotations:23.1.1'
      androidTestCompile
            'com.android.support.test.espresso:espresso-core:2.2.2'
      androidTestCompile
            'com.android.support.test.espresso:espresso-
                  intents:2.2.2'
      androidTestCompile 'com.android.support.test:runner:0.5'
      androidTestCompile 'com.android.support.test:rules:0.5'
}
```

Po úpravě souboru je možné začít vytvářet testy. Projekt obsahuje dvě třídy obsahující sady testů. První třída `PrvniSpusteni` obsahuje tři testy, které jsou určeny pro otestování před prvním spuštěním aplikace:

- `test1KratkeOsobniCislo()`
  Otestuje zobrazení chybové hlášky po zadání krátkého osobního čísla.
- `test2ChybneOsobniCislo()`
  Otestuje zobrazení chybové hlášky po zadání chybného osobního čísla.
- `test3SpravneOsobniCislo()`
  Otestuje zobrazení rozvrhu po zadání správného osobního čísla.

Druhá třída `PoStazeni` pak obsahuje další sadu testů, které jsou určeny pro testování po stažení funkčního rozvrhu (respektive po spuštění první sady testů):

- `test1zobrazenRozvrh()`
  Otestuje zobrazení rozvrhu po spuštění aplikace.
- `test2zpetDoMenu()`
  Otestuje zobrazení menu po návratu z rozvrhu.
- `test3tlacitkoSpravaViditelneAFunkcni()`
  Otestuje, zda je tlačítko „Správa“ viditelné a otevře správu rozvrhů.
- `test4spravaAkcePridat()`
  Otestuje otevření aktivity `StahniRozvrhActivity` po kliknutí na tlačítko „Přidat“.
- `test5spravaAkceZobrazit()`
  Otestuje zobrazení rozvrhu po stisku tlačítka
- `test6spravaAkceHlavni()`
  Otestuje zobrazení tlačítka „Hlavní“. Netestuje jeho funkčnost, protože pro účely testování se stáhne pouze jeden rozvrh.
- `test7spravaAkceAktualizovat()`
  Otestuje úspěšné zahájení stahování. Netestuje úspěšné stažení.
- `test8spravaAkceSmazat()`
  Otestuje zobrazení zprávy (`Toast`) o úspěšném smazání rozvrhu.
- `test9rozvrhSmazan()`
  Otestuje zobrazení `StahniRozvrhActivity` po spuštění aplikace po smazání rozvrhu.

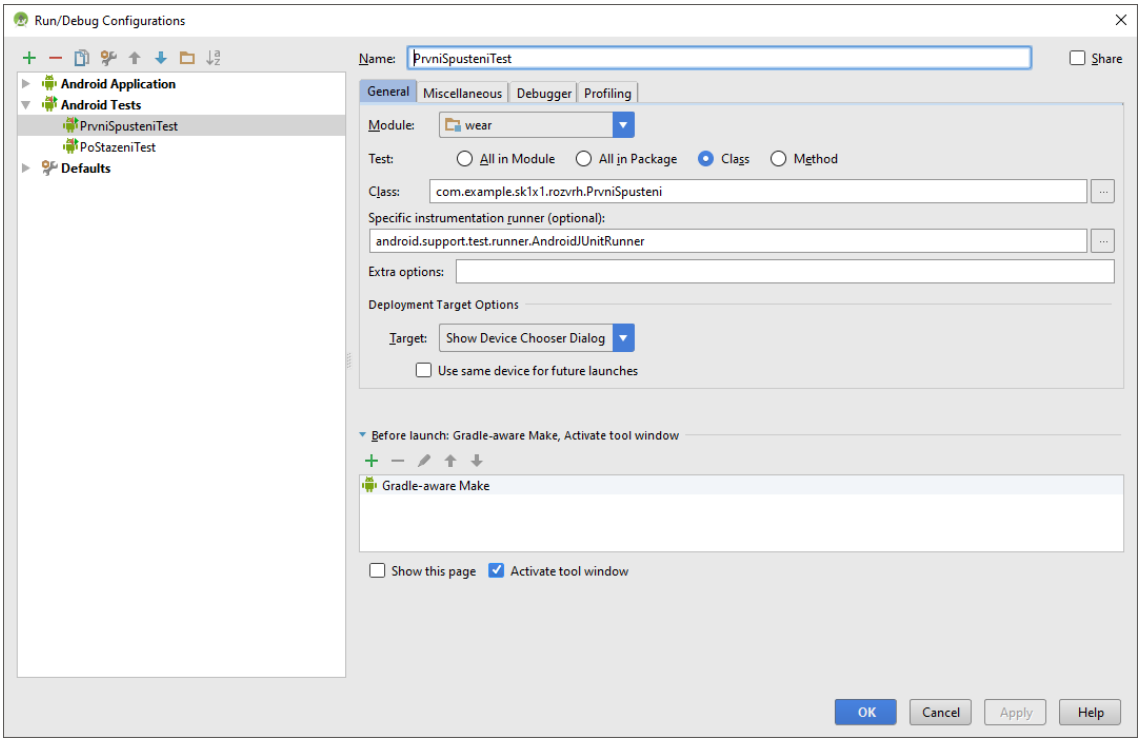
Obě třídy před svým deklarováním obsahují dvě anotace. První anotace zajistí spuštění testů třídou `JUnit4`. Druhá anotace zajišťuje spuštění testů v jejich abecedním pořadí. Od verze 4 nemusí být testy spouštěny v jejich abecedním pořadí. Testy by měly být nezávislé na pořadí spuštění, ovšem v tomto projektu, zejména testy ve třídě `PrvniSpusteni`, je nutné spustit ve správném pořadí, protože poslední test ovlivní příští spuštění aplikace.

```
@RunWith(JUnit4.class)
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
```

Před metodami je navíc v obou třídách nastavena aktivita, která se s každým testem spustí. V tomto případě jde o aktivitu `MenuActivity`, která je výchozí třídou aplikace.

```
@Rule
public ActivityTestRule<MenuActivity> activityRule =
      new ActivityTestRule<>(MenuActivity.class);
```

Před samotným spuštěním testů je také nutné nastavit jejich spuštění. Toto nastavení v Android Studiu nastavit v nabídce „Run/Edit Configurations…“. V následně otevřené nabídce lze přidat konfiguraci tlačítkem „+“ v levém horním rohu. Na obrázku je vidět ukázková konfigurace (obr. 7.1).



Obr. 7.1 Ukázka konfigurace spuštění testů z jedné třídy

7.2. Spojení emulátoru hodinek a fyzického mobilního zařízení

Protože jsem při vývoji a testování aplikace neměl možnost využít fyzického zařízení Android Wear, využíval jsem během celé práce emulátory zařízení dostupné v Android Studiu. Aplikace umožňuje získání rozvrhu z mobilního zařízení a proto je nutné emulátor s tímto zařízením spojit. Aby bylo možné zařízení spojit, je potřeba mobilní zařízení připojit k PC kabelem a povolit režim „Pro vývojáře“. Postup k povolení tohoto režimu může být na různých zařízeních a verzích systému odlišný, proto je nutné si návod vyhledat pro konkrétní zařízení a systém. Po aktivování tohoto režimu je nutné v nově odemčeném nastavení povolit ladění přes USB. Po těchto krocích by mělo být

mobilní zařízení připojené k PC dostupné v nabídce při spuštění projektu. Posledním krokem je pak instalace aplikace Android Wear z obchodu Google Play.

Emulátor hodinek je možné vytvořit přímo v Android Studiu. Po jeho vytvoření a spuštění je na počítači nutné nastavit přesměrování příkazem:

```
adb -d forward tcp:5601 tcp:5601
```

Po nastavení přesměrování stačí na mobilním zařízení spustit aplikace Android Wear a spojit ji s emulátorem. Podrobnější postup lze nalézt na webových stránkách Android Developers [22].

7.3. Nastavení týdne

Protože aplikace zobrazuje rozvrh pro aktuální týden, bylo nutné zajistit pro testovací účely možnost změny zobrazovaného týdne. Ve třídě `Konstanty` je tedy proměnná `TESTOVACI_TYDEN`, která umožňuje nastavit vlastní týden. Pokud je týden v této proměnné menší než nula, zobrazí se týden aktuální.

7.4. Průběh testování

Testování modulu *wear* jsem prováděl na emulátorech různých nastavení (tab. 7.1):

Displej	Rozlišení	DPI	Verze systému
Kulatý	320x320	hdpi	6.0
Čtvercový	280x280	hdpi	6.0
Čtvercový	320x320	280	6.0
Kulatý	400x400	280	5.1.1
Kulatý	320x320	hdpi	5.1.1

Tab. 7.1 Testovaná zařízení

Na všech výše uvedených zařízení byla aplikace otestována automatickými testy. Testy byly spuštěny třikrát za sebou a všechny proběhli bez selhání. U emulátoru zařízení s verzí Android 5.1.1 není po výchozím nastavení a spuštění emulátoru možnost simulace sítě a proto bylo nutné rozvrh odeslat ručně z mobilního zařízení.

Po automatických testech proběhlo testování ostatní funkcionality. Byl testován správný chod následujících funkcí:

- Otevření a prohlížení rozvrhu
- Přidání nového rozvrhu
- Zobrazení jiného než hlavního rozvrhu
- Aktualizace rozvrh
- Smazání rozvrhu

Po otestování funkcí výše byl emulátor odpojen od sítě a bylo otestováno odeslání dvou různých rozvrhů z mobilního zařízení. Během testování této funkce jsem narazil

na problém s používanou verzí Google Play služeb. Emulátor s verzí Android 5.1.1 neobsahoval aktuální verzi těchto služeb a tak bylo nutné snížit verzi v závislostech aplikace. Na reálném zařízení by ovšem tyto služby měly být aktuální.

8. Možnosti rozšíření

Vytvořená aplikace poskytuje základní funkcionalitu pro zobrazení rozvrhu. Aplikaci by bylo možné rozšířit například o následující funkce:

- Zobrazení notifikací o začátku předmětu
- Zobrazení vypsaných (zapsaných) zkoušek
- Nahrávání hlasových poznámek k jednotlivým předmětům
- Využití webových služeb ostatních univerzit využívajících systém IS STAG

Aplikaci by také bylo možné rozšířit o komplexnější komunikaci s mobilním zařízením. Například získání rozvrhu na požádání, či automatické aktualizování a sdílení rozvrhů mezi oběma zařízeními.

9. Závěr

V rámci práce jsem se seznámil s rostoucí platformou Android wearable, hlavně s vývojem aplikací na zařízení Android Wear. V první části práce jsem prozkoumal zařízení, na kterých je systém Android dostupný a jednotlivá zařízení popsal. V druhé části práce jsem prozkoumal možnosti tvorby aplikací na zařízení Android Wear. Tyto možnosti jsem popsal i s praktickými ukázkami kódů, které lze využít jako příklady při programování vlastních aplikací. Poslední část práce byla věnována vytvoření nové aplikace využitelné v prostředí ZČU a jejímu otestování.

Vytvořená aplikace nabízí studentovi možnost stáhnout svůj, či kamarádův rozvrh do hodinek Android Wear. Stažený rozvrh je pak možné prohlížet, aktualizovat, či smazat. Aplikace také obsahuje mobilní verzi, která umožňuje odeslání rozvrhu do hodinek, které například nemají možnost přístupu k internetu. Pro stažení rozvrhu jsou využívány webové služby IS STAG.

Aplikace byla také otestována sadou automatických testů a její funkcionalita byla testována průběžně během celého vývoje. Během práce jsem bohužel neměl možnost využití reálného zařízení, celý vývoj a testování tak probíhalo pouze na emulátoru dostupném v Android Studiu.

Platforma Android Wear je stále vyvíjena a rozšiřována a proto si myslím, že aplikace má potenciál k užívání větší základnou uživatelů a v případě jejího otestování na reálných zařízeních by mohla být skvělým pomocníkem pro studenty využívající hodinky Android Wear.

Seznam zkratek

WiFi - Technologie bezdrátového přenosu dat

GPS - Globální polohovací systém

RAM - Počítačová paměť umožňující čtení i zápis

PPI - Jednotka pro měření hustoty obrazových bodů (body na palec)

API - Rozhraní pro programování aplikací

OLED - Typ displeje využívající technologii elektroluminiscenčních diod

SDK - Sada softwarových nástrojů pro tvorbu aplikací pro specifickou platformu

XML - Značkovací jazyk

URL - Adresa pro jednoznačné určení zdroje

Literatura

[1] IDC (2015). *Smartphone OS Market Share, 2015 Q2* [online] [cit. 02.12.2015] Dostupné z http://www.idc.com/prodserv/smartphone-os-market-share.jsp,)

[2] Aligator. *Aligator S4000 Duo* [online] [cit. 11.12.2015] Dostupné z http://www.aligator.cz/produkty/aligator-s4000-duo/59

[3] Developer Android. *Build.VERSION_CODES* [online] [cit. 11.12.2015] Dostupné z http://developer.android.com/reference/android/os/Build.VERSION_CODES.html#GINGERBREAD_MR1

[4] Samsung. *Samsung Galaxy S6 Edge* [online] [cit. 11.12.2015] Dostupné z http://www.samsung.com/global/galaxy/galaxys6/galaxy-s6-edge

[5] Samsung. *Samsung Galaxy Tab 7.0 plus* [online] [cit. 11.12.2015] Dostupné z http://www.samsung.com/global/microsite/galaxytab/7.0/spec.html?type=find

[6] Samsung. *Samsung Galaxy Tab S (10.5, LTE)* [online] [cit. 11.12.2015] Dostupné z http://www.samsung.com/cz/consumer/mobile-devices/tablets/galaxy-tab-s/SM-T805NTSAXEZ

[7] Google. *Nexus Player* [online] [cit. 11.12.2015] Dostupné z https://www.google.com/nexus/player

[8] Android Central. *Android Auto* [online] [cit. 2.12.2015] Dostupné z http://www.androidcentral.com/android-auto

[9] Hyundai. *2016 Sonata Specs & Trim* [online] [cit. 2.12.2015] Dostupné z https://www.hyundaiusa.com/sonata/specifications.aspx

[10] Ion, Florence (2015). 11 things you need to know about Android Auto. *Greenbot*. 30.4.2015 [online] [cit. 2.12.2015] Dostupné z http://www.greenbot.com/article/2914832/11-things-you-need-to-know-about-android-auto.html

[11] Pioneer. *AVH-X8700BT* [online] [cit. 11.12.2015] Dostupné z http://www.pioneer-car.eu/eur/products/avh-x8700bt

[12] Ouya (2012). OUYA: A New Kind of Video Game Console. *Kickstarter*. 12.7.2012 [online] [cit. 11.12.2015] Dostupné z https://www.kickstarter.com/projects/ouya/ouya-a-new-kind-of-video-game-console

[13] Ouya. *Ouya* [online] [cit. 11.12.2015] Dostupné z https://www.ouya.tv

[14] Jawbone. *UP2 by Jawbone* [online] [cit. 11.12.2015] Dostupné z https://jawbone.com/store/buy/up2

[15] Spree Wearables. *Spree Smartcap* [online] [cit. 11.12.2015] Dostupné z http://spreewearables.com/products/smartcap

[16] LACKO, Ľuboslav. *Vývoj aplikací pro Android*. 1.vyd. Brno: Computer Press, 2015, 472 s. ISBN 978-80-251-4347-6

[17] IS/STAG. *Webové služby nad IS/STAG* [online] [cit. 11.12.2015] Dostupné z https://stag-demo.zcu.cz/ws/web

[18] Developer Android. *Receive Voice input in a Notification* [online] [cit. 1.06.2016] Dostupné z https://developer.android.com/training/wearables/notifications/voice-input.html

[19] Developer Android. Preview API Overview [online] [cit. 20.05.2016] Dostupné z https://developer.android.com/wear/preview/api-overview.html#stand-alone

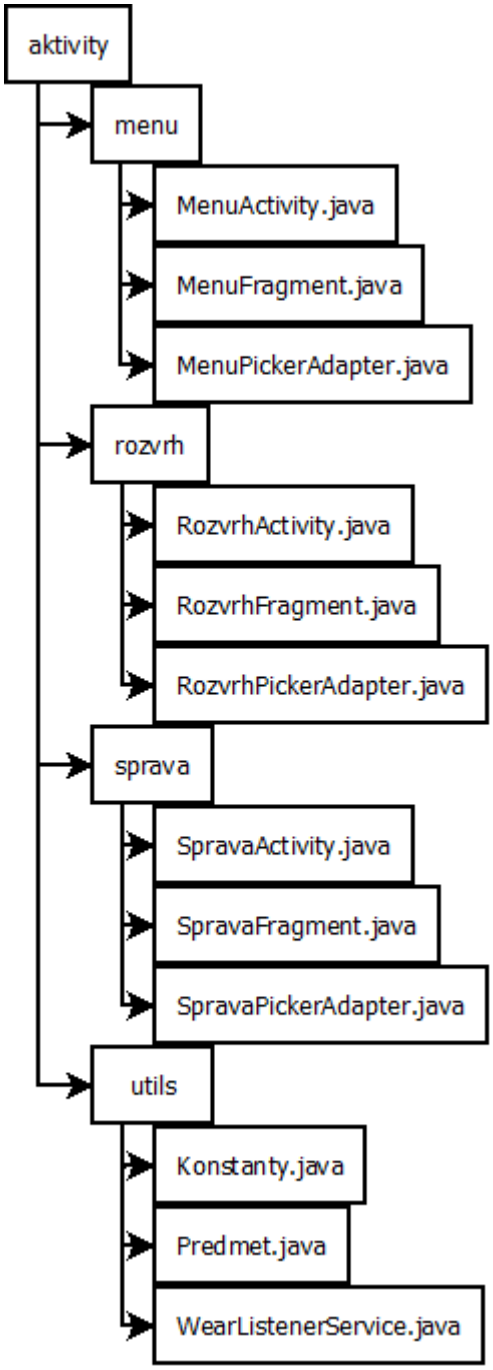
[20] Developer Android. *SharedPreferences* [online] [cit. 13.04.2016] Dostupné z https://developer.android.com/reference/android/content/SharedPreferences.html

[21] Espresso. [online] [cit. 1.06.2016] Dostupné z https://google.github.io/android-testing-support-library/docs/espresso/index.html

[22] Developer Android. *Creating and Running a Wearable App* [online] [cit. 10.05.2016] Dostupné z https://developer.android.com/training/wearables/apps/creating.html

Přílohy

A – Struktura modulu *wear*



B – Uživatelská příručka

Po prvním spuštění aplikace se zobrazí výchozí obrazovka s textovým polem pro zadání rozvrhu. Pokud je aplikace spuštěna na emulátoru, stačí do textového pole zadat osobní číslo a stisknout tlačítko „Stáhni“ (obr. 1).

Obr. 1 Úvodní obrazovka

Pokud je zařízení (emulátor) připojen k internetu a osobní číslo je platné (osobní číslo musí být dlouhé devět znaků a musí pro něj existovat rozvrh), rozvrh se stáhne a zobrazí se rozvrh aktuálního dne pro aktuální týden (obr. 2). Pokud by aktuálním dnem byla sobota nebo neděle, je zobrazen rozvrh pro příští týden. V případě, že emulátor nemá připojení k internetu, je možné do aplikace odeslat rozvrh z mobilního zařízení (kap. **Mobilní verze aplikace**). Je-li aplikace spuštěna s již staženým a nastaveným rozvrhem, je tato obrazovka přeskočena a je zobrazen rovnou rozvrh.

Obr. 2 Rozvrh aktuálního dne

V rozvrhu je možné se pohybovat všemi směry:

- Nahoru / dolů
  Pohyb mezi dny týdne
- Vpravo
  Předměty aktuálního dne

- Vlevo
  Přesun zpět do hlavního menu

Pokud uživatel přechází z předmětu na jiný den (gesto nahoru či dolů), přesune se na začátek řádku a je tedy zobrazen den a počet předmětů v tomto dnu.

Po opuštění rozvrhu je zobrazeno hlavní menu. V tomto menu je možné otevřít znovu hlavní rozvrh nebo otevřít správu rozvrhů.

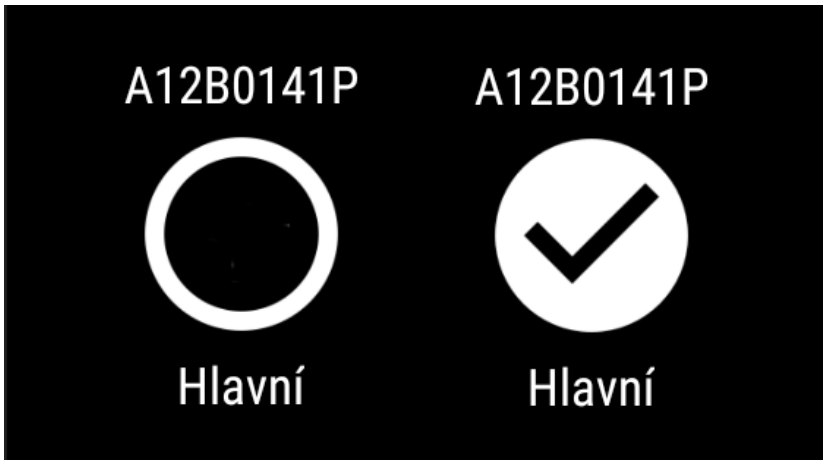
Správa rozvrhu

Po otevření správy je jako první zobrazeno tlačítko pro přidání nového rozvrhu. Po stisknutí tlačítka je otevřena obrazovka shodná s obrazovkou na obrázku 1. Po úspěšném přidání je nově stažený rozvrh zobrazen a je nastaven jako hlavní rozvrh.

Pohybem směrem dolů je pak možné procházet jednotlivé rozvrhy. U každého rozvrhu je možné přesunem vpravo vybrat akci k tomuto rozvrhu (obr. 3).

Obr. 3 Seznam dostupných akcí

První možností je rozvrh zobrazit. Po stisknutí je rozvrh zobrazen a lze ho prohlížet. Druhou akcí je nastavení rozvrhu jako hlavního. Hlavní rozvrh je zobrazen po stisknutí tlačítka „Rozvrh“ z hlavního menu aplikace a je zobrazen po spuštění aplikace. Na obrázku (obr. 4) je možné vidět stavy tohoto tlačítka.



Obr. 4 Rozvrh nenastavený jako hlavní (vlevo) a rozvrh nastavený jako hlavní (vpravo)

Třetí akcí je možnost rozvrh aktualizovat. Po stisknutí se spustí stažení a uložení rozvrhu. Na úspěšné stažení je uživatel upozorněn zobrazením textu „OK“. Pokud by nebylo dostupné internetové připojení, zobrazí se chybová hláška. Poslední dostupnou akcí je smazání rozvrhu. Po stisknutí je rozvrh smazán a správa zobrazena znovu s aktualizovaným seznamem.

Mobilní verze aplikace

Mobilní verze aplikace umožňuje stáhnout rozvrh na mobilním zařízení a odeslat ho do hodinek. Aplikace obsahuje jedinou obrazovku (obr. 5), která umožňuje zadání rozvrhu a jeho stažení a odeslání. Po kontrole osobního čísla (osobní číslo musí být dlouhé devět znaků a musí pro něj existovat rozvrh) je číslo odesláno.

Obr. 5 Obrazovka mobilní aplikace

Pokud je rozvrh úspěšně doručen, aplikace na hodinách se restartuje a zobrazí se rozvrh přijatý od mobilního zařízení.