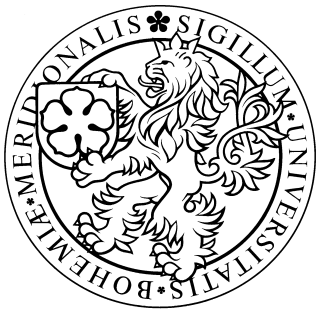
JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH



Mobilní informační banka

Bakalářská práce

Jaroslav Hůna

Školitel: Ing. Václav Novák, CSc.

ČESKÉ BUDĚJOVICE, DUBEN 2013

Bibliografické údaje

Jaroslav Hůna, 2013: Mobilní informační banka

[Mobile Information Bank. Bc. Thesis, in Czech] – 39 pages, Faculty of Science,

The University of South Bohemia, České Budějovice, Czech Republic.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práv k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 25.04.2013

_____

Jaroslav Hůna

Abstrakt

Mobilní informační banka je projekt sjednocující aktuality z celé Jihočeské univerzity, navrhující zároveň i řešení navigace mezi učebnami Jihočeské univerzity. Všechny informace předává přes server klientským mobilním aplikacím platformy Android. Navržené a implementované řešení lze po úpravě aplikovat i na jiných školách.

Klíčová slova

Android, informace, navigace, Jihočeská univerzita

Abstract

Mobile information bank is a project of unifying news generated by the University of South Bohemia, and at the same time proposing solutions of navigation between university classrooms. All information are transmitted via server to client mobile applications build upon Android platform. Designed and implemented solution can be modified and applied to other schools as well.

Keywords

Android, information, navigation, University of South Bohemia

Poděkování

Poděkovat bych chtěl v prvé řadě svému vedoucímu. Za obrovskou trpělivost, kterou se mnou měl a za cenné rady, které mi poskytl.

Z celého srdce děkuji i všem, kteří věřili, že tuto práci dokončím včas. Bez vaší podpory bych tyto řádky nepsal.

Moje díky ale patří i těm, kteří mi nevěřili. Dokázat vám, že se pletete, byla výzva, kterou jsem nemohl odmítnout.

Obsah

1 Úvod

Držíte v rukou bakalářskou práci s nejasným názvem – **Mobilní informační banka.** Hned si tato tři slova vysvětlíme. V textu pak budeme používat převážně jejich zkratku – MIB.

Mobilní je přídavné jméno od *mobility*. Nejbližšími synonymy jsou *pohyblivost* nebo *přemístitelnost*. Používanějším významem pak *přístupnost z chytrých přístrojů* v rukou (pohybujících se) studentů. Tu v našem případě umožní mobilní aplikace svázaná s MIB.

Druhé přídavné jméno – **informační** – odhaluje hlavní aspekt MIB – *informace***.** Běžnými synonymy jsou *vědění* či *znalosti*. Odborněji lze hovořit o údaji, který snižuje nebo odstraňuje neurčitost systému. Tím systémem, který nás zde zajímá, je Jihočeská univerzita.

Důležité je i slovo **banka**. Ve svém původním významu instituce shromažďující dočasně volné peníze, se kterými dále různě nakládá. V naší analogii ale banka shromažďuje volné informace z různých zdrojů a umožňuje k nim snadný a centralizovaný přístup skrze mobilní aplikaci MIB.

1.1 Zadání práce

Rozložení pracovišť na Jihočeské univerzitě je relativně složité a hledání informací o organizaci studia je velmi náročné a to zejména pro zahraniční studenty. Rovněž informace o změnách v rozvrhu a jiných aktualitách, jsou zatím jen na webových stránkách jednotlivých kateder. Student má za úkol vytvořit mobilní informační banku tak, aby se situace zlepšila.

1.2 Cíle práce

1. Navrhnout architekturu mobilní informační banka použitím příslušných diagramů.
2. Naprogramovat aplikaci do mobilních zařízení běžících na OS Android. Systém ponechat otevřený směrem k ostatním užívaným mobilním OS.
3. Doplnit webovou stránku Přírodovědecké fakulty nebo ústavu Aplikované informatiky o možnost stahování této aplikace a její instalaci v mobilních telefonech studentů.
4. Provést příslušné akceptační testy a aplikaci uvést do trvalého provozu v českém a anglickém jazyce.
5. Vše řádně zdokumentovat.

2 Stav

2.1 Distribuce aktualit

2.1.1 Aktuality

Jako aktuality si v tomto kontextu představme chronologicky seřazený seznam informací relevantních pouze v určitém čase. Jsou obsahem většiny webových stránek a právě myšlenka jejich sběru a lepší redistribuce dala vzniknout celému projektu MIB.

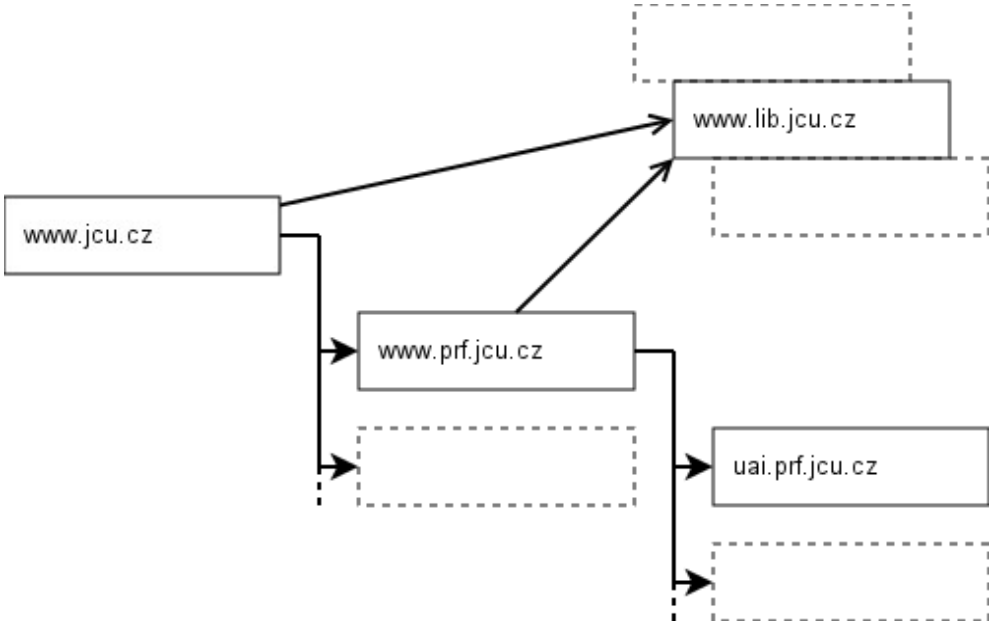
2.1.2 Distribuce

V nejobecnější rovině lze distribuci aktualit rozdělit do dvou skupin. Na tradiční – neelektronickou – z dob před internetem a na moderní – elektronickou – postavenou právě na internetu.

Do první – tradiční – skupiny by spadaly nástěnky, papírová pošta apod. Tyto informace mají nezanedbatelnou hodnotu, ale jejich společný jmenovatel – elektronická nezpracovatelnost – je z projektu MIB diskvalifikuje.

Zato druhá – moderní – kategorie je v tomto ohledu téměř ideální. Obsahuje pouze elektronicky zpracovatelné zdroje informací, jako jsou webové stránky nebo elektronická pošta. Email je přímo vzorem rychlé a cílené distribuce informací a vedle něj je dobře patrná slabina webových stránek – pasivita šíření obsahu. Očekává se, že si čtenář sám přijde nový obsah přečíst, ale co když nepřijde? MIB chce tento nedostatek vykompenzovat transformováním aktuálního obsahu webových stránek do jediného proudu informací, který by čtenář mohl odebírat skrze mobilní aplikaci ve svém telefonu.

V univerzitním kyberprostoru jsou desítky stránek s různou důležitostí. MIB může teoreticky spravovat každou z nich, ale v této práci použijeme jen čtyři. Budou to weby Jihočeské univerzity (www.jcu.cz), Přírodovědecké fakulty (www.prf.jcu.cz), Akademické knihovny (www.lib.jcu.cz) a jednoho z ústavů, konkrétně Ústav aplikované informatiky (uai.prf.jcu.cz). Byly pečlivě vybrány tak, aby co nejlépe prověřili celý princip MIB.



Ilustrace 1: Vybraná část univerzitního kyberprostoru

2.2 Hledání učeben

Úplně jiným příběhem je hledání učeben a celkově orientace ve školních prostorách. Fakulty tento problém řeší po svém. Nabízí studentům pomoc, v lepším případě formou mapek, jindy alespoň napoví slovní popis, ale někdy studentům nezbývá než zamířit s adresou na internetové mapy.

Jihočeská univerzita a nejen ona, prakticky každá univerzita, je „městem“ ve městě. Nebo se to tak alespoň může jevit studentům uvnitř. Zkusme se této analogie chvíli držet. Skutečné město má své budovy a ulice zakresleny v mapách. Budovy univerzity tedy najdeme snadno, ale představme si, že každá budova je městem, chodby ulicemi a učebny budovami. Pak teprve začne dávat smysl tvrzení, že univerzita je „městem“ ve městě. Pointou je, že mapy tohoto „města“ uvnitř města neexistují. A přitom by je zejména noví studenti určitě ocenili.

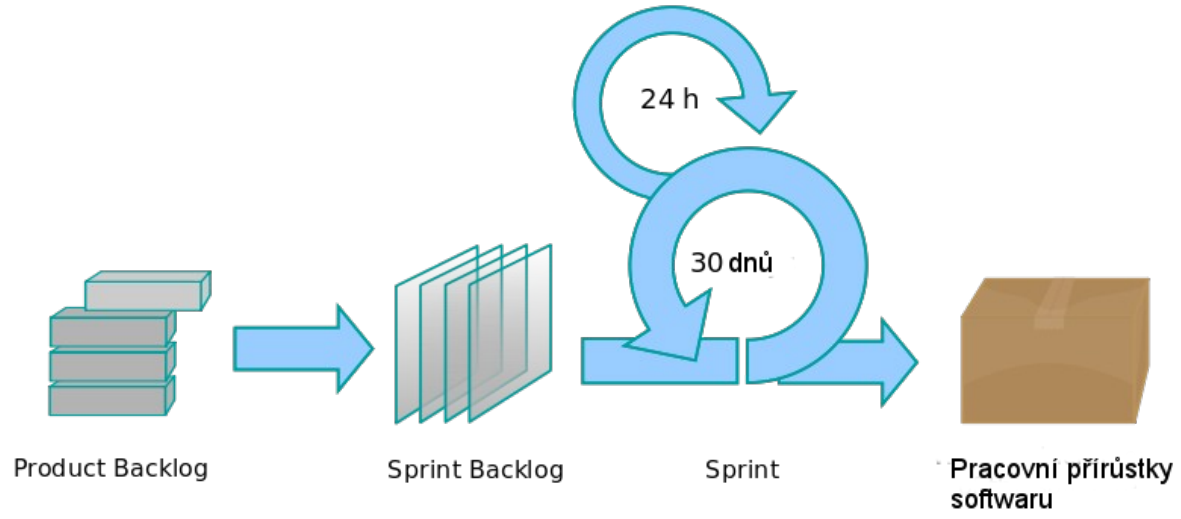
Je to prostor pro zlepšení, které je vskutku úkolem MIB, jak si mohl pozorný čtenář všimnout v zadání. Na rozdíl od aktualit, kde MIB obsah transformuje, zde se bude muset stát přímo jeho zdrojem. Protože jiný zdroj map, základního kamene navigace, není. Ano, existují různé indicie, od seznamu místností a budov až k nákresům a textovým popisům, ale chybí jim jednotnost. MIB proto musí nejprve takový jednotný informační model navrhnout a až poté ho implementovat.

3 Technologie

3.1 SCRUM

Na začátku neměl vývoj SW jasná pravidla. Byl velice nákladný a výsledný produkt nemíval odpovídající kvalitu, což vedlo k dalšímu prodražování. V sedmdesátých letech proto vznikl tzv. Vodopádový model, který zachycuje sekvenci fází vývoje – analýza, návrh, implementace, validace, údržba. Na jeho principu byla postavena řada metodik pro vývoj SW, dnes označovaných jako tradiční metodiky. Další rozvoj přinesl cykly a jiná vylepšení, až v roce 2001 vznikl tzv. Agilní manifest. Tehdejší způsob vývoje SW, připadal tvůrcům agilního manifestu zastaralý a pro řadu projektů naprosto nevhodný. Navrhli proto principy, ze kterých vzešel nový druh metodik, podle manifestu pojmenovaný – agilní metodiky.

Jednou z nejpoužívanějších agilních metodik je i SCRUM.



Ilustrace 2: Metodika SCRUM

Ve SCRUM jsou vlastnosti SW produktu psány z pohledu uživatele – říká se jim User Stories – a všechny jsou evidovány v seznamu zvaném Product Backlog. Ten funguje podobně jako seznam přání. Můžeme ho doplňovat i aktualizovat. Před začátkem vývoje musíme však z Product Baclogu vybrat jen ty User Stroies, které chceme zahrnou do chystaného vydání. Stanovit pro každý User Story prioritu a odhadnout dobu vývoje. Vzniká tak tzv. Release Backlog, který se již nemění.

#	NÁZEV	POPIS	PODMÍNKY SPLNĚNÍ
3	Zobrazení detailu aktuality	Umožní uživateli přechod na stránku, která je zdrojem aktuality.	**- Zobrazení seznamu aktualit** - Klient je po kliknutí na vybranou aktualitu ze seznamu místností přesměrován na zdroj

Tabulka 1: Příklad User Story z Product Backlogu (Celý Product Backlog je součástí přílohy)

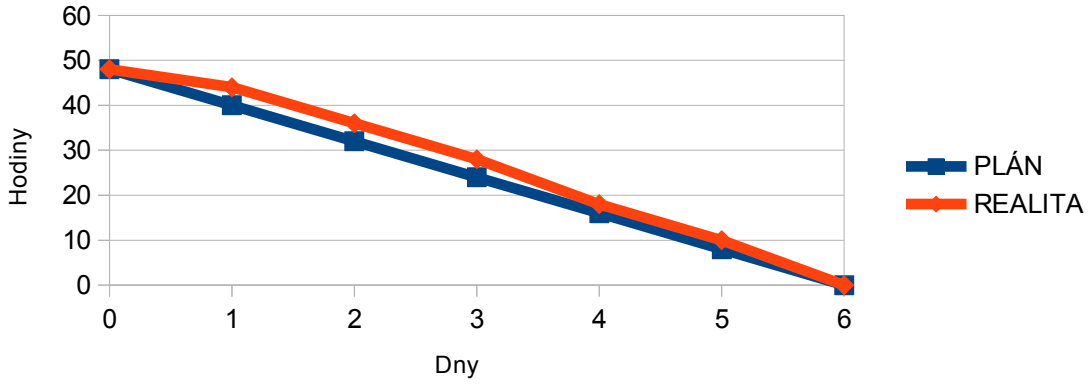
#	NÁZEV	POPIS	PODMÍNKY SPLNĚNÍ	PRIORITA	ESTIMACE
3	Zobrazení detailu aktuality	Umožní uživateli přechod na stránku, která je zdrojem aktuality.	**- Zobrazení seznamu aktualit** - Klient je po kliknutí na vybranou aktualitu ze seznamu místností přesměrován na zdroj	2	8 hodin

Tabulka 2: Příklad User Story z Release Backlogu prvního vydání (Více viz kapitola 4.2 První vydání)

MIB používá prioritu číselnou, kde 1 je priorita nejvyšší. Časová estimace se udává v hodinách, dnech nebo i měsících a pro zvýšení přesnosti se relativizuje – vytvářejí se skupiny podobně dlouho trvajících User Stories.

Samotný vývoj vydání pak probíhá ve stejně dlouhých cyklech, nebo-li iteracích, kterým se ve SCRUM říká sprinty. Každý sprint má vlastní Sprint Backlog, naplněný vybranými, na úlohy rozloženými, User Stories z Release Backlogu. Úloha už je brána z pohledu vývojáře a stanovuje konkrétní kroky. Také u úkolů se odhaduje doba jejich vývoje, priority zůstávají shodné s User Story. Sprint končí splněním všech úloh. Po skončení všech sprintů je hotové i celé vydání a mohou následovat další, až do ukončení podpory celého produktu.

Postup a rychlost vývoje lze ve SCRUM sledovat pomocí speciálního nástroje – tzv. Burn Down Chartu. To je graf zachycujícího úbytek práce v čase – tempo – ve srovnání s plánem.



Ilustrace 3: Burn Down Chart

3.2 PHP

PHP je otevřeně vyvíjený (open source) skriptovací jazyk určený zejména pro dynamickou úpravu obsahu internetových stránek a tvorbu webových aplikací. V této oblasti se těší velké oblibě. Dokáže: vyhodnocovat data z formulářů, vytvářet vlastní webový obsah, komunikovat s databází i zasílat a přijímat cookies. Skripty se vykonávají na straně serveru a uživatel ve svém prohlížeči vidí pouze výsledky jejich činnosti.

3.2.1 PHP Simple HTML DOM Parser

Pod tímto označením se ukrývá šikovná PHP knihovna určená k analýze (parsování) HTML kódu. S její pomocí je vyjmutí jeho specifické části velmi snadné. Po načtení celého HTML souboru do jedné proměnné stačí k definování hledaného sektoru jediný příkaz kombinovaný s cyklem (foreach). V těle cyklu pak postupně dostáváme všechny odpovídající oddíly.

```
include('simple_html_dom.php');
$dates = array();
$html = file_get_html('http://www.prf.jcu.cz');
foreach($html->find('ul#newsy') as $news){
     foreach($news->find('div.datumNews') as $d){
          $date = iconv('windows-1250','UTF-8',$d->innertext);
          $date = str_replace('. ','-',$date);
          array_push($dates,$date);
     }
}
$html->clear();
```

Pomocí dalších příkazů ze získaného výstřižku extrahujeme požadovanou část. Metoda innertext() ji například omezuje na samotný obsah ležící mezi HTML tagy. Text je pak možné dále modifikovat a ukládat, jak vidíme v ukázce. Kvůli vlastnostem PHP je zvláště důležité vytvořený DOM objekt na konci vymazat voláním metody clear();

3.3 RSS

RSS (Really Simple Syndication) je webový standard pro publikaci a čtení novinek nebo článků. Má pevnou, přesně definovanou strukturu, postavenou na dialektu standardního a otevřeného formátu/jazyka pro výměnu informací – XML.

Weby, publikující svůj obsah skrze RSS, tím dávají čtenářům možnost, přihlásit se k jeho odběru. Výhodou je, že si čtenář ve své RSS čtečce může takových odběrů přihlásit více, a mít tak všechny články a novinky na jednom místě.

Mezi koncepcí RSS a MIB můžeme vidět velkou podobnost. MIB proto místo vymýšlení vlastního řešení, které by se zde rovnalo vymýšlení kola, využije právě standardu RSS.

3.4 Java

Java je všestranný programovací jazyk. Přednostmi Javy jsou: Přenositelnost napříč operačními systémy, přes zařízení různých typů až po odlišné architektury procesorů. Otevřený (open source) vývoj, který umožňuje přizpůsobení Javy vlastním potřebám nebo i vlastní přispívání. A výhodou je i obrovská popularita Javy, ke které přispěl i Android, když si Javu zvolil za hlavní programovací jazyk pro své aplikace. Využil přitom právě její otevřenost.

3.5 Android

Android je otevřená (open source licence) mobilní platforma. Slovo platforma zahrnuje operační systém, middleware (propojovací prvek), uživatelské rozhraní a aplikace.

Jádrem OS Android je Linux (mimochodem, další známý open source projekt). Můžeme tedy o OS Android říci, že je velmi specializovanou linuxovou distribucí. Jedním z praktických dopadů využití linuxového jádra je i bezpečná správa uživatelů. Obyčejně jsou jimi lidé, ale v OS Android je uživatelem i každá aplikace. To znamená, že k využití systémových prostředků (např. přístup k internetu, kontaktům apod.) nebo společných dat (např. SD karta) potřebují aplikace oprávnění, která mohou získat jedině od uživatele během své instalace. Každá aplikace navíc běží v odděleném procesu, ve vlastním virtuálním stroji, kde neohrožuje systém ani ostatní spuštěné aplikace.

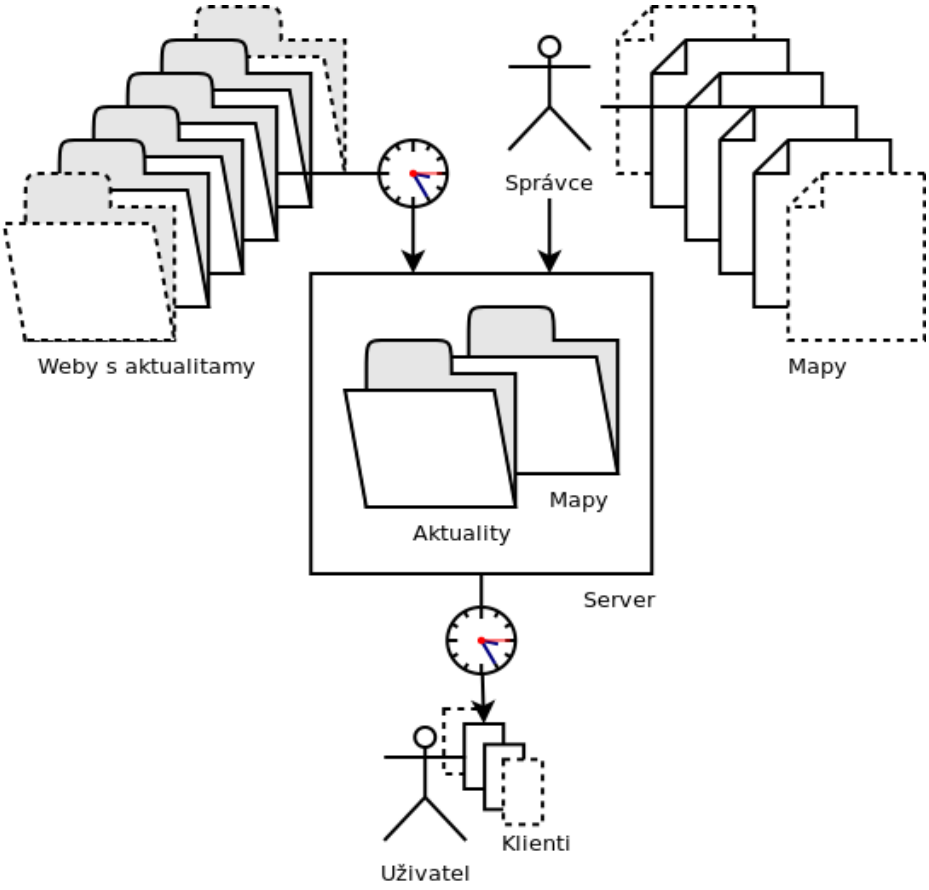
Pro jejich vývojáře je k dispozici rozsáhlá dokumentace na developer.android.com. Základem je jazyk Java, značně rozšířený o speciální knihovny platformy Android.

4 Návrh

Metodika SCRUM nemá tradiční jednu fázi návrhu, po které by následoval celý vývoj. Podle SCRUM je návrh součástí každého sprintu, kde řeší aktuálně implementované funkce. Ovšem i u SCRUM musí nejprve existovat jasná představa o celém systému, ze které mohou návrhy ve sprintech vycházet a určovat konkrétní vlastnosti. Ta představa je zachycena pomocí vysoce abstraktního modelu obsahujícího jen základní elementy systému a jejich vazby.

4.1 Model

Model je sestavován podle Product Backlogu. Ze zde definovaných User Stories je zřejmé, že MIB má dvě hlavní části. Aktuality na straně jedné a mapy na straně druhé. Úkolem MIB bude nejen načíst a sjednotit aktuality, ale i umožnit správci vkládání nových map, tak jak budou postupně vznikat. Vhodným proto bude model klient-server.



Ilustrace 4: Abstraktní model MIB

Server bude aktuality z každé webové stránky transformovat pomocí PHP skriptů do nového RSS kanálu. Zároveň bude server obsahovat seznam těchto zdrojů, který si aplikace vždy před načítáním RSS zdrojů aktualizuje.

Výhodou tohoto řešení je možnost správy skriptů bez zásahu do mobilní aplikace. Můžeme kdykoli přidat nový nebo opravit některý ze stávajících zdrojů a klient se o změnách dozví při nejbližší aktualizaci. Pokud bychom to samé chtěli provést v aplikaci, museli bychom vydat novou verzi, kterou by si musel uživatel znovu stáhnout a nainstalovat.

Server bude současně zdrojem navigačních dat, která na něj vloží správce. Budou to zejména mapy, ale i XML seznam místností, který klientu řekne, jaké místnosti existují, kde najde jejich mapy a další podrobnosti.

Výhodou je opět možnost jednotné správy obsahu, bez zásahu do mobilní aplikace. Klient postřehne změnu při nejbližší aktualizaci dat.

4.2 První vydání

V prvním vydání navrhujeme podrobně jen ty části aplikace, které chceme v prvním vydání implementovat (viz stejnojmenná podkapitola 5.1, kapitoly 5 Implementace), ale snažíme se přitom postupovat tak, aby do našeho návrhu zapadali i požadavky příštích vydání.

4.2.1 Sprint 1-1

4.2.1.1 Struktura XML seznamu zdrojů

U zdroje **<source>** potřebujeme znát jeho jméno **<title>** a adresu **<link>**, proto byla pro XML dokument vytvořena následující struktura:

```
<xml>
     <source>
          <title>Jihočeská univerzita</title>
          <link>http://www.jcu.cz/news/aggregator/RSS</link>
     </source>
</xml>
```

4.2.1.2 Databáze

Pro uložení načítaných dat – seznamu zdrojů, všech aktualit, a jak víme, v příštím vydání i seznamu místností – nám postačí tři nezávislé tabulky. Na první pohled se tu nabízí spojení mezi tabulkami **NEWS** a **SOURCES**, ale tabulka **SOURCES** se může v čase měnit, a odkaz v tabulce **NEWS** zastarat. Toto by mohli vyřešit další tabulky a chytřejší algoritmy ukládání,

ale pro zachování jednoduchosti, a protože jde pouze o dva atributy, bylo rozhodnuto takto. (Ze stejného důvodu byla zamítnuta i vlastní tabulka budov.)

ROOMS
* _id (klíč) ° building * title * floorplan ° address ° floor

SOURCES
* _id (klíč) * title * link

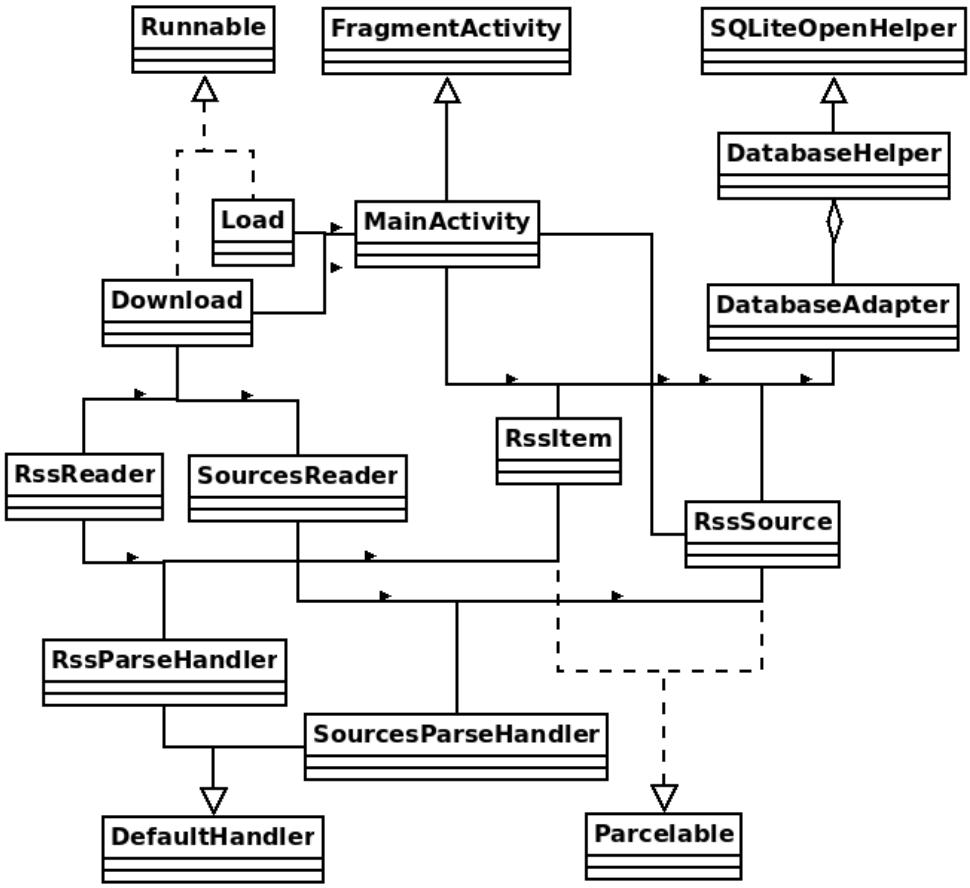
NEWS
* _id (klíč) * title * link ° description ° content * date ° author ° source_title ° source_link

Tabulka 3: Tabulky klientské databáze

4.2.1.3 Diagram tříd

V prvním sprintu začínáme pracovat na aplikaci, a potřebujeme proto navrhnout základní diagram tříd. Obsahuje tyto třídy:

- **MainActivity**, rozšiřující FragmentActivity pro zobrazení obsahu
- Dvě třídy implementující Runnable – **Load** a **Download** pro paralelní načítání dat bez blokování hlavního vlákna s uživatelským rozhraním. (V tomto bodě byl návrh přepracován, když se původní řešení se speciální třídou AsyncTask ukázalo v testech jako nevhodné.)
- **DatabaseAdapter** (obsahující **DatabaseHelper** rozšiřující standardní SQLiteOpenHelper) pro práci s databází.
- Třídy reprentující objekty **RssItem** a **RssSource** s implementací statických metod pro jejich ukládání a načítání a implementijící rozhraní Parcelable pro jednodušší předávání těchto objektů uvnitř aplikace.
- Pro metody třídy Download nezbytné sesterské třídy **RssReader** a **SourcesReader** určené k získávání obsahu ze serveru, respektive RSS zdrojů.
- **RssParseHandler** a **SourcesParseHandler,** poskytující RssReaderu a SourcesReaderu data převedená z XML na seznamy objektů typu RssItem a RssSource.



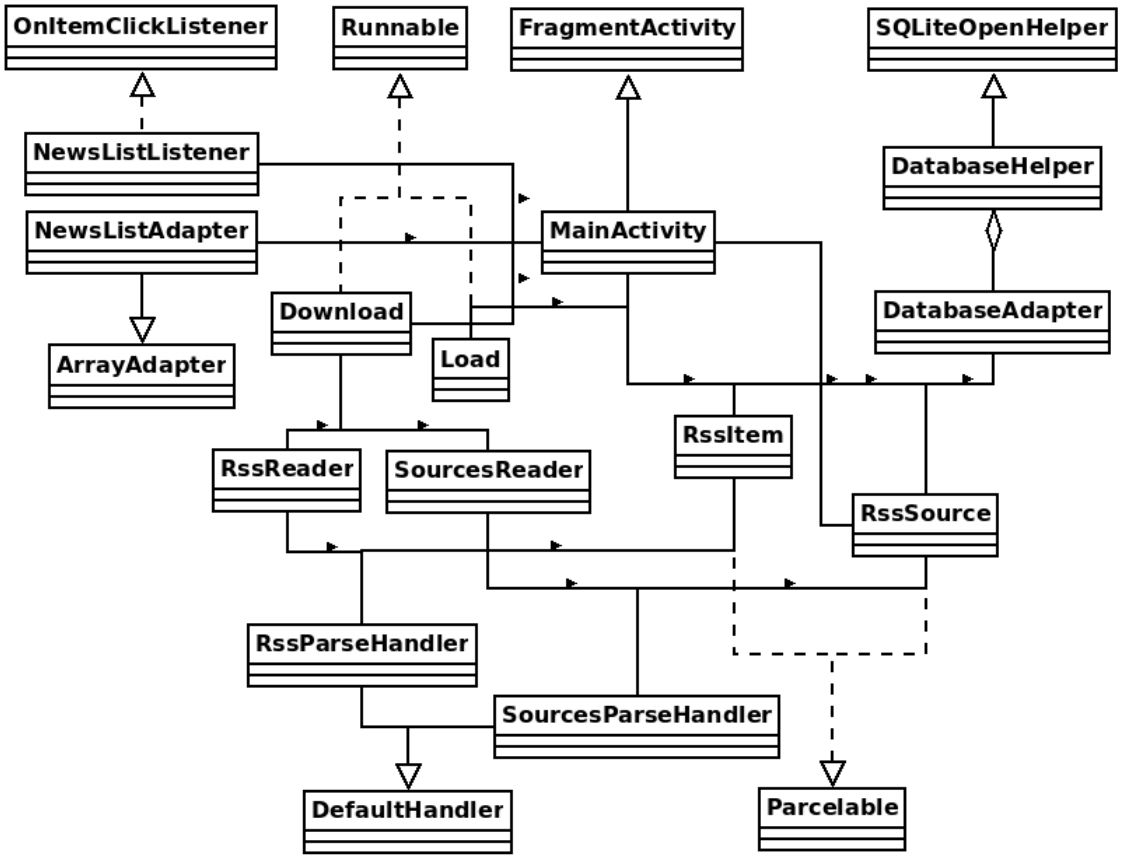
Ilustrace 5: Diagram tříd – 1. vydání – 1. sprint

4.2.2 Sprint 1-2

4.2.2.1 Diagram tříd

Ve druhém sprintu rozšíříme náš diagram o další dvě třídy:

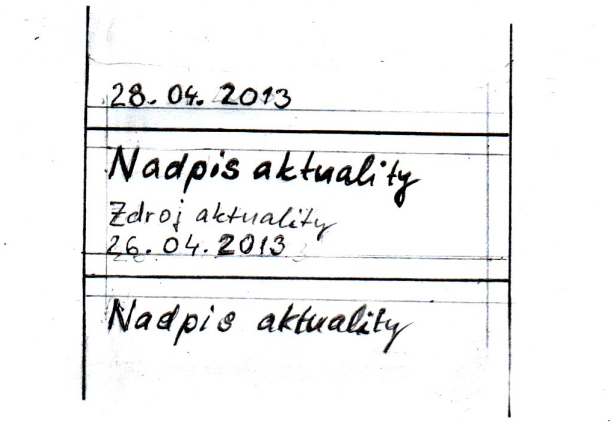
- **NewsListAdapter**, rozšiřující ArrayAdapter, pro naplnění seznamu aktualit.
- **NewsListListener**, implementující OnItemClickListener, pro zpracování události při kliknutí na položku v seznamu aktualit.



Ilustrace 6: Diagram tříd – 1. vydání – 2. sprint

4.2.2.2 Zobrazení aktualit

V tomto sprintu máme zobrazit seznam aktualit, který bude nakonec součástí finální aplikace. Tím se poprvé dostáváme k návrhu uživatelského rozhraní. Využijeme zde standardní kontejner pro seznam – ListView, ale pro jeho položky si navrhneme vlastní rozložení (layout):



Ilustrace 7: Rozložení položky v seznamu aktualit

4.3 Druhé vydání

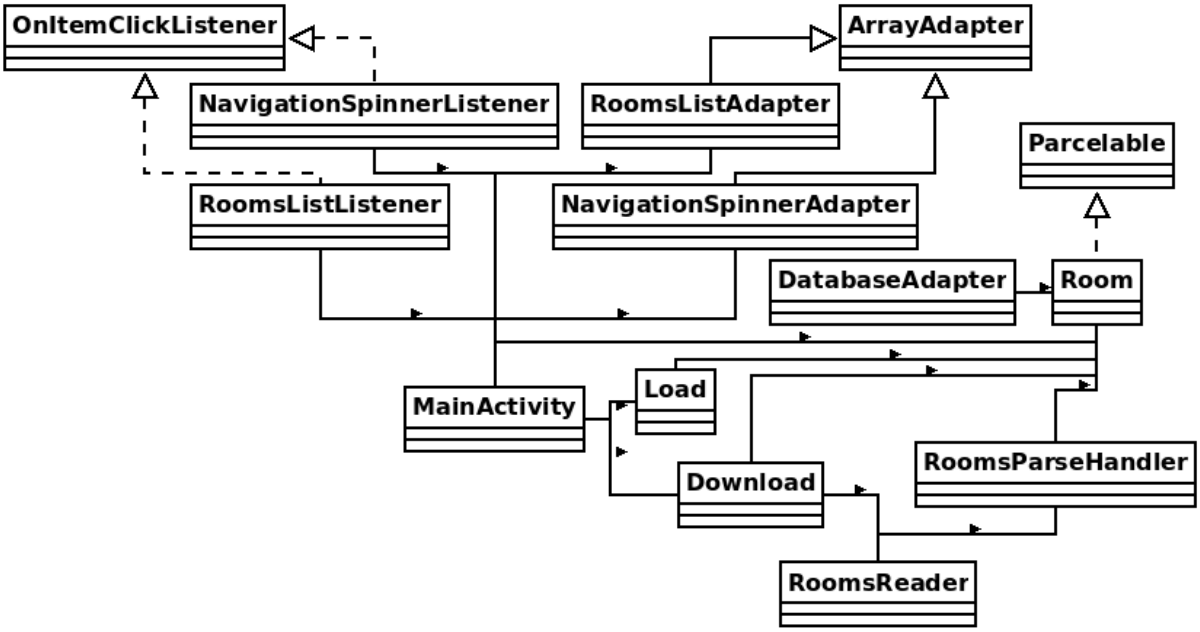
I implementaci vlastností druhého vydání MIB (viz kapitola 5.2) musí v každém sprintu předcházet podrobný návrh.

4.3.1 Sprint 2-1

4.3.1.1 Diagram tříd

Abychom si ušetřili práci, přidáme při tvorbě diagramu v tomto sprintu hned všechny třídy figurující ve druhém vydání. Pro přehlednost tentokrát z diagramu vynecháme třídy, které nemají žádnou vazbu na ty nové:

- Třída reprezentující objekt **Room** s implementací statických metod pro jeho ukládání a načítání, implementující rozhraní Parcelable pro jednodušší předávání tohoto objektu uvnitř aplikace.
- **RoomsReader** pro získání XML souboru s místnostmi ze serveru.
- **RoomsParseHandler** nezbytný pro převedení XML na seznam objektů typu Room.
- **RoomsListAdapter**, rozšiřující ArrayAdapter, pro naplnění seznamu místností.
- **RoomsListListener**, implementující OnItemClickListener, pro zpracování události při kliknutí na položku v seznamu místností.
- **NavigationSpinnerAdapter**, rozšiřující ArrayAdapter, pro naplnění rozbalovací navigace v horní liště aplikace.
- **NavigationSpinnerListener**, implementující OnItemClickListener, pro zpracování události při kliknutí na položku v rozbalovací navigaci.



Ilustrace 8: Diagram tříd – 2. vydání

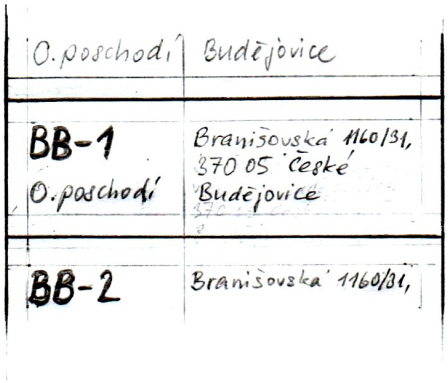
4.3.1.2 Struktura XML seznamu místností

Místnosti **<room>** mají dlouhou řadu parametrů, ale pro MIB jsou důležité jen ty navigační: jméno místnosti **<title>,** budova **<building>,** odkaz na plán **<floorplan>**, poschodí **<floor>** a adresa **<address>**. Z toho plyne navržená struktura XML dokumentu:

```
<xml>
     <room>
          <title>1</title>
          <building>BB</building>
          <floorplan>
          http://binolupa.napotom.com/maps/BB-1.png
          </floorplan>
          <floor>0</floor>
          <address>
          Branišovská 1160/31, 370 05 České Budějovice
          </address>
     </room>
</xml>
```

4.3.1.3 Zobrazení místností

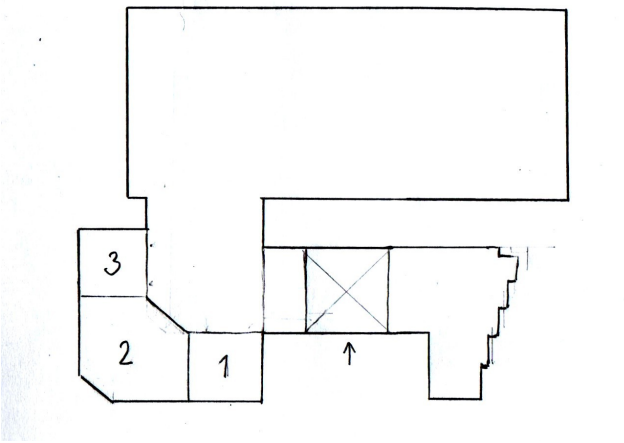
Stejně jako u zobrazení aktuality, i zde narážíme na návrh rozhraní. Seznam místností využije stejný kontejner ListView, jako aktuality, ale také potřebuje vlastní rozložení (layout) položky:



Ilustrace 9: Rozložení položky ze seznamu místností

4.3.1.4 Mapy místností

Mapy jsou klíčovou komponentou celé navigace. Mají znázorňovat polohu místnosti v budově. Zatím neexistují a není ani cílem MIB je vytvářet, ale přesto byla pro ukázku jedna mapa nakreslena:



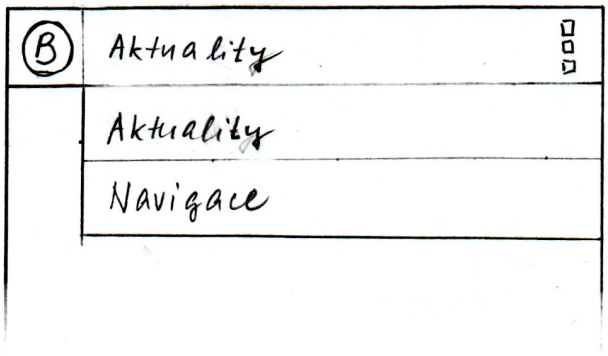
Ilustrace 10: Ukázková mapa budovy BB

Co ale MIB může udělat, je stanovit určitý standard pro budoucí tvůrce. Ten bude velmi jednoduchý – mapa musí být v jakékoli podobě dostupná kdekoli na internetu a odkaz na ní poté přidán správcem MIB, spolu s příslušným popisem, do XML seznamu místností (viz 4.3.1.2 Struktura XML seznamu místností). Aplikace pak bude moci mapu ihned využívat. Její zobrazení se bude realizovat prostým otevřením daného odkazu ve webovém prohlížeči.

4.3.2 Sprint 2-2

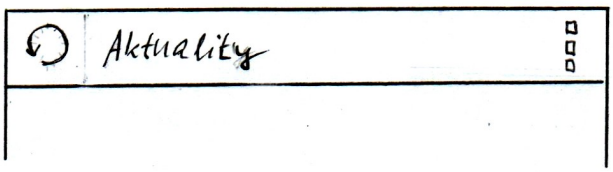
4.3.2.1 Vzhled a funkce horní lišty

Po navržení obou seznamů s obsahem, přichází na řadu i horní lišta aplikace. Má umožnit přecházení mezi aktualitami, navigací a nebránit ani možnosti rozšíření MIB o další sekce. Například o jídelníčky Menz nebo o akademický kalendář. Proto bylo navrženo z lišty se rozbalující navigační menu:



Ilustrace 11: Horní lišta aplikace - navigace

V levém rohu bude ikona aplikace, která se při aktualizaci změní na kroužící ukazatel probíhajícího procesu (ProgressBar). A ikona v pravém rohu informuje uživatele o samotné přítomnosti rozbalovacího menu. Tento návrh tak plní zadané cíle a úspěšně prošel i uživatelským testováním.



Ilustrace 12: Horní lišta - aktualizace

Druhé menu bude mít aplikace standardní, spouštěné systémovým tlačítkem „menu“. Zobrazí se na dolním okraji obrazovky a obsahovat bude položky aktualizace a nastavení. (Implementace nastavení není úkolem tohoto vydání, a proto položka nastavení nebude prozatím fungovat.)

5 Implementace

Implementace podle metodiky SCRUM nemá tradiční jednu fázi. SCRUM dělí vývoj společně s návrhem na jednotlivá vydání a tzv. sprinty. Toto svou strukturou již zohledňuje kapitola 4 Návrh a stejné dělení bylo zachováno i v této kapitole.

5.1 První vydání

V prvním vydání řešíme pouze aktuality. Navigaci si necháváme do vydání druhého. Release Backlog prvního vydání je proto na estimacích a prioritách založený výběr, pro aktuality nepostradatelných User Stories, z Product Backlogu:

#	NÁZEV	POPIS	PODMÍNKY SPLNĚNÍ	PRIORITA	ESTIMACE
1	Zobrazení seznamu aktualit	Umožní uživateli procházet seznam aktualit.	- Server převádí aktuality na RSS - Server obsahuje XML seznam RSS zdrojů - Klient načítá XML seznam zdrojů do DB - Klient načítá aktuality z RSS zdrojů do DB - Klient zobrazuje seznam aktualit z DB	1	5 dnů
3	Zobrazení detailu aktuality	Umožní uživateli přechod na stránku, která je zdrojem aktuality.	**- Zobrazení seznamu aktualit** - Uživatel je, po kliknutí na vybranou aktualitu v seznamu, přesměrován na zdroj	2	8 hodin

Tabulka 4: Release Backlog – 1. vydání

Odhadovaná doba vývoje je šest dnů. Vydání proto rozdělíme do dvou sprintů po třech dnech. Ve sprintech se už nepracuje s User Stories, ale s úkoly, které vznikly jejich transformací. Pro plánování a sledování postupu se přidávají i nové časové estimace, a tak seznam úkolů prvního vydání vypadá následovně:

- Server: Převod www.prf.jcu.cz na RSS (4 hodiny)
- Server: Převod www.lib.jcu.cz na RSS (4 hodiny)
- Server: XML seznam zdrojů (2 hodiny)
- Klient: Načítání XML seznamu RSS zdrojů (4 hodiny)
- Klient: Ukládání XML seznamu RSS zdrojů (4 hodiny)
- Klient: Načítání aktualit z RSS zdrojů (4 hodiny)

- Klient: Ukládání aktualit z RSS zdrojů (4 hodiny)
- Klient: Převody časů (4 hodiny)
- Klient: Načítání zdrojů z databáze (2 hodiny)
- Klient: Načítání aktualit z databáze (2 hodiny)
- Klient: Zobrazení seznamu aktualit (4 hodiny)
- Klient: Přesměrování na web s aktualitou (2 hodiny)

5.1.1 Sprint 1-1

První sprint má trvat přibližně tři dny. Vybereme tedy nejdůležitější úkoly tak, aby tento čas (24 hodin) co nejlépe vyplnily a pustíme se do jejich plnění:

- Server: XML seznam zdrojů (2 hodiny)
- Klient: Načítání XML seznamu RSS zdrojů (4 hodiny)
- Klient: Ukládání XML seznamu RSS zdrojů (4 hodiny)
- Klient: Načítání aktualit z RSS zdrojů (4 hodiny)
- Klient: Ukládání aktualit z RSS zdrojů (4 hodiny)
- Klient: Načítání zdrojů z databáze (2 hodiny)
- Klient: Načítání aktualit z databáze (2 hodiny)

Byl zprovozněn webový lokální Apache server a vytvořen seznam zdrojů, v této fázi obsahující pouze zdroje stránek, které již RSS používají – Jihočeské univerzity a Ústavu aplikované informatiky. (Později se server přesunul na adresu binolupa.napotom.com.)

Ve vývojovém prostředí Eclipse vznikl nový projekt pro Android, a započala tak dlouhá cesta k vytvoření mobilní aplikace. Jako první přibyl v manifestu projektu řádek povolující aplikaci přístup k internetu:

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

Klíčová třída *DatabaseAdapter*, obsluhující databázi, dostala metody pro ukládání, načítání a mazání obsahu tabulek (viz také kapitola 4.2.1.2 Databáze).

Třídám reprezentujícím objekty *RssSource* a *RssItem* byly doplněny parametry společně s příslušnými metodami pro nastavení a získání jejich hodnot (set/get). *RssSource* a *RssItem* se také staly místem implementace statických metod pro operace s těmito objekty a databází. Výsledná implementace se dá využít například takto:

```
RssItem item = new RssItem(context);
item.setTitle("Titulek aktuality");
item.setId(RssItem.putOneToDatabase(item));
```

Těchto nových metod mohou pak využít další vytvořené třídy – *RssReader* doplňovaný třídou *RssParseHandler* a *SourcesReader* dopněný třídou *SourcesParseHandler*. Třídy s přívlastkem Reader využívají vestavěný objekt typu SAXParser ze SAXParseFactory a již zmíněné pomocné handlery k získání obsahu webových RSS a XML dokumentů.

Třídě *MainActivity* pak teoreticky stačí jednoduché volání:

```
List<RssItem> parsedItems;
RssReader rssReader = new RssReader(context, url);
parsedItems = rssReader.getItems();
```

V kombinaci se statickými metodami třídy *RssItem* může následovat i uložení:

```
RssItem.putAllNewToDatabase(parsedItems, olditems, context);
```

Ale v tomto případě aplikace skončí chybou, jak se záhy ukázalo při testech. Příčinou je, že Android preventivně ukončuje programy, které příliš zatěžují hlavní vlákno aplikace. A stahování, jako časově náročná úloha, je jednou z takto kritických operací.

A nebyl to jediný problém. Při návratu k návrhu a hledání řešení se zdálo nejlepší, použít implementaci speciální třídy AsyncTask určenou k vykonání části kódu paralelně, ve vlastním vlákně. Teprve reálná implementace celého řešení ukázala další, dosud ne zcela jasnou, chybu při pokusu spustit paralelní úlohu opakovaně.

Druhý návrat k rýsovacímu prknu už znamenal dvoudenní zpoždění, ale použitím klasických a osvědčených metod, skrze obyčejná vlákna Thread a vlastní úlohy typu Runnable – *Download*, byl celý problém vyřešen. Nyní můžeme v *MainActivity* napsat:

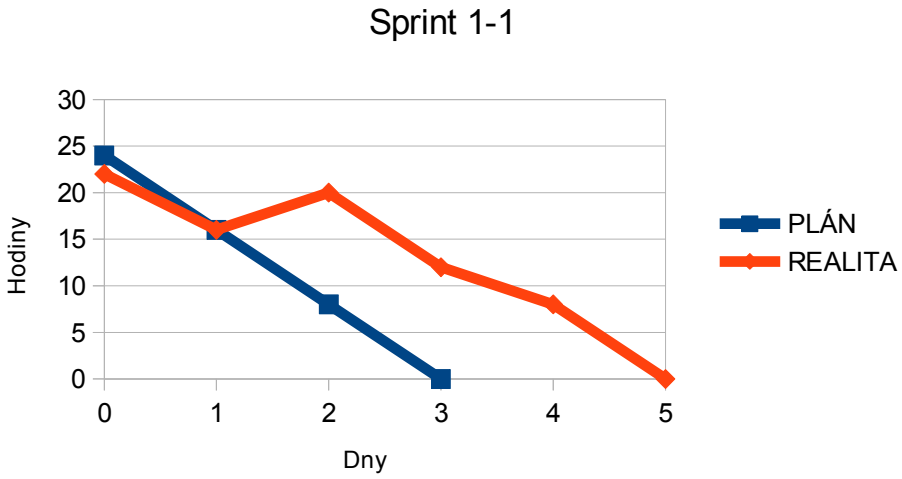
```
Runnable download = new Download(this, sources, items);
Thread thread = new Thread(download));
thread.start();
```

Později bylo pro zvýšení výkonu aplikace, přesunuto do vlastního vlákna i načítání dat z databáze. Nová implementace Runnable má podobně výstižný název, jako ta první – *Load*.

Ani tento způsob řešení ale nebyl úplně bez komplikací. Data bylo možné získat, dokonce opakovaně, ale bylo je nutné předat třídě *MainActivity*, což skutečnost, že běží v jiném vlákně, neulehčovala. Tento úkol plní do tříd *Load* a *Download* zabudované třídy typu

Handler. Pomocí handlerů dokážeme posílat z běžících vláken zprávy do vlákna hlavního, kde je můžeme vyhodnotit a zpracovat. V tomto případě vyhodnocujeme stav stahování a zpracováváme přijatá data. Aby pak mohla být společně se stavy předána *MainActivity*, potřebujeme i zvláštní druh reference – WeekReference. Obyčejná reference by mohla způsobit problémy s pamětí a tím i pád aplikace.

Zpoždění, které sprint nabral můžeme vidět níže. Ukázalo se, že časové odhady zde byly příliš optimistické. V dalších sprintech jsou proto více nadsazené.



Ilustrace 13: Burn Down Chart: Sprint 1-1

5.1.2 Sprint 1-2

Druhý sprint má trvat znovu přibližně tři dny (24 hodin) a jeho cílem bude splnění zbývajících úkolů celého prvního vydání:

- Server: Převod www.prf.jcu.cz na RSS (4 hodiny)
- Server: Převod www.lib.jcu.cz na RSS (4 hodiny)
- Klient: Převody časů (4 hodiny)
- Klient: Zobrazení seznamu aktualit (4 hodiny)
- Klient: Přesměrování na web s aktualitou (2 hodiny)

Od minulého sprintu máme běžící server, nyní přidáme skripty pro generování nových RSS kanálů ze stránek Přírodovědecké fakulty a Akademické knihovny. Podstata získání obsahu webových stránek byla vysvětlena v teoretické části (3.2.1 PHP Simple HTML DOM Parser) a nyní bychom se pouze opakovali. Ukážeme si ale druhou část skriptu – vytvoření RSS:

```
header('Content-type: text/xml; charset=utf8');
$rssfeed = '<?xml version="1.0" encoding="UTF-8"?> ';
$rssfeed .= '<rss>';
$rssfeed .= '<channel>';
$rssfeed .= '<title>Přírodovědecká fakulta JU</title>';
$rssfeed .= '<link>'. $link .'</link>';
$rssfeed .= '<description>Aktuality</description>';
     $count = count($dates);
for($i = 0; $i < $count; $i++){
     $rssfeed .= '<item>';
     $rssfeed .= '<title>' . $titles[$i] . '</title>';
     $rssfeed .= '<pubDate>' . date(DATE_ISO8601,
          strtotime($dates[$i])) . '</pubDate>';
     $rssfeed .=    '<description><![CDATA[' . $contents[$i] .
          ']]></description>';
     $rssfeed .=    '<content><![CDATA[' . $contents[$i] .
          ']]></content>';
     $rssfeed .= '<link>' . $links[$i] . '</link>';
     $rssfeed .= '<author/>';
     $rssfeed .= '</item>';
}
$rssfeed .= '</channel>';
$rssfeed .= '</rss>';
echo $rssfeed;
```

Zajímavým problémem byl čas a jeho jeho převod do formátu vyhovujícího aplikaci. RSS 2.0 používá standard RFC 822 – vypadá takto: Sat, 07 Sep 2002 00:00:01 GMT – ale my používáme ISO 8601 – například: 2013-03-30T21:11Z. Proč? Během vývoje aplikace byla totiž odhalena chyba starších verzí OS Android. Přesněji, na OS Android 2.3.3 načítání dat ve formátu RFC 822 selhává, i když shodný kód, spuštěný na OS Android 4.2.2, funguje. Bez delšího rozmýšlení bylo proto sáhnuto po formátu ze standardu Atom. Je jím již zmíněný ISO 8601 a aplikace ho spolehlivě načítá v obou verzích OS Android.

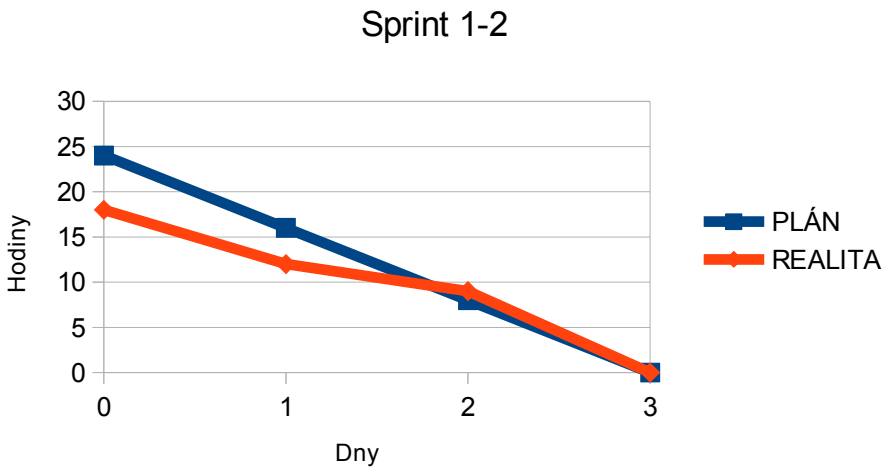
Metody pro převod času byly učiněny statickými a umístěny do třídy *RssItem*. Pomocí podobných šablon umí pak jedna z metoda převést čas získaný z RSS na objekt typu Date:

```
Date date = new Date();
SimpleDateFormat formatter =
     new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'",
          Locale.ENGLISH);
date = formatter.parse(pubDate);
```

Formát Date velmi ulehčuje práci zbylým metodám pro operace s časem. Takto například z Date vzniká čitelné datum pro zobrazením v seznamu aktualit:

```
String when = "";
SimpleDateFormat formatter =
     new SimpleDateFormat("dd. MM. yyyy", Locale.getDefault());
when = formatter.format(date);
```

O zobrazení seznamu aktualit a obsluhu události při kliknutí na položku, se starají běžné komponenty systému – *ListView*, *NewsListListener* implementující OnItemClickListener a *NewsListAdapter* rozšiřující ArrayAdapter, ve kterém se uplatní vlastní rozložení (layout) zobrazované položky (blíže rozebrané v kapitole 4.2.2.2 Zobrazení aktualit).



Ilustrace 14: Burn Down Chart: Sprint 1-2

5.2 Druhé vydání

Ve druhém vydání přidáme aplikaci navigační lištu, možnost aktualizace a podporu map, kterou doplníme i u serveru. Přesněji to vystihuje samotný Release Backlog druhého vydání:

#	NÁZEV	POPIS	PODMÍNKY SPLNĚNÍ	PRIORITA	ESTIMACE
5	Zobrazení seznamu místností	Umožní uživateli procházet seznam místností.	- Server umožňuje vkládání map - Server obsahuje XML seznam map - Klient načítá XML seznam map do DB - Klient zobrazuje seznam map z DB	1	4 dny
10	Aktualizace obsahu	Umožní uživateli získat aktuální obsah.	**- Zobrazení seznamu aktualit** **- Zobrazení seznamu místností** - Klient na požádání aktualizuje obsah - Klient má možnost ručního spouštění aktualizace - Klient informuje o nezdaru pokusu o připojení k serveru - Klient informuje o zdaru aktualizace	2	8 hodin
8	Zobrazení mapy S polohou místnosti	Umožní uživateli zobrazit mapu vybrané místnosti.	**- Zobrazení seznamu místností** - Uživatel je, po kliknutí na vybranou místnost v seznamu, přesměrován na mapu	3	8 hodin

Tabulka 5: Release Backlog – 2. vydání

Práce na druhém vydání má podle plánu trvat znovu šest dní, jako u vydání prvního. Zůstaneme proto u dvou sprintů, které si mezi sebe rozdělí následující úkoly:

- Server: XML seznam místností (2 hodiny)
- Klient: Načítání XML seznamu místností (4 hodiny)
- Klient: Ukládání seznamu místností do databáze (4 hodiny)
- Klient: Načítání seznamu místností z databáze (4 hodiny)
- Klient: Zobrazování seznamu místností (4 hodiny)
- Server: Mapy místností (4 hodin)
- Klient: Přesměrování na plán místnosti (4 hodiny)

- Klient: Horní lišta s aplikační navigací (8 hodiny)
- Klient: Přepínání aktuality/místnosti (4 hodiny)
- Klient: Nabídka spuštění aktualizace (2 hodiny)
- Klient: Indikace aktualizace (2 hodiny)
- Klient: Ošetření možných chybových stavů aktualizace (4 hodiny)

5.2.1 Sprint 2-1

Zde se zaměříme na úkoly související s navigací. Do 24 hodin můžeme stihnout tyto:

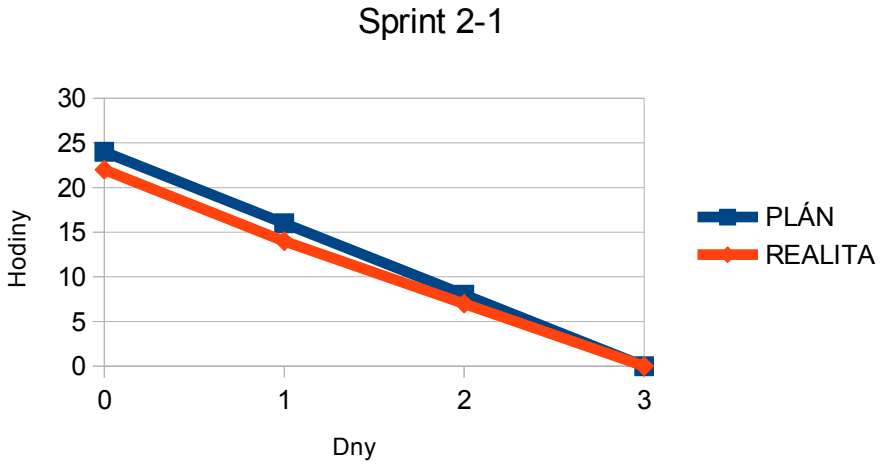
- Server: XML seznam místností (2 hodiny)
- Server: Mapy místností (4 hodin)
- Klient: Načítání XML seznamu místností (4 hodiny)
- Klient: Ukládání seznamu místností do databáze (4 hodiny)
- Klient: Načítání seznamu místností z databáze (4 hodiny)
- Klient: Zobrazování seznamu místností (4 hodiny)

Minulé vydání nám je dobrým základ pro další práci. Nejprve doplníme na server seznam místností v navržené podobě (viz kapitola 4.3.1.2 Struktura XML seznamu místností). Jde jen o krátký seznam o třech položkách, ale na ukázku nám úplně stačí. Vzniknou-li další mapy, stačí je do něj přidat.

Nové třídy – *Room*, *RoomsReader* a *RoomsParseHandler* – budou principiálně shodné se třídami kolem *RssSource* nebo *RssItem*. Všechny plní stejné poslání, ale nad různými daty.

I pro *RoomsList*, zobrazující seznam místností, již existuje příbuzný – *NewsList*. Stačí implementovat vlastní třídy *RoomsListAdapter* a *RoomsListListener*, za stejným účelem, jako vznikli jejich protějšky u *NewsListu*. Adaptér pro určování podoby položek (viz 4.3.1.3 Zobrazení místností) a plnění seznamu. A Listener, který se ale do tohoto sprintu už nevešel.

Vytvořit ukázkovou mapu je posledním úkolem tohoto sprintu a vytvořena byla (viz 4.3.1.4 Mapy místností). Na serveru je použita hned třikrát. Jednou, pro každou místnost, která je na ní navíc vybarvena. Je to velmi prostý trik, v budoucnu může být nahrazen sofistikovanějším řešením, ale funguje.



Ilustrace 15: Burn Down Chart: Sprint 2-1

4.3.2 Sprint 2-2

V posledním sprintu druhého vydání dokončíme navigaci z minulého sprintu a doplníme aplikaci o další nezbytné funkce. Přesné znění úkolů je zde:

- Klient: Přesměrování na plán místnosti (4 hodiny)
- Klient: Horní lišta s aplikační navigací (8 hodiny)
- Klient: Přepínání aktuality/místnosti (4 hodiny)
- Klient: Nabídka spuštění aktualizace (2 hodiny)
- Klient: Indikace aktualizace (2 hodiny)
- Klient: Ošetření možných chybových stavů aktualizace (4 hodiny)

Začneme dokončením práce započaté v minulém sprintu – implementací listeneru pro naslouchání, respektive čekání na chvíli, kdy uživatel klikne na položku v seznamu místností, aby ji otevřel. V tu chvíli se spustí internetový prohlížeč a zobrazí mapu.

Tím je pozadí (backend) aplikace kompletní a na řadu přichází její popředí (frontend).

Spojovacím prvkem celé aplikace je horní lišta. Vznikne složením obyčejných grafických kontejnerů (widgetů) do požadované podoby (viz 4.3.2.1 Vzhled a funkce horní lišty). Aplikační navigaci vdechnou život vlastní třídy typu ArrayAdapter a OnItemClickListener – *NavigationSpinnerAdapter* a *NavigationSpinnerListener*.

Ukazatel průběhu, ve výchozím stavu neviditelný a nahrazený logem, zviditelníme vždy při

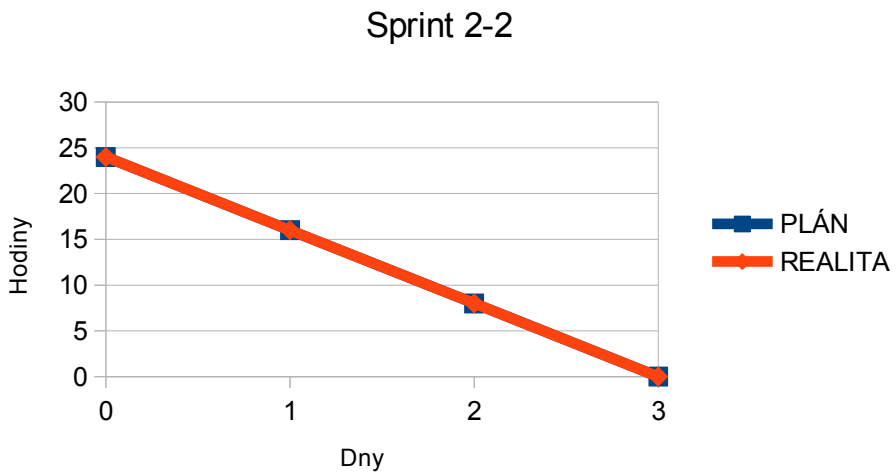
startu aktualizace. S informací o startu, chybě nebo dokončení aktualizace, která by popisovanou akci spustila, je to ale složitější. Pokud si vzpomínáte, stahování obsahu probíhá v odděleném vlákně a nastřádaná data se předávají skrze handler. Pro monitorování stavu se musí handler naučit vyhodnocovat nové stavové signály a provádět na jejich základě příslušné operace v *MainActivity* – zviditelnění a skrytí našeho ukazatele průběhu. Pro vyvolání neviditelného nebo zneviditelnění viditelného postačí dvě „zaklínadla“:

```
progress.setVisibility(ProgressBar.VISIBLE);
logo.setVisibility(ImageView.GONE);
```

Tento „magický“ princip se využije i při přepínání aktuality – navigace. Oba seznamy budou stále existovat, ale viditelný bude vždy jen jeden z nich.

Posledním bodem druhého vydání je ošetření možných chybových stavů, spojených zejména s nedostupností sítě nebo serverů, ale najdou se i další nedoladěné části aplikace. Ve třídě Download přibyla metoda *isNetworkConnected()*, která před pokusem o aktualizaci ověří, zda jsem připojeni k síti a pokud ne, vyšle příslušný stavový signál.

Dalšími úpravami a opravami se zabývá kapitola 6 Testy.



Ilustrace 16: Burn Down Chart: Sprint 2-2

6 Testy

Testování MIB probíhalo souběžně s vývojem. Pro odhalení problémů byla každá spustitelná verze kódu ihned vyzkoušena, u mobilní aplikace ve virtuálním zařízení s OS Android 4.2.2 (API 17) a zároveň ve fyzickém zařízení s OS Android 2.3.3 (API 10), a server pomocí internetového prohlížeče s kontrolou syntaxe XML.

Ve spuštěné mobilní aplikaci se navozovaly všechny stavy, které mohly nastat. Chybová hlášení z konzole LogCat pak poskytovala informace potřebné k nalezení a odstranění problémů – druh selhání, třída, metoda, řádek. Toto během vývoje prvních dvou vydání k odladění aplikace stačilo, ale při dalším rozvoji by se tento postup stal nakonec, kvůli zvyšujícímu se objemu kódu, neefektivním. Proto by nejbližší vydání mělo začít s implementací testovacích tříd, aby se jimi testování zautomatizovalo.

Testování ale není jen o kódu. Testovat se musí i „pohodlí“ uživatelů (UX). Cílem takových zkoušek je zjistit, jak si uživatelé s aplikací budou rozumět. Jde především o rozhraní, jeho jednoduchost a intuitivnost. Výsledkem byla například ikona v pravém rohu horní lišty (viz 4.3.2.1 Vzhled a funkce horní lišty), bez které uživatelé nemohli vědět, že v ní najdou nějaké menu.

6.1 První vydání

Ve sprintu 1-2 (viz 5.1.2 Sprint 1-2) byl testy odhalen problém s převodem času. Jeho řešení bylo již v uvedené kapitole popsáno, ale další problém související také s časem, zmíněn nebyl. Šlo o pořadí aktualit v seznamu. Pokud aktualita neměla od zdroje žádné datum vydání, nastoupil, jako záloha, čas jejího načtení. Problém spočíval v tom, že si aktuality při načítání prohodili pořadí. Aktualita, která byla v RSS úplně nahoře, byla najednou dole, protože byla nejstarší. Řešení je logické – stupňující se zpoždění:

```
currentItem = new RssItem(context);
Date date = currentItem.getDate();
long time = date.getTime();
long delay = count * 2000;
count++;
currentItem.setDate(new Date(time-delay));
```

První vydání mělo i další mouchy. Například problém ukládání aktualit. Ty by se při aktualizaci ukládaly vždy znovu a vytvářely duplicity, nebýt algoritmu v metodě

RssItem.PutAllNewToDatabase(). Ta nad každým prvkem seznamu aktualit před uložením provádí následující kontrolu:

```
for (RssItem parsedItem : parsedItems) {
     boolean isNew = true;
     for (RssItem savedItem : savedItems) {
          if ((parsedItem.getTitle()
                    .equals(savedItem.getTitle()))
               &&(parsedItem.getSource_title()
                    .equals(savedItem.getSource_title()))) {
               isNew = false;
               break;
          } else
          isNew = true;
     }
}
```

A zapomenout nemůžeme ani na největší objevený problém prvního vydání – paralelizaci. Ten byl zmíněn už v kapitole Návrh (viz 4.2.1.3 Diagram tříd) a řešen v kapitole Implementace (viz 5.1.1 Sprint 1-1). Šlo o stahování dat z internetu, které poprvé nefungovalo kvůli vlastnostem OS Android a podruhé kvůli vlastnostem třídy AsyncTask. Až třetí implementace byla úspěšná.

6.2 Druhé vydání

Ve druhém vydání se na řadu dostaly méně kritické nedostatky aplikace.

Musel být ošetřen případ, kdy jsme připojení k síti, ale ne k internetu, nebo servery nejsou dostupné z vlastních příčin. Metodám třídy *Download* – *downloadRooms()*, *downloadItems()* a *downloadSources()* – byla proto přidána vlastnost vyvolat chybu NullPointerException, kterou zachycujeme při jejich volání. Díky tomu aplikace už nespadne a navíc můžeme tento stav zachytit a oznámit uživateli:

```
try {
     mSources = downloadSources();
} catch (NullPointerException e) {
     e.printStackTrace();
     status = UPDATE_ERROR;
}
```

Při událostech typu překlopení displeje nebo přepnutí do jiné aplikace a zpět, je vlastností

OS Android, že je opuštěná aktivita zničena a při návratu znovu vytvořena. Uživatel toto ani nepostřehne, pokud je životní cyklus aktivity správně ošetřen. Je však nutné implementovat odpovídající metody. Stav aktivity uložíme při *onSaveInstanceState()* a byl-li nějaký uložen, načteme ho zpět v *onCreate()*. Například takto:

```
protected void onSaveInstanceState(Bundle outState) {
     outState.putInt(KEY_SELECTED,
          navigationSpinner.getSelectedItemPosition());
     super.onSaveInstanceState(outState);
}

protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     navigationSpinner =
          (Spinner) findViewById(R.id.navigation);
     if (savedInstanceState != null
          && !savedInstanceState.isEmpty()) {
          setLoadedData(savedInstanceState, false);
          navigationSpinner.setSelection(savedInstanceState
               .getInt(KEY_SELECTED));
     }
}
```

7 Rozšíření

Druhé vydání, byť funkční a splňující zadání práce, není zdaleka možné prohlásit za finální podobu MIB. Jedná se spíše o prototyp MIB. Možností rozšíření je proto mnoho.

Prvním a nejdůležitějším rozšířením by měla být možnost nastavení zdrojů aktualit, aby si uživatel mohl zvolit, co ho zajímá. Nejlépe by asi fungovalo přidání příznaku odebírat/neodebírat do tabulky zdrojů.

Toto vylepšení by ale nemělo velké uplatnění, kdybychom také nepřidali další zdroje. Již existující RSS stačí přidat na serveru do seznamu zdrojů (viz 4.2.1.1 Struktura XML seznamu zdrojů). A vytvoření nových RSS také nic nebrání (viz kapitoly 3.2.1 PHP Simple HTML DOM Parser a 5.1.2 Sprint 1-2).

Nutným rozšířením je i přidání nových map. Zde platí dvě pravidla: Musí jít o obrázek se zvýrazněnou polohou místnosti (viz 4.3.1.4 Mapy místností). A ten musí být přidán do seznamu místností (viz 4.3.1.2 Struktura XML seznamu místností).

Bude-li přidán větší počet map, začneme potřebovat funkci vyhledávání. Její implementace může být založena na vestavěném filtru ListView, nebo sofistikovaněji využít databázi.

Se všemi popsanými úpravami naroste projekt do rozměrů, kdy se skutečně vyplatí použít automatizované testy tříd, a proto by měli být rovněž přidány.

A v budoucnu může být MIB obohacena o další funkce i obsah. Například o podporu pro tablety nebo o rozvrhy u místností.

8 Závěr

Mobilní informační banka je ambiciózní projekt a její implementace v této práci není 100%. Navržená koncepce ale má potenciál skutečně usnadnit studentům život. Před masivním rozšířením bude nutné ještě řadu věcí dodělat (viz kapitola 7 Rozšíření), ale přesto práce své cíle plní:

1. Navrhnout architekturu mobilní informační banka použitím příslušných diagramů.

Byl vytvořen abstraktní model MIB jako celku (viz 4.1 Model) a diagram tříd mobilní aplikace (viz 4.2.2.1 Diagram tříd a 4.3.1.1 Diagram tříd). Úlohu diagramu případů užití (UseCase) zde zastoupil seznamem User Stories – Product Backlog (viz Přílohy).

2. Naprogramovat aplikaci do mobilních zařízení běžících na OS Android. Systém ponechat otevřený směrem k ostatním užívaným mobilním OS.

Mobilní aplikace, ve svém druhém vydání, plní zadané úkoly. A server je na mobilní platformě zcela nezávislí.

3. Doplnit webovou stránku Přírodovědecké fakulty nebo ústavu Aplikované informatiky o možnost stahování této aplikace a její instalaci v mobilních telefonech studentů.

Toto záleží na rozhodnutí vedení ústavu, potažmo fakulty. Prozatím je aplikace dostupná ze svých stránek: binolupa.napotom.com.

4. Provést příslušné akceptační testy a aplikaci uvést do trvalého provozu v českém a anglickém jazyce.

Celá MIB je nyní v testovacím provozu na adrese binolupa.napotom.com. Primárním jazykem mobilní aplikace je angličtina, ale podpora češtiny byla také zahrnuta.

5. Vše řádně zdokumentovat.

Dokumentace serveru je ve formě dokumentu README.txt přímo na serveru a mobilní aplikace má vysvětlující komentáře přímo v kódu. Základní principy celé MIB pak shrnuje tato práci.

Nad rámec cílů je univerzálnost celého řešení. Mobilní aplikace MIB není pevně vázaná na Jihočeskou univerzitu, jen zobrazuje data ze serveru. Pro nasazení na jiné škole proto stačí spustit vlastní server s příslušnými daty a nastavit jeho adresu v mobilní aplikaci.

9 Literatura

- KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. 1. vyd. Brno: Computer Press, 2004, 278 s. ISBN 80-251-0342-0.
- GOOGLE, Inc. *Android Developers* [online]. 2013 [cit. 2013-04-24]. Dostupné z: http://developer.android.com/
- SPELL, Brett. *Java. Programujeme profesionálně*. 1. vyd. Praha: Computer Press, 2002, 1022 s. ISBN 80-722-6667-5.
- ORACLE. *Oracle Documentation* [online]. 2013 [cit. 2013-04-24]. Dostupné z: http://docs.oracle.com/
- CHEN, S.C. PHP Simple HTML DOM Parser Manual. *PHP Simple HTML DOM Parser* [online]. 2013 [cit. 2013-04-24]. Dostupné z: http://simplehtmldom.sourceforge.net/manual.htm
- STACK EXCHANGE, Inc. *Stack Overflow* [online]. 2013 [cit. 2013-04-24]. Dostupné z: http://stackoverflow.com/
- ARLOW, Jim a Ila NEUSTADT. *UML a unifikovaný proces vývoje aplikací: průvodce analýzou a návrhem objektově orientovaného softwaru*. Vyd. 1. Brno: Computer Press, 2003, 387 s. ISBN 80-722-6947-X.
- Seriál: Vyvíjíme pro Android. DEVEL.CZ LAB S.R.O. *Zdroják.cz* [online]. 2012 [cit. 2013-04-24]. Dostupné z: http://www.zdrojak.cz/serialy/vyvijime-pro-android/

Přílohy

Product Backlog

#	JMÉNO	POPIS	PODMÍNKY
1	Zobrazení seznamu aktualit	Umožní uživateli procházet seznam aktualit.	- Server převádí aktuality na RSS - Server obsahuje XML seznam RSS zdrojů - Klient načítá XML seznam zdrojů do DB - Klient načítá aktuality z RSS zdrojů do DB - Klient zobrazuje seznam aktualit z DB
2	Filtrování zdrojů aktualit	Umožní uživateli vybrat, které aktuality uvidí.	**- Zobrazení seznamu aktualit** - Klient má seznam zdrojů - Klient má rozhraní pro výběr zdrojů aktualit - Klient zobrazuje pouze aktuality z vybraných zdrojů
3	Zobrazení detailu aktuality	Umožní uživateli přechod na stránku, která je zdrojem aktuality.	**- Zobrazení seznamu aktualit** - Uživatel je, po kliknutí na vybranou aktualitu v seznamu, přesměrován na zdroj
5	Zobrazení seznamu místností	Umožní uživateli procházet seznam místností.	- Server umožňuje vkládání map - Server obsahuje XML seznam map - Klient načítá XML seznam map do DB - Klient zobrazuje seznam map z DB
6	Filtrování místností podle budov	Umožní uživateli vybrat budovy, jejichž místnosti uvidí.	**- Zobrazení seznamu místností** - Klient vytváří seznam budov z XML seznamu map - Klient má rozhraní pro výběr budov - Klient zobrazuje pouze místnosti z vybraných budov
7	Vyhledávání místnosti	Umožní uživateli zadat název místnosti a zobrazí odpovídající výsledky.	**- Zobrazení seznamu místností** - Klient má rozhraní pro zadání hledaného výrazu - Klient podle hledaného výrazu zobrazuje zúžený seznam místností
8	Zobrazení mapy S polohou místnosti	Umožní uživateli zobrazit mapu vybrané místnosti.	**- Zobrazení seznamu místností** - Uživatel je, po kliknutí na vybranou místnost v seznamu, přesměrován na mapu
9	Uložení mapy	Umožní uživateli uložit mapu jako obrázek.	**- Zobrazení mapy s polohou místnosti** - Klient při zobrazení mapy nabízí možnost mapu uložit - Mapa se ukládá na SD kartu přístroje
10	Aktualizace obsahu	Umožní uživateli získat aktuální obsah.	**- Zobrazení seznamu aktualit** **- Zobrazení seznamu místností** - Klient na požádání aktualizuje obsah - Klient má možnost ručního spouštění aktualizace - Klient informuje o nezdaru pokusu o připojení k serveru - Klient informuje o zdaru aktualizace
11	Odlišení přečtených a nepřečtených aktualit	Umožní uživateli zjistit, které aktuality ještě nečetl.	**- Zobrazení seznamu aktualit** - Klient u aktualit sleduje stav přečtená/nepřečtená - Klient v zobrazovaném seznamu aktualit vizuálně odlišuje přečtené a nepřečtené aktuality