VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SPRÁVA ČASU - ANDROID APP S FILOSOFIÍ FIRST-THINGS-FIRST (S. COVEY)

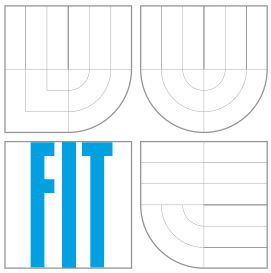
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN SLADEČEK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SPRÁVA ČASU - ANDROID APP S FILOSOFIÍ FIRST-THINGS-FIRST (S. COVEY)

TIME MANAGEMENT - ANDROID APP WITH THE FIRST-THINGS-FIRST (S. COVEY)

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN SLADEČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. IGOR SZŐKE, Ph.D.

BRNO 2015

Abstrakt

Cílem práce je návrh a implementace mobilní aplikace pro správu času pro Android. Vychází z filosofie First Things First od Stephena Coveye. Snahou je navrhnout aplikaci, která pokryje co možná největší počet zařízení jako jsou mobilní telefony, tablety či hodinky. Součástí práce je i monitorování chování uživatelů v aplikaci, na základě čehož dochází k úpravám uživatelského rozhraní, opravám chyb a přidávání nových funkcí.

Abstract

The aim of this thesis is the design and implementation of mobile application for time management on Android. It is based on First Things First philosophy by Stephen Covey. The main effort is to design application for various devices like mobile phones, tablets and smart watches. Modification of the design, bugfixes and new functions are based on user's monitoring, which is part of this thesis.

Klíčová slova

Správa času, First Things First, Android, kalendář, todo list, Google Calendar API, Android Wear, Content provider, uživatelské testování

Keywords

Time management, First Things First, Android, calendar, todo list, Google Calendar API, Android Wear, Content provider, user testing

Citace

Martin Sladeček: Správa času - Android app s filosofií First-Things-First (S. Covey), diplomová práce, Brno, FIT VUT v Brně, 2015

Správa času - Android app s filosofií First-Things-First (S. Covey)

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Igora Szőkeho, Ph.D. Uvedl jsem veškeré literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Sladeček
25. května 2015

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu práce za rady a podněty při vytváření práce. V neposlední řadě pak své rodině za skvělé podmínky k práci a shovívavost během celého studia.

Obsah

Kapitola 1

Úvod

Správa času je proces plánování a řízení všech činností s cílem zvýšit naši efektivitu a produktivitu. Většina knih a filosofií na toto téma se zaměřuje na co možná nejúsporněji využitý čas, což nemusí být vždy nejlepším řešením. Filosofie First-Things-First, ze které práce čerpá, bere navíc ohledy na základní hodnoty lidské existence. Efektivní se v tomto podání rovná kvalitní.

Cílem projektu je vytvořit aplikaci na správu času pro platformu Android. Aplikace implementuje vlastnosti 4. generace plánovacích systémů a principy z filosofie First-Things-First (více v kapitole 2). Cílová platforma podporuje zařízení nejrůznějšího druhu, jako jsou mobilní telefony, tablety či hodinky. Snahou je navrhnout aplikaci, která pokryje co možná největší počet těchto zařízení a maximalizuje potenciál každého z nich. Součástí práce je i monitorování chování uživatelů v aplikaci, na základě čehož dochází k úpravám uživatelského rozhraní, opravám chyb a přidávání nových funkcí. Cílovou skupinou jsou náročnější uživatelé, kterým nestačí základní kalendář a rádi využijí pokročilý nástroj ke správě svého času.

Následující kapitola se zabývá správou času. Shrnuje poznatky získané z knihy *7 návyků skutečně efektivních lidí* důležité pro tuto práci. Zaměřuje se na filosofii First-Things-First, která se zabývá sebeřízením, problematikou plánování a řízením času. Kapitola 3 blíže popisuje specifikaci a cíle aplikace. Představuje konceptuální návrh aplikace, včetně návrhu uživatelského rozhraní. V kapitole 4 jsou nezbytné teoretické informace nutné k implementaci aplikace, přibližuje zvolenou platformu a její základní stavební prvky. Následně je v kapitole 5 nastíněn postup a techniky použité při vývoji. V kapitole 6 je popsán princip a metody testování, jak na úrovni kódu, tak uživatelského testování. V závěru jsou diskutovány možná rozšíření a úpravy.

Kapitola 2

Správa času

Stephen Covey, uznávaný odborník na vedení lidí a rodinnou problematiku, je autorem knihy *7 návyků skutečně efektivních lidí (7 Habits of Highly Effective People)* [2], ze které práce čerpá. V knize dokazuje, že primárním předpokladem úspěchu je osvojení a využívání základních principů – návyků, které nám umožňují, abychom jednali co možná nejefektivněji nejen v pracovním, ale i v osobním životě. Rozlišuje a popisuje 4 generace plánovacích systémů, přičemž každá přináší nové prvky, které obohacují předchozí generaci o další možnosti správy času.

- 1. generace plánovacího systému využívá poznámky a seznamy úkolů. Jednotlivé úkoly nemají priority a nezabývají se tím, jak přispívat k naplnění životních hodnot a cílů. Reaguje pouze na to, co se musí udělat. Zástupcem je například aplikace GTasks (více v podkapitole 2.1).
- 2. generace je generací plánovacích kalendářů a diářů, které mají za cíl mít přehled o budoucích událostech, činnostech a závazcích. Ani zde nejsou stanoveny priority a činnosti se zabývají převážně tím, co se musí v daný den udělat. Rozdíl oproti první generaci je v přesně stanovených časech. Mezi zástupce lze zařadit mj. vestavěný kalendář v systému Android.
- 3. generace vychází z předchozí a doplňuje ji o možnost stanovovat priority k jednotlivým událostem. Zaměřuje se na pravidelná denní plánování s maximální mírou využití času. Často je stanoven příliš ambiciózní, nerealistický plán, což může vést ke stresu a pocitu nestíhání. Zástupcem je kalendář aCalendar (více v podkapitole 2.1).
- 4. generace, podle knihy [2] ideální systém plánování, se spíše než na maximální využití času soustředí na zkvalitňování vztahů a dosahování výsledků. Zástupcem je mj. aplikace MyEffectiveness (více v podkapitole 2.1).

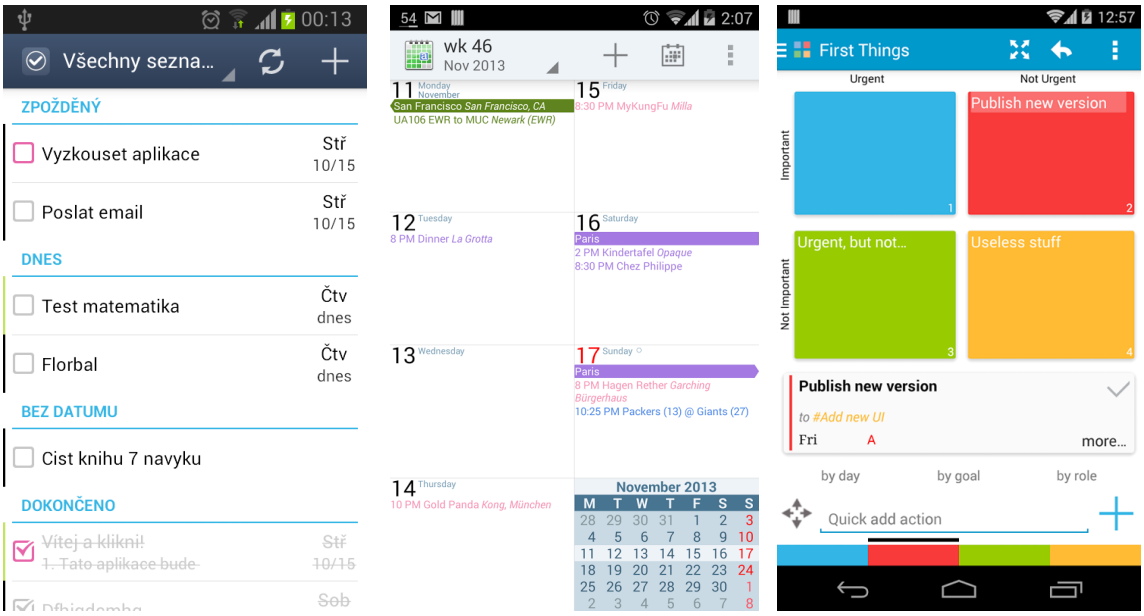
Následující podkapitola stručně popisuje vybrané konkurenční aplikace, které jsou inspirativní pro tuto práci. Podkapitola 2.2 shrnuje poznatky z knihy [2], přičemž se zaměřuje na pasáže podstatné pro tuto práci, zejména filosofii First-Things-First. Jelikož se jedná o výtah, může dojít ke zjednodušení myšlenek.

2.1 Analýza konkurenčních řešení

Před návrhem a vývojem aplikace byla analyzována konkurenční řešení. Zdrojem byl elektronický obchod Google Play[1]. Počet aplikací, které mají za cíl zvýšit produktivitu je celá řada. Jak bylo zmíněno dříve, existuje několik generací plánovacích systémů, pro něž byl vybrán jeden zástupce. Při výběru jsem vycházel z hodnocení ostatními uživateli, celkového počtu stažení i osobních zkušeností získané po čase užívání.

GTasks

Aplikace GTasks[2] (testována verze 2.1.80, uživatelské hodnocení 4,4) je jednoduchý správce úkolů a tedy spadá do 1. generace plánovacích systémů. V aplikaci lze vytvářet úkoly, organizovat je do seznamů, nastavovat upomínky a opakování úkolů. Všechno lze synchronizovat pomocí Google účtu s Google Tasks. Vzhled aplikace je velmi jednoduchý a plní svůj účel. Jak lze vidět na obrázku 2.1a, jedná se o seznam úkolů řazený podle data. Pomocí posunu do boku se přechází mezi jednotlivými seznamy (lokálními i vzdálenými). Na obrázku 2.2a je obrazovka pro přidání nového úkolu, velmi jednoduchá, umožňuje zadat pouze text a datum splnění úkolu.



(a) GTasks (b) aCalendar (c) MyEffectiveness

Obrázek 2.1: Základní obrazovka konkurenční aplikace

aCalendar

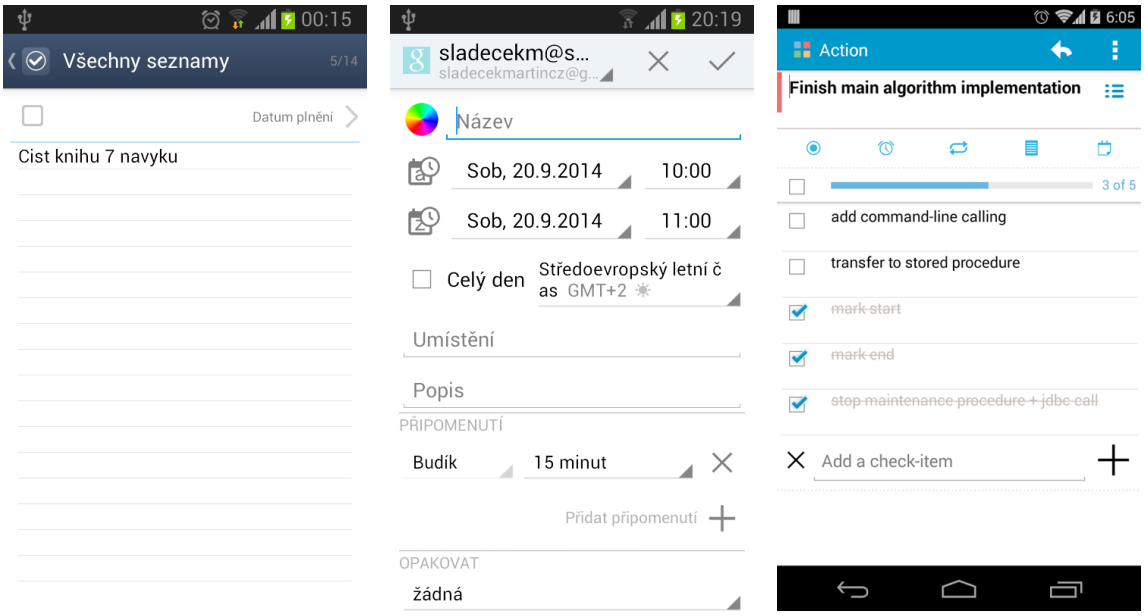
Aplikace aCalendar[3] (testována verze 1.0.2, uživatelské hodnocení 4,4) je propracovaný kalendář. Řadí se do 3. generace plánovacích systémů, jelikož umožňuje nastavovat priority. V aplikaci lze přidávat a editovat události a to jak v lokálních, tak vzdálených kalendářích

[1] https://play.google.com/
[2] https://play.google.com/store/apps/details?id=org.dayup.gtask
[3] https://play.google.com/store/apps/details?id=org.withouthat.acalendar

(propojených se systémem Android). Obrazovka pro přidávání nové události (obrázek 2.2b) poskytuje množství nastavení, včetně opakování událostí, přidávání účastníků a upomínek. Kalendář nabízí 3 pohledy - denní, týdenní (obrázek 2.1b) a měsíční. Ovládání probíhá pomocí gest. Pohybem stranou měníme pohledy, vertikálním pohybem se posouváme v čase. Aplikace umožňuje upravovat vzhled a nabízí několik widgetů. Dostupná je i placená verze aCalendar+[4], která umožní přidávat úkoly, včetně synchronizace s Google Tasks. Umožňuje další pokročilá nastavení včetně možnosti zasílat upomínky pomocí emailu či SMS.

MyEffectiveness

Aplikace MyEffectiveness[5] (testována verze 0.10.5, uživatelské hodnocení 4,3) je správce úkolů a řadí se do 4. generace plánovacích systémů. Je založen na filosofii First-Things-First, omezuje se však pouze na úkoly a neumožňuje práci s kalendářem. Úkolům lze nastavovat prioritu, upomínky, opakování či další dílčí úkoly (obrázek 2.2c). Umožňuje měnit barevné schéma aplikace a nabízí několik povedených widgetů. Ovládání aplikace je dle mého názoru neintuitivní a místy nepřehledné. Aplikace kombinuje 2 druhy ovládání – boční panel a menu v podobě ikon. Pro některé často používané akce je potřeba provést mnoho kliknutí. Snahou mé aplikace je minimalizovat počet kliknutí potřebných k nejpoužívanějším akcím. Podrobnější porovnání z pohledu ovládání a filosofie First-Things-First lze najít v kapitole 6. Nedostatky MyEffectiveness a absence obdobných aplikací vedla k nápadu a vývoji tohoto projektu.



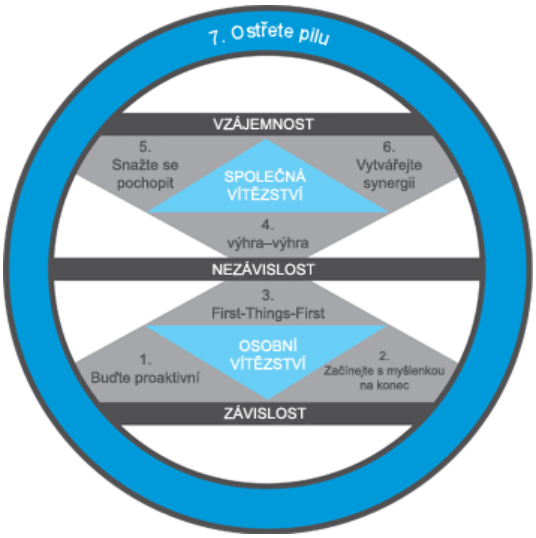
(a) GTasks (b) aCalendar (c) MyEffectiveness

Obrázek 2.2: Obrazovka přidávání záznamu u konkurenční aplikace

[4]https://play.google.com/store/apps/details?id=org.withouthat.acalendarplus
[5]https://play.google.com/store/apps/details?id=com.andtek.sevenhabits

2.2 7 návyků skutečně efektivních lidí

Kniha *7 návyků skutečně efektivních lidí (7 Habits of Highly Effective People)* [2] se zabývá osobním sebeřízením. Vychází z principu směřování *zevnitř ven*, které říká – chceme-li změnit chování lidí okolo nás, musíme nejdříve začít sami od sebe. Obsah knihy shrnuje obrázek 2.3. První část knihy se věnuje osobnímu vítězství a zabývá se převážně sebekontrolou. Návyky 1, 2 a 3 nás dovedou od závislosti k nezávislosti. Druhá část knihy (návyky 4, 5 a 6) popisuje společná vítězství – týmovou práci, spolupráci a komunikaci, tedy problémy spojené s charakterem a osobností druhých lidí. Poslední část popisuje návyk 7, který se zaměřuje na obnovu sil a nepřetržité zdokonalování.



Obrázek 2.3: Diagram posloupnosti a provázanosti 7 návyků. Převzato z knihy [2]

Návyk 1 – Buďte proaktivní

Podle knihy [2] je proaktivita nejdůležitější princip skutečně efektivního člověka. Být proaktivní znamená jednat iniciativně a nést odpovědnost za věci, které se stanou. Na rozdíl od reaktivních lidí, kteří jsou poháněni svými pocity a vnějšími vlivy, se proaktivní člověk podřizuje hodnotám, které si sám vybral a přijal za své.

Návyk 2 – Začínejte s myšlenkou na konec

Začít s myšlenkou na konec znamená mít jasnou představu cíle. Vychází z principu, že vše vzniká dvakrát. Nejdříve mentálně (návrh) a poté fyzicky (realizace). Nejefektivnější způsob, jak začít v praxi aplikovat tento princip je vypracovat si **osobní poslání**. Určitou filosofii, krédo zaměřující se na to, jakými chceme být, jaký chceme mít charakter. To nám umožní lépe pochopit, kde se v současné době nacházíme, a nasměrovat naše kroky správným směrem.

Osobní poslání by mělo být osobní, pozitivní a sepsané v přítomném čase. Musí obsahovat nejzákladnější a nejdůležitější dlouhodobé cíle, kterých chceme v životě dosáhnout. S posláním lze lépe pracovat, pokud se rozčlení na specifické oblasti, které kopírují naše životní role. K těmto rolím je třeba určit několik cílů. Efektivně stanovený cíl se zaměřuje

především na výsledky, nikoliv činnosti. Čas od času je dobré osobní poslání přepracovat, doplnit. Kromě osobního lze vytvářet i poslání rodinná či firemní.

Návyk 3 – First-Things-First

Návyk 3 je osvojením a praktickým naplněním návyků 1 a 2. Filosofie First-Things-First, neboli „to nejdůležitější dávejte na první místo“ se zabývá efektivním sebeřízením, problematikou plánování a řízením času. Vést znamená určovat, co je nejdůležitější. Řídit znamená zajistit, že nejdůležitější věci budou skutečně na prvním místě. Principům filosofie First-Things-First a popisu ideálního plánovacího systému 4. generace se věnuje podkapitola 2.3.

Návyk 4 – Myslete způsobem výhra–výhra

Návyk 4 je popsán v druhé části knihy a spadá do společných vítězství. Zaměřuje se na hledání prospěšných řešení. Přijaté dohody mají být prospěšné pro všechny zúčastněné. Jinými slovy, nemáme uvažovat stylem „buď bude po mém nebo po tvém“, ale snažit se najít vhodnou alternativu. Na příkladech v knize je ukázáno, že z dlouhodobého hlediska jiné způsoby jako prohra-výhra či výhra-prohra negativně ovlivní obě strany. Podstatou vyjednávání je oddělit lidi od problému a zaměřit se na zájmy, ne na postoje. Při jednání a snaze nalézt řešení výhra-výhra doporučuje podívat se nejprve z pohledu druhé strany a snažit se ji pochopit. Následně identifikovat zájmy a určit přijatelná řešení pro obě strany. Návyky 5 a 6 umožňují prakticky realizovat uvažování výhra-výhra.

Návyk 5 - Nejdříve se snažte pochopit, potom být pochopeni

Tento návyk se zabývá zlepšením komunikace s lidmi. K lepšímu porozumění mezi lidmi je dobré naslouchat empaticky – s cílem pochopit (jejich úhel pohledu, pocit), což vyžaduje ohleduplnost. Dříve než nastanou problémy, než začneme hodnotit, radit a předkládat vlastní myšlenky, můžeme se vcítit a upravit tak naši reakci podle potřeby. Návyk umožňuje rozvíjet mezilidské vztahy a budovat vzájemnou důvěru. Když lidem porozumíte, snáz naleznete prospěšná řešení – alternativu (výhra-výhra).

Návyk 6 - Vytvářejte synergii

Synergie je završením všech předchozích návyků. Vychází z předpokladu, že celek je více, než součet jeho částí. Základem jsou tvůrčí, otevřená jednání a komunikace, při které se snažíme dosáhnout řešení výhra-výhra. Spojováním různých názorů, myšlenek a konceptů více lidí dosáhneme několikanásobně lepšího výsledku, než v případě jednotlivých prvků, což je hlavní motiv týmové práce.

Návyk 7 - Ostřete pilu

Návyk ostřete pilu zastřešuje zbývající návyky. Zaměřuje se na obnovu sil a sebezdokonalování. Umožňuje obnovovat 4 dimenze lidské podstaty – fyzickou (tělesná cvičení, výživa, zvládání stresu), duchovní (ujasňování hodnot a závazků, studium a meditace), mentální (četba, představivost, plánování) a společenskou (vedení lidí, týmová spolupráce, synergie, vnitřní jistota). Obnova sil by měla být vyvážená a neměla by zanedbávat ani jednu z dimenzí. Tyto činnosti naplňují naše osobní poslání a díky nim zlepšujeme i ostatní návyky.

2.3 First-Things-First

Základním principem filosofie First-Things-First je **matice plánování** (obr. 2.4), která vychází z představy, že čas využíváme jedním ze čtyř způsobů. Bere v úvahu 2 faktory – naléhavost a důležitost. Naléhavé věci si vyžadují naši okamžitou pozornost. Často se jedná o nenáročné záležitosti a nečiní nám tak problém se jimi zabývat. Mohou však být nedůležité. Pokud je však něco důležité, pomáhá nám to realizovat naše poslání a cíle. V případě, že nejednáme s návykem 2, často upřednostňujeme naléhavé záležitosti na úkor důležitých.

Obrázek 2.4: Matice plánování času [2]. Čas využíváme jedním ze 4 způsobů. Snahou je věnovat se činnostem z kvadrantu II, které nám pomohou realizovat naše dlouhodobé cíle.

Kvadrant I obsahuje naléhavé a důležité záležitosti – hovoříme o krizích a problémech. Značná část lidí tráví většinu času řešením krizí z kvadrantu I a žije od jednoho problému k druhému. Výsledkem je stres a vyčerpání. Jedinou úlevou je plnění nedůležitých a nenaléhavých záležitostí z kvadrantu IV. Lidé trávící většinu času řešením úkolů z kvadrantu III a IV žijí v podstatě nezodpovědný způsob života. V případě, že se někdo věnuje pouze naléhavým, ale nedůležitým úkolům z kvadrantu III, uvažuje v krátkodobém horizontu a má pověst nestálého člověka. Základ efektivního osobního řízení je kvadrant II obsahující nenaléhavé, ale důležité věci (vztahy, příprava, učení, plánování atd.), které vyžadují proaktivitu a stanovení životních cílů.

Ideální **plánovací systém 4. generace** [2] by měl umožnit poznamenat si osobní poslání, své role, krátkodobé i dlouhodobé cíle. Neustále by vám měl připomínat, abyste nezanedbávali důležité věci (cvičení, rodina, profesní i osobní rozvoj), tedy záležitosti z kvadrantu II. Je zaměřen na týdenní plánování (namísto denních), aby se aktivity zasadili do širšího kontextu. V neposlední řadě musí být systém přizpůsoben vašim potřebám, přenosný a vždy po ruce.

Kapitola 3

Specifikace a návrh

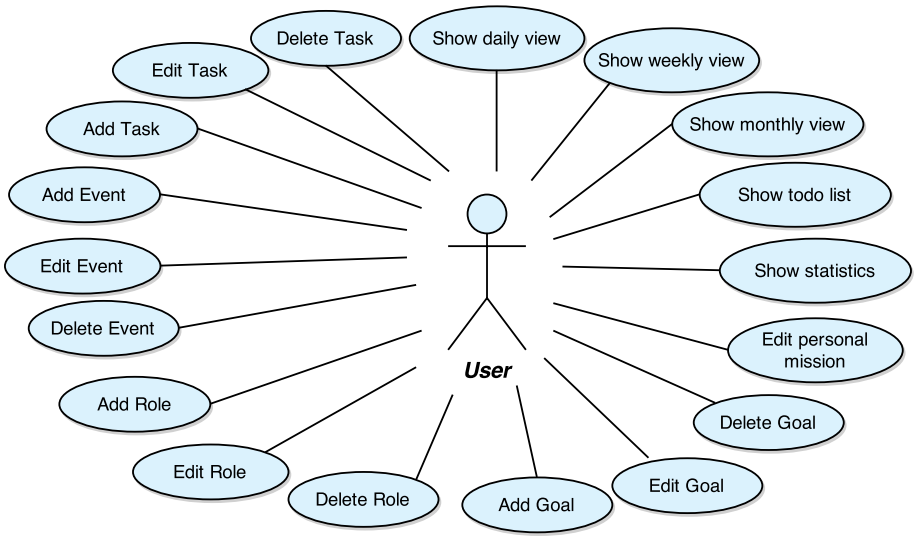
Kapitola popisuje specifikaci a návrh aplikace. Vychází z poznatků získané v předcházející kapitole. Teoretické poznatky se snaží převést do konkrétních požadavků na funkce v aplikaci. Při návrhu jsem se také inspiroval nejlepšími vlastnostmi konkurenčních aplikací popsaných v podkapitole 2.1. Následující podkapitola 3.1 má za cíl seznámit čtenáře s detailnějším popisem funkčnosti. Podkapitola 3.2 popisuje strukturu ukládaných dat a v podkapitole 3.3 je nastíněn návrh uživatelského rozhraní a koncept fungování celé aplikace.

3.1 Specifikace řešení

Jednou z požadovaných vlastností plánovacího systému je možnost sepsat si vlastní osobní poslání. K tomuto účelu aplikace poskytuje prostor pro vkládání delšího textu a jeho trvalé uložení, přičemž bude možné text v případě potřeby upravit či doplnit. Umožňuje vytvářet, editovat a mazat životní role, přičemž lze definovat krátkodobé i dlouhodobé cíle ke každé z nich. Tyto funkce jsou důležitou součástí filosofie First-Things-First a proto by měly být kdykoliv dostupné. Aby i neznalý uživatel dokázal využít veškerou funkčnost, bude zde stručná nápověda.

Aplikace bude plnit funkce kalendáře. Bude možné přidávat, upravovat a mazat jednotlivé události – činnosti v přesně vymezeném časovém období. Každá událost je propojena s kalendářem, který může být lokální či vzdálený (Google Calendar). U každé události lze přidat podrobnější popis či místo konání. Pro činnosti, kdy nevíme čas vykonání, lze využít úkoly. Úkol, na rozdíl od události, není zasazen přesně do časového intervalu, nicméně může být zadán nejzazší termín splnění. Zobrazení úkolů a událostí se bude vhodně kombinovat. Určení priority u úkolů se provádí výběrem jednoho ze čtyř kvadrantů v matici plánování času (obr. 2.4). Každý úkol může být propojeny s některým z uživatelských cílů. Při plánování bychom se měli zaměřit na týdenní plány. Z toho důvodu bude v kalendáři týdenní pohled. Týdenní pohled by měl zobrazovat události a úkoly naplánované na daný týden. Navíc bude přidán i denní a měsíční pohled, jak je běžné u většiny kalendářních aplikací a plánovacích systémů.

Aby mohl uživatel lépe plánovat svůj čas, bude aplikace poskytovat statistiky. Ty mu budou zobrazeny v přehledných grafech a jejich cílem je zobrazit uživateli podíl času stráveného v jednotlivých kvadrantech matice plánování. Taky bude poskytovat statistiku o počtu hotových úkolů, a to jak celkově, tak i za období v nejbližší době.

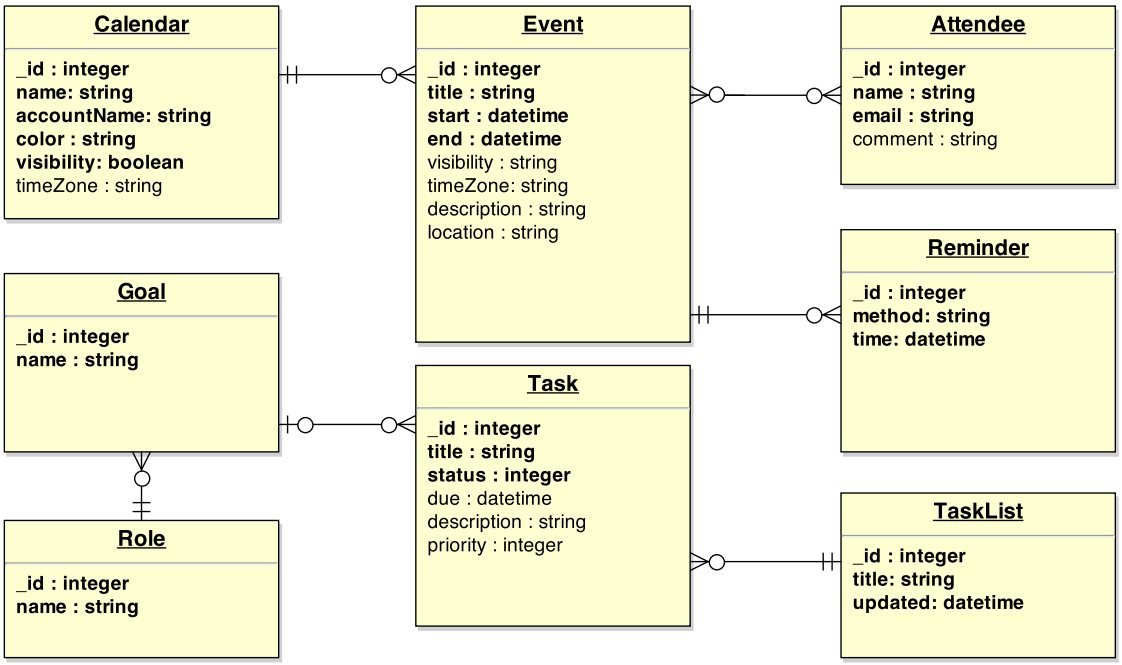


Obrázek 3.1: Diagram případů užití

3.2 Návrh datové struktury

Možnosti uživatele shrnuje diagram případů užití na obrázku 3.1. Mezi nejčastější akce patří přidávání, editace a mazání nejrůznějších dat. Z toho důvodu je třeba řešit ukládání a práci s daty. Ukládaná data mají mezi sebou vazby, a proto bude nejlepší způsob uložit data do relační databáze. Návrh lokální databáze zobrazuje ER diagram na obrázku 3.2 (povinné atributy jsou tučně zvýrazněny). Následuje stručná charakteristika jednotlivých ukládaných objektů.

- Uživatelské *role* a *cíle* jsou základem filosofie First-Things-First. Jedná se o jednoduché objekty (obsahují ID a název), které slouží k logickému uskupování prováděných úkolů. Každá role může mít libovolný počet krátkodobých a dlouhodobých cílů.
- *Kalendáře* seskupují uživatelské události. Mohou být lokální (uloženo pouze v daném zařízení) nebo vzdálené (synchronizované napříč zařízeními, např. Google Calendar). Kalendář má výchozí barvu a časové pásmo, které se přiřazuje událostí. Změnou atributu viditelnosti lze skrýt všechny události v kalendáři.
- *Události* jsou časově omezené (atributy start a end) akce. Obsahují název, popisek, místo konání aj. Každá událost je zařazena do jednoho z uživatelských *kalendářů*. K události lze přiřazovat i její *účastníky* a nastavovat libovolný počet *upozornění*.
- *Úkoly* nejsou časově omezené. Mohou však obsahovat datum, do kterého by měly být splněny. Mimo samotného nadpisu úkolu obsahují i popisek a priority podle matice plánování. Úkol může být zařazen do jednoho ze *seznamu úkolů*. Každý úkol může pak lze zařadit do jednoho z krátkodobých nebo dlouhodobých cílů.



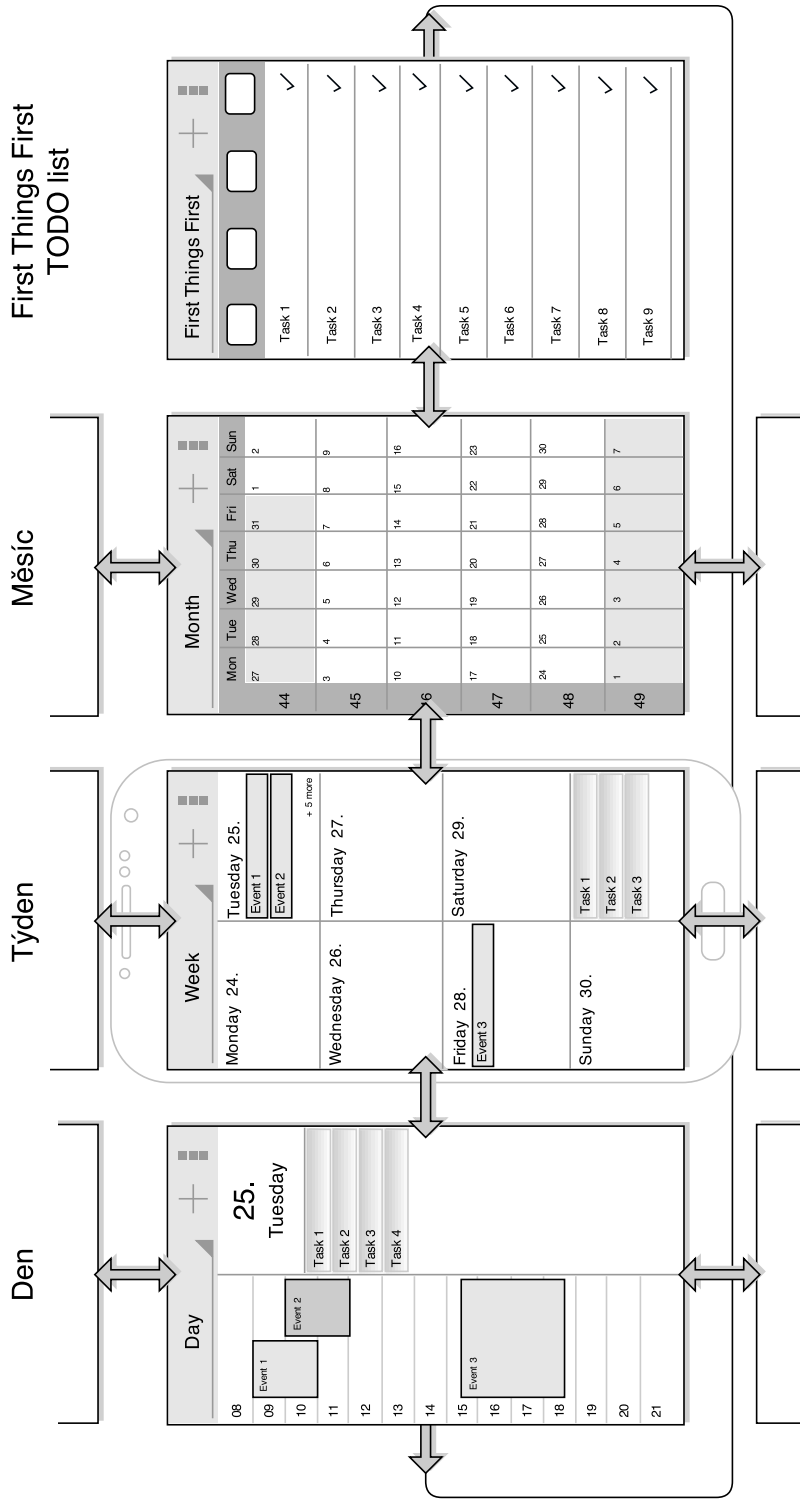
Obrázek 3.2: ER diagram lokální databáze (Crow's Foot notace)

3.3 Návrh vzhledu aplikace

Jak bylo zmíněno v úvodu, aplikace má za úkol podporovat nejrůznější zařízení – mobilní telefon, tablet či hodiny. Každé z těchto zařízení má jiné rozměry, rozlišení a způsob ovládání a uživatelské rozhraní musí být vhodně uzpůsobeno. Návrh uživatelského rozhraní vychází z doporučení na oficiálních stránkách[1]. Níže jsou popsány a zobrazeny konceptuální návrhy nejdůležitějších částí aplikace.

Koncept ovládání je zaměřen na maximální využití dotykových gest a je zobrazen na obrázku 3.3. Pohybem do stran se přepínají jednotlivé typy zobrazení (denní, týdenní a měsíční pohled + todo list) rotující v cyklické smyčce. Pohybem nahoru, respektive dolů dochází k posunu v čase (předcházející, respektive následující den, týden či měsíc). Na liště v horní části aplikace jsou umístěny nejdůležitější akce v aplikaci – přepínání mezi pohledy, přidání nové události či úkolu a uživatelské menu. Další akce, jako je správa kalendářů, rolí a cílů či editace osobního poslání je na dalších obrazovkách. Snahou je minimalizovat počet kliknutí nutných ke spuštění nejpoužívanějších akcí (více v kapitole 6, která se zabývá testováním efektivnosti uživatelského rozhraní).

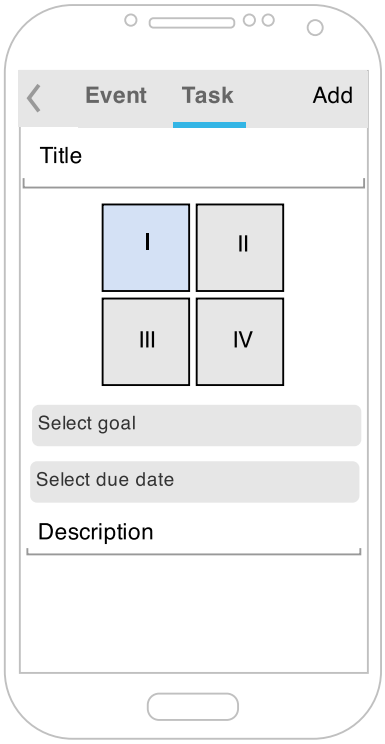
[1] https://developer.android.com/design/index.html



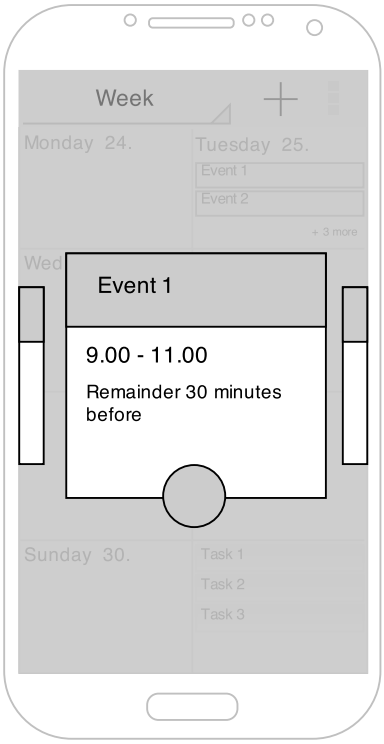
Obrázek 3.3: Návrh hlavní obrazovky aplikace. Pohybem do stran se přepínají jednotlivé typy zobrazení rotující v cyklické smyčce. Pohybem nahoru, respektive dolů dochází k posunu v čase.

Na obrázku 3.3 vlevo je zobrazen denní pohled. Je rozdělen na dvě logické části. První zobrazuje události s přesně vymezeným časem na časové ose, zatímco druhá část zobrazuje úkoly, které jsou na daný den naplánovány, ale nemají přesně stanovený čas. Pořadí těchto úkolů respektuje priority podle matice plánování času (viz 2.3). Vzhled na tabletu je stejný, samozřejmě v adekvátním zvětšení. Při otočení zařízení naležato (landscape) se vytvoří 3 sloupce. Dojde k rozdělení časové osy na 2 části, čímž lépe využijeme prostor. Uprostřed obrázku 3.3 je týdenní pohled. Pro každý den jsou zobrazeny události a naplánované úkoly. Dny jsou zobrazeny v matici 4x2 (při otočení zařízení naležato 2x4). V pravé dolní části jsou zobrazeny týdenní úkoly. Vpravo na obrázku 3.3 je zobrazen měsíční pohled. Jedná se o tradiční zobrazení známé z papírových kalendářů (matice 6x7 dnů), které umožňuje zobrazovat události pro celý měsíc. Tento pohled je stejný i ve verzi pro tablet.

Obrazovka pro přidávání úkolu je zobrazena na návrhu 3.4. Na liště v horní části je možnost přepínat mezi obrazovkou pro přidávání úkolu nebo události. Při vytváření nového úkolu jsou zde formulářová políčka pro nadpis, poznámku a volitelně lze zadat i nejzazší den splnění úkolu. Priorita se zadává pomocí výběru jednoho z kvadrantů matice plánování času. Pro vytváření nové události jsou zde políčka pro výběr času – počátečního a koncového. V případě potřeby lze vytvořit celodenní událost. Událost lze zařadit do jednoho z kalendářů. Pokud žádný neexistuje, vytvoří se nový lokální kalendář.



Obrázek 3.4: Návrh obrazovky pro přidávání úkolu

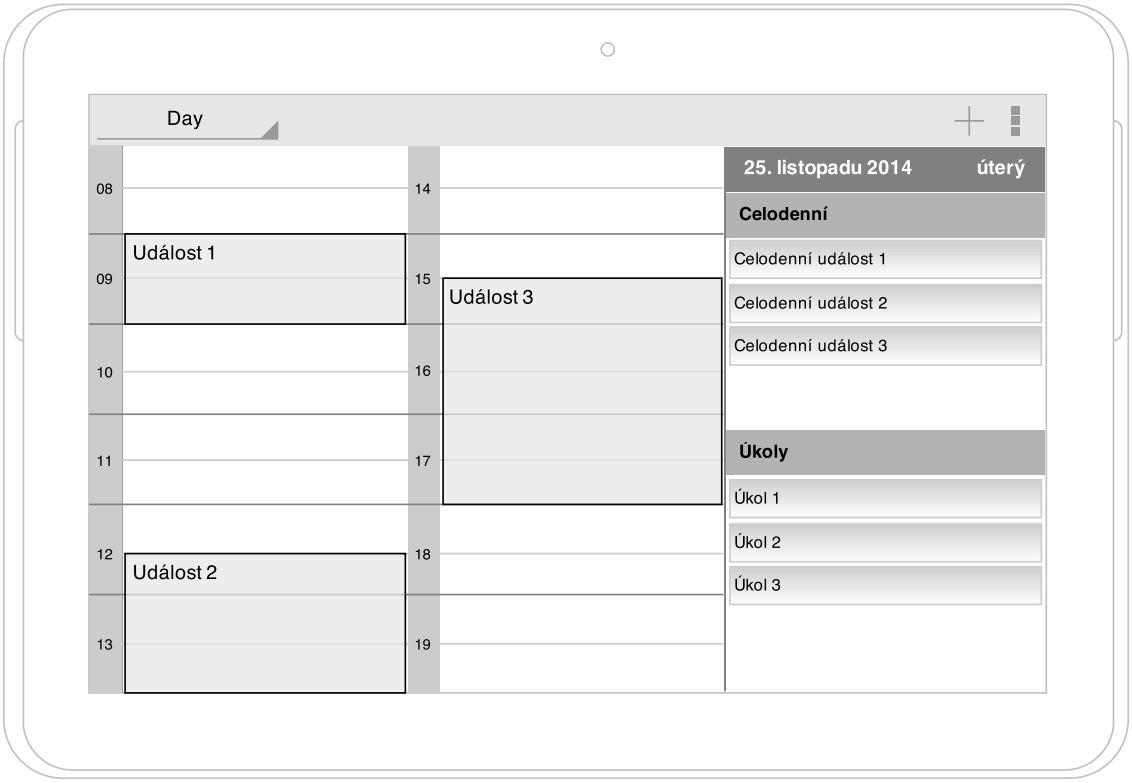


Obrázek 3.5: Návrh obrazovky pro zobrazení detailu

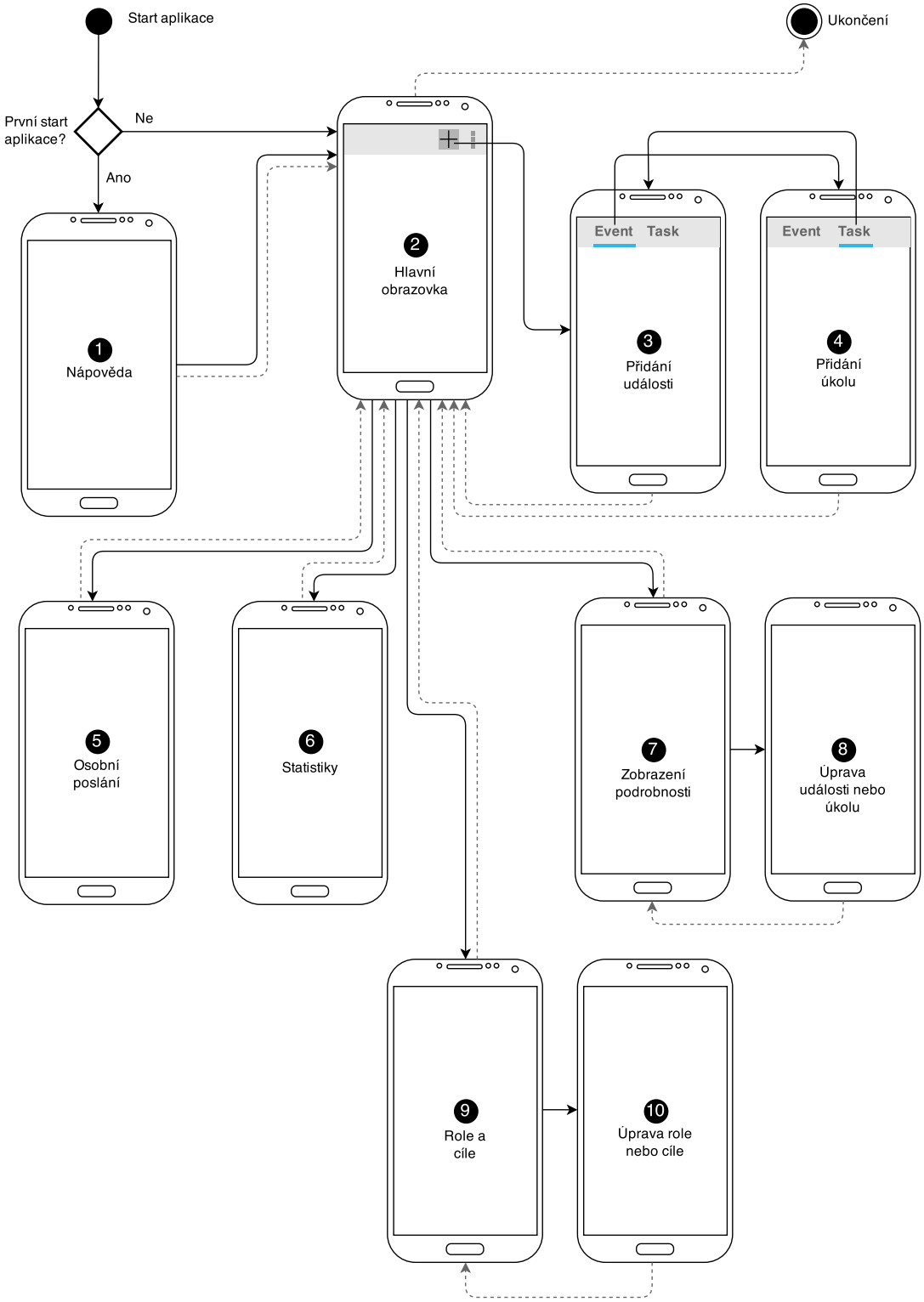
Zobrazení detailu události či úkolu je vidět na obrázku 3.5. Menší okno překryje celou obrazovku a zobrazí podrobnější informace k události či úkolu – čas, upomínky, účastníky aj. Umožňuje události a úkoly editovat a pohybem do stran přecházet na další položky. Aby uživatel mohl snáze pochopit tento způsob ovládání, budou na okrajích viditelné části dalších položek.

Funkčnost aplikace pro hodinky je značně omezena a uzpůsobena velikosti těchto zařízení. Jak lze vidět na obrázku 3.6, umožňuje zobrazit úkoly a události naplánované na daný den a to v pořadí podle priorit. Mezi úkoly a událostmi lze listovat a povést akci – úkol lze nastavit jako splněný, událost lze smazat.

Obrázek 3.6: Zobrazení událostí na chytrých hodinkách



Obrázek 3.7: Návrh obrazovky denního zobrazení pro tablety



Obrázek 3.8: Zobrazení funkčnosti celé aplikace (application flow). Plné čáry zobrazují přechod mezi obrazovkami při určité akci, čerchované pak vracení se v hierarchii pomocí akce zpět.

Na obrázku 3.8 je zobrazeno celkové fungování aplikace. V případě prvního spuštění dojde ke spuštění nápovědy (č. 1), v opačném případě ke spuštění hlavní obrazovky (č. 2). Ta zobrazuje různé pohledy kalendáře v cyklické smyčce (obrázek 3.3). Z hlavního menu lze spustit novou obrazovku pro přidávání nového záznamu. Pomocí přepínače lze měnit, zda se přidává nová událost (č. 3) nebo nový úkol (č. 4). Obrazovka se uzavře při přidání nového záznamu nebo při stisknutí tlačítka zpět. V případě, že dojde ke kliknutí na událost nebo úkol v kalendáři, otevře se poloprůhledné okno s detailem a možností přecházet na další položky (č. 7). Odtud lze událost upravit nebo smazat. Z hlavní obrazovky lze zobrazit obrazovku s editací osobního poslání (č. 5), statistikami (č. 6) nebo správou rolí a cílů (č. 9). Obrazovky lze spouštět pomocí ikonek nebo menu v hlavním panelu.

Kapitola 4

Mobilní aplikace

Následující text se zabývá popisem vývoje mobilní aplikace pro systém Android. Vychází z požadavků a návrhu v kapitole 3 a pro jejich naplnění hledá vhodná řešení. Informace čerpá z oficiální dokumentace [1] a z knih [3] a [6].

V první podkapitole je obecně popsán operační systém Android. Jsou zde stručně charakterizovány vlastnosti systému i nástroje nutné k vývoji. Kapitola 4.2 popisuje základní prvky každé aplikace a kapitola 4.3 práci s uživatelským rozhraním. Jak bylo zmíněno v kapitole 3, je nutné trvale ukládat data, proto se kapitola 4.4 zabývá možnostmi trvalého ukládání dat a zaměřuje se na relační databázi. V závěru je popsána odlehčená verze systému pro chytré hodinky Android Wear.

4.1 Platforma Android

Android je open source mobilní operační systém založený na Linuxovém jádře [6]. Vývoj zastřešuje konsorcium Open Handset Alliance v čele se společností Google. Android je určen pro dotyková zařízení, především mobilní telefony a tablety. Existují i upravené edice se speciálním uživatelským rozhraním pro hodinky a náramky (Android Wear), televize (Android TV) a auta (Android Auto). Primárním vývojovým jazykem je Java.

V době psaní této práce vychází nová verze systému Android 5.0 Lollipop (API 21), která byla zvolena cílovou platformou. Nejrozšířenější[1] je Android 4.4 KitKat a Android Jelly Bean (4.1.x, 4.2.x a 4.3). Jako minimální verze proto bylo zvoleno API 14. Díky tomu aplikace pokryje 92% zařízení se systémem Android. Aby bylo možné vyvíjet i pro starší verze a využívat nové vlastnosti systému, poskytuje Android tzv. Support Library. Přilinkování této sady knihoven k aplikaci umožní využívat nové funkce.

Architektura operačního systému Android je rozdělena do vrstev [3]. Nejnižší vrstvou je jádro operačního systému (Linux) tvořící abstraktní vrstvu mezi hardwarem a softwarem. Stará se např. o správu paměti, sítí, procesů a ovladačů. Nad jádrem operačního systému jsou knihovny a middleware napsané v C/C++ jako jsou libc, Media Framework, SQLite, OpenSSL, WebKit, OpenGL ES aj. Vrstvu nad knihovnami tvoří Application framework. Poskytuje přístup k funkcím a službám, které lze v aplikaci využít. Mezi prvky patří např. Activity Manager, Resource manager, Package Manager, Fragment manager, Notification Manager, View System aj. Důležitou součástí platformy je virtuální stroj Dalvik – JIT (Just-in-time) compiler, který je primárním běhovým prostředím do verze 5.0. Od této

[1] http://developer.android.com/about/dashboards/index.html

verze jej nahradil ART (Android Runtime), který na rozdíl od svého předchůdce provádí AOT (ahead of time) kompilaci a tím dosahuje značného urychlení.

Vše potřebné pro vývoj aplikací je součástí Android SDK (Software Development Kit)[2], obsahuje knihovny, příklady i nástroje zajišťující překlad, profilování a testování. Součástí je i emulátor, který umožňuje spouštět aplikace ve virtuálním stroji s nastavitelnou verzí systému a dalšími parametry (velikost paměti, rozlišení obrazovky atd.). V době psaní této práce bylo uvolněno nové oficiální vývojové prostředí Android Studio[3], založeno na IntelliJ IDEA[4], které všechny potřebné nástroje integruje. Využívá překladové prostředí Gradle (nahradilo dříve používaný Ant) pro řízení překladu, testování, spouštění i vytváření finálního `.apk` balíčku.

Struktura Android projektu vychází z požadavků překladového prostředí Gradle. Člení se na moduly – logicky uskupené celky. Každý modul má přesně stanovenou adresářovou strukturu. Obsahuje mimo jiné složku `libs` pro knihovny třetích stran, složku `build` ve které lze nalézt vygenerovaný kód a složku `src` obsahující veškerý obsah programu. Nachází se zde konfigurační soubor `AndroidManifest.xml`. V tomto souboru je zachyceno základní nastavení aplikace – obsahuje název, jmenný prostor, ale také používané komponenty (viz podkapitola 4.2) a práva určující k jakým zdrojům může aplikace přistupovat. V podsložce `java` jsou zdrojové kódy programu a podsložka `res` obsahuje prostředky (anglicky resources) jako jsou obrázky, layout, barvy, textové řetězce aj. Jsou odděleny od samotného kódu aplikace a podle typu se umísťují do různých podadresářů. V názvech těchto podadresářů se používají tzv. kvalifikátory, díky kterým lze využívat různé zdroje v závislosti na konfiguraci zařízení (např. obsah složky `values-cs` se použije v případě, že zařízení je nastaveno na češtinu, v opačném případě se použije obsah složka `values`). Platforma sama zajišťuje vyhledání nejlépe vyhovujících zdrojů. Při překladu je vygenerována třída `R.java`, která obsahuje odkazy na tyto zdroje. Pomocí ní lze k zdrojům přistupovat z kódu nebo se na ně odkazovat v layoutech.

4.2 Součásti aplikace

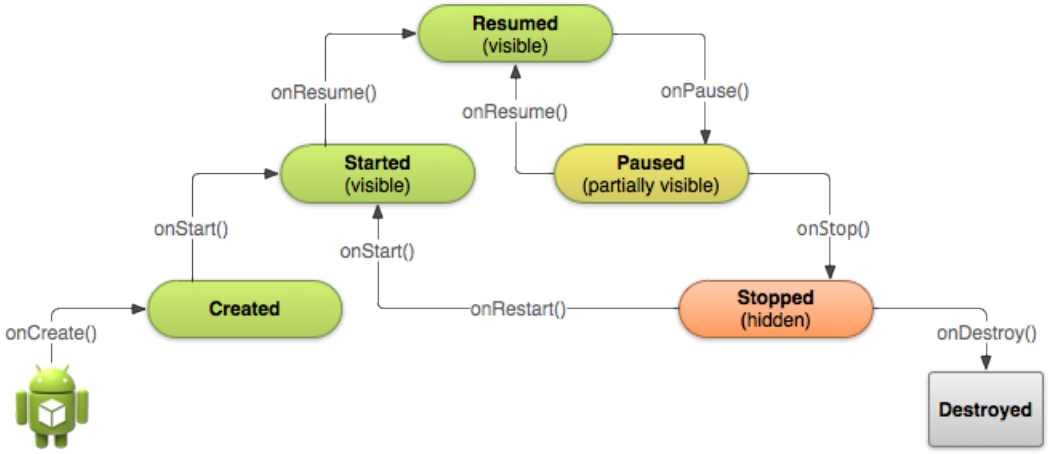
Podle oficiálních stránek[5] existují 4 základní komponenty – `Activity`, `Service`, `Broadcast receiver` a `Content provider`. Mezi další důležité součásti většiny aplikací dále patří `Fragment`, `Intent`, `AsyncTask` či `Loader`. Každá komponenta má svou specifickou funkci a životní cyklus (předpis jak komponenta vzniká a zaniká). Níže jsou popsány nejdůležitější vlastnosti každé z vybraných komponent.

Activity

Komponenta `Aktivity`[6] reprezentuje obrazovku s uživatelským rozhraním. Jedna z aktivit je nastavena jako výchozí a ta se spustí po startu programu. Celá aplikace se obvykle skládá z více různých aktivit, které jsou mezi sebou propojeny. Spuštění jiné aktivity se provádí pomocí metod `startActivity()` a `startActivityForResult()`. Při spuštění nové aktivity dojde k ukončení původní, která je vložena na vrchol zásobníku, což umožňuje rychlé obnovení při akci zpět.

[2] http://developer.android.com/sdk/index.html
[3] http://developer.android.com/tools/studio/index.html
[4] http://www.jetbrains.com/idea/
[5] http://developer.android.com/guide/components/fundamentals.html
[6] http://developer.android.com/guide/components/activities.html

Životní cyklus je znázorněn na obrázku 4.1. Při spuštění aktivity dojde k volání callback funkcí `onCreate()`, `onStart()` a `onResume()` (inicializace uživatelského rozhraní). V případě překrytí části aktivity je volán callback `onPause()`. Při ukončení aktivity (metodou `finish()`) dojde k zavolání metod `onStop()` a `onDestroy()`.



Obrázek 4.1: Životní cyklus aktivity

Service

`Service`[7] je komponenta, která běží na pozadí a vykonává dlouhotrvající operace (komunikaci se serverem, práce s databází aj.). Nemá žádné uživatelské rozhraní. Služba je nezávislá na aplikaci, která ji spustila a může běžet i po jejím ukončení.

Existují 2 typy služeb – spouštěné a připojované. Spouštěné služby, které aktivujeme pomocí `startService()`, se po ukončení operace sami ukončí. K připojované službě se aktivity napojí pomocí `bindService()` a mohou čerpat její data. Po odpojení všech aktivit dojde k ukončení služby. Životní cyklus služby se liší, zda se jedná o spouštěnou či připojovanou. V případě spouštěné dojde k zavolání callback funkce `onStartCommand()`. Pokud implementujeme tuto metodu, jsme zodpovědní za řádné ukončení služby (příkazy `stopSelf()` nebo `stopService()`). Při napojování aktivity na připojovanou služby se volá `onBind()`. Oba typy služeb volají metody `onCreate()` (inicializace) a `onDestroy()` (řádné ukončení).

Broadcast receiver

`Broadcast receiver`[8] komponenta zajišťuje příjem systémových či jiných událostí (stav baterie, odpojení od sítě atd.) a zajišťuje adekvátní odpověď. Komponenta neposkytuje uživatelské rozhraní. Aplikace mohou vytvářet vlastní oznámení a informovat tak jiné aplikace o vykonané události. Zaregistrovat receiver lze dynamicky pomocí `registerReceiver()` nebo staticky v Android manifestu (`<receiver>`).

[7] http://developer.android.com/guide/components/services.html
[8] http://developer.android.com/reference/android/content/BroadcastReceiver.html

Content provider

`Content provider`[9] je mechanismus, který umožňuje přistupovat k datům jiné aplikace. Jedná se o standardizované rozhraní pro výměnu dat mezi aplikacemi. Objekty reprezentuje jako záznamy v tabulce nezávisle na fyzickém uložení (soubory, databáze či vzdálený server).

Metodou `getContentResolver()` získáme objekt `ContentResolver` a s ním dále pracujeme. Na základě Content URI ve tvaru `content://authority/optionalPath/optionalId` rozhodne, jaký `Content provider` vybrat. Část *authority* identifikuje požadovaný Content provider, *optionalPath* definuje typ požadovaných dat a zadáním *optionalId* získáme jediný záznam podle unikátního čísla. `ContentResolver` poskytuje metody pro CRUD (Create Read Update Delete) operace `insert()`, `update()`, `delete()` a `query()`. K zvýšení efektivnosti lze využít metody `applyBatch()` a `bulkInsert()`, které seskupují více volání do jednoho. Metoda `query()` je syntaxí totožná s metodami pro práci s databází (viz. 4.4). Jako parametry přijímá Content URI, projekci (výběr sloupců), selekci (filtr řádků), agregaci (seskupování) a pořadí. Vrací `Cursor` objekt reprezentující výsledná data.

Systém Android nabízí několik vestavěných `Content provider` a mimo ně umožňuje si definovat vlastní a poskytnou tak data jiným aplikacím. Mezi vestavěné Content provider patří např. `Settings` (globální nastavení zařízení), `ContactsContract` (správa kontaktů), `MediaStore` (muzika, videa, obrázky) a od API 14 `CalendarContract`, který zajišťuje správu událostí v lokálním kalendáři. Vyžaduje definovat dodatečná práva v Android manifestu pro čtení a zápis dat do kalendáře.

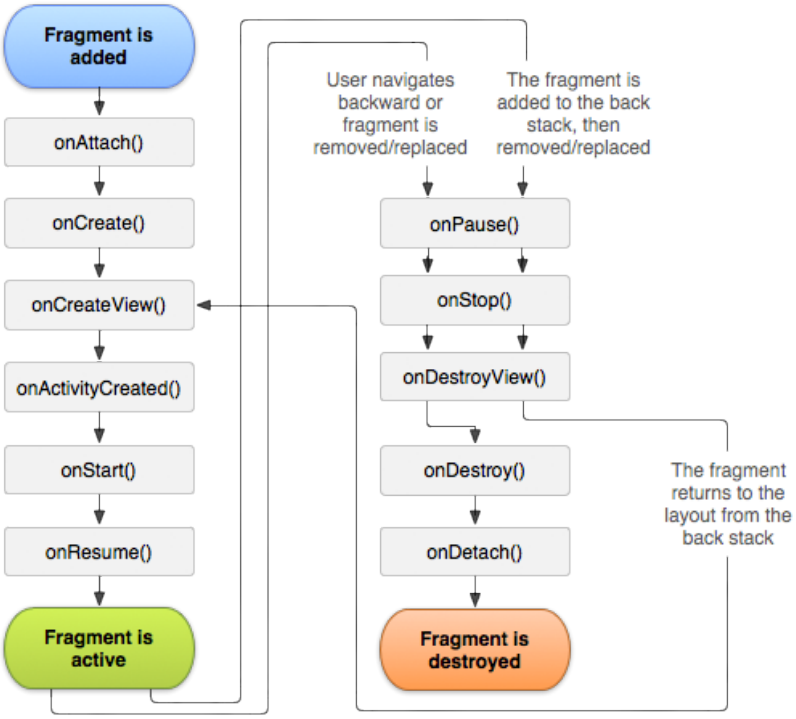
Intent

`Intent`[10] je asynchronní zpráva, která umožní vyvolat určitou akci jiné komponenty (Activity, Service či Broadcast receiver). Jedná se o jednoduchý objekt, který umožní uložit primitivní data a předat je jiné komponentě, a to jak v rámci aplikace, tak i napříč systémem. Existují 2 druhy: explicitní a implicitní. U explicitní je přímo nastavena třída, kterou chceme spustit. Tento způsob se nejčastěji využívá ke spuštění Activity (objekt `new Intent(context, SomeActivity.class)` je předán do metody `startActivity()` a daná třída je spuštěna). Zajímavější jsou implicitní, které umožňují pouze zadat jakou akci chceme vykonat (např. otevření webové stránky) a systém umožní otevřít Intent v aplikacích, které dokážou na požadovanou událost odpovědět.

Fragment

`Fragment`[11] je modul s vlastním uživatelským rozhraním a logikou. Tato znovupoužitelná komponenta byla poprvé představena v Android 3.0, první verze s podporou tabletů. Fragmenty jsou tu proto, aby bylo možné vytvářet složitější uživatelské rozhraní pro větší zařízení. Každá obrazovka může obsahovat více fragmentů. V případě tabletů často bývá více fragmentů na jedné obrazovce, zatímco na mobilních telefonech se prochází zanořenou hierarchií s jedním fragmentem na obrazovce. Nastavují se staticky pomocí tagu `<fragment>` či dynamicky třídou `FragmentManager` (metody jako `add()`, `replate()` aj. pro přidávání, úpravu atd). Pomocí unikátního jména lze získat již vytvořené fragmenty a dále s nimi pracovat. K vytvoření vlastního fragmentu je třeba zdědit třídu `Fragment` a přetížit potřebné metody.

[9] http://developer.android.com/guide/topics/providers/content-provider-basics.html
[10] http://developer.android.com/guide/components/intents-filters.html
[11] http://developer.android.com/guide/components/fragments.html



Obrázek 4.2: Životní cyklus fragmentu

`Fragment` má specifický životní cyklus (obrázek 4.2), který je úzce spjat s životním cyklem aktivity. Jakmile je aktivita spuštěna, ve fragmentu je zavolána metoda `onAttach()` (lze získat referenci na aktivitu) a metody `onCreate()` a `onCreateView()`, ve kterých dochází k vytvoření vzhledu fragmentu. Vzhled lze vytvořit dynamicky inicializací `View` objektu nebo pomocí `LayoutInflater`, který se využívá k načtení vzhledu z formátu XML. Po vytvoření vzhledu jsou zavolány metody `onStart()` a `onResume()` (nejprve metody aktivity a následně metody fragmentu). V případě ukončování jsou nejdříve volány ukončovací metody `onPause()` a `onStop` u fragmentu a nakonec u aktivity.

AsyncTask a Loader

Třída `AsyncTask`[12] usnadňuje práci s vlákny k vykonávání dlouhotrvajících výpočtů. Na rozdíl od třídy `Service`, která může běžet nepřetržitě, je určena pro kratší úkoly v řádu několika sekund (načtení dat z databáze, stažení dat z internetu atd.). Pro vytvoření je třeba zdědit `AsyncTask` a implementovat metodu `doInBackground()`. Úkon se spouští metodou `execute()`, pomocí které lze předat vláknu parametry. Přetížením metody `onPreExecute()` lze vykonat akci před spuštěním vlákna (např. zobrazit načítání). Po dokončení výpočtu je zavolán `onPostExecute()` callback, který je již spuštěn v hlavním vlákně a tedy lze měnit uživatelské rozhraní.

`Loader` je třída uzpůsobená pro načítání dat, která dokáže automaticky detekovat jejich změnu a reagovat na ni. Vnitřně využívá `AsyncTask` pro dlouhotrvající operace. Každá komponenta `Fragment` nebo `Activity` může `Loader` zaregistrovat pomocí `LoaderManager`. Jednou z variant je `CursorLoader`, který je uzpůsoben pro čtení dat z `Content provider`.

[12] http://developer.android.com/reference/android/os/AsyncTask.html

4.3 Uživatelské rozhraní

Architektura uživatelského rozhraní je podle návrhové vzoru Model-View-Controller (MVC). Model reprezentuje informace. Jedná se o jednoduché objekty, které obsahují data a zajišťují operace nad nimi. Model často implementuje serializaci dat (v prostředí Android rozhraní Parcelable). View zobrazuje uživateli data uložená v modelu. Každý model může mít více různých zobrazení. View se mohou lišit i v závislosti na velikosti obrazovky či rozlišení. Controller obsahuje aplikační logiku a zajišťuje aby získaná data (model) byla uživateli zobrazena (view). Tuto funkčnost zajišťuje třída Activity nebo Fragment (blíže popsané v podkapitole 4.2). V případě aktualizace dat, ať už na základě uživatelské interakce nebo automaticky na pozadí, dojde k překreslení obrazovky aktuálními daty.

Android 5.0 Lollipop představil nový vzhled uživatelského rozhraní – Material design[13]. Má za cíl sjednotit design, ať už jde o webové stránky, mobilní či desktopové aplikace. Vzhled aplikací se snaží přiblížit reálnému světu – pracuje s hloubkou. Každá komponenta obrazovky má určitou hloubku a pomocí stínování dosahuje pocitu plastičnosti. Součástí knihoven jsou i nové standardizované grafické prvky nejen pro mobilní telefony a tablety (RecyclerView, CardView, Floating Action Button aj.), ale také pro chytré hodinky (GridViewPager).

Definovat uživatelské rozhraní lze pomocí XML, programově v Javě či kombinací. Každý grafický element je potomek třídy `View`. Tato třída zajišťuje vykreslení prvku na obrazovku zařízení. Vykreslování probíhá v metodě `onDraw()`. Jako parametr dostane objekt `Canvas`, který obsahuje metody pro kreslení. Vlastnosti vykreslovaných primitiv lze ovlivnit nastavením objektů `Paint`. Při změně rozlišení (např. při otočení zařízení) dochází k přepočítání všech objektů v metodě `onSizeChanged` a překreslení. Celkový vzhled aplikace je dán kolekcí objektů `View`, resp. `ViewGroup` (a od nich odvozených) uložených ve stromové struktuře. `ViewGroup` a od něj zděděné třídy seskupují více prvků `View` a zajišťují jejich korektní vykreslení. Při vykreslování celé obrazovky je struktura procházena a vykreslovány jednotlivé položky. Procházení stromovou strukturou je preorder, čímž je zajištěno, že rodič je vykreslen dříve než potomek. Tedy nejvíce zanořené objekty jsou v nejvyšší viditelné vrstvě. Opačným způsobem jsou zpracovávány doteky na obrazovku. Nejdříve je zavolán nejvíce zanořený objekt. V případě, že dotek nezpracuje, je zavolán jeho rodič.

Jak již bylo zmíněno dříve, platforma Android podporuje zařízení nejrůznějšího druhu, jako jsou mobilní telefony, tablety či hodinky. Každé z těchto zařízení má jiné rozměry a rozlišení. Android poskytuje nástroje, jak pracovat s uživatelským prostředím tak, aby se korektně vykreslovalo na všech zařízeních. Při definici designu pracujeme namísto jednotky `px` s jednotkou `dp` (density independent pixel). $1dp = 1px$ na displeji s hustotou 160 dpi (dots per inch). Pokud bude hustota displeje jiná, dojde k přepočítání podle vztahu $px = dp * (dpi/160)$. Platforma Android rozlišuje displeje podle hustoty do několika generických kategorií: `ldpi, mdpi, hdpi, xhdpi a xxhdpi` (přesné hodnoty lze dohledat na této stránce[14]). S tím souvisí i práce s obrázky. Pro každou kategorii je třeba nahrát obrázek v příslušném rozlišení (např. u xhdpi musíme definovat 2x větší rozlišení obrázku než u mdpi – základu pro výpočet). Na základě velikosti displeje v dp jednotkách se rozlišuje následující typy: `small, normal, large a xlarge`. Pro různé velikosti lze definovat rozdílný vzhled (mobilní vs. tablet verze).

[13] http://www.google.com/design/spec/material-design/

[14] http://developer.android.com/guide/practices/screens_support.html

4.4 Ukládání perzistentních dat

Platforma Android umožňuje, podle stránek[15], ukládat data různými způsoby. Zvolení je závislé na dané situaci. Záleží na velikosti, struktuře nebo na tom, zda chcete data poskytovat aplikacím třetích stran.

Pro uložení privátních dat ve formátu klíč-hodnota lze využít `SharedPreferences`. Tato třída zajišťuje ukládání a načítání primitivních datových typů (boolean, float, int, long, string). Referenci na třídu získáme metodou `Context.getSharedPreferences()`. Pro čtení/zápis využíváme metody `getString()`/`putString()` (obdobně pro další datové typy), po ukončení zápisu je nutné potvrdit změny pomocí `commit()`. Interně se data ukládají v XML souboru v privátní datové oblasti aplikace.

V případě velkých souborů Android umožňuje ukládat a načítat soubory z interního či externího úložiště (SD karty). Pro práci se soubory lze využít standardní Java I/O API (`openFileOutput()`, `read()`, `write()`, `close()`) či další pomocné metody [6]. V případě interního úložiště jsou data uložena v privátní datové oblasti aplikace, zatímco u externího úložiště (vyžaduje `WRITE_EXTERNAL_STORAGE` povolení) jsou data ve sdílené paměti a dostupná tak všem aplikacím. Obdobným způsobem se pracuje i v případě využívání cache souborů (`getCacheDir()`).

SQLite

Pro ukládání strukturovaných dat je vhodné využít relační databázový systém SQLite [7]. Je implementován v jazyce C a pro jeho nenáročnost se využívá převážně na mobilních zařízeních. Implementuje téměř vše ze standardu SQL. Na rozdíl od jiných databází není jeho architektura typu client-server, ale je součástí programu. Celá databáze je uložena v jednom souboru, což s sebou nese výhody (přenositelnost) i nevýhody (fragmentace). Podporuje pouze datové typy BLOB (binární data), REAL (double), INTEGER (int) a TEXT (string), vše ostatní je nutné převést na jeden z nich (např. datum má přesně stanovený formát textového zápisu). Jelikož SQLite neprovádí typovou kontrolu, je nutné ošetření v programu.

Pro práci s SQLite se využívají třídy z balíčku `android.database.sqlite`. Nejlepší způsob je zdědit třídu `SQLiteOpenHelper` pro každou vytvářenou databázi. Při první pokusu získat databázi se nejdříve zavolá metoda `onCreate()`, ve které se spustí inicializační scripty k vytvoření databáze. V případě potřeby změnit databázové schéma změníme verzi databáze a při aktualizaci aplikace dojde k zavolání metody `onUpgrade()`. Zde lze spouštět aktualizační scripty na základě staré a nové verze databáze. Tímto způsobem lze udržovat databázi konzistentní i po aktualizaci programu.

Pomocí metod `getReadableDatabase()`, popřípadě `getWritableDatabase()` získáme přístup k `SQLiteDatabase` objektu, který poskytuje funkce pro vkládání (`insert()`), úpravu (`update()`) či mazání (`delete()`) dat v databázi. V případě nutnosti umožňuje spouštět přímo SQL příkazy pomocí `execSQL()`. Dotazy lze provádět pomocí metod `query()` nebo `rawQuery()`. Ty vrací `Cursor` objekt, který reprezentuje výsledek. Jedná se o ukazatel na řádky, které vyhovují zadaným kritériím v dotazu. Poskytuje metody k získání dat (`get*()`) a posunu na další položky. Po ukončení práce je nutné `Cursor` uzavřít (`close()`), stejně tak i celou databázi.

[15] http://developer.android.com/guide/topics/data/data-storage.html

4.5 Android Wear

Android Wear je odlehčenou verzí operačního systému Android uzpůsobený pro chytré hodinky a náramky, anglicky označované jako wearables. Pro plnohodnotný chod je třeba zařízení spárovat s mobilním telefonem či tabletem s Android 4.3+, se kterým komunikuje skrze rozhraní bluetooth. Instalace dodatečných aplikací probíhá během spárování. Ty jsou součástí běžných `.apk` balíčků, které lze stáhnout z Google Play. Spouštět aplikaci lze z menu nebo pomocí hlasového ovládání. Aplikace pak běží přímo na zařízení a mají přístup k dostupným senzorům.

Programování pro Android Wear rozlišuje v zásadě dva hlavní přístupy – `Suggest` a `Demand`. První přístup využívá tzv. `Context Stream`, relevantní informace, které se zobrazují v případě potřeby. Jedná se například o notifikace zaslané z mobilního telefonu či data závislá na poloze uživatele. Druhý přístup, `Demand` (na vyžádání), jsou aplikace spuštěné uživatelem, často pomocí hlasového ovládání. Komunikace mezi zařízeními probíhá pomocí mechanismu na zasílání zpráv (podrobněji v podkapitole 5.2). Jelikož se jedná o velmi malá zařízení, ovládání je omezeno pouze na velmi jednoduchá gesta – kliknutí, vertikální pohyb a horizontální pohyb. Snahou je nenutit uživatele na malých zařízeních přesně zacílit a proto musí být grafické prvky dostatečně velké. Nejčastěji jsou data uživateli prezentována ve 2D mřížce. Vertikálním pohybem přecházíme mezi kartami a horizontálním pohybem získáváme podrobnější informace.

Kapitola 5

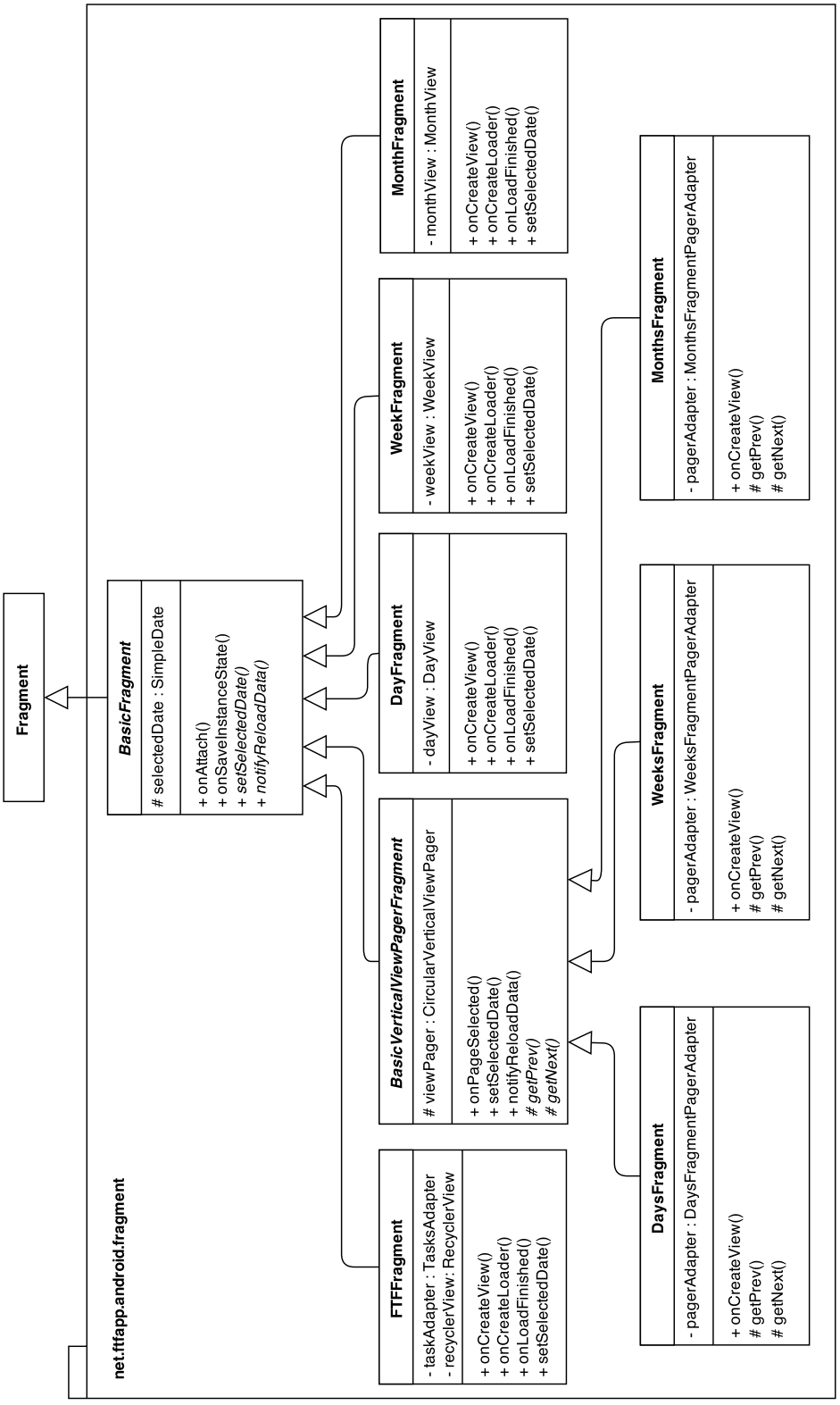
Implementace

V této kapitole je popsána implementace aplikace. Vychází z kapitoly 3, která popisuje návrh řešení. Taktéž čerpá z předchozí kapitoly 4, která popisuje teoretické znalosti nutné k vývoji mobilní aplikace pro platformu Android. Celý projekt kopíruje standardní strukturu Gradle projektu. Je rozdělen na 2 moduly. Modul **mobile** obsahuje verzi aplikace pro mobilní telefony a tablety. Druhá část projektu, modul **wear**, je určena pro wearables zařízení se systémem Android Wear. Tento modul je ve finálním `.apk` balíčku přibalen k mobilní verzi. Jednotlivé moduly jsou podrobněji popsány v následujících podkapitolách.

5.1 Mobilní verze

Mobilní verze vychází z obrázku 3.8, který zobrazuje celkové fungování programu. Základem celé aplikace je třída `MainActivity`, která je spuštěna po stratu. Hlavní funkcí této třídy je zobrazovat různé pohledy kalendáře a todo list, které se přepínají horizontálním pohybem. Princip byl podrobněji popsán dříve (obrázek 3.3). Možnost přepínat pohledy zajišťuje třída `ViewPager`, která je součástí Support Library. Zobrazovaná data se získávají z `PagerAdapter`, respektive z třídy `FragmentPagerAdapter`, které poskytují zobrazované fragmenty. Aby bylo možné se pohybovat v cyklické smyčce, je vytvořena třída `CircularViewPager`, která rozšiřuje původní třídu `ViewPager` a tuto funkčnost zajišťuje. Data jsou poskytována třídou `CircularPagerAdapter`, která poskytuje čtyři fragmenty – `FTFFragment`, `DaysFragment`, `WeeksFragment` a `MonthsFramgnet`.

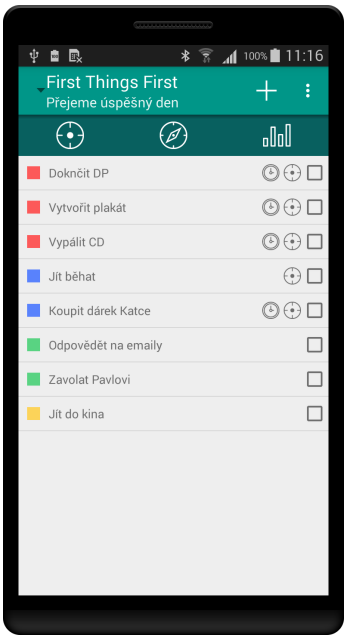
Třída `DaysFragment` zobrazuje denní, `WeeksFragment` týdenní a `MonthsFramgnet` měsíční pohled. Změna času na předcházející, resp. následující den, týden nebo měsíc probíhá vertikálním pohybem. Jelikož na platformě Android není vertikální stránkování standardní komponenta, byla vytvořena třída `CircularVerticalViewPager`, která vychází z původní a modifikuje ji pro vertikální pohyb. Společná funkčnost pro všechny výše popsané fragmenty je v třídě `BasicVerticalViewPagerFragment`. Obsahuje logiku pro vertikální stránkování. Jednotlivé stránky jsou pak vnořené fragmenty – `DayFragment` pro den, `WeekFragment` pro týden a `MonthFragment` pro měsíc. Všechny fragmenty pak dědí od základní třídy `BasicFragment`. Ta implementuje společné vlastnosti pro všechny fragmenty, jako je např. uložení a obnovení stavu při otočení zařízení (dojde k ukončení `Activity` a opětovného spuštění s obnoveným stavem). Celková struktura všech důležitých fragmentů je zobrazena v diagramu tříd na obrázku 5.1.



Obrázek 5.1: Diagram tříd nejdůležitějších fragmentů. Zachycuje dědičnost a nejdůležitější metody a atributy. Položky označené kurzívou jsou abstraktní.

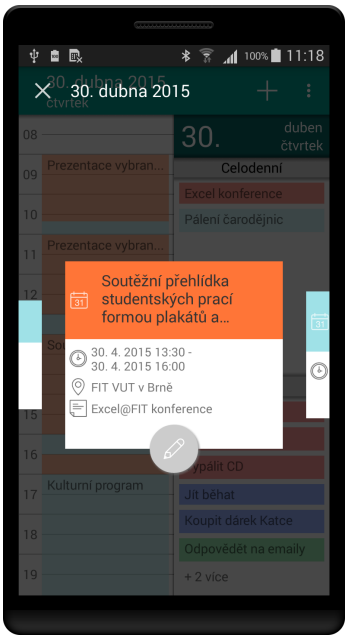
Práce s kalendářem probíhá pomocí `Content provider` (viz podkapitola 4.2), resp. pomocí vestavěné implementace `CalendarContract`. Každý fragment zobrazující data si udržuje vlastní kopii v rámci svého časového rozsahu – `DayFragment` pro den, `WeekFragment` pro týden a `MonthFragment` pro měsíc. Data načítá třída `CursorLoader`, která automaticky zajišťuje opětovnou aktualizaci při jejich změně. K prvotnímu načtení dojte, pokud se fragment stane viditelným. Každý z těchto fragmentů obsahuje třídu pro zobrazení korektního pohledu – `DayView` (obrázek 5.3), `WeekView` a `MonthView`. V prvotním prototypu aplikace byl vzhled definován pomocí XML, což se ukázalo, díky velmi složité struktuře, jako velmi pomalé řešení (zvláště na starších zařízeních). Ve finální verzi jsou tyto třídy potomky `View` a zobrazení je zajištěno vykreslováním na `Canvas` pomocí metod jako `drawRect()`, `drawLine()`, `drawText()` aj. Díky grafické akceleraci bylo urychlení značné. K vykreslování grafů ve fragmentu `StatsFragment` byla využita knihovna MPAndroidChart[1].

K zobrazení úkolů ve formě seznamu se v třídě `FTFFragment` (obrázek 5.2) využívá `RecyclerView`. Ten je určen pro seznamy o velkém počtu položek. Tato třída byla představena v novějších verzích systému Android jako součást Support Library v7 a na rozdíl od klasického `ListView` implementuje návrhový vzor ViewHolder. V něm se vytváří jednoduché objekty (`ViewHolder`), které drží reference na grafické prvky pro každou položku v seznamu (inicializace probíhá v metodě `onCreateViewHolder()`). V případě potřeby se tyto objekty znovu použijí. Systém se tak vyhýbá neustálému vytváření nových grafických objektů a využívá nepoužívané, čímž dosahuje značného urychlení. V metodě `onBindViewHolder` dojde k naplnění korektními daty.



Obrázek 5.2: FTFFragment zobrazující úkoly v seznamu.

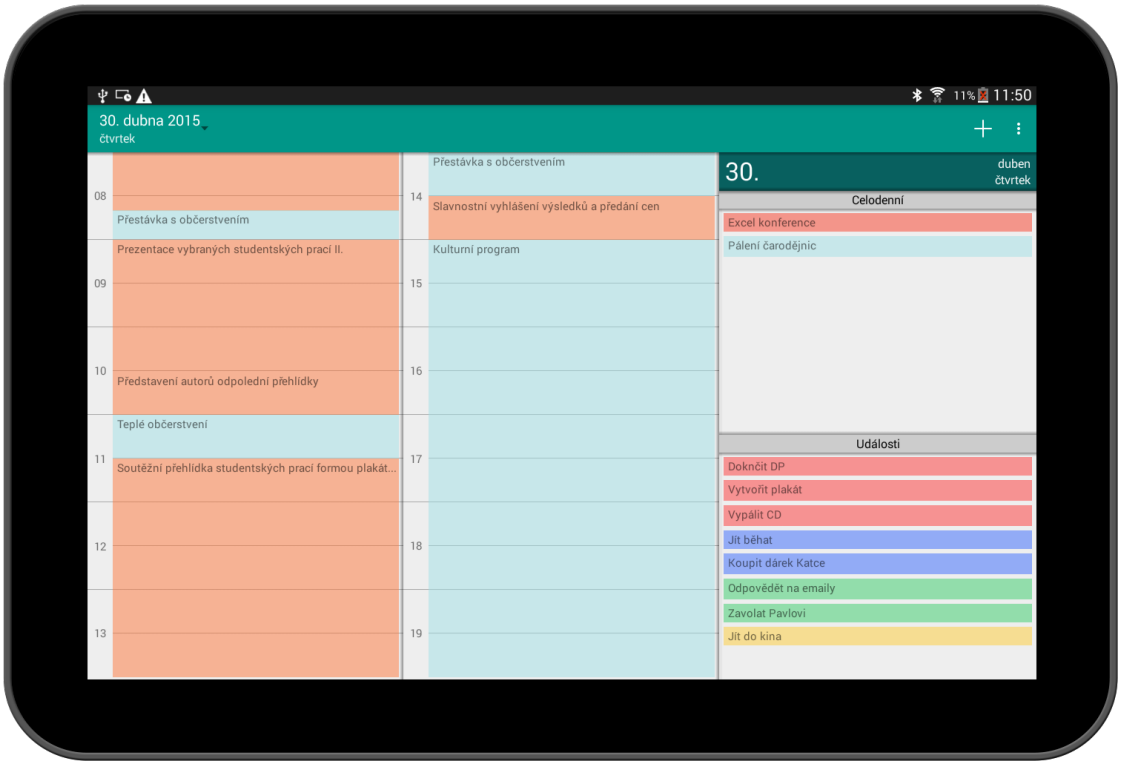
Obrázek 5.3: DayFragment zobrazující denní pohled DayView.



Obrázek 5.4: Zobrazení detailu s poloprůhledným pozadím.

[1] https://github.com/PhilJay/MPAndroidChart/

Jelikož platforma Android nemá v základu podporu pro správu úkolů, rozhodl jsem se implementovat vlastní `Content provider`, který tento nedostatek vyřeší. Vlastní `Content provider`, v našem případě třída `FTFContentProvider`, musí být zaregistrována v souboru `AndroidManifest.xml` pomocí tagu `<provider>`. Lze nastavit, zda má být dostupný i aplikacím třetích stran či zda musí aplikace vyplňovat dodatečná práva. Při vytváření je třeba vytvořit tzv. `Contract` (třída `FTFContract`). Je to předpis, který je dodán ostatním vývojářům jako API pro práci s vaší komponentou. Poskytuje identifikátory AUTHORITY a CONTENT_URI k identifikaci a obsahuje vnitřní třídy, které reprezentují prezentovaná data. Záměrně uvádím prezentovaná, nikoli ukládaná, jelikož lze vytvářet i fiktivní tabulky, které se vytváří složitějším dotazem na databázi. Mnou implementovaný `FTFContract` umožňuje přistupovat k úkolům (podtřída `Task`), rolím (podtřída `Role`), cílům (podtřída `Goal`) a kombinacím rolí a cílů (podtřída `RoleGoal`). Pro každý tento typ je nutné v třídě `FTFContentProvider` zaregistrovat tzv. UriMatcher (`URI_MATCHER.addURI(FTFContract.AUTHORITY, "roles", ROLES)`), pomocí kterého se na základě zadané URI provede požadovaná akce. Při vytváření (`onCreate()`) je nutné získat objekt `DatabaseHelper` pro práci s databází. Nejdůležitější částí třídy jsou pak metody `insert()`, `update()`, `delete()` a `query()`, které zajišťují CRUD (Create Read Update Delete) operace. Na základě zadaného URI provedou požadovanou akci v databázi a vrátí výsledek – Cursor s daty v případě `query()`, počet aktualizovaných, respektive smazaných položek u `update()`, respektive `delete()` a URI nově přidané položky u metody `insert()`.



Obrázek 5.5: Denní pohled uzpůsobený pro tablety

5.2 Android Wear verze

Vývoj pro Android Wear se mnoha ohledech podobá vývoji klasické Android aplikaci. Mohou se zde využívat základní komponenty (viz podkapitola 4.2) a až na výjimky lze využívat Android API. Základní rozdíl je v uživatelském rozhraní, které využívá speciální třídy uzpůsobené malým rozlišením. Dalším specifikem je komunikace se spárovaným zařízením pomocí `Wearable Data Layer API`. To je součástí `Google Play services`, které jsou nutností pro vývoj Android Wear aplikace. Jedná se o komunikační kanál mezi spárovanými zařízeními. K umožnění komunikace je třeba zařízení zaregistrovat do `Google Play services`. Pro tuto práci se využívá třída `GoogleApiClient`. Při startu programu, který chce komunikovat, se zaregistruje do wearable služby (`addApi(Wearable.API)`) a připojí se (`connect()`). Komunikace probíhá pomocí několika různých rozhraní:

- `Node API` se využívá k získání informací o připojených zařízeních do `Google Play services`. Pomocí metody `getConnectedNodes()` lze získat symbolická jména všech zařízení. Ta jsou potřeba pro další komunikaci. Aby šlo s API pracovat, musí být zaregistrován callback po připojení (`Wearable.NodeApi.addListener()`), resp. odregistrován v případě odpojení (`Wearable.NodeApi.removeListener()`).

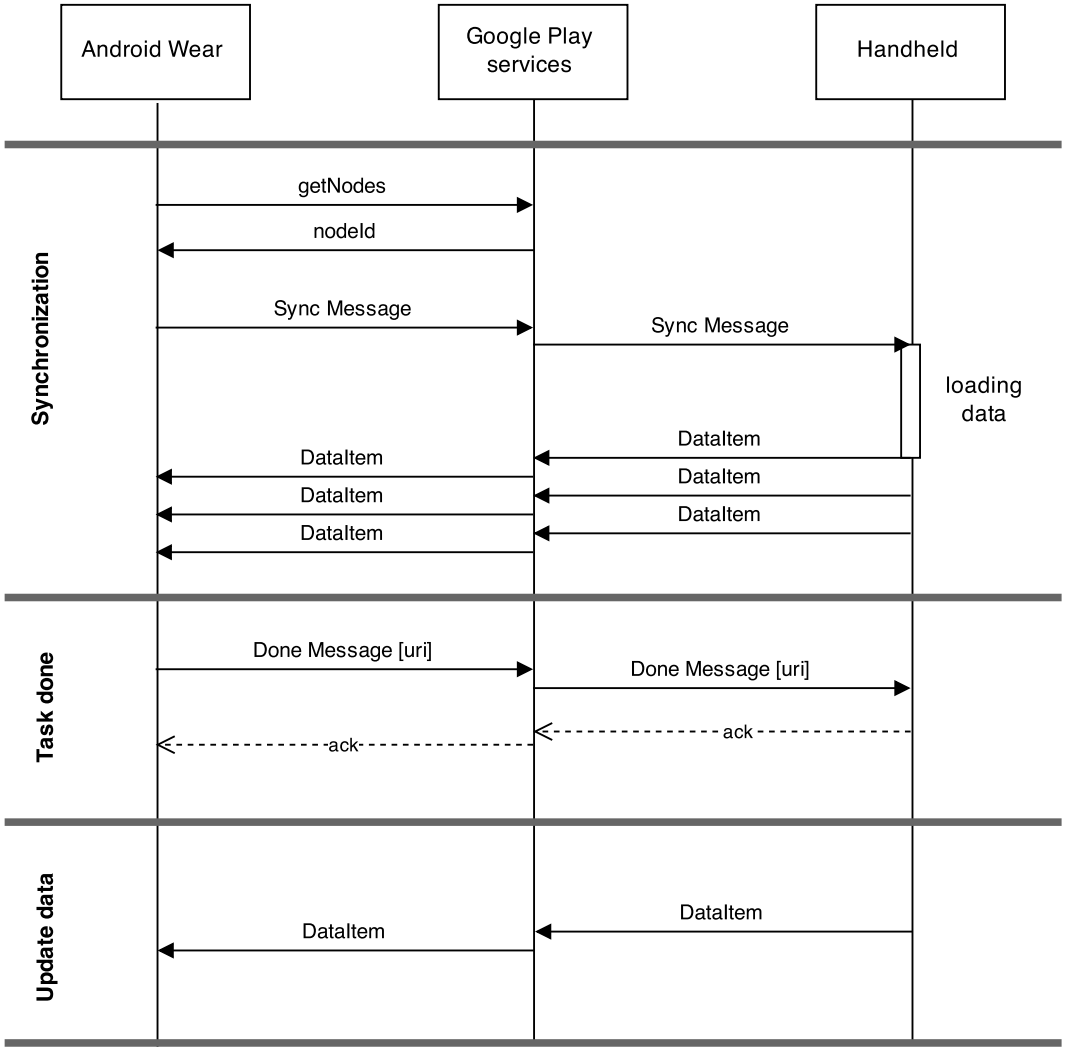
- `Message API` je nejjednodušší způsob výměny dat mezi zařízeními. Jedná se o jednosměrnou zprávu zaslanou pomocí metody `sendMessage()`. K identifikaci je třeba znát symbolické jméno uzlu se kterým chceme komunikovat (získané pomocí `Node API`). Každá zpráva se pak skládá z řetězce popisující zprávu a dodatečných dat (byte[] o velikosti max 100KB). Aby bylo možné zprávy přijímat, musí aktivita či služba implementovat rozhraní `MessageListener`, konkrétně metodu `onMessageReceived()`. Ta je spuštěna při přijetí jakékoliv zprávy. Pomocí řetězce získané ze zprávy lze pak na danou zprávu reagovat.

- `DataLayer API` je mechanismus pro obousměrnou výměnu dat. Data jsou doručena všem naslouchajícím uzlům. V případě změny dat, ať už ze strany mobilního zařízení nebo chytrých hodinek, dojde k zaslaní aktualizovaných dat. Mechanismus je připraven pro více zařízení napojených do služby. Díky tomu jsou data synchronizována mezi všemi zařízeními (mobil, tablet, hodinky). Data lze do služby nahrát metodou `putDataItem()`, která přijímá jako parametr `DataMap` – jednoduchá struktura klíč-hodnota určená k serializaci. Aby bylo možné data přijímat, musí aktivita či služba implementovat rozhraní `DataListener`, konkrétně metodu `onDataChanged()`. Ta je spuštěna při aktualizaci či smazání dat. Pomocí řetězce lze data identifikovat a zpracovat je.

Základem Android Wear verze je třída `MainActivity`. Implementuje všechna potřebná rozhraní pro práci s `Node API`, `Message API` i `DataLayer API`. Po startu se zaregistruje do `Google Play services` a zajišťuje veškerou komunikaci na straně hodinek. Komunikuje s mobilní částí, konkrétně službou `FTFWearableListenerService`, která zajišťuje logiku na straně mobilní aplikace.

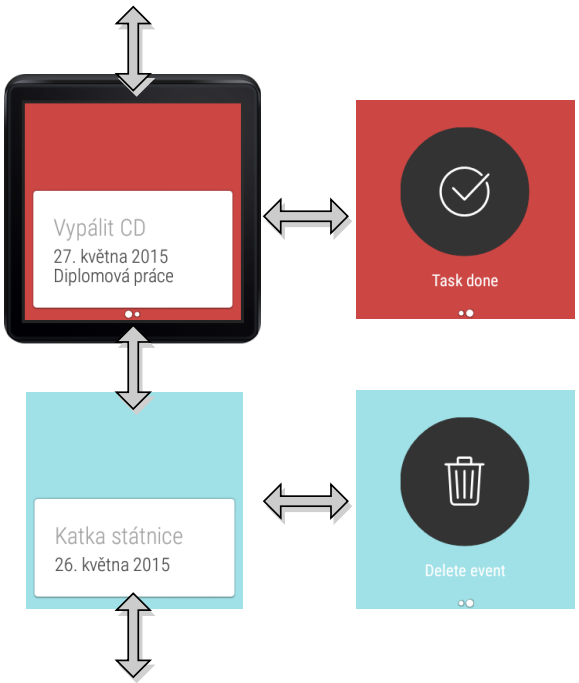
Princip komunikace zachycuje diagram 5.6. Po spuštění programu v hodinkách je nutné zajistit synchronizaci dat (na obrázku v horní části). Je třeba získat unikátní identifikátor mobilního zařízení pro další komunikaci. Po jeho úspěšném získání (a tedy potvrzeném spárováním) je zaslána synchronizační zpráva danému uzlu. Mobilní aplikace po jejím přijetí načte data z databáze. Načtené úkoly a události na následující den jsou převedeny do

objektů `WearableAction`, které zajišťují serializaci. Jakmile `Google Play services` získá nová data, informuje všechny zaregistrované uzly. Na straně hodinek je zavolána metoda `onDataChanged()`, která data zpracuje a uloží. V případě, že je potřeba vyvolat z hodinek nějakou akci (označit úkol jako hotový či smazání události), dojde k zaslání asynchronní zprávy (obrázek 5.6 uprostřed), která jako parametr přenese URI daného objektu. V mobilním zařízení je pak akce provedena. Pokud dojde v mobilním zařízení k úpravě či smazání dat, mobilní zařízení nahraje nová data do `Google Play services` a ta informuje všechna připojená zařízení (obrázek 5.6 dole).



Obrázek 5.6: Ukázka komunikace mezi Android Wear zařízením, Google Play services a mobilním zařízením

Data jsou uživateli prezentovány v 2D mřížce pomocí třídy `GridViewPager` – speciální třída určená pro wearable zařízení. Pro každou získanou `WearableAction` je vytvořen řádek `GridRow`, který zobrazuje data v kartě a umožňuje spouštět akce (obrázek 5.7).



Obrázek 5.7: Android Wear aplikace. Data jsou prezentovány ve 2D mřížce. Vertikálním pohybem přepínáme mezi řádky, horizontálním pohybem přecházíme mezi kartou s daty a akcemi.

5.3 Publikování aplikace

Finální verze byla publikována v katalogu Google Play[2]. K uveřejnění aplikací v katalogu je třeba vlastnit vývojářský účet (jednorázový poplatek $25). Každá nahrávaná aplikace musí být digitálně podepsána. V případě ztráty privátního klíče již není možné aplikaci upravovat. Zveřejnění probíhá během několika hodin od nahrání. Součástí karty aplikace jsou i grafické materiály, které bylo potřeba vytvořit. V rámci snahy oslovit větší počet zákazníků byly vytvořeny i jednoduché webové stránky propagující aplikaci – www.ftfapp.net. Analýze finální aplikace se podrobněji věnuje následující kapitola.

[2]http://play.google.com/store/apps/details?id=net.ftfapp.android

Kapitola 6

Testování a experimenty

Testování aplikace probíhá v několika rovinách. Kód programu je testován pomocí automatizovaných testů, které kontrolují stěžejní části aplikace. Během vývoje byl program spouštěn a testován na různých zařízeních. V rámci semestrálního projektu bylo provedeno testování prototypu (podkapitola 6.3). Na základě zpětné vazby docházelo k úpravám uživatelského rozhraní a funkčnosti. V podkapitole 6.4 je finální verze aplikace porovnána s konkurenčními. Podkapitola 6.5 popisuje experiment, který měl za cíl zjistit, jak finální verze aplikace dokázala splnit stanovené cíle. Je zde popsán postup, vyhodnocení i možné budoucí rozšíření programu.

6.1 Automatizované testy

Automatizované testování[1], které ověřuje očekávané výstupy jednotlivých izolovaných částí systému, je v systému Android založeno na frameworku JUnit 3[2]. Potřebné nástroje jsou součástí Android SDK a integrovány do vývojového prostředí Android Studio. Je třeba rozlišovat testy, které vyžadují Android API a které nikoliv. V případě ze chceme otestovat kód, který nevyužívá Android API, vystačíme si s třídou `TestCase`. Využívá se pro test běžných Java tříd či výpočtů. `TestCase` poskytuje metody pro počáteční inicializaci (`setUp()`) a řádné ukončení testů (`tearDown()`). Metody s prefixem `test*()` jsou považovány za izolovaný modul k otestování. Ke kontrole očekávaných hodnot se využívá třída `Assert` a metody jako `assertEquals()`, `assertNotNull()`, `assertTrue()` aj.

Základem testování kódu využívající Android API je třída `AndroidTestCase`. Dědí od třídy `TestCase`, kterou rozšiřuje o možnost testovat specifika pro Android. Platforma poskytuje další třídy pro testování všech komponent (popsané v podkapitole 4.2) – Service (`ServiceTestCase`), Aktivity (`ActivityInstrumentationTestCase2`) nebo Content provider (`ProviderTestCase2`). Ty dědí od třídy `AndroidTestCase` a umožňují testovat životní cyklus dané komponenty. Testovací framework poskytuje řadu připravených mock objektů – fiktivní objekty, které umožňují izolovat testované části od zbytku reálného systému (`MockApplication`, `MockContext`, `MockCursor`, `MockContentProvider` či `MockResources`).

Při testování této aplikace jsem se zaměřil na testování modelových tříd (`Task`, `Event`, `SimpleDate` aj) a ověření korektního ukládání, serializace a výpočtů. Taktéž jsem ověřoval stěžejní části aplikace, jako je práce s databází nebo třídu `MainActivity`. Netestoval jsem vykreslování GUI.

[1] http://developer.android.com/tools/testing/testing_android.html
[2] http://junit.org/

6.2 Testovaná zařízení

Aplikace je vyvíjena s cílem pokrýt velké množství zařízení s platformou Android – mobilní telefony, tablety a hodinky. Počet zařízení, které jsou kompatibilní s vyvíjenou aplikací je velké množství (podle Google Play karty aplikace téměř 8000 typů zařízení). Otestovat všechny není možné. Během vývoje jsem se však snažil otestovat různé konfigurace, ať už reálných zařízení či emulovaných (různou velikost a rozlišení displeje, verzi systému a další parametry). Pro emulovaná zařízení byl využit jak emulátor v Android SDK, tak i komerční emulátor Genymotion[3]. Testovaná zařízení jsou uvedena v tabulce 6.1.

Tabulka 6.1: Testovaná zařízení

Zařízení	**Typ**	**Android**	**Rozlišení [px]**
Samsung Gallaxy S3 mini	mobil, reálné	4.1 (API 16)	480x800
Samsung Gallaxy S3 Neo	mobil, reálné	4.4 (API 19)	720x1280
Samsung Gallexy Tab4	tablet, reálné	4.4 (API 19)	1280x800
LG G WATCH W100	hodinky, reálné	5.0 (API 21)	280x280
Android Wear Round	hodinky, emulátor	5.0 (API 21)	320x320
Google Nexus konfigurace	tablet, emulátor	5.0 (API 21)	1440x2560
Google Nexus konfigurace	mobil, emulátor	4.3 (API 18)	1200x1920

6.3 Prototyp a beta testování

V rámci semestrálního projektu bylo provedeno testování prototypu aplikace metodou beta testování. Vybraní uživatelé dostanou možnost si aplikaci stáhnout ještě před oficiálním uveřejněním na trh. Program má v sobě zabudované mechanismy, které umožňují sledovat uživatelovu aktivitu a odesílat ji vývojářům. Tímto způsobem lze odhalovat chyby v reálném provozu a na základě zpětné vazby od testerů upravovat aplikaci.

Jedním ze způsobů beta testování je pomocí oficiální distribuční služby Google Play[4]. Lze nahrát aplikaci v módu beta testování a zpřístupnit tak aplikaci vybrané skupině uživatelů. Pozvaným uživatelům (ať už emailem či pomocí Google+ komunity) se zpřístupní aplikace v jejich Google Play katalogu, jako běžná aplikace. Beta testování umožňuje zaznamenávat pády aplikace a zobrazovat stack trace. Pro další monitorování a analýzu jsou potřeba další dodatečné nástroje integrované do aplikace. Jedním z nich je Google Analytics[5]. Zaměřuje se převážně na analýzu uživatelů, dokáže monitorovat jejich počet v reálném čase, geografickou lokalitu, průchod aplikací, nejčastěji navštěvované obrazovky nebo pády aplikace. Výhoda beta testování pomocí Google Play je jednoduchost a standardní instalace, na kterou jsou uživatelé zvyklí.

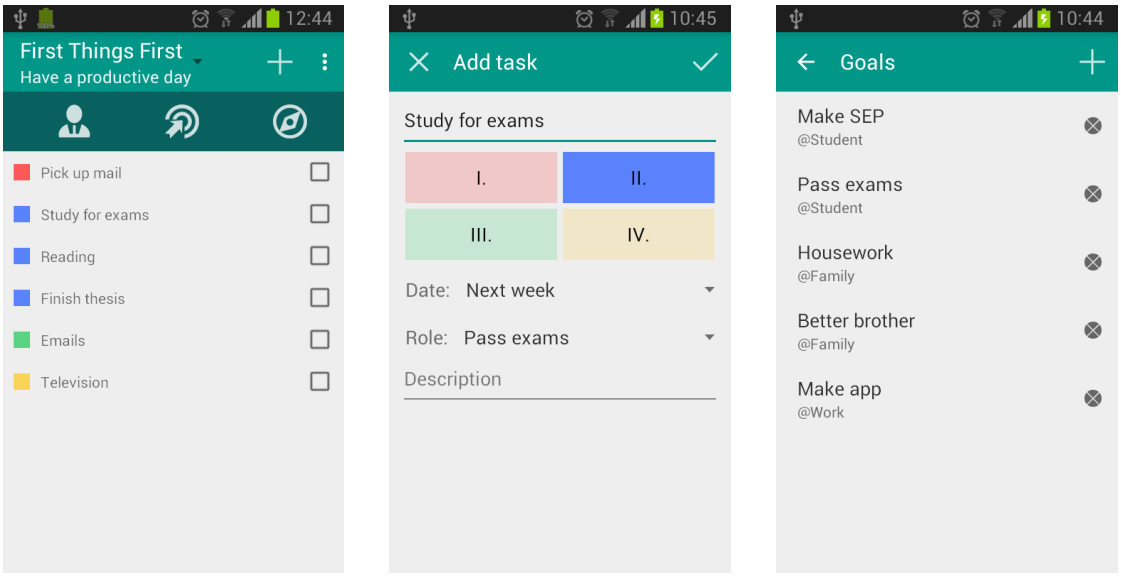
Další možností je využít specializovanou službu na testování. TestFairy[6] je jednou z nich. Jedná se o propracovanou službu, která poskytuje podrobné informace ke každému testu. Test je identifikován emailem testera, verzí Android, rozlišením obrazovky a modelem zařízení. Ukládá logovací zprávy, vytížení CPU, paměti a sítě. Každé sezení zaznamenává a

[3] https://www.genymotion.com/
[4] https://play.google.com/
[5] http://www.google.com/analytics/
[6] http://testfairy.com/

vytváří během testu screenshoty. V případě pádu se ukládá stack trace a odesílá se email s upozorněním správci. Veškeré výsledky zobrazuje v podrobných grafech a statistikách. K běhu nepotřebuje žádné speciální SDK integrované do aplikace. Stačí nahrát finální `.apk` na stránky TestFairy, nastavit parametry testu a rozeslat pozvánky testerům. Kvůli mnohem podrobnějším statistikám a záznamům z každého testu jsem se rozhodl využít testovací službu TestFairy. Za cenu podrobných statistik se platí nestandardní instalací, která vyžaduje více práce na straně testerů.

Do beta testování prototypu se zapojilo přes 30 dobrovolníků. Prototyp aplikace byl jednoduchým správcem úkolů, který lze vidět na obrázku 6.1. Oproti plánované finální verzi byla funkčnost omezena. V aplikaci bylo možné přidávat, upravovat a mazat úkoly. Každý úkol mohl uživatel přiřadit k jednomu z cílů a na základě důležitosti a naléhavosti zařadit do matice plánování. Uživatel si mohl sepsat osobní poslání a upravovat své role a cíle. Správa rolí a cílů byla oddělena na samostatné obrazovky.

Z testů vyplynulo několik poznatků. Prvním z nich bylo nedostatečné pochopení filosofie First-Things-First. I když je aplikace cílená především na znalé uživatele, byla by škoda přijít o další potencionální zákazníky, a proto jsem se rozhodl do finální verze lépe zakomponovat nápovědu. Ta by neměla uživatele příliš vyrušovat a měla by mu pomoci v době, kdy to potřebuje. Ve finálním testování bude užitečnost nápovědy ověřena. Dalším poznatkem bylo, že je aplikace stabilní a až na několik drobných chyb nedochází k pádu aplikace. Díky logování chyb bylo možné dohledat patřičná místa a chyby odstranit. Testovací framework zaznamenává i vytížení CPU a spotřebovanou paměť RAM. V tomto případě se aplikace nevymykala běžným hodnotám. Jeden z účastníků testu mi napsal, že se mu aplikace líbí, ale chybí mu možnost zobrazit počet hotových úkolů. Tento komentář mě přivedl k myšlence zakomponovat do aplikace podrobné statistiky. Jejich účelovost a přehlednost bude ve finálním testování ověřena. Z komentářů jsem získal i další drobné poznatky, které budou zapracovány do finální verze (přehlednější rozložení prvků ve formuláři pro vkládání, výstižnější popisky a překlad aj.).



(a) Správa úkolů (b) Přidávání úkolu (c) Správa cílů

Obrázek 6.1: Obrázky z prototypu aplikace, která byla využita k prvotnímu testování

6.4 Porovnání s konkurenčními aplikacemi

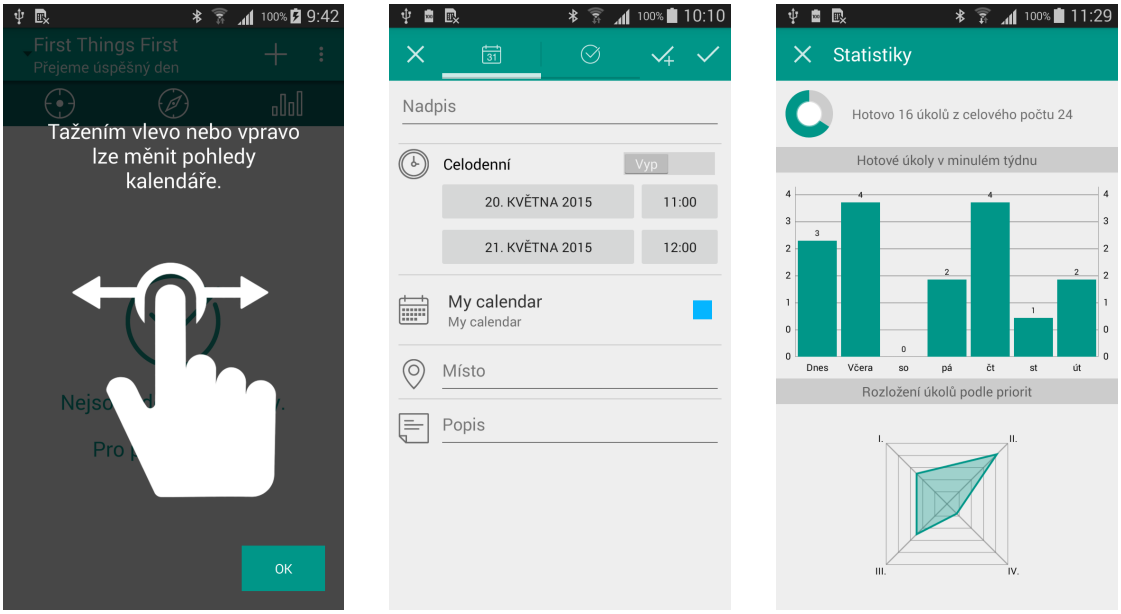
Tato podkapitola porovnává finální aplikaci s některými konkurenčními, které byly analyzovány v podkapitole 2.1. Obecné funkce každého kalendáře porovnává s aplikací aCalendar, kterou považuji za velmi propracovaný kalendář. Prvky z filosofie First-Things-First porovnává s aplikací MyEffectiveness, která je taktéž implementuje.

Jedním z požadavků systému 4. generace je efektivnost. Porovnal jsem složitost provedení nejpoužívanějších akcí, respektive počet kliknutí nutných k provedení akce. Výsledky jsou zaneseny v následující tabulce 6.2. Z ní zle vyčíst, že vyvíjená aplikace dosahuje stejných výsledků jako aplikace aCalendar, což považuji za úspěch. Pravděpodobně se jedná o minimální možný počet kliknutí pro provedení všech akcí. Aplikace MyEffectiveness, díky své složitě zanořené struktuře, vyžaduje mnohem více kliknutí (pro smazání až 5 kliknutí). Správu událostí a kalendář neimplementuje vůbec.

Tabulka 6.2: Počet kliknutí nutných k provedení akce v porovnávaných aplikacích

Akce	**FirstThingsFirst**	**aCalendar**	**MyEffectiveness**
Zobrazení úkolů	0	0	1
Podrobnosti úkolu	1	1	2
Přidání úkolu	1	1	1
Editace úkolu	2	2	3
Smazání úkolu	2	2	5
Zobrazení událostí	0	0	N/A
Podrobnosti události	1	1	N/A
Přidání události	1	1	N/A
Editace události	2	2	N/A
Smazání události	2	2	N/A

Další kritérium experimentu je funkčnost z pohledu filosofie First-Things-First. Zde má smysl porovnávat moji aplikaci pouze s MyEffectiveness. Obě umožňují uživateli definovat osobní poslání, životní role a cíle. MyEffectiveness umožňuje k rolím přidávat obrázky, což moje neumožňuje. Aplikace MyEffectiveness umožňuje přidávat úkoly na několika místech v aplikaci, avšak zařazení do priorit se musí vybrat před zobrazením formuláře a během editace (obrázek 2.2c) nelze změnit, což považuji za velmi nešťastné řešení. V mém případě se výběr priorit provádí kliknutím na jeden z kvadrantů matice plánování. MyEffectiveness podporuje práci pouze s úkoly a neslouží tedy jako kalendář, zatímco moje se snaží kombinovat úkoly i události (lokální i vzdálené). MyEffectiveness umožňuje vkládat samostatné poznámky, zatímco v případě mojí aplikace jsou poznámky součástí úkolů. MyEffectiveness umožňuje navíc definovat okruh vlivu a okruh zájmu (více v knize [2]), zatímco moje aplikace zobrazuje statistiky a obsahuje prvky gamifikace. Celkově obě implementují obdobné prvky z filosofie, nicméně moje se snaží o větší přehlednost, což se snažím ověřit v následujícím experimentu. Nespornou výhodou mé aplikace je pak možnost práce s kalendářem a podrobné statistiky.



(a) Nápověda (b) Přidávání událostí (c) Statistiky

Obrázek 6.2: Obrázky z finální aplikace, ve které byly zapracovány změny oproti prototypu

6.5 Uživatelské testování

Finální verze aplikace byla podrobena uživatelskému testování. Experiment vychází z metody *myšlení nahlas* [5]. Základem je předem stanovený scénář, který musí tester v rámci aplikace plnit. Autor aplikace pozoruje postup při práci a zaznamenává si získané poznatky. V ideálním případě je celý test nahráván pro další analýzu (několik záznamů pořízených během testů je na přiloženém CD). Při výběru respondentů jsem se snažil vybírat osoby z cílové skupiny. Tedy převážně osoby, které mají zkušenost s mobilní platformou Android a se zájmem o aplikaci na správu času. Testu se účastnilo i několik respondentů, kteří však neměli žádné zkušenosti s digitálními kalendáři ani aplikacemi podobného typu. Jejich pozorování bylo obzvláště užitečné. Pro test byl stanoven následující scénář:

1. Seznam se s aplikací. Můžeš dělat cokoliv, co tě napadne.
2. Sepiš si vlastní osobní poslání.
3. Přidej novou roli a cíl.
4. Přidej nový úkol a v případě úspěchu:
   (a) přidaný úkol uprav
   (b) upravený úkol smaž
5. Přidej novou událost a v případě úspěchu:
   (a) přidanou událost uprav
   (b) upravenou událost smaž
6. Zobraz statistiky a popiš, co zobrazují.

Na závěr testu dostal každý účastník k vyplnění dotazník. Cílem dotazníku je zjistit, jak se povedlo implementovat efektivní uživatelské rozhraní a filosofii First-Things-First. Dotazník je inspirován metodou *Likert Scale* [4]. Základem této metody je předkládat hodnotícímu tvrzení s kterými se více či méně ztotožňuje. Počet možných odpovědí musí být lichý a obsahovat i neutrální odpověď. Vyplňovaný dotazník, včetně všech otázek a možných odpovědí, je uveden v tabulce 6.3.

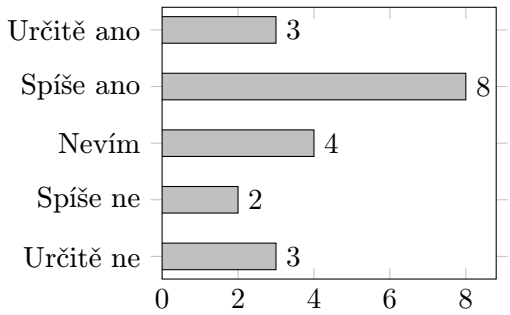
Tabulka 6.3: Dotazník předkládaný testerům inspirovaný metodou *Likert Scale*

Akce	**Určitě ano**	**Spíše ano**	**Nevím**	**Spíše ne**	**Určitě ne**
Celkový dojem z aplikace je pozitivní					
Ovládání aplikace mi přijde intuitivní					
Pochopil jsem koncept filosofie First-Things-First					
Formulář pro přidávání úkolů mi přijde přehledný					
Formulář pro přidávání událostí mi přijde přehledný					
Denní pohled mi přijde přehledný					
Týdenní pohled mi přijde přehledný					
Měsíční pohled mi přijde přehledný					
Zobrazování úkolů a událostí mi přijde přehledné					
Správa rolí a cílů mi přijde intuitivní					
Zobrazované statistiky mi přijdou užitečné					
Nápověda se mi zdá užitečná					

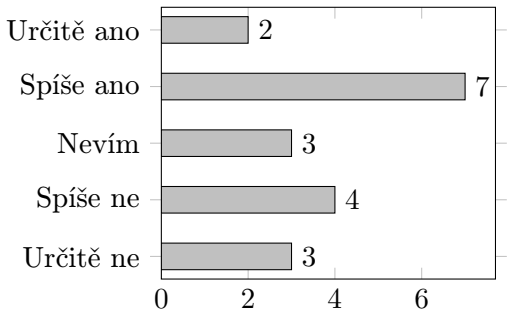
V následujícím odstavci jsou shrnuty poznatky, které jsem získal během pozorování testů. Každý tester dostal čistou instalaci aplikace a v první fázi mu byl dán prostor k seznámení se s aplikací. V testované aplikaci je předvyplněno několik rolí, cílů a úkolů. Při prvním spuštění se objevuje nápověda, která se snaží velmi stručně popsat filosofii v několika krocích (obrázek 6.2a). Testeři se zde rozdělili na dvě odlišné skupiny. První z nich si nápovědu podrobněji procházela a druhá ji okamžitě přeskočila a neobtěžovala se se čtením. Výsledky užitečnosti nápovědy, jedna z otázek dotazníku, pak kopírovaly tyto skupiny. Po ukončení nápovědy si všichni testeři začali procházet aplikaci. Každý strávil touto činností tolik času, kolik chtěl. Při plnění druhého úkolu, sepsat si vlastní osobní poslání, bylo zajímavé, že část respondentů začala upravovat přednastavený úkol *Sepište si své osobní poslání* a své poslaní psala do pole *Popis*. Záměrem předpřipravených úkolů (vytvářených během inicializace databáze) bylo usnadnit prvotní spuštění aplikace. Ukázalo se však, že část respondentů tím byla spíše zmatená. Díky tomuto poznatku jsem se rozhodl odstranit předinstalované úkoly a nechat vše na uživatelích. Většině uživatelů se však podařilo sepsat osobní poslání korektně. Osobní poslání se ukládá automaticky bez nutnosti změny potvrdit, nicméně jednomu z uživatelů takové tlačítko chybělo (nebyl si jist, zda dojde k uložení), proto bylo do finální verze přidáno. Již před testováním jsem úkol 3, vytvořit novou roli a cíl, považoval za nejtěžší hlavně kvůli složitější struktuře. Uživatelé neznalí filosofie First-Things-First příliš nerozlišovali roli a cíl. Data se zobrazovala v seznamu s hierarchickou strukturou, přičemž šlo části seznamu skrýt. Možnost skrytí byla spíše na obtíž a méně technicky zdatné uživatele mátla. Byla odstraněna a místo ní bylo přidáno intuitivnější vkládací formulář. Několika uživatelům se nepodařilo vložit vlastní cíl. Tito

uživatelé patřili k těm, kteří hodnotili aplikaci jako neintuitivní. Následně měli uživatelé přidat úkol. Všichni testeři věděli, kde hledat přidávání nového úkolu. Dokázali zadat název a zařadit jej pomocí matice plánování. V horním panelu je možnost ukládat dvěma způsoby (obrázek 6.2b vpravo nahoře). První z nich úkol uloží a zavře obrazovku, zatímco druhý úkol uloží a nechá obrazovku aktivní, aby uživatel mohl přidávat další úkoly či události. Tuto neobvyklou funkci ne všichni testeři pochopili, avšak ti co ji pochopili, ji hodnotili jako přínosnou. Editace a smazání úkolu nebyl u velké většiny problém. U formuláře pro přidání nové události nenastaly větší komplikace, každý dokázal změnit začátek a konec a doplnit ji o dodatečná data, jako je místo konání či poznámka. S úpravou a smazáním, stejně jako u úkolů, nebyly problémy. Z toho lze vyvodit, že zobrazování detailu a ikonky pro editaci a mazání byly přehledné a každý poznal jejich účel. Posledním úkolem bylo zobrazit statistiky a popsat, co znázorňují. U tohoto úkolu byl u některých problém, že dříve neoznačili žádné úkoly jako splněné, a proto byly grafy prázdné. Pokud jsem jim naznačil, že je třeba některý z úkolů označit jako splněný, po provedení dokázali skoro všichni popsat, co grafy zobrazují. V názoru na užitečnost statistik se uživatelé lišili. Zatímco ti, kteří měli zkušenost s filosofií First-Things-First nebo využívají jinou aplikaci ke správě času, je považovali za zajímavé, část uživatelů nevěřila v jejich užitečnost. Celkový dojem z testování byl pozitivní, až na výjimky se všem podařilo splnit zadané úkoly.

Dotazník, který následoval po samotném testu, vyplnilo všech 20 účastníků experimentu. Kompletní výsledky jsou zobrazeny v grafech v příloze B. Na následujících grafech 6.3 a 6.4 jsou zachyceny nejdůležitější otázky – zda uživatele aplikace zaujala a zda jim ovládání přijde intuitivní. Odpovědi jsou převážně pozitivní, což považuji za úspěch. Kladně odpovídali převážně lidé, kteří již využívají nějaký systém pro správu času, a kteří mají alespoň základní povědomí o filosofii First-Things-First. Negativně pak odpověděli lidé, kteří žádný systém nevyužívají.



Obrázek 6.3: Celkový dojem z aplikace je pozitivní



Obrázek 6.4: Ovládání aplikace mi přijde intuitivní

Další otázky v dotazníku měly za úkol zjistit přehlednost jednotlivých pohledů kalendáře. Týdenní a měsíční pohled dosahují velmi pozitivních výsledků. Důvodem zřejmě bude to, že vypadají standardně jako v jiných aplikacích. Denní přehled byl nejméně standardní (rozděluje událostí, celodenní události a úkoly do různých částí), avšak i tak ho většina testerů hodnotila spíše pozitivně. Za úspěch považuji i dobré hodnocení o intuitivnosti v případě obrazovek pro přidávání událostí a úkolů. Většina lidí ihned pochopila možnost vybrat jeden z kvadrantů matice plánování a možnost přiřadit úkol k cíli, který si dříve vytvořili. Obrazovka pro přidávání událostí je obdobná jiným kalendářním aplikacím a nenastaly zde

žádné viditelné problémy. Nejhorších výsledků v dotazníku získala otázka, zda správa rolí a cílů přijde uživatelům intuitivní. Část uživatelů nevěděla, proč by měla vůbec nějaké role a cíle v kalendáři vytvářet. Nedošlo tak k úplnému pochopení filosofie. Zajímavým výsledkem je hodnocení užitečnosti nápovědy. Jak jsem již psal dříve, nápovědu buď lidé uvítají nebo je obtěžuje, a proto jsou výsledky dotazníku buď příliš negativní nebo naopak velmi pozitivní. Nápovědu nicméně považuji za užitečnou. Uživatele, kteří ji nevyužijí, zdrží jen na několik vteřin při prvním spuštění. Avšak uživatelé, kteří ji četli, pak kladně odpověděli na otázku, zda pochopili filosofii First-Things-First. Užitečnost statistik hodnotilo většina respondentů kladně, i když přiznávali, že neví, zda by je pravidelně využívali.

Během testů jsem získal užitečnou zpětnou vazbu. Některé nedostatky jsem již do aplikace zapracoval. Do budoucna by bylo potřeba upravit uživatelské rozhraní pro přidávání rolí a cílů, které z testů vyplynulo jako největší překážka. Zřejmě by bylo dobré zapracovat například metodu drag&drop pro přesun položek a celkovou organizaci. Nenutit tak uživatele procházet dialogová okna pro vytváření nových cílů a přiřazovat je k rolím. Další úpravu by měl podstoupit měsíční pohled. I když z hodnocení vyplynulo relativně kladné hodnocení, několika respondentům chybělo zobrazení dlouhotrvajících úkolů přes několik dní, jak je to např. v aplikaci aCalendar (viz 2.1). Do budoucna bude potřeba přidat podporu pro vkládání opakujících se úkolů i událostí. Bylo by zajímavé vytvořit widget s důležitými úkoly pro daný den.

Kapitola 7

Závěr

Na základě analýzy filosofie First-Things-First byla navrhnuta mobilní aplikace pro správu času a následně úspěšně implementována. Aplikace byla prezentována na školní soutěži Excel@FIT 2015, kde postoupila do užšího výběru. Finální aplikaci je možné stáhnout z katalogu Google Play[1]. K projektu jsou vytvořeny i propagační webové stránky www.ftfapp.net, kde je možné získat další dodatečné informace.

Vývoj mobilní aplikace zahrnuje nejen implementaci samotného programu, ale také ladění, testování, distribuci a propagaci. Všechny tyto aspekty jsem se snažil v textu popsat. Za zajímavou část považuji pasáž věnující se vývoji aplikace pro chytré hodinky se systémem Android Wear. Za úspěch považuji i převod filosofie First-Things-First z myšlenek a rad v knize do podoby moderní aplikace, která má za cíl usnadnit lidem život. Uživatelské testování, které bylo součástí práce, potvrdilo, že grafická podoba aplikace je použitelná a pro uživatele zajímavá.

V budoucnu bych rád pokračoval ve vývoji a zdokonalování aplikace. Chtěl bych vytvořit službu, která bude dostupná na většině dnes využívaných platformách – Android, iOS a Windows Phone. Díky dobře navržené databázové struktuře, nezávislé na platformě, nebude problém se synchronizací a rozšiřitelností.

[1] http://play.google.com/store/apps/details?id=net.ftfapp.android

Literatura

[1] Android Developers: Develop Apps [online]. 2015 [cit. 2015-03-12].
URL http://developer.android.com/develop/index.html

[2] Covey, S.: *7 návyků skutečně efektivních lidí*. Management press, 2014,
ISBN 978-80-7261-268-0.

[3] Komatineni, S.; MacLean, D.: *Expert Android*. Apress Media LLC, 2013,
ISBN 978-1-4302-4951-1.

[4] Likert, R.: A technique for the measurement of attitudes. *Archives of Psychology*,
ročník 22, č. 140, 1932: s. 5–43.

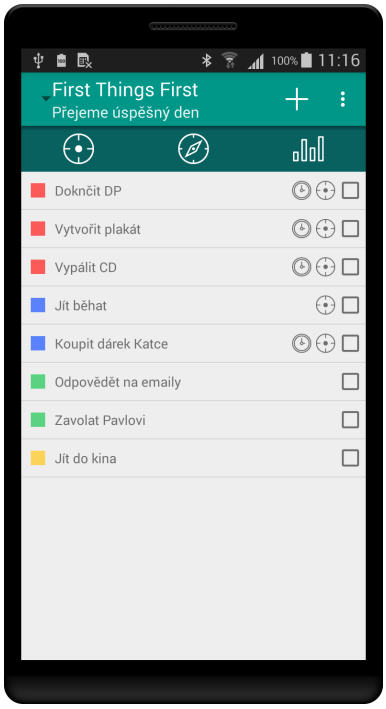
[5] Preece, J.: *Human-Computer Interaction*. Addison-Wesley, 1994, ISBN 0-201-62769-8.

[6] Smith, D.; Friesen, J.: *Android Recipes: A Problem-Solution Approach, 3rd Edition*.
Apress Media LLC, 2014, ISBN 978-1-4302-6322-7.

[7] SQLite Developers: About SQLite [online]. 2015 [cit. 2015-04-20].
URL http://www.sqlite.org/about.html

Příloha A

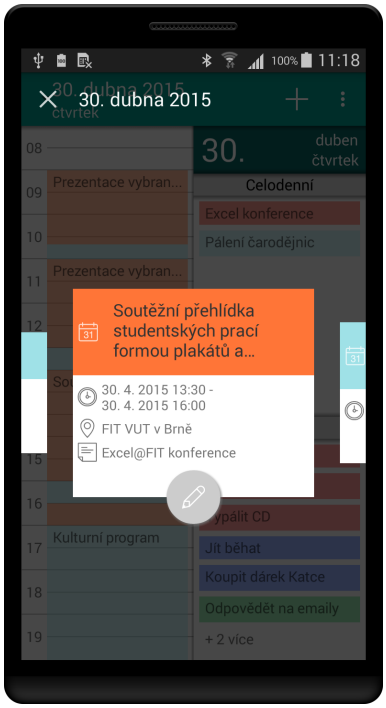
Obrázky z aplikace



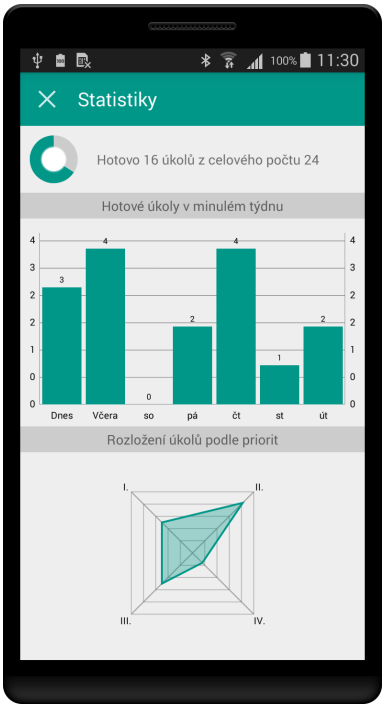
Obrázek A.1: Úvodní obrazovka. Seznam všech úkolů.

Obrázek A.2: Denní pohled kalendáře.

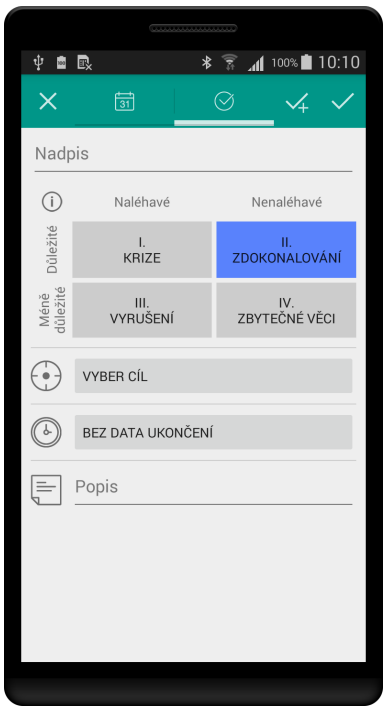
Obrázek A.3: Android Wear verze aplikace. Zobrazuje podrobnosti k dnešním úkolům.



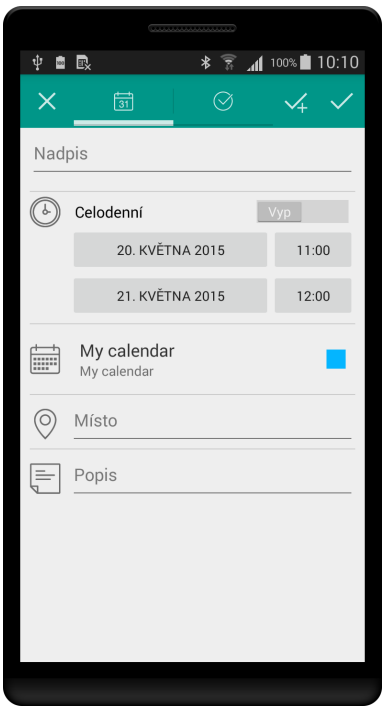
Obrázek A.4: Detail události s možností přecházet na další položky.



Obrázek A.5: Statistiky.

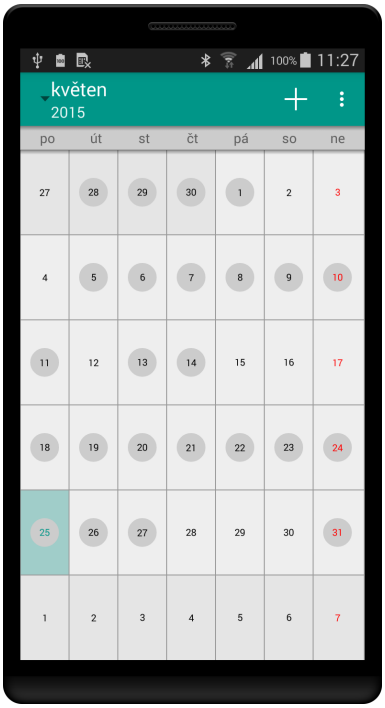


Obrázek A.6: Formulář pro přidání nového úkolu.

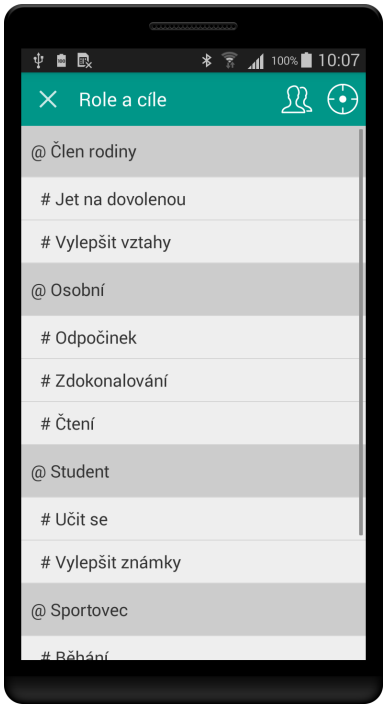


Obrázek A.7: Formulář pro přidání nové události.

Obrázek A.8: Týdenní pohled kalendáře.

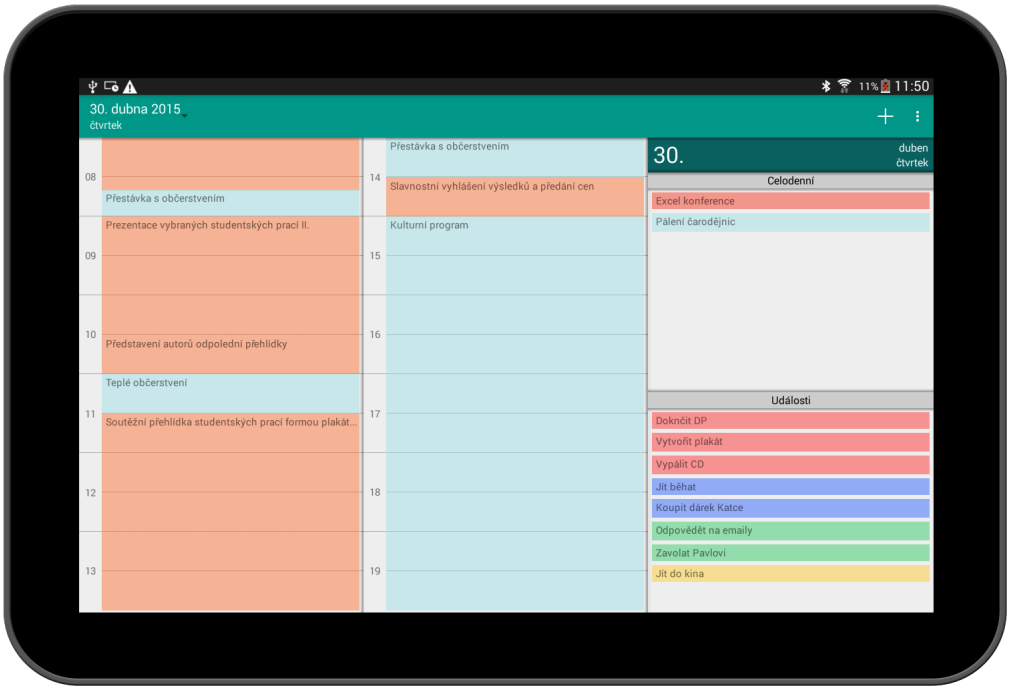


Obrázek A.9: Měsíční pohled kalendáře.

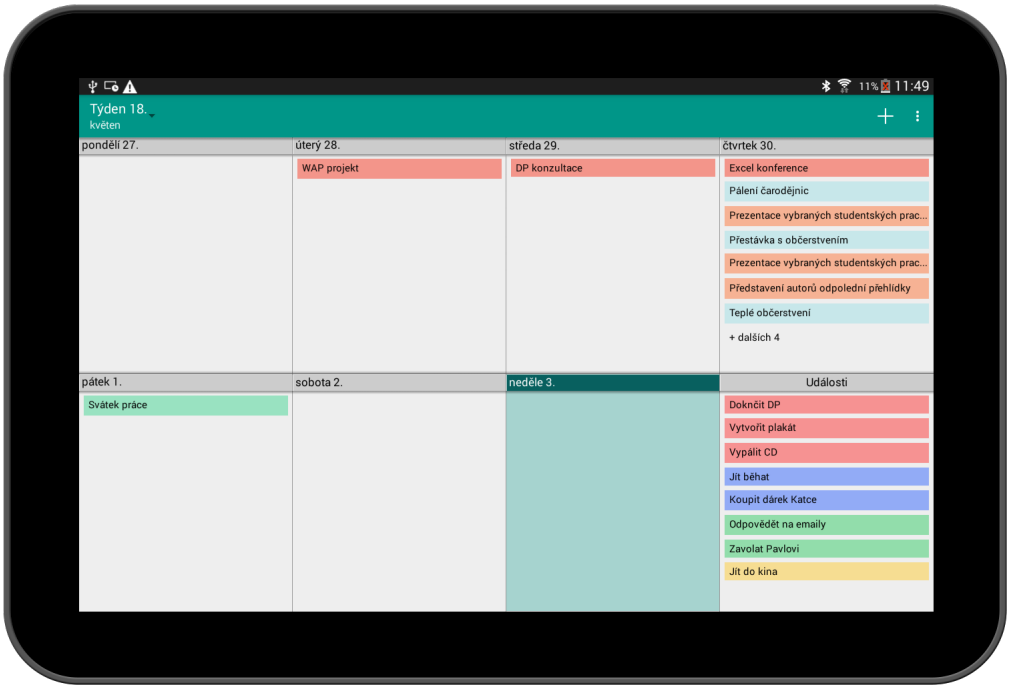


Obrázek A.10: Správa rolí a cílů.

Obrázek A.11: Nápověda po prvním spuštění programu.



Obrázek A.12: Denní pohled uzpůsobený pro tablety

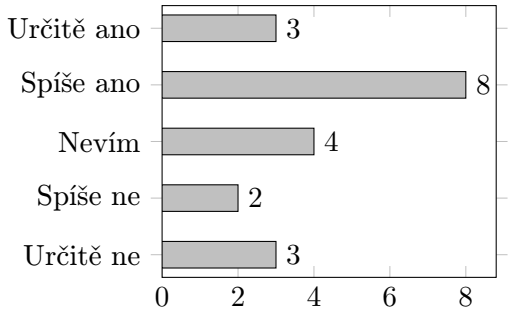


Obrázek A.13: Týdenní pohled uzpůsobený pro tablety

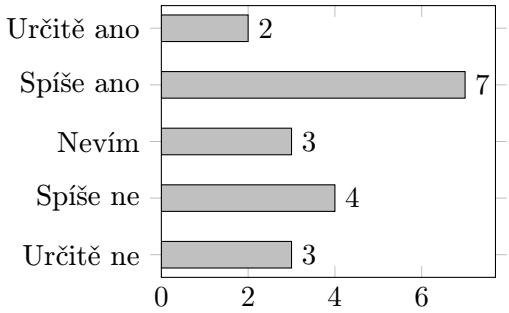
Příloha B

Výsledky experimentu

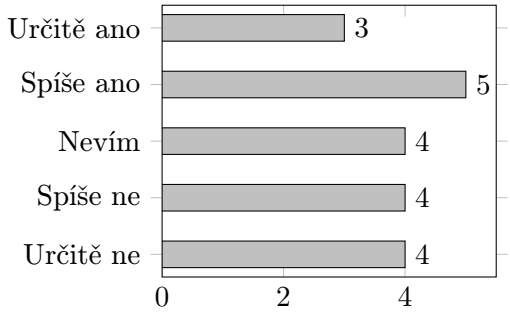
Příloha obsahuje data získaná během uživatelského testování. Experiment je podrobně popsán v podkapitole 6.5, včetně zhodnocení výsledků.



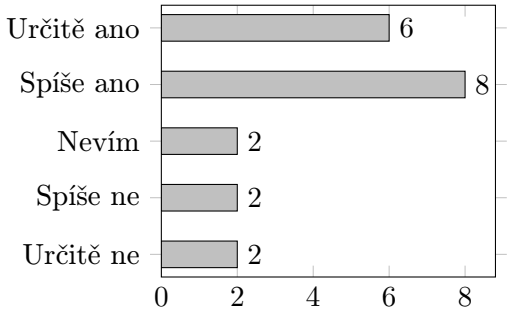
Obrázek B.1: Celkový dojem z aplikace je pozitivní



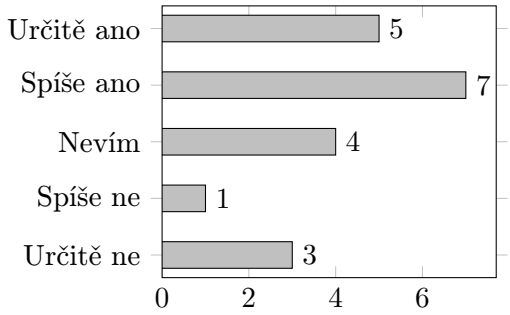
Obrázek B.2: Ovládání aplikace mi přijde intuitivní



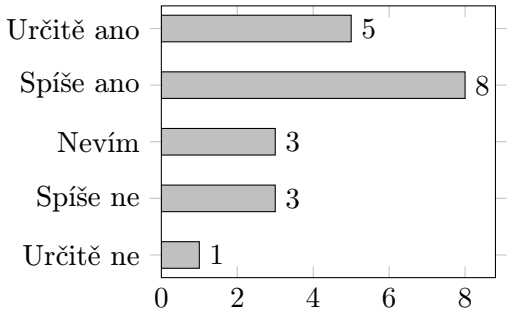
Obrázek B.3: Pochopil jsem koncept filosofie First-Things-First



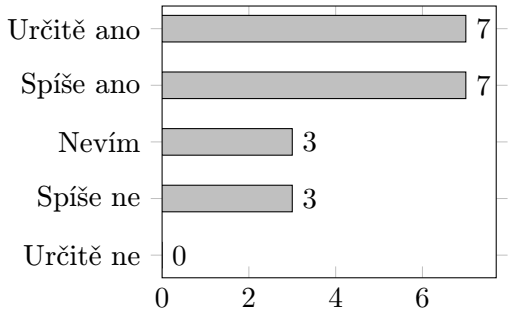
Obrázek B.4: Formulář pro přidávání úkolů mi přijde přehledný



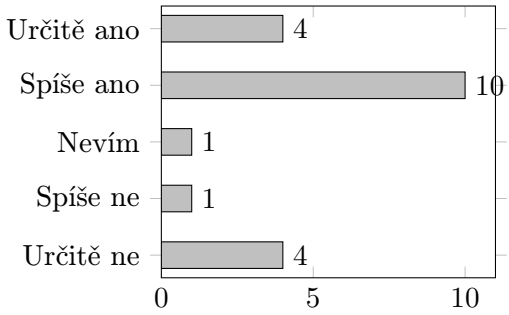
Obrázek B.5: Formulář pro přidávání událostí mi přijde přehledný



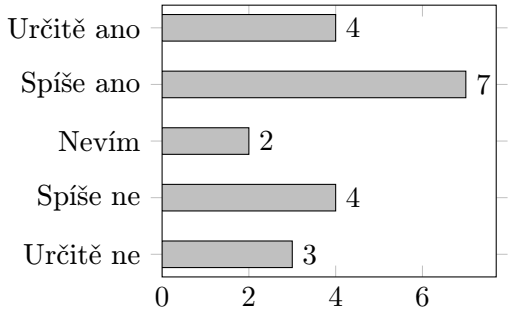
Obrázek B.6: Denní pohled mi přijde přehledný



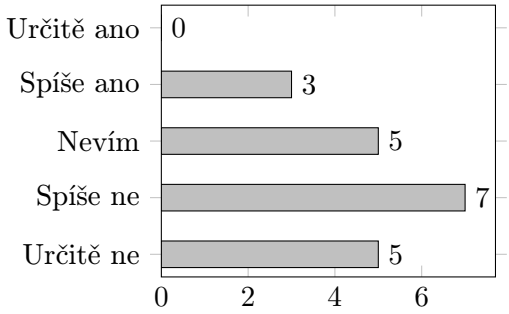
Obrázek B.7: Týdenní pohled mi přijde přehledný



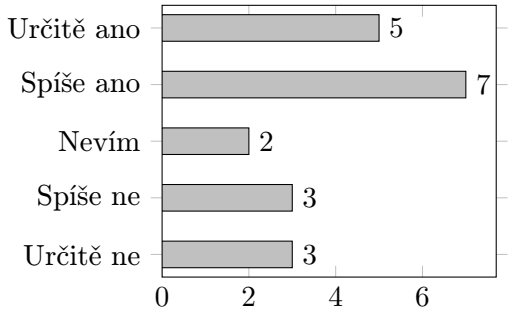
Obrázek B.8: Měsíční pohled mi přijde přehledný



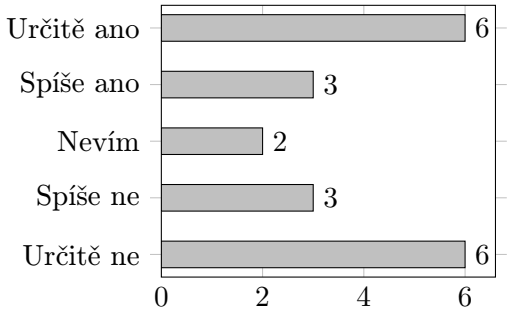
Obrázek B.9: Zobrazování úkolů a událostí mi přijde přehledné



Obrázek B.10: Správa rolí a cílů mi přijde intuitivní



Obrázek B.11: Zobrazované statistiky mi přijdou užitečné



Obrázek B.12: Nápověda se mi zdá užitečná

Příloha C

Obsah CD

- **/android/** – Android Studio projekt
  - **/mobile/** – mobilní verze
  - **/wear/** – Android Wear verze
  - **/FTFAppNet.apk** – spustitelná verze aplikace
- **/thesis/**
  - **/thesis.pdf** – text práce
  - **/poster.pdf** – plakát
  - **/src/** – zdrojové soubory LaTeX
- **/video/**
  - **/video.mp4** – propagační video
  - **/tests/** – záznamy z uživatelského testování