Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Servisní deník techniků

Plzeň, 2014 Štěpán Tuháček

Prohlášení

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

V Plzni dne

Abstract

The main purpose of this thesis is development of request tracking system. The first part of my thesis consists of analysis of existing applications, which could be used for same purpose and the main goals of this work are determined. In the next section are described technologies used for the development. In the following part there is described the implementation of the system and the classes and their methods are defined. Problems which occurred during testing are clarified. The conclusion consists of description of results and the need of individual adjustments for companies is mentioned.

Abstrakt

Práce se zabývá vývojem systému evidence požadavků, jenž by mohl být využit pro skupinu techniků. V první části práce je popsána analýza existujících aplikací, které by bylo možno využít pro tento účel, a jsou stanoveny hlavní cíle práce. V další části práce jsou popsány použité technologie. Následně je nastíněna implementace aplikací a jsou popsány třídy a jejich metody. Jsou objasněny problémy, které se vyskytly během testování. V závěru je zmíněna potřeba individuální úpravy pro případné nasazení aplikace do provozu ve společnosti.

Poděkování

Na tomto místě bych rád poděkoval vedoucímu své bakalářské práce Ing. Ladislavu Pešičkovi za odborné vedení a cenné rady, které mi pomohly tuto práci úspěšně dokončit.

Obsah

1. Úvod

Téměř v každé větší firmě je dnes potřeba zaznamenávat a sledovat požadavky, či vést evidenci vnitřních úkolů, které mají stanovenou určitou prioritu. Bez takovéto evidence by dříve či později mohla nastat situace, kdy například zaměstnanec nebude vědět, zda je potřeba něco akutně napravit, nebo je třeba dodat nějaký požadovaný materiál, atp.

Pro zkvalitnění efektivity, ale i zlepšení obecné komunikace ve firmě či společnosti, se v současné době běžně používají systémy pro sledování pracovních požadavků. Možností výběru druhu sledování je již velké množství – od zadávání úkolů přes e-mail až po sofistikované systémy s možností úpravy pro přesné požadavky dané firmy. Existují jak volně šiřitelné (Request Tracker, Bugzilla), tak komerční aplikace (JIRA). Je třeba podotknout, že každá firma má své specifické požadavky takovéto evidence, tudíž se tyto produkty mohou někdy i výrazně lišit.

Zaměření bakalářské práce se týká možností těchto evidencí s využitím mobilních aplikací, kterých však dosud na trhu není mnoho. Nejprve bylo nutné provést analýzu již existujících aplikací pro lepší zmapování funkčností tohoto druhu softwaru. Bylo také zapotřebí řešit problém synchronizace databází.

Výsledná aplikace by měla být schopna plynulého a spolehlivého běhu. Z testování by v závěru mělo být vidět, že by bylo možné ji nasadit do provozu v menší společnosti.

2. Analýza

Provedl jsem průzkum existujících aplikací, které lze více či méně použít pro podobnou činnost. Dále jsem analyzoval svou aplikaci a stanovil cíle práce.

2.1. Analýza existujících aplikací

Do analýzy zahrnuji pouze aplikace dostupné na platformu Android. U každé z těchto aplikací uvádím základní shrnutí, možnosti evidence, způsob synchronizace a licencování, či výši poplatku za danou aplikaci.

2.1.1. Evernote

Tato aplikace od společnosti Evernote Corporation umožňuje uživateli ukládat připomínky a poznámky mnoha způsoby. Velkou předností je zde přenositelnost – je možné ji použít jak v mobilních zařízeních (Android, Windows Phone, iOS, Blackberry), tak v počítači (Windows, Mac OS X), či jako doplněk v prohlížečích Chrome, Firefox a Opera.

Organizace poznámek je provedena pomocí poznámkových bloků a štítků. Jednotlivá poznámka může obsahovat text s možností kvalitního formátování (odrážkový, číslovaný, či zaškrtávací seznam, tučné písmo, atd.), dále několik typů příloh (obrázek, zvuk, video, soubor, PDF soubor, či odkaz) a v neposlední řadě možnost přidat k dané poznámce nastavení připomenutí na daný datum a čas, či zapsat místo konání v podobě GPS souřadnic.

Synchronizace je prováděna přes cloudové úložiště mezi více zařízeními uživatele, pokud je v nich aplikace nainstalována. Lze k tomu nastavit možnost, zda chce uživatel synchronizovat pouze při připojení přes Wi-Fi síť nebo i přes datové připojení. Je možné odesílat poznámky do Evernote e-mailem – díky této funkci se stává aplikace schopným GTD[1] nástrojem. Aplikace disponuje možností sdílet poznámky pomocí sociálních sítí.

[1] GTD - Getting things done; dnes již populární metoda organizace práce

Aplikace je zdarma, pro prémiový účet je třeba zaplatit částku $45 za rok. Tím získá uživatel výhody jako offline přístup k poznámkám, větší úložný prostor - 1GB měsíčně, velikost jedné poznámky max 100MB (uživatelé bez prémiového účtu 60MB měsíčně), možnost upravovat poznámky jiným uživatelům (uživatelé bez prémiového účtu je mohou pouze zobrazit), uzamknout poznámky PIN kódem, či editovat PDF soubory (uživatelé bez prémiového účtu mohou PDF soubory pouze prohlížet).

Evernote je kvalitní aplikace s příjemným designovým zpracováním, spolehlivou synchronizací a velkou škálou možností zadání poznámek. Uživatelé Androidu by mohli ocenit také podporu widgetů. Aplikace má na Google Play přes 10 mil. stažení, což vypovídá o její kvalitě a rozšířenosti mezi uživateli.

2.1.2. Google Keep

Společnost Google Inc. vsadila na jednoduchost a na jaře roku 2013 vypustila na trh aplikaci Google Keep, přesto má již také přes 10 mil. stažení. Je dostupná pro zařízení se systémem Android, Chrome OS, existuje také webová verze, či doplněk pro prohlížeč Chrome.

Uživatel může zaznamenávat úkoly pomocí poznámek, seznamů a fotek. K důležitým poznámkám je možnost přidat připomenutí. Aplikace disponuje funkcí automatického přepisu hlasové poznámky. Poznámky lze zaznamenat pomocí widgetů, barevně odlišit, či přesunout do archivu položek, které již uživatel nepotřebuje.

Co se týče synchronizace, poznámky lze používat odkudkoli – jsou uloženy v cloudu a dostupné na webu. Aplikace je zdarma. Její předností je již zmíněná jednoduchost, na rozdíl od Evernote, který je pro spoustu uživatelů až příliš komplexní. Spolupráce mezi uživateli přímo v aplikaci není možná. Jde však danou poznámku sdílet např. přes poštovního klienta.

2.1.3. Podio

Tato aplikace umožňuje evidovat úkoly v týmu, obsahuje spoustu pracovních návrhů pro projekty, prodejní procesy, vývoj produktů, správu uchazečů o zaměstnání a další. Je schopna zefektivnit práci na projektu funkcemi, jako jsou: přehled aktivity

týmu a zobrazení stavu úkolu v reálném čase, konverzace v týmu, přehled delegovaných či nesplněných úkolů, atd.

Poznámky je opět možné vkládat různými způsoby, jako např. vložení obrázků, GPS lokace, atp. Tyto možnosti se odvíjí od výběru specifického pracovního návrhu. Podio má přehledného správce poštovní schránky a kontaktů, umožňuje je také synchronizovat s telefonním seznamem. Aplikace je zdarma pro tým obsahující maximálně 5 uživatelů. Pro získání nadstandardních funkcí je poplatek $9 za měsíc na jednoho uživatele. Je přístupná z mobilních zařízení s Androidem a iOS (viz Obrázek 2.1), či z webových stránek společnosti Podio.

Obrázek 2.1: Aplikace Podio

2.1.4. JIRA Issue Tracker

JIRA Issue Tracker od společnosti RK Integrations je aplikace, která rozšiřuje systém pro zadávání úkolů JIRA i na platformu Android. Software JIRA je vyvíjen již od roku 2002 a je využíván velkým množstvím společností, např. NASA, BMW, Ebay, Cisco a Adobe. Po přihlášení do systému může uživatel prohlížet jemu přiřazené projekty a jim odpovídající úkoly, které může prohledávat, upravovat, či vytvářet.

Při vytváření úkolu lze zadat velké množství parametrů (termín, priorita, typ požadavku/úkolu, verze ke které se požadavek vztahuje, přiřazená osoba, atd.) Aplikace během testování zobrazovala chybové hlášky, například při přechodu na detail požadavku. Design sice není u takovéto aplikace tolik důležitý, avšak na mě působil tak, že se pro mne stávala nepřehlednou.

2.1.5. Wings for JIRA

Obdobně jako u JIRA Issue Trackeru tato aplikace rozšiřuje systém JIRA na platformu Android. Funkčně je téměř shodná, ale na rozdíl od výše zmíněné je Wings for JIRA designově přívětivější. Funkcí navíc je offline cachování požadavků (tzn., že jsou k dispozici i při nedostupném síťovém připojení). Tyto dvě aplikace jsou pro uživatele systému JIRA zdarma se základními funkcemi, u Wings for JIRA je však možné po zakoupení Premium ($2/měsíc) či Pro verze ($20/doživotně) využívat vylepšené funkce.

2.1.6. Shrnutí analýzy

Ač tyto analyzované aplikace slouží k podobnému účelu, svými funkčnostmi jsou až na dvě poslední zmíněné dosti odlišné. Aplikace Evernote by byla použitelná pro zadání této bakalářské práce. Její velká přednost je přístup k datům z velkého množství zařízení a po zaplacení prémiového účtu se stává silným nástrojem pro správu úkolů ve skupině více lidí.

2.2. Cíle práce

Cílem práce je vytvořit systém, který bude umožňovat jednoduchým způsobem evidenci úkolů. Každá firma, či společnost, by měla na tento systém evidence různé požadavky, které by bylo zapotřebí před samotnou implementací stanovit. Bylo tedy nutné nejprve uvést rozsáhlost funkcí systému, na čemž jsem se dohodl společně s vedoucím mé práce, kdy jsme probírali jednotlivé možnosti a výhody, či důležitosti daných funkcí.

Jako hlavní požadavky na systém byly uvedeny funkce znázorněné v Tabulce 2.1.

Uživatel	- Vytvoření uživatele - Přihlášení uživatele - Přiřazení úkolu jinému uživateli
Úkol	- Jednoznačná identifikace úkolu - Různé typy úkolů

	- Více priorit (např. A – kritická, B – vysoká, …) - Termín splnění s možností změny - Stav úkolu - Možnost vložení fotografie z mobilního zař. - Přidání komentáře
Systém	- Možnost nastavení adresy serveru - Zobrazení statistiky úkolů - Seznam úkolů - Zobrazení detailu úkolu a možnost jeho úpravy - Synchronizace úkolů mezi zařízeními - Možnost zadat úkol i při nedostupném síťovém připojení

Tabulka 2.1: Funkce systému

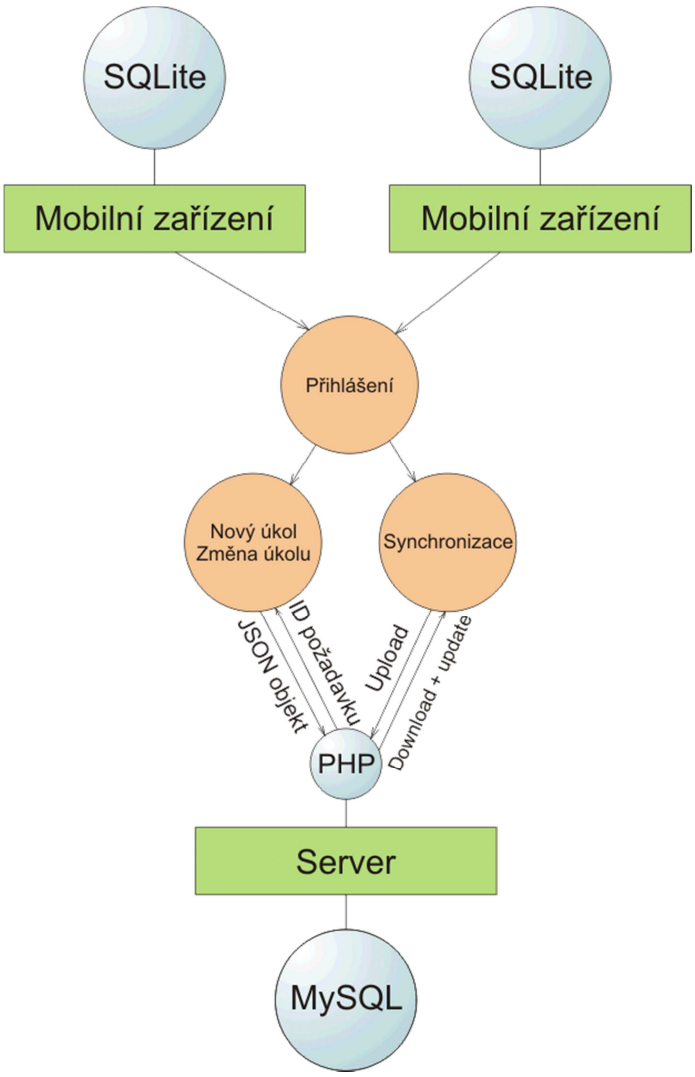
2.3. Předběžný návrh systému

Systém bude sestávat ze dvou aplikací – jedna pro mobilní platformu Android, druhá pro operační systém Windows. Bude zde také server, na kterém bude umístěna hlavní databáze (viz Obrázek 2.2).

Aplikace na mobilní zařízení bude využívat lokální databázi SQLite. S hlavním serverem bude komunikovat pomocí PHP skriptů, které budou umístěny na straně serveru. Způsob této komunikace bude prostřednictvím objektů JSON. Vytvoření požadavku bude pomocí formuláře, ve kterém bude možno přiřadit úkol jinému uživateli. Dále zde bude možnost využití fotoaparátu, díky kterému půjde nahrát k úkolu fotografie jako příloha. Pokud nebude mít mobilní zařízení během vytváření požadavku dostupnou mobilní síť, dojde k lokálnímu uložení. Při dostupné síti dojde k nahrání tohoto požadavku na server. Dále bude možné úkoly prohlížet a případně upravit, či přidat komentář.

Na straně serveru bude hlavní databáze MySQL, ve které budou uloženy záznamy o všech uživatelích a veškeré požadavky. Tato databáze bude zásadní z hlediska poslední změny daného požadavku.

Aplikace na operační systém Windows bude napsána v jazyce C# a bude díky ní možné ovládat databázi na hlavním serveru (přidat uživatele, požadavek, provádět změny).



Obrázek 2.2: Diagram systému

3. Použité technologie

V této kapitole jsou popsány veškeré technologie, které jsem při své práci použil. Je kladen důraz na objasnění použitých komponent a vysvětlení základních pojmů.

3.1. Android

Android je operační systém založený na Linuxovém jádře[5]. Je vyvíjen primárně pro dotyková mobilní zařízení. Jeho zdrojový kód je vyvíjen v soukromí společnosti Google, která jej po uvedení nové verze zveřejní a uvolní tak širokému spektru vývojářů.

Vývoj aplikací na tuto platformu je značně ulehčen vývojářskými nástroji dodávanými již zmíněnou společností. Po založení projektu, např. v prostředí Eclipse s ADT pluginem, je automaticky vygenerováno několik komponent, které lze následně upravovat. Jednou z hlavních takových komponent je třída, která se z pohledu vývojáře pro Android nazývá „aktivita“.

Aktivita je komponenta aplikace, jež uživateli zajišťuje interakci. Dá se také chápat jako jednotlivá obrazovka, ve které je vykresleno grafické rozhraní. Obvykle se jedna aplikace skládá z několika aktivit, přičemž jedna je definována jako hlavní. Ta je uživateli zobrazena po startu aplikace.

3.1.1. Android Manifest

Jedná se o soubor `AndroidManifest.xml`, který musí být obsažen v kořenovém adresáři každé aplikace platformy Android. Tento soubor poskytuje operačnímu systému základní informace o dané aplikaci, přesněji řečeno informace, které jsou potřebné pro systém dříve, než spustí aplikaci samotnou. Jedná se například o jméno aplikace, jméno balíčku a samotných tříd, popis komponent aplikace, rozhodnutí o tom, jaké procesy budou obstarávat dané komponenty, práva potřebná pro aplikaci ke spuštění chráněných částí API a k interakci s ostatními aplikacemi. V neposlední řadě je zde uvedena informace o minimální potřebné verzi systému Android a údaje o knihovnách, které aplikace potřebuje mít k dispozici.

Pokud by například dané mobilní zařízení nedosahovalo minimální požadované verze systému Android uvedené v tomto souboru, nebude možné aplikaci na zařízení použít. Je tedy poměrné důležité zvážit, na jakou verzi chceme cílit, tedy zda funkce vydávané v novějších verzích systému opravdu potřebujeme, nebo je možné se bez nich obejít.

Soubor `AndroidManifest.xml` může obsahovat velké množství elementů, nelze si však vymyslet vlastní, jedná se o definovaný seznam s těmito povolenými elementy.

Příslušný `.xml` soubor má jasně definovanou strukturu (viz Listing 3.1), jelikož se jedná o značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C[2]. Daný soubor musí obsahovat jediný kořenový element, do kterého jsou další vkládány (nemohou se však navzájem překrývat). Každý element je ohraničen počáteční a ukončovací značkou, parametry atributů musí být umístěny v uvozovkách.

```
<korenovyElement atribut1="x">
   <vlozenyPrvek atribut2="y">
       <dalsiPrvek>Toto je dalsi prvek</dalsiPrvek>
   </vlozenyPrvek>
</korenovyElement>
```

Listing 3.1: Ukázka XML dokumentu

3.1.2. Grafické rozhraní

Android poskytuje několik designových šablon, ze kterých lze při tvoření aplikace vycházet. Při zakládání každé activity se k ní vytvoří příslušný `.xml` soubor, ve kterém je tato šablona definována. Lze buď použít již zmíněné předpřipravené, či vytvořit si kompletně vlastní. Výhodou vývojového prostředí Eclipse je možnost instalovat Android Developement Tools plugin a s jeho pomocí zobrazit rozvržení tak,

[2] Dostupné z - http://www.w3.org/XML/

jak by vypadalo na mobilním zařízení. Zároveň zde vidíme každou změnu provedenou v souboru s definicemi prvků.

Pokud mluvíme o výše zmíněných šablonách v kontextu s platformou Android, jedná se o základní uživatelské rozhraní, takzvaný layout[3], v němž se nacházejí jednotlivé prvky, které dovolí uživateli interakci – jsou tedy do tohoto layoutu vnořeny. Existuje 5 základních typů rozvržení:

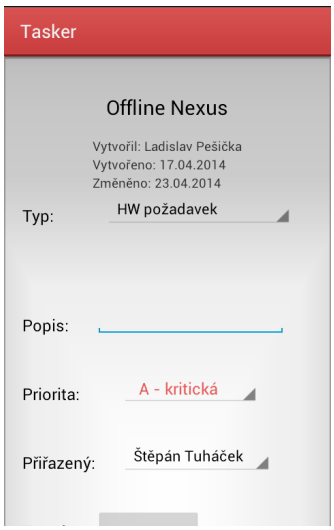
- lineární:

- prvky jsou vnořovány vertikálně, či horizontálně (tuto vlastnost určuje parametr `android:orientation` specifikovaný v `.xml` souboru s rozvržením náležící dané aktivitě)

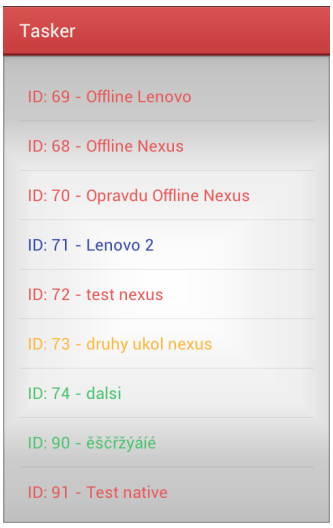
- pokud dojde k nedostatku místa na obrazovce pro všechny prvky, automaticky se umožní posun obrazovky

- relativní (viz. Obrázek 3.1):

- umožňuje specifikovat pozici vnořených prvků relativně vzhledem k sobě samým



Obrázek 3.1: Relativní layout



Obrázek 3.2: List View

[3] Layout (angl. plán) – grafické rozvržení stránky

- web vzhled (web view):

- pro tvorbu a zobrazení webových aplikací

- forma seznamu (list view):

- obecný jednořádkový seznam, vhodný např. pro výčet položek (viz. Obrázek 3.2)

- forma tabulky (grid view):

- možnost zobrazení do klasické tabulkové struktury

Nejčastěji je v layoutu upravována šířka, vzdálenost a pozice prvků. Tyto rozměry se mohou uvádět v různých jednotkách, nejvhodnější je však jednotka *dp*, což je abstraktní jednotka založená na hustotě pixelů obrazovky daného mobilního zařízení, relativně ke 160 dpi (body na palec) obrazovky.

Důležitou hodnotou atributu je také samotný parametr daného layoutu. Umožňuje říci nadřazenému prvku, jakým způsobem mají být umístěny. Základní parametry určují, jak velké se mají dané prvky zobrazit:

`FILL_PARENT`: podřazený prvek má být stejně velký jako jeho nadřazený prvek

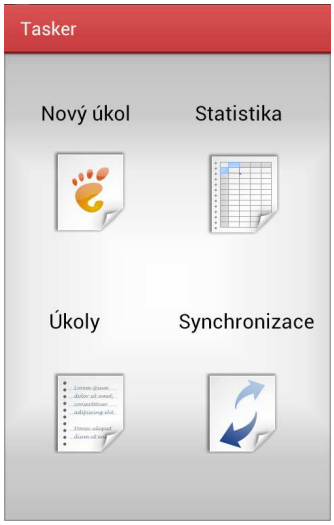
`WRAP_CONTENT`: podřazený prvek má být tak dostatečně velký, aby se zobrazil jeho obsah

Dále je zde možnost využít velké množství prvků samotného uživatelského rozhraní, které lze navíc do značné míry upravit podle vlastních požadavků. Dle potřeb způsobu zadávání úkolů jsem volil převážně prvky formulářové – textové pole, výběr z možností, tlačítka.

Pro atraktivnější design jsem se snažil nepoužívat tlačítka nabízená Androidem, ale vytvořit si vlastní v grafickém editoru, viz Obrázek 3.4. Tlačítka menu nabídky (viz Obrázek 3.3) jsou převzata z volně dostupných zdrojů na internetu.

Veškeré obrázky, které chceme v Android aplikaci využít, ukládáme do adresáře `drawable`, který navíc obsahuje několik podadresářů rozdělených podle hustoty pixelů obrazovky. Toho využijeme například tehdy, pokud chceme, aby se ikona naší aplikace zobrazovala stejně kvalitní a dostatečně velká jak na mobilním zařízení

s malou obrazovkou, tak na zařízení s obrazovkou velkou. To, jaký adresář s obrázky bude při spuštění aplikace využit, rozhoduje samotný systém, který tyto zdroje vhodně zvolí. Pokud se v daném adresáři potřebný obrázek nenachází, systém jej najde v adresáři výchozím a změní jeho velikost tak, aby odpovídala příslušné velikosti obrazovky a hustotě jejích bodů.

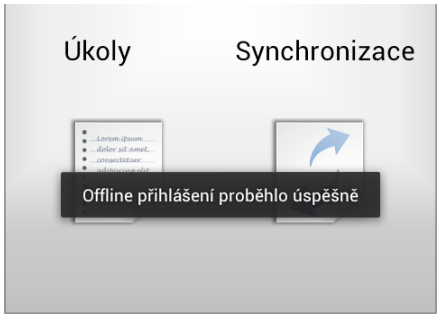


Obrázek 3.3: Menu aplikace

Obrázek 3.4: Přihlašovací obrazovka

3.1.3. Toast, dialog a spinner

Pro dostatečnou informovanost uživatele jsem využil také objekt zvaný `Toast` (viz. Obrázek 3.4), jenž poskytuje jednoduchý způsob zobrazení zpětné vazby, či upozornění na určitou událost. Standardně se zobrazuje v dolní části obrazovky, zabere pouze místo potřebné pro korektní zobrazení celé zprávy a automaticky zmizí po určité době. Je vhodný v případě, kdy chceme uživatele pouze informovat o určité události a neočekáváme od něj jakoukoliv odpověď.

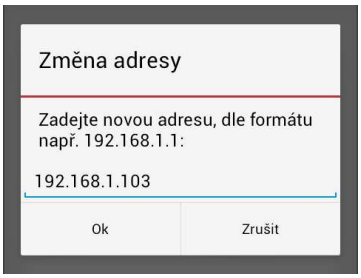


Obrázek 3.4: Ukázka objektu Toast

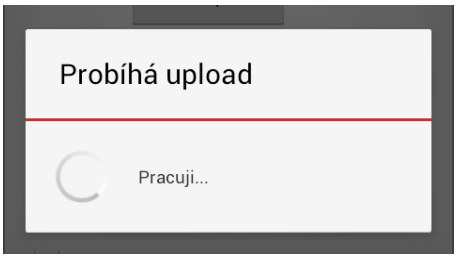
Pro určitou dynamičnost slouží také objekt `Dialog`. Jedná se o malé vyskakovací okno, které může být upraveno dle potřeby a sloužit pak pro více účelů. Nejčastěji chceme po uživateli, aby učinil nějaké rozhodnutí, zadal požadovanou hodnotu, či vyčkal na dokončení operace. Tento objekt a jeho podtřídy jsem využil v aplikaci mnohokrát:

- klasický styl s potvrzovacími tlačítky (Obrázek 3.5)

- klasický styl bez možnosti potvrzení (Obrázek 3.6)



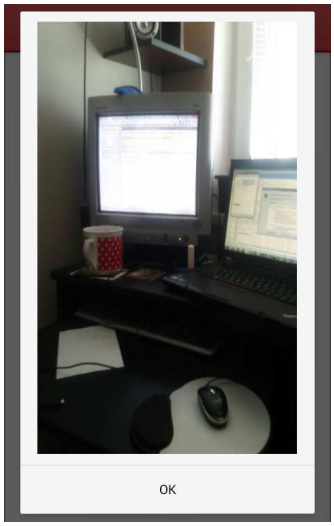
Obrázek 3.5: Dialog s potvrzením



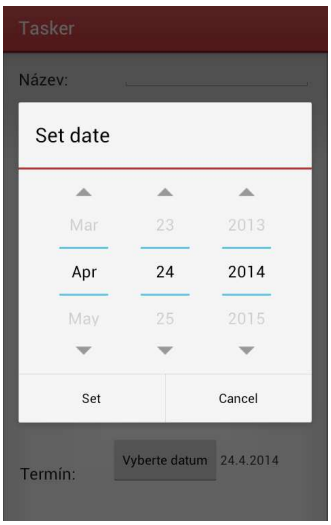
Obrázek 3.6: Dialog bez potvrzení

- dialog se zobrazením fotografie (Obrázek 3.7)

- výběr data (Obrázek 3.8)



Obrázek 3.7: Dialog pro fotografii



Obrázek 3.8: Dialog pro datum

Zmínil jsem také použití výběru z možností - tzv. objekt `Spinner`. Ten nabízí uživateli rychlý způsob vybrání hodnoty z nabízených možností. Ve výchozím stavu

zobrazuje aktuálně zvolenou hodnotu. Při doteku prstem na tuto hodnotu se rozevře nabídka, ve které je možno zvolit hodnotu jinou, z dalších dostupných.

Jeden z nejpoužívanějších způsobů, jak tento objekt naplnit dostupnými hodnotami, je uvedení těchto hodnot v `.xml` souboru a poté je možno se na něj odkázat v kódu. Druhým způsobem je přidání takovéhoto výčtu hodnot přímo v kódu aplikace, v neposlední řadě pak lze plnit tyto hodnoty z databáze.

Obrázek 3.9: Spinner s barevnými prioritami

Použil jsem způsob první, jelikož by bylo v případě pozdějších úprav snadné tento soubor pouze pozměnit. V pozdější fázi implementace jsem potřeboval takovýto objekt upravovat v kódu dle designových požadavků. Jednalo se o barevné rozlišení priorit jednotlivých úkolů dle závažnosti (viz. Obrázek 3.9).

3.1.4. Fotografie

Jak jsem již uvedl, použil jsem dialog pro zobrazení fotografie, která slouží jako případná příloha k úkolu, např. pro zdokumentování daného problému. Pro vyfocení fotografie bylo třeba nastavení příslušných práv v souboru `AndroidManifest.xml`. Po nastavení těchto práv je možno využít výchozí aplikaci systému pro focení fotografií. Vlastní aplikace se pozastaví a předá řízení této výchozí aplikaci, s jejíž pomocí uživatel zachytí snímek. Po jeho potvrzení se do popředí dostane opět aplikace vlastní, která má již zachycený snímek k dispozici a je schopna s ním dále pracovat.

Usoudil jsem, že by bylo vhodné, aby mohl uživatel s touto fotografií pracovat i nadále, například pro případ pozdějšího popisu servisního zásahu. Fotografie je tedy po zachycení uložena do výchozí galerie obrázků mobilního zařízení, kde si aplikace sama případně vytvoří příslušnou složku `/tasker`, díky čemuž bude snadnější danou fotografii dohledat a případně na těchto sítích sdílet, či použít v jiných aplikacích pro možnou úpravu.

3.1.5. Použité externí knihovny

Jednou z částí této aplikace je také statistika úkolů. Uživatel by měl mít přehled o veškerých jeho úkolech, či o úkolech celkově. Po konzultaci s vedoucím jsme se dohodli na následující podobě statistiky:

- přehled úkolů dle priorit

- přehled stavů úkolů

- informace o úkolech příslušného uživatele

Dále bylo nutno zvážit, jakým způsobem dané informace zobrazím. Jelikož jsem nechtěl, aby byly tyto informace pouze v textové podobě, analyzoval jsem dostupné knihovny pro kreslení grafů a tabulek v systému Android. Dnes existuje již velké množství jak volně dostupných, tak placených knihoven pro kreslení komplexních grafů pro tento systém.

Obrázek 3.10: Statistika úkolů

Jako nejlepší řešení se zdála být knihovna HoloGraphLibrary[1] napsaná v programovacím jazyce Java. Tato knihovna je šířena pod licencí Apache, což znamená, že umožňuje vývojáři používat tuto knihovnu bez jakýchkoliv omezení za předpokladu, že bude přiložena k danému produktu, například ve zdrojovém kódu.

Knihovna poskytuje tři základní typy grafů - lineární, sloupcový a výsečový. Po importování knihovny do projektu a vyzkoušení základních funkcí jsem již byl rozhodnut, jak ji využiji. Vývojář má možnost barevných úprav podle svých nadefinovaných barevných vzorů, u sloupcového grafu poskytuje i hezké popisky jednotlivých hodnot. Knihovna je stále ve vývoji a nové funkce postupně přibývají.

Pro přehled úkolů dle priorit jsem zvolil graf výsečový, ale jelikož tento typ grafu nepodporuje popisky jednotlivých výsečí, bylo zapotřebí pod graf přidat vysvětlivky. Tento graf se zobrazí pouze pokud je k dispozici více jak jeden úkol s různou prioritou. Přehled stavů úkolů zobrazuji pomocí grafu sloupcového. Tento typ podporuje dokonce popisky ve stejné barvě jako samotné sloupce. K nim je možno zobrazit jejich hodnotu. Zvolil jsem číselnou hodnotu nad sloupcem a název jsem umístil pod sloupec (viz Obrázek 3.10).

Aby měl každý uživatel přehled o úkolech, které má na starost, je pod oběma grafy zobrazena informace o jejich počtu úkolů, které jsou na daného uživatele delegovány. Níže je pak zobrazeno, kolik úkolů bylo z tohoto počtu zrušeno, či splněno.

Dále byla použita knihovna BouncyCastle (kolekce rozhraní pro kryptografické účely), o ní více v kapitole 3.2.4 Synchronizace, proměnlivost sítě.

3.2. Databáze

Všechna data, s kterými systém požadavků pracuje, bylo třeba nějakým způsobem přehledně uchovávat v určité struktuře. To všechno umožňuje databáze.

3.2.1. SQLite

V systému Android je vestavěna databáze SQLite, což je relační databázový systém obsažený v relativně malé knihovně. Ta se přilinkuje k dané aplikaci, přesněji

řečeno je uložena v `.dbm` (Database Manager) souboru a umístěna v adresáři `/databases` každé aplikace. Výhodou tohoto souboru je fakt, že je multiplatformní, čili kdybychom jej potřebovali přenášet mezi servery, bude to velké usnadnění. Dalšími charakteristickými rysy je absence možnosti konfigurace a absence serveru. Její zvláštností je tzv. „typeless“ vlastnost. To znamená, že můžeme ve sloupci deklarovaném jako číselný typ nalézt i datový typ string, jelikož sama databáze nekontroluje, jaký typ do sloupce ukládá. Výjimka je však u sloupce, který je označen jako `INTEGER PRIMARY KEY`, v němž je vždy celé číslo. Tyto neobvyklosti mohou poté vést na problém s uložením data.

Pomocí jednoduchého rozhraní vývojového prostředí Eclipse ji lze snadno prohlížet. Pro tento účel existuje i velké množství softwarů. Tento databázový systém je dostupný pod licencí public domain, čili je možné jej volně užívat, dokonce i pro komerční účely. Je vhodný pro operace nad menšími daty, výhodou je pak jeho schopnost šetřit se systémovými prostředky. V případě mojí mobilní aplikace tvoří tato databáze důležitou část. Jsou využívány tři tabulky:

`Task` – tabulka obsahující veškerá data týkající úkolů

`User` – obsahuje veškerá data o uživatelích

`TaskOffline` – tabulka s úkoly využívaná při nedostupnosti síťového připojení

3.2.2. MySQL

Aby bylo umožněno sdílet a delegovat úkoly na ostatní uživatele, nestačilo je pouze lokálně ukládat do SQLite databáze, ale bylo je nutné i nahrávat na veřejně dostupný server, na kterém je umístěna hlavní databáze. Zvolil jsem databázi MySQL, která je k dispozici jak pod bezplatnou licencí, tak i tou komerční placenou.

MySQL je označení pro RDBMS[4], což je databázový server spravující databáze, komunikaci s klienty, přenos dat a jejich integritu. Je distribuován bez nástrojů pro správu databáze s grafickým rozhraním. Administrátor takovéto databáze může pak

[4] RDBMS (angl. Relational DataBase Management System) – systém pro správu relační databáze

využít obsažený nástroj pro ovládání v příkazovém řádku, nebo je zde možnost využít nadstaveb, což jsou programy usnadňující práci s tímto příkazovým řádkem tak, že pro zadávání parametru a funkcí poskytují přehlednější grafické rozhraní. Oficiální sada nástrojů pro ovládání MySQL databáze je volně dostupný program MySQL Workbench. Při implementaci této aplikace jsem však použil webově založenou nadstavbu phpMyAdmin[12].

Abych zachoval strukturu tabulek uvnitř aplikace v databázi SQLite, ponechal jsem ji i zde u MySQL, pouze ubyla tabulka pro úkoly vytvořené při nedostupné síti.

Tabulky:

`task` – obsahuje pole: id, name, type, priority, description, state, reporter, assignee, date_create, date_due, date_last_update, attachment1, attachment2, comment, valid

`user` – obsahuje pole: id, username, password, name, email, department, used, valid

pozn.: Tabulka `TaskOffline` v SQLite databázi má navíc příznak `update,` jenž značí, zda byl úkol již nahrán do hlavní databáze.

3.2.3. PHP, JSON

Bohužel se mi nepodařilo nalézt způsob, kterým by mohla komunikovat Android aplikace přímo s databází MySQL umístěnou na nějakém veřejném serveru na přímo. Dá se však využít PHP služeb/skriptů, které budou na tomto serveru umístěny a díky nim bude aplikace schopna data z databáze načítat, či ukládat[11].

PHP je skriptovací programovací jazyk, který je převážně využíván pro tvorbu dynamických internetových stránek a aplikací. Skripty jsou prováděny na straně serveru a až poté je výsledek přenesen k uživateli. Mezi jeho hlavní výhody patří multiplatformnost a podpora mnoha databázových systémů. Použil jsem verzi 5.5.8.

Samotná PHP služba však není dostačující pro komunikaci „aplikace – MySQL“. Je třeba zmínit, že je v ní navíc využito JSON objektů. JSON[5] je dnes hojně využívaný formát pro výměnu dat, který je nezávislý na použitém jazyce. Byl vyvinut jako odlehčená náhrada XML a mezi jeho hlavní výhody patří menší velikost přenášených dat. Mezi jeho nevýhodu osobně považuji nemožnost definovat znakovou sadu přenášených dat. JSON má několik podob (struktur), které je možno využít (viz Listing 3.2). Je však nutné dodržet syntaxi, jinak by se stal objekt neplatný a tím pádem nepoužitelný.

```
{
     “studenti“: [
         {"jmeno":"Jan","prijmeni":"Novak"},
         {"jmeno":"Anna","prijmeni":"Novotna"}
      ]
}
```

Listing 3.2: Struktura JSON objektu

Data od aplikace se dají přijímat několika způsoby. V PHP existují takzvané *super-globální proměnné*, což jsou předdefinovaná pole, uvnitř kterých jsou hodnoty proměnných získané od volající aplikace[4]. Rozdíl použití záleží na tom, odkud chceme data přijmout, např. ze serveru, z url, atd. U svých skriptů získávám data pomocí super-globální proměnné `$_REQUEST`, jenž může obsahovat kombinaci všech předaných hodnot - `$_POST`, `$_GET` a `$_COOKIE`.

V takovémto formátu jsem již byl schopen přenést data z databáze SQLite (potažmo své aplikace) na server s databází MySQL a zpět za využití PHP funkcí pro převod těchto objektů do potřebného formátu. Později se však objevil problém s výše zmíněnou znakovou sadou (kódováním). PHP skripty, přesněji řečeno použité funkce na převod JSON objektů, nekontrolují předávaný obsah, tudíž provedou svou

[5] JSON (angl. Java Script Objet Nation) – formát pro výměnu dat

činnost, ať už je obsah jakýkoliv. Bylo zapotřebí použít správné hlavičky dokumentů a nastavit příznaky definující kódování daného obsahu. A to platilo i o samotných definicích hlaviček na straně aplikace.

3.2.4. Synchronizace, dostupnost sítě

Poté, co jsem již byl schopen předávat data mezi aplikací a hlavní databází (MySQL), bylo potřeba navrhnout synchronizaci tabulek mezi oběma těmito databázemi (SQLite – MySQL). Musel jsem brát v potaz fakt, že uživateli musí být umožněno přihlášení a vytvoření úkolu i při nedostupném síťovém připojení. Analyzoval jsem tento problém a implementoval následující řešení.

Uživatelé:

Při prvním spuštění aplikace na mobilním zařízení je potřeba, aby bylo dostupné síťové připojení. To z důvodu, že během procesu přihlašování se aktualizuje tabulka v lokální databázi SQLite s údaji o uživatelích. Bylo nutné také myslet na bezpečnost, proto jsou ukládány veškeré informace kromě hesel. Aby ale bylo možno provést přihlášení pomocí hesla, musel jsem vymyslet způsob, jak jej uložit bezpečněji. Rozhodl jsem se tedy použít šifrovací algoritmus.

Knihovna BouncyCastle[2] je šířena pod licencí MIT, což znamená, že může být volně použita právě tehdy, když bude text této licence dodán společně s daným softwarem, který takovou knihovnu používá. Tato knihovna nabízí velké množství šifrovacích algoritmů - jak symetrických, tak asymetrických. Vybral jsem algoritmus (hashovací funkci) SHA-512, který se jinak souhrnně označuje jako SHA-2. Délka klíče je, jak již číslo napovídá, 512 bitů[6].

Po úspěšném přihlášení se uloží konkrétní uživatel do databáze SQLite spolu se zašifrovaným heslem. Dále je aktualizována tabulka se všemi ostatními uživateli a následně zpřístupněno menu aplikace.

Pokud není na mobilním zařízení dostupné síťové připojení a již někdy v minulosti proběhlo úspěšné přihlášení, je opět tato tabulka využita. Heslo vložené

uživatelem se zašifruje tím samým šifrovacím algoritmem a následně je porovnáno s otiskem uloženým v SQLite databázi. Poté je opět při shodě zpřístupněno menu.

Vytvoření úkolu:

Při vytvoření úkolu proběhne kontrola dostupnosti mobilní sítě. Pokud je síť dostupná, úkol se se nahraje do hlavní databáze MySQL. Ihned po tomto uložení odešle PHP služba zpět aplikaci zprávu o potvrzení úspěšného uložení spolu s ID číslem úkolu. Pod tímto jedinečným číslem je úkol uložen do lokální databáze SQLite. Tak je zaručeno, že nenastane konflikt, kdy by dvě různá mobilní zařízení měla uvnitř své lokální databáze úkol se stejným ID číslem.

Pokud není síť dostupná, dojde k uložení úkolu do lokální SQLite databáze. Na rozdíl od předchozího případu je však uložen do tabulky `TaskOffline`, kde je úkolu nastaven příznak, který značí, že tento úkol nebyl dosud nahrán do hlavní databáze. Úkol zde zůstává a není možno jej editovat ani prohlížet nikým, dokud uživatel nezvolí funkci synchronizace.

Synchronizace úkolů:

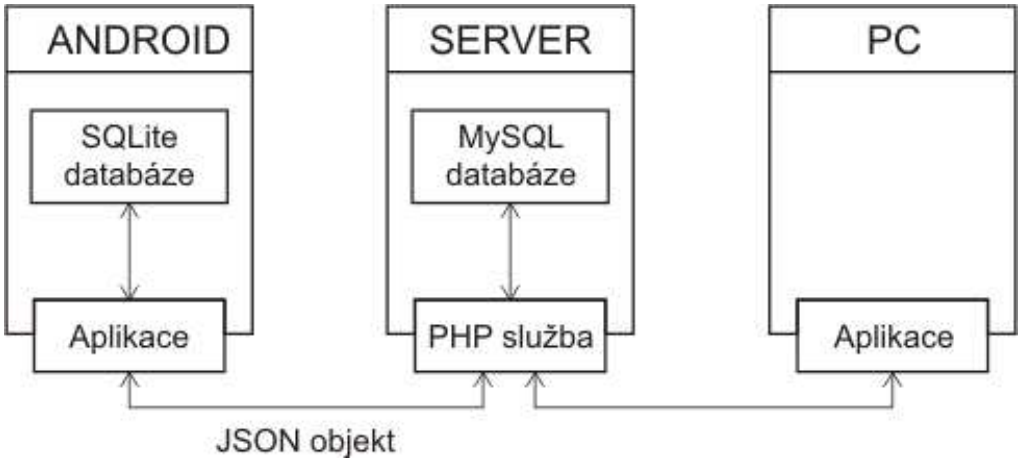
Synchronizace je možná pochopitelně pouze při dostupném připojení. Nejprve dojde k nahrání všech úkolů z tabulky `TaskOffline` do databáze MySQL (požadavky, které byly vytvořeny při výpadku síťového připojení). Pokud proběhne nahrání do tabulky `Task` v obou databázích úspěšně, je tento úkol z původní tabulky smazán.

Poté aplikace vyžádá ID všech nahraných požadavků z hlavní databáze MySQL. Pokud žádné nemá a tedy neobdrží (případ prvotního spuštění aplikace, smazání záznamů v PC aplikaci), tento proces končí. V opačném případě je aplikace získá a zjistí, které úkoly jí ve své lokální databázi SQLite chybí. Následně jsou doplněny.

V konečné fázi je porovnáváno pole `last_update_date` dle databáze MySQL, které obsahuje datum a čas poslední změny příslušného úkolu. Aplikace provede obnovu každého úkolu, u kterého je tento datum aktuálnější v hlavní databázi.

Update úkolu:

Je možné také zobrazit, upravit a následně pak opětovně uložit detail každého nahraného požadavku. To lze pouze však při dostupném síťovém připojení.



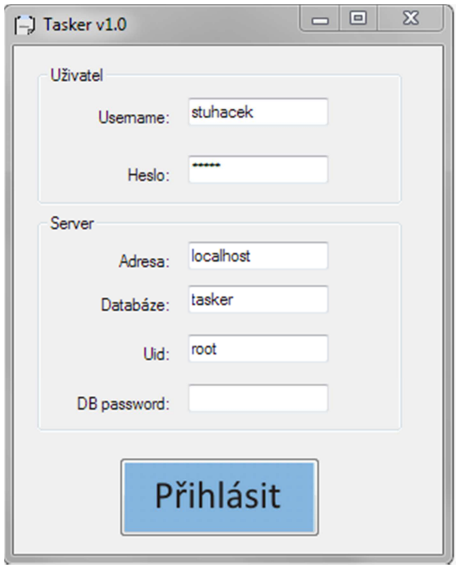
Obrázek 3.11: Znázornění komunikace

3.3. C# aplikace

Součástí zadání bakalářské práce byl také požadavek, aby bylo možné k systému přistupovat též z prostředí PC na platformě Windows (viz Obrázek 3.11). Měl jsem možnost výběru jazyka, ve kterém napíši aplikaci, která tento přístup umožní. Zvolil jsem tedy C#, což je jednoduchý objektově orientovaný jazyk, jehož základy pocházejí z jazyka C[8]. Je to jeden z často používaných jazyků pro tvorbu databázových programů, webových služeb a formulářových aplikací.

Přihlášení:

Pro přihlášení do aplikace je potřeba znát přihlašovací údaje uživatele a údaje k připojení k databázi (viz Obrázek 3.12).

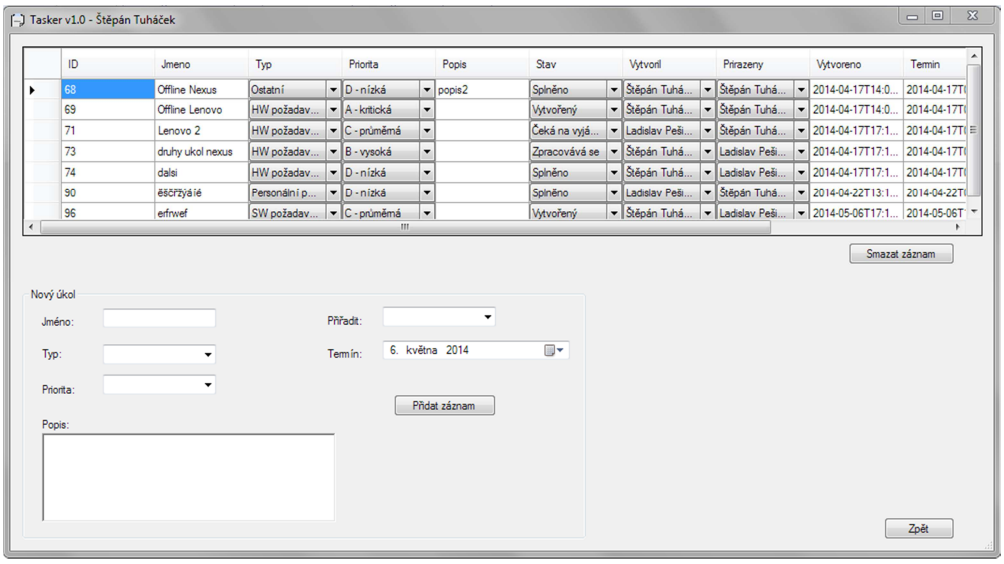


Obrázek 3.12: C# aplikace – přihlášení

Správa úkolů:

Po vybrání tabulky úkolů se zobrazí pohled, ze kterého je možno vytvořit, upravit, či

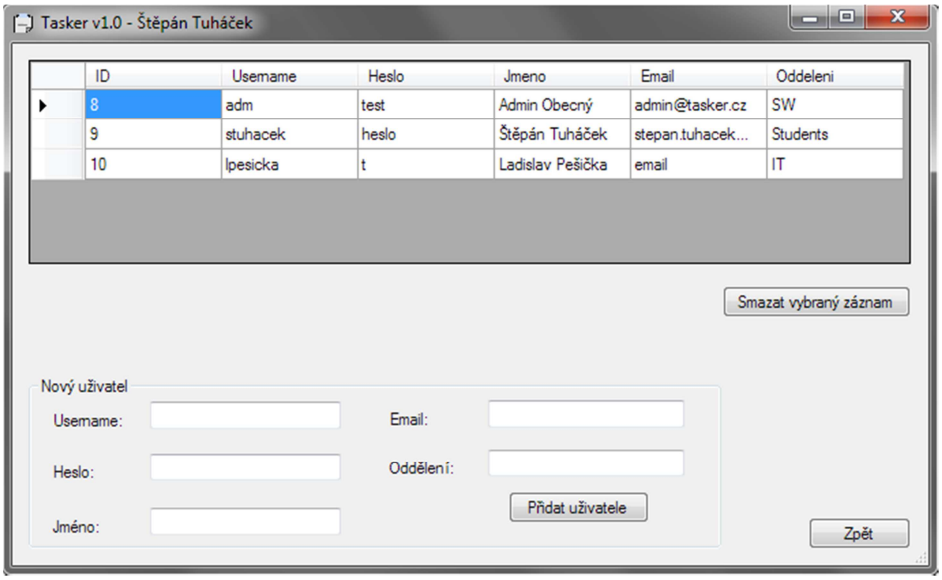
smazat úkol (viz Obrázek 3.13). Pro vytvoření úkolu je potřeba vyplnit jednoduchý formulář a následně potvrdit příslušným tlačítkem. Pro úpravu některého z políček úkolu je třeba stisknout „Enter“ po příslušné úpravě. Následně se zobrazí okno s informací, že úprava proběhla úspěšně. Smazání úkolu je možno provést tak, že označíme celou řádku pomocí kliknutí do prvního sloupce požadovaného záznamu a stiskneme tlačítko „Smazat vybraný záznam“.



Obrázek 3.13: C# aplikace – tabulka úkolů

Správa uživatelů:

Správa uživatelů (viz Obrázek 3.14) je obdobná, jako u předchozí tabulky.



Obrázek 3.14: C# aplikace - uživatelé

4. Vývoj aplikací

Tato kapitola obsahuje popis vývoje aplikací (Android, C#) a služeb (PHP). Jsou zde uvedeny třídy jednotlivých projektů a popsány jejich příslušné metody. U Android aplikace také popíši strukturu projektu.

4.1. Android aplikace

Jak jsem zmínil výše, bylo třeba brát v potaz rozdílné hardwarové a softwarové schopnosti různých mobilních zařízení a verzí systému.

Pro vývoj aplikace jsem používal vývojové prostředí Eclipse spolu s balíkem Android SDK. Aplikace je psána v jazyce Java, pro návrh uživatelských rozhraní a definici řetězců je použito XML. Po zkompilování je vygenerován soubor s příponou `.apk`, který obsahuje veškeré potřebné zdroje pro samostatný chod aplikace. Jedná se o soubor `AndroidManifest.xml`, zkompilované zdrojové kódy, zdrojové soubory uložené mimo kód, knihovny a meta informace[7].

V dnešní době rozšířenosti informací je pro vývojáře obrovská výhoda, že téměř každý problém, na který narazí, již někdo řešil. Velkou oporou je tudíž portál Stackoverflow[6], kde vývojáři pokládají otázky a ostatní se snaží poradit je vyřešit. Nezbytným zdrojem informací jsou také stránky Googlu pro vývojáře[7].

V této kapitole popíši strukturu adresáře s projektem, třídy a jejich metody, případně vysvětlím důvod použití, pokud jsem tak již neučinil výše.

Struktura projektu:

`/bin` - adresář generovaný kompilátorem obsahující `.class` soubory zkompilované z java kódu, které jsou později spolu s potřebnými prostředky (např. obrázky) zabaleny do `.apk` souboru

`/gen` – adresář generovaný vývojovým prostředím Eclipse, obsahuje třídu `R.java`, ve které jsou odkazy k prostředkům

[6] Dostupné z - http://stackoverflow.com/
[7] Dostupné z - http://developer.android.com/index.html#

`/joda-time-2.3-dist` – adresář knihovny `joda-time-2.3.jar`

`/libs` – obsahuje externí knihovny

`/res` – obsahuje zdroje využívané v kódu (např. obrázky, definice řetězců, atd.)

`/src` – zde jsou veškeré třídy se zdrojovým kódem aplikace v jazyce Java

`AndroidManifest.xml` – soubor se základními informacemi o aplikaci potřebný pro systém Android dříve, než dojde k samotnému spuštění aplikace

`ic_launcher-web.png` – ikona, která se zobrazí u příslušné aplikace ve službě Google play, pokud je zde aplikace nahrána

`.classpath` – soubor obsahující jména souborů a jiné závislosti potřebné pro úspěšnou kompilaci a spuštění aplikace

`.project` – automaticky generovaný soubor popisující vlastnosti projektu

Aplikace sestává ze čtrnácti tříd a využívá pět knihoven:

`holographlibrary.jar` – knihovna využívaná při vykreslování grafů

`android-support-v7-appcompat.jar` – podpůrná knihovna, kterou využívá HoloGraph knihovna

`android-support-v4.jar` – podpůrná knihovna, kterou využívá HoloGraph knihovna

`joda-time-2.3.jar` – knihovna pro převod dat a času na řetězce a opačně

`bcprov-ext-jdk15on-1.46.jar` – knihovna poskytující šifrovací algoritmy

4.1.1. Obecné třídy

Do této skupiny řadím veškeré základní třídy (aktivity) projektu, které zasahují do uživatelského rozhraní.

SplashActivity.java

Třída (aktivita), která původně sloužila pro rychlé nastavení adresy hlavního serveru MySQL v kódu. Později bylo však toto nastavení implementováno pomocí Shared preferences. V konečné fázi je tato třída však pouhou uvítací obrazovkou s logem aplikace a informací o verzi aplikace.

SignupActivity.java

Aktivita zajišťující přihlášení uživatele. Volá PHP služby potřebné pro přihlášení uživatele, nahrání informací o všech uživatelích a zaslání hesla pro jeho pozdější zašifrování a následné uložení do databáze SQLite. Obsahuje dvě textové pole a tři tlačítka – Přihlášení, Offline přihlášení a Změna adresy

`accessWebService()`

- metoda pro online přihlášení, v případě úspěchu volá metodu `addUserLocal(User user)`

`localSignup()`

- metoda, která zajišťuje lokální přihlášení – využito například při nedostupném síťovém připojení

- vložené heslo zašifruje algoritmem SHA-512 a poté jej porovná s heslem uloženým v lokální databázi SQLite; po případném úspěchu volá menu aktivitu

`addUserLocal(User user)`

- spouští asynchronní úlohu, která volá službu pro zaslání hesla, které poté zašifruje a spolu s ostatními informacemi uloží do lokální databáze; volá následující metodu

`addUsers()`

- spouští asynchronní úlohu, která volá službu pro uložení základních informací o všech uživatelích, kteří jsou obsaženi v hlavní databázi MySQL

- volá menu aktivitu

MenuActivity.java

Hlavní menu aplikace, resp. rozcestník funkcí aplikace. Obsahuje volby Nový úkol, Statistika, Úkoly a Synchronizace.

`onBackPressed()`

- po prvním stisku tlačítka „Zpět“ zobrazí objekt Toast s informací o možném ukončení, po druhém stisku tohoto tlačítka aplikaci ukončí

CreateTaskActivity.java

Aktivita umožňující vytvořit úkol. Obsahuje několik formulářových prvků, dokáže také přidat fotografii do přílohy, stejně tak tuto fotografii uložit do galerie. Při nedostupné síti volá metodu `localSave()`, jinak je volána metoda `webServiceCreate()`.

`localSave()`

- uloží úkol do lokální SQLite databáze, přesněji do její tabulky `taskOffline` a přepne na menu aktivitu

`static JsonAddTask()`

- asynchronní úkol, který ukládá požadavek do lokální databáze (tabulka `Task`)

- následně volá službu, která jej uloží i do hlavní databáze MySQL

- přílohu převádí z byte pole na řetězec pomocí metody `Base64.encodeToString(byte[] input, int flags)`

- statická z důvodu volání ze synchronizační části, kde je jí úkol předáván v parametrech

ListActivity.java

Vytváří adaptér, který je naplněn všemi úkoly. Ty jsou následně zobrazeny v podobě seznamu barevně odlišeného dle priorit. Tento adaptér je pro takovou

funkčnost upraven, více viz. třída **Utilities.java**. Po stisku na daný úkol se zobrazí jeho detail, resp. je volána následující aktivita.

TaskDetailActivity.java

Načte detail požadavku z lokální databáze a následně naplní získané hodnoty do formuláře. Ten je poté možno upravit a znovu uložit. Pokud požadavek obsahuje přílohu (fotografii), je tato příloha dekódována a převedena na bitmapu, která je upravena dle velikosti aktuální obrazovky. Zobrazení fotografie probíhá do objektu `Dialog`, který lze potvrzovacím tlačítkem opět zavřít. Před každým nahráním fotografie probíhá spuštění garbage collectoru.

StatisticsActivity.java

Aktivita, která používá knihovnu HoloGraph, s jejíž pomocí zobrazuje statistiky úkolů. Pokud aplikace neobsahuje dostatek úkolů pro zobrazení poměru priorit (více než 2 rozdílné priority), tento první graf není zobrazen. Dále zobrazuje informace o úkolech daného uživatele. K tomu slouží metody tříd s ovládáním databáze, tedy viz **DatabaseTaskHandler.java.** Příklad použití knihovny – vytvoření výsečového grafu viz Listing 4.1.

```
// inicializace grafu
PieGraph pg=(PieGraph)findViewById(R.id.piegraph);
// vytvoření první výseče
PieSlice slice = new PieSlice();
// nastavení hodnoty první výseče
slice.setValue(4);
//přidání výseče do grafu
pg.addSlice(slice);
// vytvoření druhé výseče
slice = new PieSlice();
// nastavení hodnoty druhé výseče
slice.setValue(12);
//přidání výseče do grafu
pg.addSlice(slice);
```

Listing 4.1: Příklad užití knihovny HoloGraph

SynchronizeActivity.java

Synchronizační aktivita, která se stará o aktuálnost obou databází (hlavní MySQL/lokální SQLite). Popis algoritmu synchronizace viz kapitola 3.2.4 Synchronizace, proměnlivost sítě.

`addOfflineTasks()`

- načte všechny požadavky z tabulky `TaskOffline` v lokální databázi

- pro každý tento požadavek volá asynchronní úkol `JsonAddTask`, který jej nahraje do hlavní a lokální databáze (tabulka `task`) a následně je z původní tabulky smazán

`accessWebService1JsonListId()`

- získá z hlavní databáze všechna identifikační čísla požadavků

- předá identifikační čísla chybějících požadavků následující metodě

`accessWebService2CompleteTasks(String[] taskIds)`

- doplní úkoly chybějící v lokální databázi podle ID čísel získaných z předávaných parametrů

`accessWebService3JsonLastUpdate(String[] taskIds)`

- porovná data posledních změn u všech požadavků

- případné zastaralé úkoly aktualizuje

4.1.2. Databázové třídy

V této podkapitole jsou popsány veškeré třídy, které slouží k obsluze vnitřní databáze aplikace SQLite.

DatabaseTaskHandler.java

Třída sloužící k obsluze lokální databáze, přesněji řečeno tabulky `Task`. Při zcela prvním spuštění aplikace a přístupu k databázi zajišťuje vytvoření této tabulky pomocí metody `onCreate(SQLiteDatabase db)`.

`addTask(Task task)`

- do výše zmíněné tabulky vloží daný požadavek

- pokud při přístupu k téhle metodě tabulka dosud neexistuje, dojde k jejímu vytvoření

`updateTask(Task task)`

- aktualizuje daný požadavek

- opět je zaručeno vytvoření tabulky

`getPriorities()`

- vrací pole s čísly, která udávají počet úkolů s danou prioritou

- pořadí je následující: A – kritická, B – vysoká, C – průměrná, D - nízká

`getStates()`

- obdobně jako předchozí metoda vrací pole s čísly, ovšem zde vyjadřují počet úkolů s daným stavem

- struktura je následující: Čekání, Vytvořené, Řeší se, Zrušené, Splněné

pozn.: Do stavu „Čekání“ jsou zahrnuty veškeré požadavky, které jsou v jednom z těchto stavů: Čekání, Čeká na dodání, Čeká na vyjádření.

`getMyTasks(String name)`

- vrací číslo s počtem požadavků přiřazených danému uživateli

`getMyDoneTasks(String name)`

- vrátí číslo udávající počet úkolů daného uživatele, které byly buď splněny, či zrušeny

`getAttachment(int taskId)`

- vrací přílohu požadavku s identifikačním číslem uvedeným v parametrech

`getListIds()`

- vrací pole s identifikačními čísly všech požadavků uvnitř lokální databáze

`getTask(int id)`

- metoda vrátí požadavek dle identifikačního čísla uvedeného v parametrech

`tableExists(String tableName)`

- pomocí této metody lze zjistit, zda tabulka se jménem předávaným v parametrech existuje

`getAllTasks()`

- všechny lokálně uložené úkoly uloží do struktury `List`, který poté vrátí

DatabaseOfflineTaskHandler.java

Třída téměř totožná s předchozí, její tabulka `TaskOffline` je však využívána pouze při nedostupném síťovém připojení. Její metody jsou shodné, navíc je zde však tato:

`deleteTask(Task task)`

- smaže požadavek z tabulky `TaskOffline` předaný v parametrech

DatabaseUserHandler.java

Třída slouží pro přístup k tabulce `User`. Tato tabulka udržuje záznamy o všech uživatelích. Ti jsou nahráni při každém úspěšném přihlášení uživatele do aplikace. Opět je zde metoda `onCreate(SQLiteDatabase db)` zajišťující vytvoření tabulky při prvotním spuštění a přístupu do databáze.

`deleteTableUsers()`

- metoda smaže tabulku `User`

- volána během každého úspěšného přihlášení, kdy je celá tabulka smazána a poté obnovena

`addContact(User user)`

- do tabulky vloží uživatele předaného v parametrech

`getUserByUsername(String username)`

- podle username uživatele předaného v parametrech jej načte z databáze a následně vrátí

`getAllUsers()`

- vrací objekt `List` se všemi uživateli v lokální databázi

`tableExists(String tableName)`

- pomocí této metody lze zjistit, zda tabulka se jménem předávaným v parametrech existuje

4.1.3. Pomocné třídy

Zde jsou zařazeny veškeré třídy, jejichž metody či konstruktory využívají ostatní aktivity.

Task.java

Tato třída slouží k vytvoření objektu `Task`. Obsahuje dva konstruktory, jeden s identifikačním číslem a druhý bez něj. V neposlední řadě také všechny příslušné gettery a settery. Struktura úkolu viz Tabulka 4.1.

Název	Typ	Komentář
id	int	Identifikační číslo úkolu
_name	String	Jméno úkolu
_type	String	Typ úkolu
_priority	String	Priorita dle závažnosti
_description	String	Popis problému
_state	String	Aktuální stav úkolu
_reporter	String	Zadavatel požadavku
_assignee	String	Přiřazený uživatel
_date_create	DateTime	Datum vytvoření
_date_due	DateTime	Termín splnění
_date_last_update	DateTime	Datum poslední změny
attachment1	byte[]	Příloha
valid	int	Příznak platnosti
_comment	String	Komentář k úkolu
_upload	int	Příznak nahrání do hlavní databáze

Tabulka 4.1: Struktura úkolu

User.java

Třída sloužící k vytvoření objektu User. Obsahuje čtyři konstruktory a opět příslušné gettery a settery. Struktura uživatele viz Tabulka 4.2.

Název	Typ	Komentář
id	int	Identifikační číslo uživatele
username	String	Přihlašovací jméno uživatele
password	byte[]	Heslo uživatele
name	String	Jméno uživatele
email	String	Email uživatele
department	String	Oddělení
used	String	Příznak lokálního použití
valid	int	Příznak platnosti

Tabulka 4.2: Struktura uživatele

Utilities.java

Pomocná třída, která obsahuje metody využívané některými aktivitami. Obsahuje statickou třídu `PriorityArrayAdapter` pro vytvoření adaptéru, který je využit u rozbalovacího menu s prioritami. Druhá statická třída `MyArrayAdapter` se používá u rozbalovacího menu stavů úkolů a u uživatelů. Dále je zde statická třída `StableArrayAdapter`, která je využita aktivitou `ListActivity.java` zobrazující seznam požadavků. Poslední statická třída `MyTaskParams` využita u asynchronních úloh. Díky této třídě lze zmíněným úlohám předat url adresu, pole identifikačních čísel požadavků, jednotlivý úkol, či příznak určující způsob zpracování úkolu (zda se jedná o přidání z offline databáze, či jde o online vytvoření).

`getView(int position, View convertView, ViewGroup parent)`

- metoda vrací pohled, kterému definuje způsob, jakým se zobrazuje rozbalovací menu ve výchozím stavu (jedná se o přepsanou metodu kvůli barevným rozlišením)

`getDropDownView(int position, View convertView, ViewGroup parent)`

- metoda též vrací pohled, na rozdíl od předchozí metody se zde jedná o zobrazení rozbalovacího menu v jeho expandovaném stavu (též se jedná o přepsanou metodu)

`getColorFromPriority(String item, Context c)`

- metoda vrací řetězec s hodnotou barvy dané priority

`getResizedBitmap(Bitmap bm, int newHeight, int newWidth)`

- bitmapu předanou v parametrech transformuje dle šířky a výšky zadaných v dalších parametrech; následně ji vrátí

`removeUTF8BOM(String s)`

- pokud řetězec předaný v parametrech obsahuje na jeho začátku BOM[8], odstraní jej a následně řetězec vrátí

`isAvailable(String str)`

- pokusí se o připojení na stránku předanou v parametrech

- vrací pravdivostní hodnotu „pravda“ při úspěchu a „nepravda“ při neúspěchu

4.2. C# aplikace

Aplikace byla navržena tak, aby byl servisní deník schopen obsluhovat i člověk ne příliš znalý počítačovým technologiím. Nebylo by žádoucí i z důvodu bezpečnosti, pokud by takový člověk musel například vytvářet uživatele pomocí vkládání SQL dotazů do příkazového řádku.

4.2.1. Programátorská dokumentace

Projekt se skládá ze 4 základních formulářů. Každý formulář má svůj vlastní pohled.

[8] BOM (angl. Byte order mark) – označení pořadí bajtů

FormSignup.cs

Třída slouží pro přihlášení uživatele. Obsahuje tlačítko pro přihlášení a dva graficky oddělené formuláře – jeden pro vyplnění údajů uživatele, druhý pro připojení k serveru.

```
signupClick(object sender, EventArgs e)
```

- zkompletuje údaje zadané do formuláře, zavolá metodu pro vytvoření databáze a dále metodu pro samotné přihlášení

```
openDb(MySqlConnection x)
```

- pokusí se o vytvoření připojení k databázi, případně odchytí výjimky

```
Statement()
```

- provede přihlášení pomocí dotazu, který je následně vykonán na hlavní databázi MySQL

FormMenu.cs

Slouží pouze jako rozcestník mezi tabulkami uživatelů a požadavků.

FormTasks.cs

Obsluhuje tabulku požadavků v hlavní databázi. Umožňuje vytvořit, upravit, či smazat požadavek.

```
getUsers()
```

- z hlavní databáze a její tabulky uživatelů vyzvedne všechny uživatele a vrátí je jako pole řetězců

```
Statement()
```

- naplní tabulku všemi požadavky z hlavní databáze

Dále jsou zde 3 tlačítka umožňující vytvoření a smazání požadavku. Tlačítkem „Zpět“ je případně možné se vrátit do menu aplikace.

FormUsers.cs

Obsluhuje tabulku uživatelů v hlavní databázi. Opět umožňuje vytvořit, upravit, či smazat záznam (uživatele). Třída je svými metodami téměř shodná jako předchozí.

4.3. PHP služby

O výměnu dat mezi Android aplikací a MySQL serverem se stará osm PHP skriptů. Každý soubor obsahuje příslušnou hlavičku deklarující PHP soubor a nastavení české lokalizace. Vždy po připojení k databázi je pomocí funkce `mysqli_set_charset()` nastaveno výchozí kódování UTF8 pro přenos obsahu směrem k databázi a zpět[9]. Pomocí MySQL příkazu `SET NAMES` je také toto kódování uvedeno jakožto výchozí kódování pro posílání příkazů klientem serveru.

Pomocí PHP skriptů je také řešen možný konflikt názvů požadavků, více viz **insertTask.php**.

signupScript.php

- skript, který zajišťuje přihlášení uživatele[10]

- po připojení k hlavní databázi zkontroluje, zda se údaje o uživateli předané v parametrech shodují s nějakým uživatelem uvnitř hlavní databáze

- odešle zpět zprávu o výsledku operace

addAllUsers.php

- skript získá všechny záznamy z tabulky uživatelů, vloží je do objektu JSON a poté odešle zpět aplikaci

pozn.: hesla nejsou z důvodu bezpečnosti posílána, k jejich získání by v opačném případě stačilo znát umístění tohoto skriptu a jeho následné zavolání

addLocal.php

- skript volaný pro uložení hesla uživatele uvnitř aplikace

- nejprve dojde k opětovné kontrole hesla a username, v případě úspěchu odešle kompletní údaje uživatele včetně hesla

getId.php

- z hlavní databáze získá všechna identifikační čísla požadavků a odešle je aplikaci

getLastUpdate.php

- z hlavní databáze získá datum poslední změny požadavku, jehož identifikační číslo je předáno v parametrech a následně jej odešle aplikaci

insertTask.php

- skript, který přijme úkol od aplikace a vloží jej do hlavní databáze

- pokud dojde o pokus vložit požadavek s duplicitním názvem, odpoví aplikaci chybou

- v opačném případě odešle aplikaci informaci o úspěšném uložení spolu s ID číslem, pod kterým je požadavek v lokální databázi uložen

resaveTask.php

- tento skript aktualizuje úkol v hlavní databázi – převezme veškerý obsah úkolu a dle identifikačního čísla jej přepíše

selectTaskById.php

- v parametrech přijme identifikační číslo úkolu, podle něj vyhledá v hlavní databázi jeho obsah a poté jej odešle zpět aplikaci

5. Testování

Vývojové prostředí Eclipse spolu s vývojářským balíčkem od Androidu nabízí několik možností, jak hledat chyby v aplikaci, odstraňovat pády, či zrychlit její celkový běh. Samozřejmě je možné jako ve spoustě jiných vývojářských prostředích používat ladící nástroje, zde se jedná o DDMS[9]. Tento nástroj nabízí kompletní přehled o vyvíjené aplikaci a poskytuje několik komponent pro snadnější vývoj[5], např. zobrazení aktivních procesů, možnost pozastavit probíhající vlákna, atd. Dále je zde možnost výpisu do logu, kde navíc lze využít schopnost vyfiltrovat určité slovo, pod kterým je daný záznam do logu vložen.

Při testování C# aplikace jsem využíval vývojové prostředí Visual Studio 2012. Stejně jako u Eclipse, i zde jsem při testování používal výpis do logu. Kromě toho také ladící mód, díky kterému je možné aplikaci pozastavit v jakémkoliv stavu a prohlédnout si obsah proměnných.

5.1. Testování Android aplikace při vývoji

V prvotních fázích jsem také využíval emulátor systému Android. Ten nabízí emulaci mobilního zařízení – virtuální systém běžící na počítači vývojáře. Díky tomu není poté k testování potřeba fyzického zařízení. Z důvodu emulování instrukční sady procesorů ARM je jeho velkou nevýhodou doba startu systému, načítání aplikací, či celková doba odezvy. Jednou možností, jak zlepšit tuto dobu, je použití ukládání stavu systému při vypínání emulátoru. Dalším možným faktorem zlepšení může být instalace akcelerátoru dodávaného v pomocném programu Android SDK Manager.

Bohužel ani poté není rychlost tohoto emulátoru přívětivá, poohlížel jsem se tedy po jiných možnostech. Jednou takovou se zdál být tzv. přehrávač aplikací s názvem BlueStacks App Player[3]. Rychlost spouštění aplikací se u tohoto programu nedá srovnávat s rychlostí emulátoru. Nicméně vyskytl se problém s připojením k databázi na mém počítači, tudíž jsem musel tuto variantu testování zamítnout.

Při vývoji aplikací na systém Android lze také testovat pomocí fyzického mobilního zařízení připojeného přes USB port. Zde jsem měl také problémy s připojením ke své databázi, nicméně správné nastavení vývojářského nástroje EasyPHP, jeho dílčích komponent a následná úprava nastavení firewallu počítače tyto

[9] DDMS (angl. Dalvik Debug Monitoring Service) – Ladící monitorovací nástroj

problémy odstranila. Byl jsem pak schopen ladit aplikaci na fyzickém zařízení bez jakýchkoliv omezení.

Na straně PHP služeb jsem testování prováděl pomocí výpisu proměnných do souboru. Případné chybové hlášky jsem poslal aplikaci zpět, ta je poté zapsala do logu.

Zpočátku se jednalo pouze o úpravy designu a zjišťoval jsem, jak aplikace vypadá na rozdílných obrazovkách. Dále bylo zapotřebí obstarat bezchybný přenos z Android aplikace do hlavní databáze. Zde se vyskytl problém zmíněný již v kapitole 3.2.3 PHP, JSON, kdy se do této databáze ukládaly nesmyslné znaky. Řešení tohoto problému je zmíněno v téže kapitole.

5.2. Testování výsledné Android aplikace

K testování jsem použil dvě fyzická mobilní zařízení. Prvním z nich bylo mé Lenovo A750, ve kterém se nachází Android 4.0.3. Zařízení má RAM paměť o velikosti 512 MB a 4“ displej. Na tomto zařízení jsem odladil většinu chyb, aplikace zde fungovala plynule. Jediné problémy se však vyskytovaly při občasném výpadku připojení přes USB port, což jsem následně řešil restartováním ADB[10] serveru uvnitř počítače.

Druhé zařízení jsem měl vypůjčené od vedoucího práce, jednalo se o Samsung Google Nexus S. Obsahuje verzi Androidu 4.1.2, paměť RAM i velikost obrazovky jsou totožné jako u předchozího zařízení. Aplikace zde chodila rychleji, alespoň co se nahrávání požadavků týče. Problém se však vyskytl s aktualizací záznamů v lokální databázi, což bylo zapříčiněno způsobem deklarace jednoho sloupce tabulky. Ukázalo se, že tato verze Androidu neumožňuje aktualizovat záznam v tabulce, jehož sloupec je označen jako `PRIMARY KEY`. Další problém, který se také u předchozího zařízení neobjevil, byl s nahráváním fotografie. Dle výpisu z logu bylo zřejmé, že zařízení nemá dostatek paměti. Zjistil jsem také, že systém se před načítáním bitmapy nepokusí nejprve o spuštění garbage collectoru a rovnou shodí aplikaci pro již zmíněný

[10] ADB (angl. Android Debug Bridge) – Nástroj pro komunikaci s emulátorem, či s fyzickým Android zařízením

nedostatek paměti. Problém jsem tedy vyřešil manuálním spuštěním tohoto garbage collectoru před nahráním bitmapy.

Problém se ještě vyskytl při zobrazování/skrývání dialogů a objektů `Toast`. Ukázalo se, že velmi záleží na tom, v jakém kontextu daný objekt vytvoříme, či odkud k němu přistupujeme. Během testování jsem došel k závěru, že pády jsou způsobeny špatnou politikou aplikace se zásobníkem aktivit, jelikož podle výpisu chyb bylo zřejmé, že si po čase práce s aplikací aktivity navzájem zasahují do kontextu.

Aktivity dané úlohy/aplikace jsou ukládány do zásobníku aktivit. Podle jeho pořadí se pak tyto aktivity zobrazují, když se například pohybujeme tlačítkem „Zpět“ v aplikaci. Vstupním bodem většiny aplikací je úvodní obrazovka mobilního zařízení. Uživatel například zvolí ikonu aplikace a ta je spuštěna – vytvoří se nová úloha a zobrazí se hlavní aktivita aplikace, která se stane úvodním bodem zmíněného zásobníku. V případě nedávného použití aplikace se zobrazí aktivita dle aktuálního stavu zásobníku.

Pokud daná aktivita volá jinou, tato další je přenesena do popředí a původní aktivita je zastavena. Zůstává však nadále v zásobníku aktivit. Při stisku tlačítka „Zpět“ je aktuální aktivita vyjmuta ze zásobníku, tedy zrušena. Do popředí se pak dostává aktivita na vrcholu zásobníku. Uživatel takto může projít celý zásobník, dokud se nedostane na domácí obrazovku.

Pořadí aktivit v zásobníku nelze měnit. Lze však upřesnit, v jakém módu se má daná aktivita spustit. Slouží k tomu několik příznaků, kterými se tento způsob definuje. V kódu lze například definovat tyto následující:

`FLAG_ACTIVITY_NEW_TASK` – tento příznak u startu aktivity zajistí, že se stane počáteční aktivitou v historii zásobníku

`FLAG_ACTIVITY_CLEAR_TOP` – pokud je při spuštění dané aktivity její instance již v zásobníku, nevytváří se nová, ale zásobník se vyčistí a tato aktivita se dostane do popředí

`FLAG_ACTIVITY_SINGLE_TOP` – aktivita nebude spuštěna, pokud je právě na vrcholu zásobníku

Způsob chování aktivit vůči zásobníku lze také definovat v souboru `AndroidManifest.xml`, kde je spolu s atributem `android:launchMode` možno zvolit několik různých módů. Aktivitě zobrazující menu aplikace jsem přiřadil hodnotu `singleTask`, která se u tohoto atributu může spolu s několika dalšími vyskytovat. Tento mód zaručuje, že pokud již existuje instance spouštěné aktivity, systém ji přenese do popředí. Spolu s tímto módem jsem při návratu do menu uplatnil výše zmíněný příznak `FLAG_ACTIVITY_CLEAR_TOP`.

Pro jistotu jsem také změnil chování přihlašovací obrazovky, a to tak, že tato aktivita se do zásobníku vůbec neukládá. Zákaz tohoto ukládání lze nastavit atributem `android:noHistory` a jeho příslušnou pravdivostní hodnotou.

5.3. Testování C# aplikace

Bylo zapotřebí odladit případy, kdy je server nedostupný. Dále jsem potřeboval zamezit vložení chybného obsahu uživatelem. Opět se objevil problém s kódováním, kdy se do databáze vkládaly špatné znaky. Řešením bylo vložení příznaku „`N`“ před vkládaný obsah, který indikuje, že jsou vkládána Unicode data (př. `N'Vytvořený'`).

Důležité bylo, aby se úkol, který byl vložen touto aplikací, zobrazil také na aplikaci pro mobilní zařízení. Po již zmíněném odhalení problému s kódováním toho aplikace schopna je.

6. Závěr

Hlavním cílem této práce bylo vytvořit systém evidence požadavků. Bylo třeba analyzovat dosud existující aplikace a stanovit jasné cíle. Pro implementaci takového systému bylo nezbytné se seznámit s vývojem aplikací pro mobilní zařízení s operačním systémem Android. Dále bylo nutné realizovat synchronizaci dvou použitých databází, aby bylo možné delegovat požadavky mezi více uživateli. Dle zadání práce bylo také nutné vytvořit aplikaci spustitelnou na systému Windows, která bude schopna ovládat tento systém evidence požadavků.

Implementoval jsem systém, který umožňuje provádět funkce uvedené v kapitole 2.2 Cíle práce. Sestává z aplikace na mobilní zařízení se systémem Android, kde je využita lokální databáze SQLite. Pomocí této aplikace je možné vytvořit požadavek, který lze delegovat jinému uživateli systému. K tomuto požadavku lze připojit fotografii jako přílohu. Úkol je možné také zobrazit, upravit, a následně aktualizovat. Požadavky se jinak aktualizují dle data poslední změny požadavku v hlavní databázi.

Tato hlavní databáze (MySQL) je umístěna na serveru, s kterým aplikace komunikuje pomocí PHP skriptů. Tyto skripty si předávají obsah pomocí JSON objektů, díky nim je možné předávat záznamy z lokální databáze SQLite do databáze hlavní a naopak.

Dále byla implementována aplikace, kterou lze spustit na operačním systému Windows. S její pomocí je uživatel schopen spravovat hlavní databázi, potažmo systém evidence požadavků.

Aplikace byla testována na dvou mobilních zařízeních a na emulátoru systému Android. Bylo objeveno několik chyb, které byly následně opraveny. Aplikace funguje plynule a je schopna případného nasazení do provozu v menší společnosti.

Reference

1. HoloGraphLibrary - Bitbucket. NADEAU, Daniel. *Bitbucket* [online]. [cit. 2014-04--25]. Dostupné z: https://bitbucket.org/danielnadeau/holographlibrary/wiki/Home
2. Entry. *The Legion of the Bouncy Castle* [online]. [cit. 2014-04-25]. Dostupné z: https://www.bouncycastle.org/index.html
3. App Player. *BlueStacks* [online]. Dostupné z: http://www.bluestacks.com/app-player.html
4. BABIN, Lee. *PHP 5 recipes: a problem-solution approach*. New York: Distributed to the book trade worldwide by Springer-Verlag, c2005, xxi, 646 p. ISBN 15-905-9509-2.
5. MEIER, Reto. *Professional Android application development*. Indianapolis, IN: Wiley, 2009. ISBN 978-047-0344-712.
6. STALLINGS, William. *Cryptography and network security: principles and practice*. 5th ed. Boston: Prentice Hall, c2011, xxiii, 719 p. ISBN 01-360-9704-9.
7. GARGENTA, Marko a Masumi NAKAMURA. *Learning Android: develop mobile apps using Java and Eclipse*. 2nd edition. Sebastopol, CA: O'Reilly, 2014. ISBN 978-144-9319-236.
8. HEJLSBERG, Anders. *The C# programming language*. 4th ed. Upper Saddle River, NJ: Addison-Wesley, c2011, xviii, 844 p. Microsoft .NET development series. ISBN 03-217-4176-5.
9. BORONCZYK, Tim. *Beginning PHP6, Apache, MySQL web development*. Indianapolis, IN: Wiley Pub., c2009, xxvii, 807 p. Wrox beginning guides. ISBN 04-703-9114-6.
10. WELLING, Luke a Laura THOMSON. *PHP and MySQL Web development*. 2nd ed. Indianapolis, Ind.: Sams, c2003, xxix, 871 p. ISBN 06-723-2525-X.
11. GILMORE, W. Jason. *Beginning PHP and MySQL: from novice to professional*. 4th ed. New York: Apress, 2010. ISBN 978-143-0231-141.
12. NIXON, Robin. *Learning PHP, MySQL, JavaScript, and CSS*. 2nd ed. Sebastopol, CA: O'Reilly, 2012, xxi, 556 p. ISBN 14-493-1926-2.

Slovník zkratek

ADB – Android Debug Bridge

ADT – Android Development Tools

API – rozhraní pro programování aplikací

iOS - odlehčená verze operačního systému Mac OS X, používaného v počítačích společnosti Apple

GPS - Global Positioning System

JSON – JavaScript Object Notation

MIT - Massachusetts Institute of Technology

MySQL – databázový systém

OS – Operating System

PDF – Portable Document Format

PHP – Hypertext Preprocessor

PIN – Personal Identification Number

RAM – Random-access Memory

USB – Universal Seriál Bus

URL - Uniform Resource Locator

XML – Extensible Markup Language

Přílohy

Příloha A: Uživatelská dokumentace

V této kapitole popíši, co vše jednotlivé aplikace uživateli nabízí a jak se s nimi zachází.

A.1: Android aplikace

Instalace

Nejprve je třeba v systémovém nastavení mobilního zařízení povolit funkci „Neznámé zdroje“. Soubor `Tasker.apk` uložíme na paměťovou kartu, ze které potom pomocí průzkumníka aplikaci nainstalujeme. Minimální verze systému Android pro tuto aplikaci je verze Android 3.2.

Přihlášení

Pokud se jedná o první přihlášení do aplikace, je potřeba, aby mělo mobilní zařízení dostupné síťové připojení. Dále je nutné nastavit adresu serveru stisknutím tlačítka „Změna adresy“ a její následné potvrzení. Pokud se po potvrzení objeví chybová hláška kontroly adresy, je třeba postup opakovat znovu.

Po přihlášení se zobrazí menu aplikace. Při dalším přihlášení můžeme již využít offline přihlášení, je však vhodné jej využívat pouze při nedostupném síťovém připojení.

Vytvoření požadavku

Z menu aplikace stiskneme ikonu pod nadpisem „Nový úkol“. Následně se zobrazí formulář, jehož vyplnění je intuitivní. V případě, že chceme k požadavku připojit fotografii, stiskneme ikonu fotoaparátu, jehož funkce je poté spuštěna. Po potvrzení fotografie je možné požadavek vytvořit.

Statistika

Druhou možností menu je statistika, v níž jsou zobrazeny dva grafy a informace o úkolech daného uživatele.

Detail úkolu

Pod sekcí „Úkoly“ nalezneme seznam všech úkolů, které byly lokálně uloženy. Výběrem z tohoto seznamu se dostaneme na detail tohoto úkolu. Zobrazí se opět formulář obohacený o informace o vytvoření a poslední změně požadavku. Je možné jej změnit, či například přidat komentář a potom znovu uložit, k čemuž je zapotřebí, aby mělo zařízení dostupné síťové připojení.

Synchronizace

Tato funkce je možná také pouze při dostupném síťovém připojení. Dojde k nahrání všech úkolů, které byly nahrány v režimu offline, dále se stáhnou všechny požadavky od ostatních uživatelů a pak se všechny úkoly aktualizují podle data poslední změny.

A.2: C# aplikace

Instalace

Pro spuštění této aplikace je potřeba mít nainstalovaný .NET Framework 4.5[11]. Poté stačí pouze spustit soubor `Tasker.exe`.

[11] Dostupné z - http://www.microsoft.com/cs--cz/download//details.aspx?id =30653

Přihlášení

Pro přihlášení je třeba pouze vyplnit požadované údaje o uživateli a přihlašovací údaje pro připojení k serveru.

Ovládání databáze

Z menu vybereme požadovanou tabulku a následně se zobrazí její obsah. Pod ním je formulář pro vyplnění nového záznamu a tlačítko umožňující jeho vložení. Pokud chceme smazat daný záznam, klikneme na první sloupec požadovaného řádku a stiskneme tlačítko „Smazat vybraný záznam“. Pro úpravu daného políčka v řádku stiskneme po každé úpravě „Enter“ pro potvrzení. Následně se zobrazí informace o úspěšné změně.

Obsah CD

`bin` – spustitelné soubory

`libs` – knihovny potřebné pro kompilaci

`server` – generující skript databáze, PHP skripty

`src` – zdrojové kódy

`BP_Š_Tuháček_2014` – tisknutelný i zdrojový text