UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 František Horák

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Aplikace pro sledování a záznam odpracovaného času

František Horák

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**František Horák**
Osobní číslo:	**I13131**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro sledování a záznam odpracovaného času**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je vytvoření aplikace časoměřiče pro sledování odpracované doby uživatele osobního počítače. Časoměřič s ručním ovládáním sleduje odpracovanou dobu s ohledem na jednotlivé úkoly projektového řízení, resp. další požadavky, tvořící pracovní náplň daného uživatele.

Jednalo by se o desktopovou aplikaci s grafickým uživatelským rozhraním, pracující v offline režimu, tzn. pro jednotlivé úkoly by se zaznamenávala doba, která uplynula od spuštění záznamu práce na nich. Data budou ukládána do vlastního datového úložiště (soubory/databáze). Jednotlivé úkoly bude možné načítat z databáze systému Redmine a další přidávat/odebírat ručně. Aplikace by tak vlastně tvořila klienta Redmine. Dobu strávenou na úkolech z Redmine by aplikace následně zapisovala do databáze systému Redmine. Odpracovanou dobu za jednotlivé dny, rozpadající se na jednotlivé úkoly, resp. projekty, by aplikace zobrazovala formou výpisů/reportů (s možností exportu do souboru formátu CSV).

V teoretické části práce se předpokládá stručný úvod do projektového řízení, resp. opodstatnění potřeby sledování odpracované doby na jednotlivých úkolech, dále stručné přiblížení systému Redmine a popis nezbytné části jeho úložiště dat. Součástí teoretické části bude i rešerše existujících obdobných klientů.

V praktické části práce se předpokládá návrh a implementace programového řešení v jazyce Java. Při implementaci je třeba ctít požadavek provozovat aplikaci na platformě Linux i Windows.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

Redmine [online]. [cit. 1. listopadu 2016]. Dostupné na internetu: <http://www.redmine.org/>.

MySQL 5.5 Reference Manual [online]. [cit. 1. listopadu 2016]. Dostupné na internetu: <http://dev.mysql.com/doc/refman/5.5/en/>.

Martin, R. C. Čistý kód. Brno : Computer Press, 2009. ISBN 978-80-251-2285-3.

Vedoucí bakalářské práce: **Ing. Petr Veselý**
Katedra softwarových technologií

Datum zadání bakalářské práce: **31. října 2016**
Termín odevzdání bakalářské práce: **12. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 12. 5. 2017

František Horák

PODĚKOVÁNÍ

Rád bych na tomto místě poděkoval především dvěma lidem. Panu Ing. Petru Veselému, vedoucímu mé bakalářské práce, za cenné rady a připomínky. A panu Ing. Viktoru Patrasovi, za velké množství času a rad, jež mi během bakalářské práce věnoval. Velmi si vážím vstřícnosti obou pánů, která mi byla poskytnuta.

Velké poděkování patří také mé rodině za podporu a trpělivost projevovanou po celou dobu mého studia.

ANOTACE

Bakalářská práce se zaměřuje na vytvoření desktopové aplikace pro sledování a záznam odpracovaného času. Jako programovací jazyk je zvolena Java. Nově vytvořený nástroj komunikuje s databází systému Redmine, který je v práci představen. Následně pak jsou uvedeny podobné vybrané nástroje a požadavky na novou aplikaci. Dále jsou představeny použité technologie a samotná implementace zdrojového kódu. Poslední kapitolu tvoří uživatelská příručka.

KLÍČOVÁ SLOVA

Redmine, MySQL, JavaFX, databáze

TITLE

Apps for tracking and recording of time worked

ANNOTATION

This bachelor paper focuses on creating a desktop application which serves for tracking and recording of working time. Java is chosen as a programming language. The newly created tool communicates with the database of Redmine system which is introduced in the paper. Subsequently, there are similar chosen tools as well as new application requirements listed. Further, technologies used and the actual source code implementation are presented. The last chapter of the paper is the user manual.

KEYWORDS

Redmine, MySQL, JavaFX, database

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

MySQL	My Structured Query Language
CSV	Comma-separated values
GUI	Graphical User Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
PC	Personal Computer

ÚVOD

V dnešní době chce mít každý zaměstnavatel přehled o tom, jak pracují jeho zaměstnanci. Je mnoho možností, jak tyto informace získat. Některá z těchto možností ale mají jisté nevýhody. Mezi ně patří neobliba u zaměstnanců, technické problémy, finanční náročnost, či jiné. Jedno z často používaných možností je nainstalování kamerového systému, ale to je spojeno s velkou dávkou odporu zaměstnanců. Mnoho firem využívalo a využívá zápis na papír, co kdy a kde pracovník dělal. Toto řešení, ale není přesnou a ekonomickou volbou. Zaměstnance zdržuje a údaje lze snadno zapsat chybně. S nástupem informačních technologií se začali rozvíjet aplikace, které by tento problém vyřešily. Vznikly takzvané nástroje pro sledování činnosti. Zjištěné informace neslouží jenom pro sledování pracovitosti zaměstnance, ale poskytují i přehled o tom, kolik času zabere vykonání daného úkolu. Toho může být využito například u nové podobné zakázky a stanovení časové náročnosti daného projektu.

Existuje velké množství aplikací řešící tento problém. Bohužel každá firma má svůj systém, jak a kam chce tato data zaznamenávat. Jelikož požadavky jsou velmi rozdílné, nelze vytvořit jednu univerzální aplikaci, která by vyhovovala všem klientům.

V teoretické části bude nejprve představen systém, ze kterého nově vytvořená aplikace vycházet. Dále budou stručně představeny nalezené podobné aplikace spolu s jejich nedostatky. Požadavky na novou aplikaci budou představeny v následující kapitole. Ke konci teoretické části budou popsány použité technologie.

Praktická část se bude zabývat nejprve návrhem aplikace. Poté budou představeny použité techniky, kterých bylo třeba pro plnou funkčnost aplikace. Dále pak budou představeny použité komponenty JavaFX. Samozřejmostí bude popsání použitých tříd a programátorských postupů. K závěru praktické části budou popsány problémy, které se vyskytly během implementace. Bude zde i vysvětleno, jak byly tyto problémy vyřešeny. Poslední kapitolu této části tvoří uživatelská příručka aplikace.

1 UVEDENÍ DO PROBLEMATIKY

Téma bakalářské práce vychází z konkrétního požadavku společnosti využívající systém Redmine[1]. Ten je určen pro řízení projektů, tzn. zejména plánování jednotlivých úkolů, sledování jejich plnění a odpracovaného času na nich. Pro komplexní práci se systémem slouží jeho webové rozhraní. Uživatel si však musí pamatovat od kdy a do kdy, co dělal. Bohužel není možné si vše zapamatovat. Psaní poznámek zaměstnance zbytečně zdržuje. Z toho vyplynulo téma práce jako požadavek na jednoduchého klienta, který by průběžně zaznamenával odpracovaný čas na jednotlivých úkolech a tento pak vložil do systému.

1.1 Projektové řízení

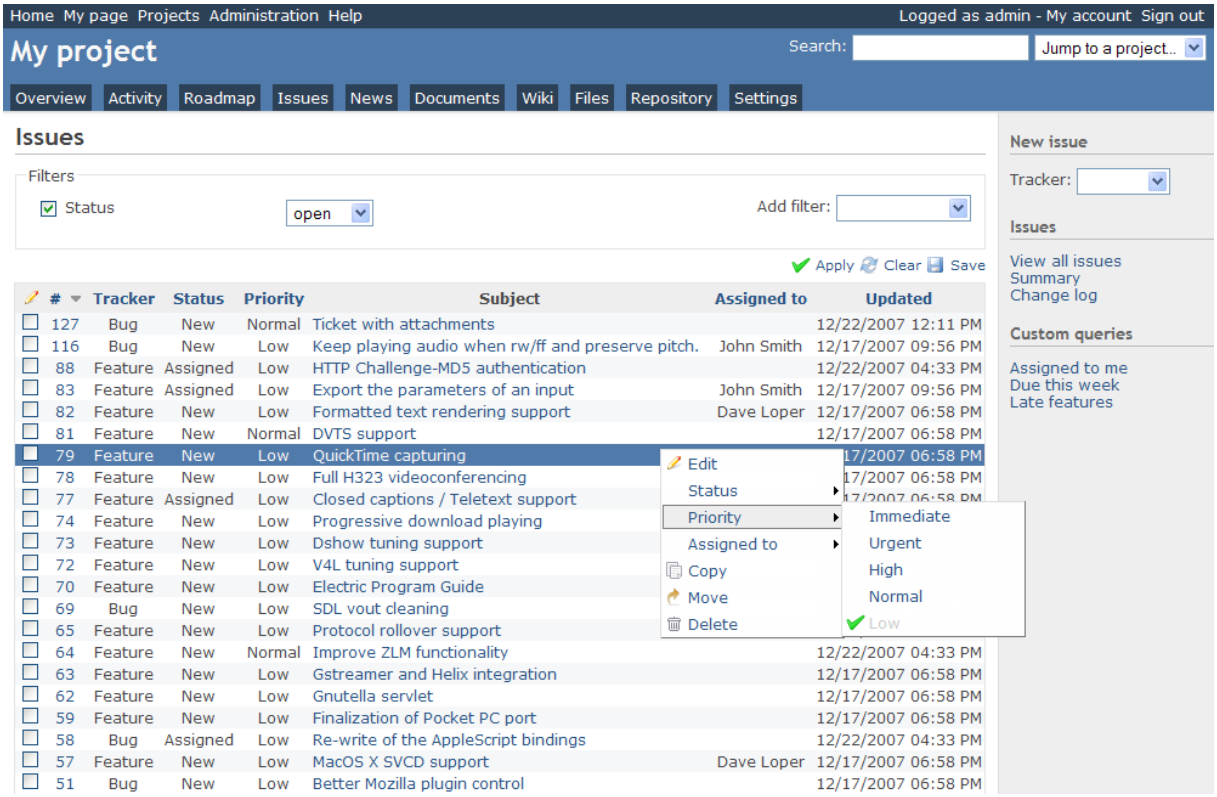
Záznam a sledování odpracované doby má několik významů. V první řadě to jsou důležité informace pro vedoucí zaměstnance. Konkrétně systém Redmine umožňuje vytvářet projekty a úkoly. Zaměstnanci jsou přiřazeny úkoly, na kterých má pracovat. Na těchto úkolech upravuje údaje, jako je míra splnění, odpracovaný čas a jiné. Vedoucí pracovník pak může sledovat, jak jsou dílčí úkoly projektu plněny. Má tak dokonalý přehled o praci zaměstnanců, aniž by s nimi musel mluvit osobně. Hlavním důvodem vzniku této bakalářské práce je zjednodušení práce zaměstnance s tímto systémem. Zjednodušení spočívá v mnoha ohledech. Nově vytvořená aplikace bude zaznamenávat veškerou odpracovanou dobu. Odpadá pak starost pamatovat si, případně psát si, co kdy pracovník dělal. Bude tak zrychlena nejen práce zaměstnance, ale předejde se i případným omylům. Další výhodou je, že nově vytvořená aplikace bude počítat odpracovaný čas automaticky. Uživatel pak jen zvolí, na kterém úkolu pracuje a po skončení jen zastaví časoměřič. Veškeré záznamy jsou ukládány na paměťové médium. Uživatel pak má možnost využít i zpětnou kontrolu. Do aplikace je možné přidat své vlastní úkoly. Toho může být využito například, když ještě daný úkol není zapsán do systému Redmine, ale už na něm zaměstnanec pracuje. Po přidání úkolu do systému, může uživatel odpracovaný čas na něj přenést. Aplikace tedy může sloužit i pro soukromé účely. Ovládání bude jednoduché a přitom plně dostatečné pro běžného zaměstnance. Ten, který bude chtít jen zobrazovat své úkoly a nahrávat své odpracované časy, nebude potřebovat webové rozhraní Redmine.

1.2 Systém Redmine

Vytvořená aplikace ve své podstatě slouží jako klient systému Redmine. Bohužel tento systém poskytuje pouze webové rozhraní a není jej možné využívat off-line, což bylo hlavním

[1] Je vysvětleno v kapitole 1.2

nedostatkem. Redmine je webový nástroj pro správu projektů a sledování práce zaměstnanců. Je poskytován pod licencí open source a využívá framework Ruby on Rails[2]. Nabízí dvě úrovně oprávnění. První je administrátorská, která má na starosti nastavení a údržbu systému. Druhá role je uživatelská, se kterou lze spravovat projekty a jeho úkoly. Při přihlášení do tohoto systému se uživateli zobrazí seznam úkolů, na kterých má pracovat. Po vybrání určitého úkolu má možnost zadat odpracovaný čas, komentář, případně upravit další vlastnosti. Nedisponuje časoměřičem, uživatel tak musí odpracovanou dobu počítat sám. Dalším nedostatkem je možnost zápisu dat jen v případě připojení do tohoto systému. Dále poskytuje vícejazyčné prostředí, podporu více databázových serverů a zásuvných modulů. Disponuje velkou škálou možností, jak sledovat stavy úkolů. Ukázka tohoto prostředí je znázorněna na obrázku 1. [1]



Obrázek 1: Prostředí Redmine

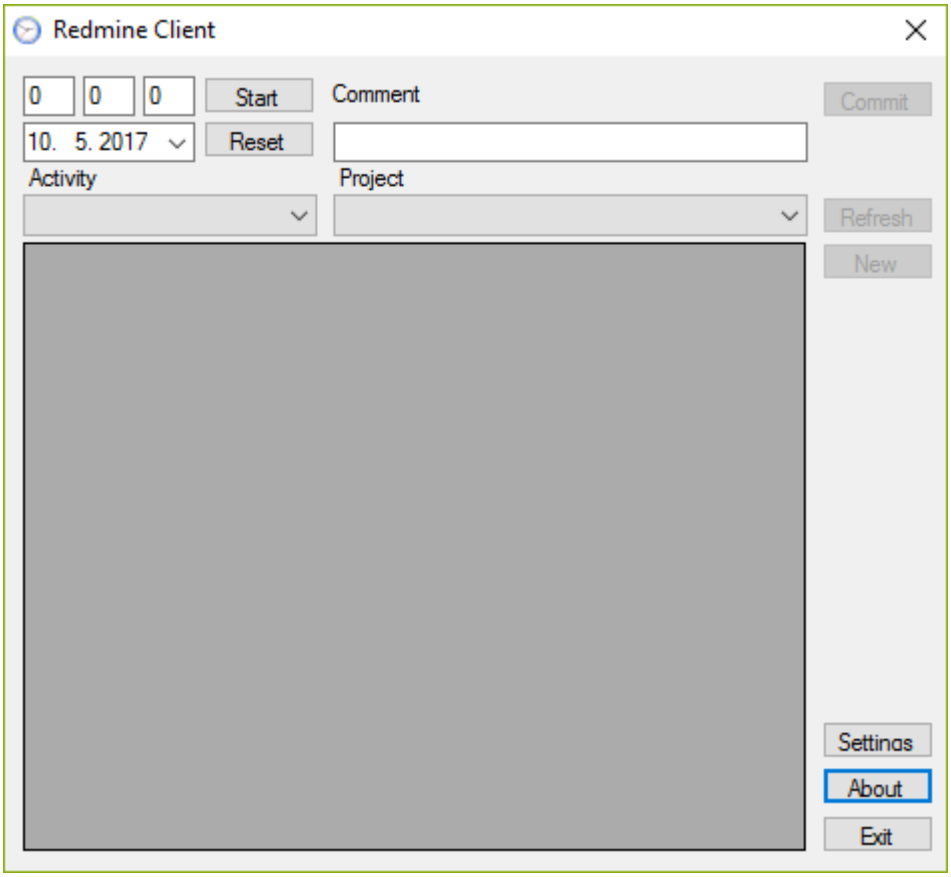
1.3 Nástroje pro sledování činnosti

V této kapitole budou stručně popsány nalezené nástroje pro sledování odpracovaného času. Uvedené aplikace byly otestovány a byly vyzkoušeny jejich možnosti. Získané informace jsou stručně popsány a pro přehled jsou uvedeny obrázky jednotlivých aplikací. První uvedenou aplikací je nástroj postavený přímo pro systém Redmine.

[2] Ruby on Rails je framework pro vývoj webových aplikací. Tento framework je napsaný v jazyce Ruby.

1.3.1 Redmine Client

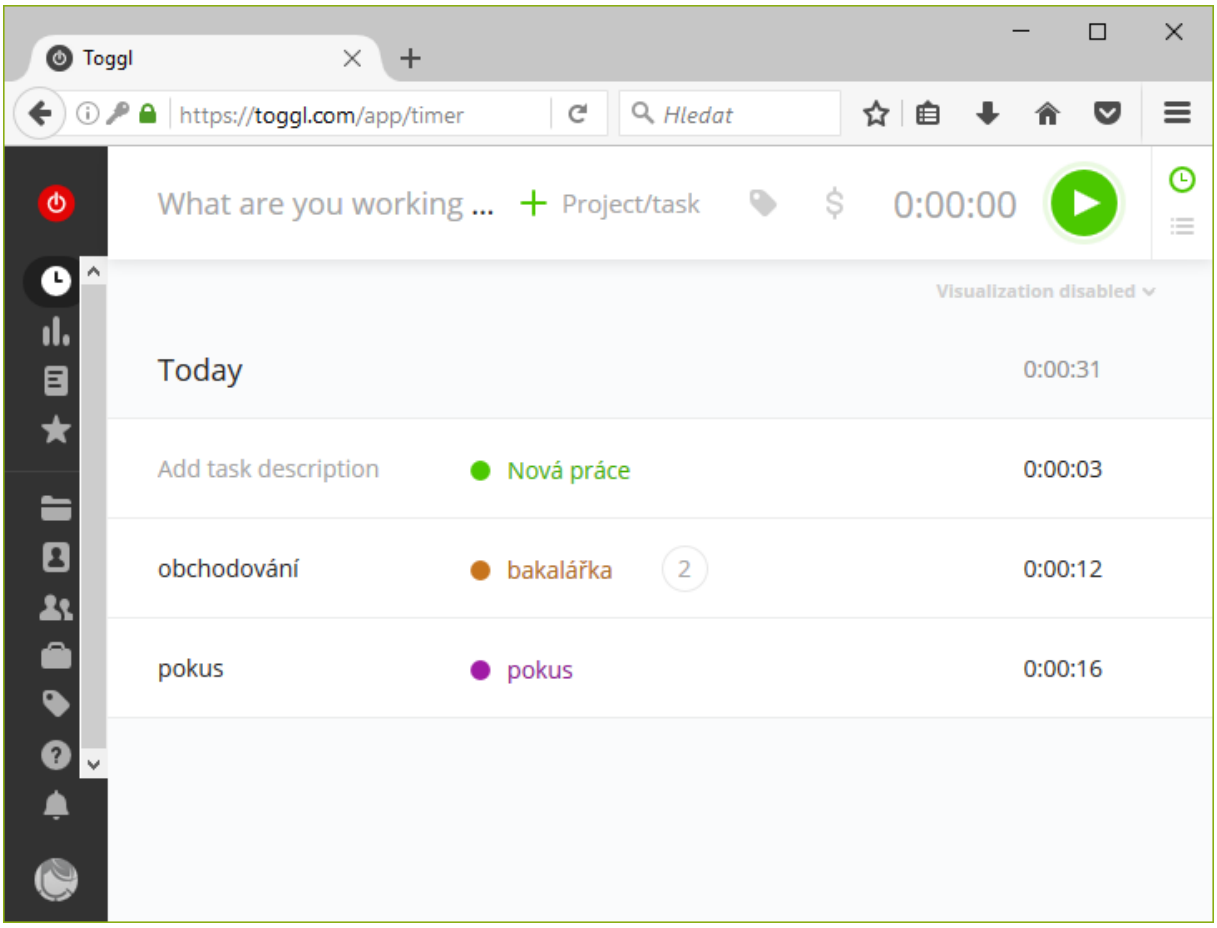
Redmine Client je aplikace vyvinutá přímo pro Redmine. Poskytuje velmi mnoho možností, jak systém Redmine ovládat. Existuje verze pro Windows, OS X i Linux. Tento nástroj je však placený. Roční poplatek činí 10€. Největší nevýhodou je, že chod aplikace je závislý na nepřetržitém spojení s databází. [2]



Obrázek 2: Redmine Client

1.3.2 Toggl

Jedná se o webovou aplikaci, ve které je možné vytvářet si projekty. Na jednotlivých projektech si pak lze měřit odpracovaný čas. Umožňuje vytvářet skupiny a poskytuje přehledné grafy, zobrazující odpracované doby. Nenabízí však propojení s databází. Zároveň ji nelze použít jako desktopovou aplikaci. Další nevýhodou je, že nelze vytvářet úkoly, týkajících se jednotlivých projektů a neumožňuje tedy stromovou strukturu projektů. Největší nevýhodou je, nemožnost použití bez připojení k internetu. Aplikace poskytuje přehledné prostředí, které je znázorněno na obrázku 3. [3]



Obrázek 3: Aplikace Toggl

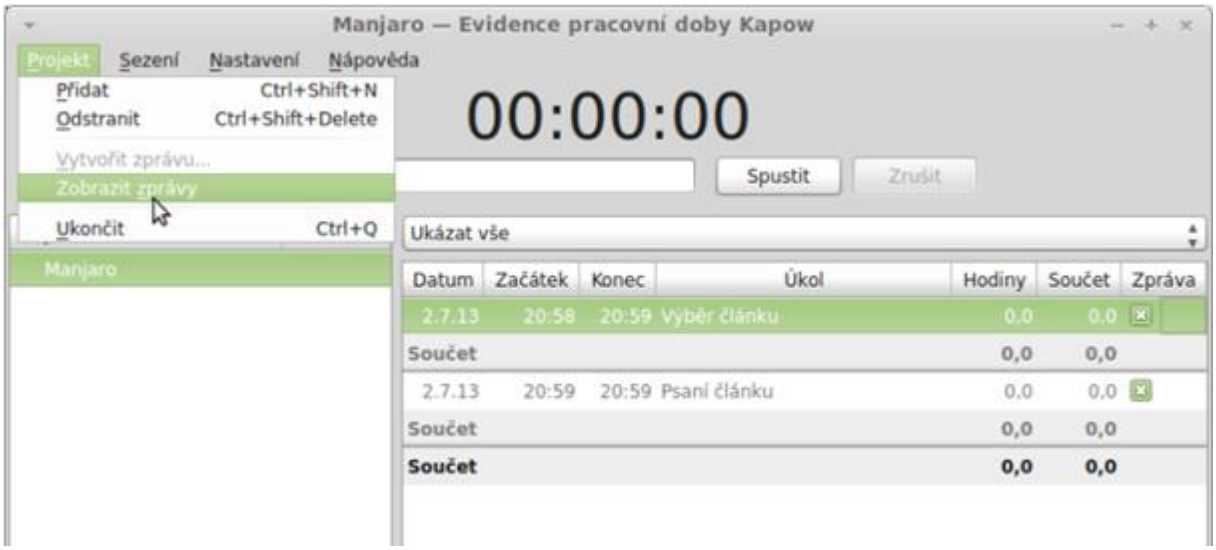
1.3.3 Time Tracker – Timesheet

Tato aplikace je určena pro zařízení se systémem Android. Uživatel si vytváří nové úkoly a zapíná a vypíná časoměřič. Tato aplikace také nekomunikuje s databází. Umožňuje exportovat data do souboru XML, či CSV. Neexistuje však verze pro jiný operační systém. [4]

Obrázek 4: Aplikace Time Tracker – Timesheet

1.3.4 Kapow

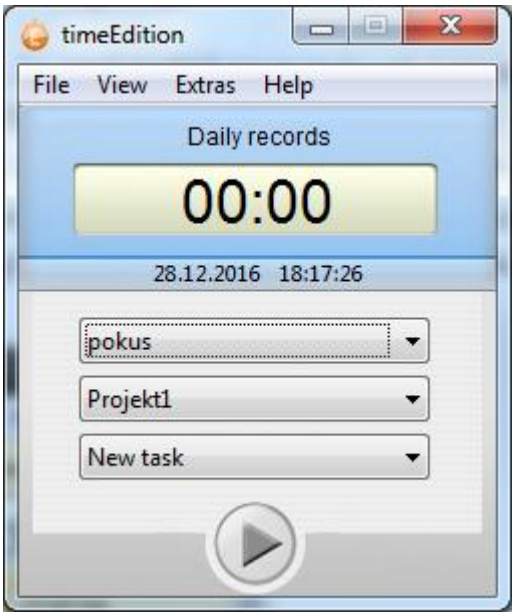
Aplikace je navržena pro operační systém Linux. Umožňuje sledovat více činností najednou. Nedisponuje však funkcí propojení s databází, ani exportováním dat. Na druhou stranu poskytuje intuitivní jednoduché ovládání. [5]



Obrázek 5: Aplikace Kapow

1.3.5 TimeEdition

Tento program je navržen pro operační systémy Windows, Linux i Mac OS. Jedná se o jednoduchou aplikaci. Licence je bezplatná, avšak postrádá český jazyk. Nabízí exportování dat do souborů CSV či XML. Rovněž umožňuje provádění zálohy a načítání z vlastní databáze. Neumožňuje práci s jinou databází a nezabývá se například problémem přechodu PC do režimu spánku. [6]



Obrázek 6: Aplikace TimeEdition

1.4 Vymezení požadavků na novou aplikaci

Aplikace by měla splňovat následující požadavky:

- desktopová aplikace naprogramovaná v jazyce Java s použitím grafického znázornění JavaFX,
- bude načítat pouze projekty a úkoly přihlášeného uživatele,
- aplikaci bude možné používat i v okamžiku nepřipojení k databázi systému Redmine,
- bude disponovat vlastním časoměřičem a zaznamenávat odpracovanou dobu,
- data budou průběžně zálohována, aby nedocházelo ke ztrátám,
- přehledné a intuitivní ovládání aplikace,
- projekty a úkoly budou zobrazeny ve stromové struktuře s možností skrývat jednotlivé větve,
- záznamy o odpracovaných časech budou ukládány do souborů, kde jeden soubor odpovídá jednomu dni, z názvů těchto souborů lze odvodit, který den je zde uložen,
- lze otevřít soubory s odpracovanou dobou i z jiného dne,
- zobrazení obsahu souborů je možní jen uživateli, jenž je autem této odpracované doby,
- záznam o práci jednotlivého dne lze exportovat do souboru CSV,
- odpracovaný čas lze upravovat dle potřeby uživatele,
- lze přidat vlastní úkoly,
- musí být vyřešen problém vypnutí PC a přechodu PC do režimu spánku bez ztráty dat,
- možné použití na operačním systému Windows i Linux.

2 POUŽITÉ TECHNOLOGIE

V této kapitole budou popsány použité technologie aplikace. Nejprve bude přiblížena databáze MySQL a grafické zobrazení JavaFX. Poté budou představena vlákna, bez nichž by nebylo možné aplikaci obsluhovat. V další části budou uvedeny knihovny, jež byly v aplikaci použity. V závěru této kapitoly budou stručně popsány programátorské techniky, bez nichž by nebyla funkčnost a spolehlivost aplikace zaručena.

2.1 MySQL

Systém Redmine i vytvořená aplikace využívají databázi MySQL[3]. Jde o relační databázový systém vlastněný společností Oracle Corporation[4]. Je šířen pod licencí Open Source. Databáze slouží pro uchování dat nemalého množství. Data jsou ukládána do jednotlivých tabulek, kde každý řádek znázorňuje právě jeden záznam a každý sloupec jednu vlastnost záznamu. S daty v tabulkách se pracuje pomocí dotazů, které pochází z jazyka SQL[5]. Jedná se tedy o neprocedurální dotazovací jazyk.

V příkladě 1 je vytvořena tabulka, která bude zaznamenávat uživatele s údaji „ID“ a „NAME“. V příkladu 2, je ukázáno, jak přidávat nové data, v tomto případě nové uživatele do tabulky. Data uložená v tabulce lze snadno získat. V příkladu 3 lze vidět, jak data získávat z databáze. První řádek vypíše všechny uživatele, druhý řádek vypíše všechny uživatele, jejichž jméno je Václav a na posledním řádku je vypsáno pouze jméno uživatele, který disponuje hodnotou 58 ve sloupci “ID“. Již zapsaná data lze snadno odstranit, jako je tomu v ukázce příkladu 4. [7]

Příklad 1
`CREATE TABLE UZIVATELE (ID INTEGER, NAME VARCHAR(30));`

Příklad 2
`INSERT INTO UZIVATELE VALUES ('20', 'Jan');`
`INSERT INTO UZIVATELE VALUES ('21','Václav');`
`INSERT INTO UZIVATELE VALUES ('58','Anna');`

[3] databázový systém uplatňující relační databázový model
[4] společnost vyvíjející především relační databáze
[5] Structured Query Language

Příklad 3
SELECT * FROM UZIVATELE;
SELECT * FROM UZIVATELE WHERE NAME='Václav';
SELECT NAME FROM UZIVATELE WHERE ID='58';

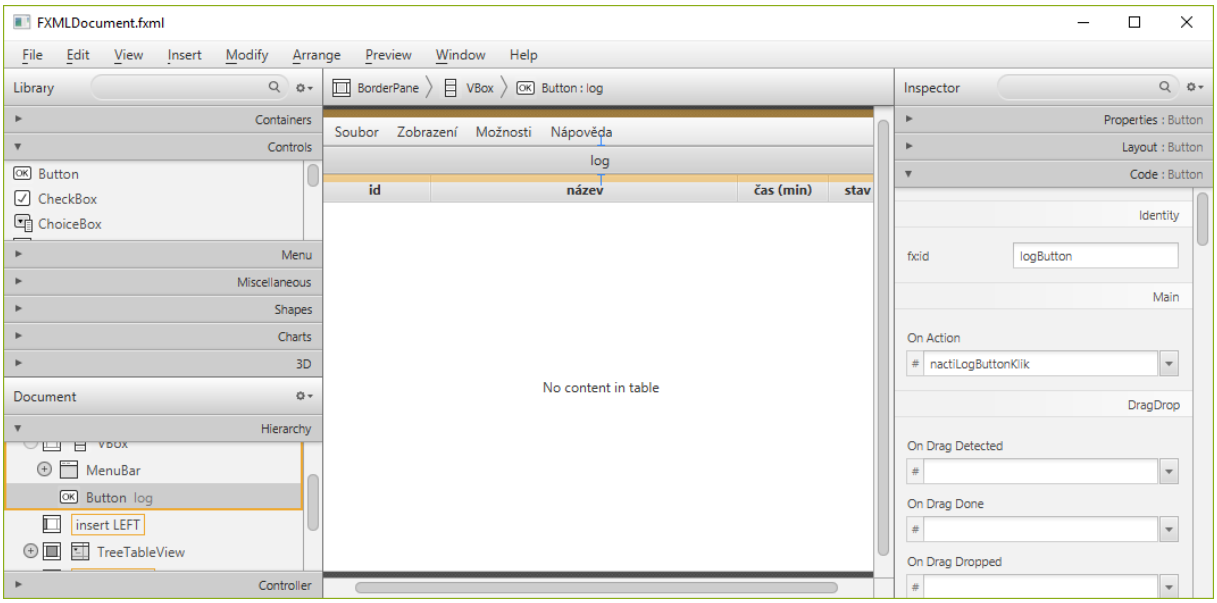
Příklad 4
DELETE FROM UZIVATELE WHERE NAME='Jan';

2.2 JavaFX

JavaFX[6] je součást programovacího jazyka Java, kterému poskytuje vykreslení grafických oken a prvků. Jde o moderní a perspektivní alternativu k rozhraní Swing[7]. JavaFX je modernější, rychlejší a umožňuje využívat CSS a HTML. Součástí jazyka Java se stala od verze 8, která vyšla v roce 2014. K jednoduchému vytváření grafického okna a rozložení jeho komponent slouží program JavaFX Scene builder. Stažení této aplikace je možné na [5]. Tato aplikace pracuje se soubory typu FXML. Je samostatná a používá se jako plugin do programů Eclipse[8], Netbeans[9], či IntelliJ Idea[10]. Při založení projektu typu JavaFX v těchto programech je automaticky vytvořen mimo jiné i soubor typu fxml. Při otevření tohoto souboru je automaticky spuštěn JavaFX Scene Builder. Na obrázku č. 7 je ukázka tohoto prostředí. V levém sloupci se nachází komponenty, které je možné umístit na okno aplikace zobrazené v prostředním panelu. Ten je zároveň náhledem na okno, které bude vykresleno po spuštění aplikace. V pravém sloupci této aplikace se nachází vlastnosti aktuálně vybrané komponenty. Hodnota v kolonce “fx:id“ slouží pro komunikaci s vývojovým prostředím, tedy například NetBeans. [8], [9], [10]

[6] moderní GUI rozhraní programovacího jazyka Java
[7] GUI jazyka Java od roku 1995
[8] vývojové prostředí umožňující využití pluginů
[9] svobodné vývojové prostředí společnosti Oracle Corporation, toto prostředí bylo použito při implementaci aplikace

[10] komerční vývojové prostředí aplikací



Obrázek 7: JavaFX Scene builder

2.2.1 Ukázka práce s komponentou TreeTableView

Pro zobrazení stromové struktury tabulky obsahuje JavaFX komponentu *Treetableview*. Prvním prvek je takzvaný *root*, který si lze představit jako kmen stromu. Na něm může být několik větví a na nich další. Toto větvení může pokračovat do velkých rozměrů. Ve spuštěné aplikaci je pak umožněno u rozvíjejících se větví jejich obsah zobrazit, nebo skrýt.

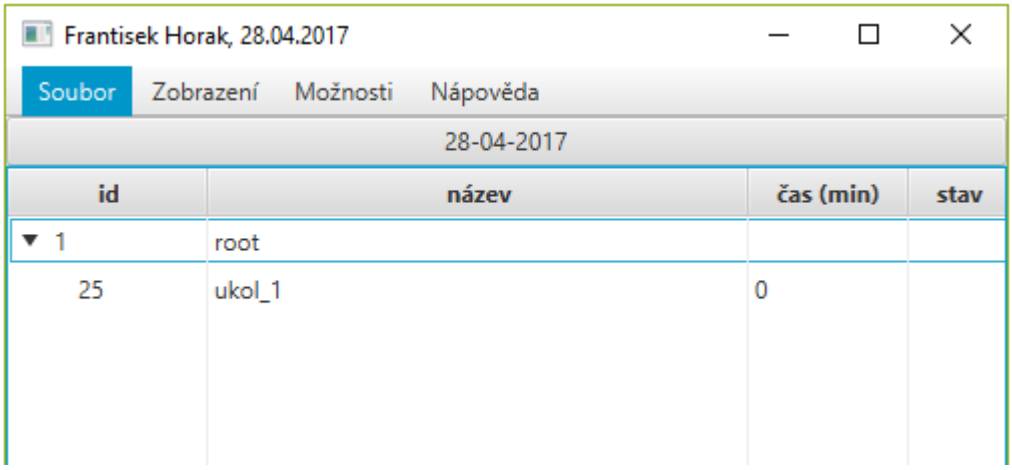
Jednotlivé sloupce v tabulce zobrazují konkrétní atributy zobrazeného prvku. Hodnoty, které jsou v tabulce zobrazovány, musí být typu *Property*. Jde-li o řetězec, proměnná musí být typu *StringProperty*. Celé číslo je pak analogicky typu *IntegerProperty*.

V příkladu 5 je uvedena ukázka, jak probíhá naplnění tabulky. Tento příklad je výtah z této bakalářské práce. *SimpleData.java* je třída, která poskytuje potřebné typy proměnných. Do této třídy jsou transformovány projekty a úkoly uživatele, aby je bylo možné v tabulce společně zobrazit. V konstruktoru třídy *SimpleData.java* je prvním parametrem *id*, dále pak název, odpracovaný čas, logická hodnota, říkající jde-li o projekt a poslední hodnotou je stav. Hodnota „id“ je zobrazena v prvním sloupci, který se jmenuje „idColumn“. Analogicky jsou zobrazeny ostatní hodnoty mimo logickou hodnotu. Ukázka, jak bude tabulka vypadat, je zobrazena na obrázku 8.

Příklad 5

```
    @FXML
    private TreeTableView<SimpleData> table;
    @FXML
    private TreeTableColumn<SimpleData, String> idColumn;
    @FXML
    private TreeTableColumn<SimpleData, String> nazevColumn;
    @FXML
    private TreeTableColumn<SimpleData, String> casColumn;
    @FXML
    private TreeTableColumn<SimpleData, String> stavColumn;

    idColumn.setCellValueFactory(param ->
param.getValue().getValue().idProperty());
    nazevColumn.setCellValueFactory(param ->
param.getValue().getValue().nazevProperty());
    casColumn.setCellValueFactory(param ->
param.getValue().getValue().casProperty());
    stavColumn.setCellValueFactory(param ->
param.getValue().getValue().stavProperty());
    SimpleData root = new SimpleData("1", "root", "0", true, "");
    TreeItem<SimpleData> rootItem = new TreeItem<>(root);
    table.setRoot(rootItem);
    table.setShowRoot(true);
    SimpleData ukol_1= new SimpleData ("25", "ukol_1", "0", false, "");
    TreeItem<SimpleData> item = new TreeItem<>(ukol_1);
    rootItem.getChildren().add(item);
```



Obrázek 8: Ukázka naplnění tabulky JavaFX

2.3 Vlákna

Samostatně běžící program je nazýván procesem. V jednu chvíli je možno vykonávat více činností pouze tehdy, pokud je k dispozici více jader a navíc pokud je použita technika vláken. Běžící proces je tedy nutné rozdělit na více částí, kterým se říká „podprocesy“, případně „vlákna“. Takto oddělené části aplikace je možné zpracovávat v jiném jádře procesoru. Aplikace pak může vykonávat více procesů v jeden čas. Další výhodou je zrychlení chodu celé aplikace. Vlákna mají stavy, ve kterých se aktuálně vyskytují. Počet těchto stavů se liší v závislosti na operačním systému počítače.

Programovací jazyk Java umožňuje implementaci vláken. Třída, která má být vláknem je potomkem třídy “Thread“. Na příkladu č. 6 lze vidět implementaci vlákna použitého v této bakalářské práci. Klíčové slovo “extends“ oznamuje překladači zdrojového kódu, že tato třída je potomkem třídy “Thread“, kde je implementována metoda `run()`. Zde je implementováno vše, co bude dané vlákno provádět. Běh tohoto vlákna lze spustit z jiného vlákna pomocí předdefinované metody `start()`, naopak zastavení pomocí metody `stop()`. [11]

Příklad 6

```
public class VlaknoAktualizaceTabulky extends Thread {

    private FXMLDocumentController fxml;
    private Calendar kalendar;

    public VlaknoAktualizaceTabulky(FXMLDocumentController fxml) {
        this.fxml = fxml;
    }
    public void run () {
        while (true) {
            try {

                Thread.sleep(1000 * 60);
                fxml.aktualizaceTabulky();
            } catch (InterruptedException ex) {
Logger.getLogger(VlaknoAktualizaceTabulky.class.getName()).log(Level.SEVERE
, null, ex);
            }
        }
    }
}
```

2.4 Použité knihovny

Knihovny slouží pro ulehčení práce programátorovi. Obsahují různé algoritmy řešící určitý problém. Je zbytečné, aby programátor ztrácel čas vymýšlením toho, co již bylo vymyšleno. Řada těchto knihoven je ke stažení na internetu.

Použitou knihovnou v aplikaci je "mysql-connector-java-5.1.39-bin.jar", tento balíček je možné stáhnout na stránkách [12]. V tomto balíčku je naimplementována komunikace s MySQL databází. Programátor neřeší, jakou formou spolu databáze a aplikace komunikuje. Předá jí jen potřebné parametry, aby došlo k úspěšnému spojení. Mezi hlavní parametry patří adresa databáze. V případě umístění databáze na serveru se uvádí IP adresa serveru, port, na kterém databáze běží a název databáze. Dále jsou potřebné přihlašovací údaje k databázi, aby bylo možné do databáze vstoupit. V příkladě 7 je uvedena metoda aplikace, díky které dojde ke spojení s databází. [13]

Příklad 7

```
private static Connection connect(String connectionString, String
databazeLogin, String databazePassword, String useSSL, String
autoReconnect, String useUnicode, String charakterEncoding) throws
SQLException {
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (ClassNotFoundException cnfe) {
            System.err.println("Error: " + cnfe.getMessage());
        } catch (InstantiationException ie) {
            System.err.println("Error: " + ie.getMessage());
        } catch (IllegalAccessException iae) {
            System.err.println("Error: " + iae.getMessage());
        }
        Properties properties = new Properties();
        properties.setProperty("user", databazeLogin);
        properties.setProperty("password", databazePassword);
        properties.setProperty("useSSL", useSSL);
        properties.setProperty("autoReconnect", autoReconnect);
        if (useUnicode.equals("true")) {
            properties.setProperty("useUnicode", useUnicode);
            properties.setProperty("characterEncoding",
charakterEncoding);
        }
        try {
            String conString = "jdbc:mysql://" + connectionString;
            conn = DriverManager.getConnection(conString, properties);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Došlo k chybě při
připojení k databázi. Aplikace bude vypnuta.");
            e.printStackTrace();
        }
        return conn;
    }
```

3 NÁVRH APLIKACE

Návrh aplikace probíhal postupně v rámci několika setkání se zástupcem zadavatele. Každá schůzka znamenala přiblížení, jak má budoucí aplikace fungovat, s čím má spolupracovat a jak má vypadat. Definice, nebo též formulace, požadavků na výsledný produkt je součástí analýzy a je nejdůležitější částí vzniku programu Nepochopení zadání znamená velkou časovou i finanční ztrátu pro obě strany. Programátor může vymýšlet něco, co nebylo požadováno. Klient pak neobdrží software, který chtěl. Analýza by tedy měla zabírat nejdelší část tvorby aplikace. Tak tomu bylo u této práce.

3.1 Použitá databáze

Jedním z hlavních kritérií bakalářské práce je propojení aplikace s databází. Aplikace je naprogramována pro databázi MySQL, avšak při drobné úpravě by mohla být použita i s jiným typem databáze. Databázová struktura popisovaná v rámci textu práce a používaná v aplikaci vychází z, resp. jedná se o část, databáze systému Redmine. Na obrázku 6 jsou schématicky znázorněny pro tuto práci potřebné tabulky a jejich vazby, včetně atributů významných pro tuto práci. V původním systému je však v uvedených tabulkách řada dalších atributů využívaných webovým rozhraním. Pro úplnost je v příloze na CD uložen soubor *databázeRedmine*.png, kde je zobrazen podrobnější databázový model.

V tabulce „users“ jsou uloženi uživatelé, kteří mají možnost přihlásit se do systému Redmine a tedy i do nově vytvořené aplikace. Výběr uživatele probíhá prostřednictvím pole „login“, které obsahuje unikátní hodnoty. Aplikaci lze spustit pod jedním uživatelským účtem.

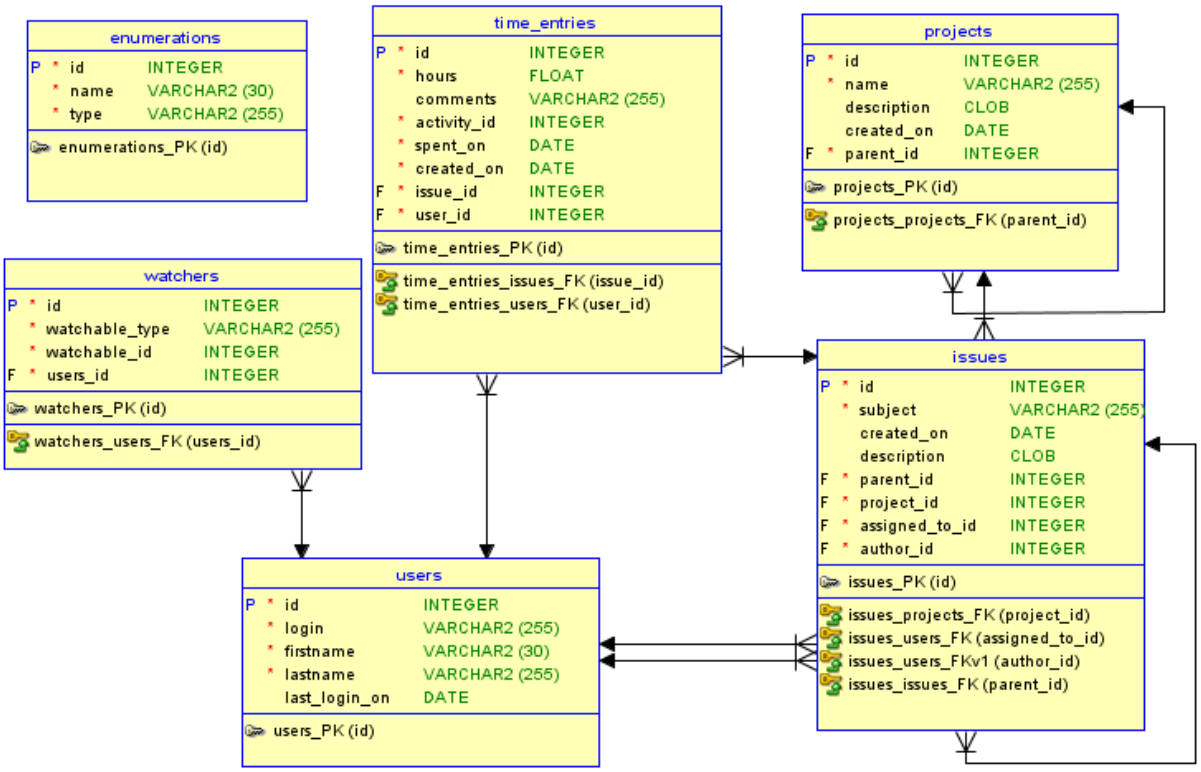
Veškeré úkoly jsou uloženy v tabulce “issues“. První cizí klíč nese informaci o projektu, jehož je součástí. Druhý říká, který uživatel je autorem a následující, komu je daný úkol přidělen. Každý úkol se může rozkládat na jednotlivé „podúkoly“, tato vazba je uložena v posledním cizím klíči tabulky.

Jednotlivé projekty jsou uloženy v tabulce “projects“. Případné dělení projektů na „podprojekty“ je také umožněno jako v tabulce úkolů. K určení, je-li projekt součástí jiného, slouží jediný uvedený cizí klíč.

Systém Redmine umožňuje sledování úkolů. Uživatel, který daný úkol sleduje, může nejen kontrolovat jeho stav, ale může na něm i spolupracovat. Informace o tom, který uživatel daný úkol sleduje, jsou uloženy v tabulce “watchers“. Shoduje-li se hodna “user_id“, což je i cizím klíčem tabulky, s hodnotou “id“ tabulky uživatelů, je tento uživatel sledujícím. Informace

o tom, jaký úkol sleduje je uložena v hodnotě “watchable_id“, ta odpovídá hodnotě “id“ z tabulky úkoly. Podmínkou je, že sloupec “watchable_type“ obsahuje hodnotu “Issue“.

Tabulka “time_entries“ obsahuje záznamy o odpracované době na daném úkolu. První cizí klíč udává uživatele, jež vykonal tento odpracovaný čas. Druhý nese informace o úkolu, na němž byla práce vykonána. Povinným údajem této tabulky je hodnota “activity_id“, která udává, jaký typ činnosti byl vykonáván. Tyto typy jsou uloženy v tabulce “enumerations“.



Obrázek 9: Informační schéma databáze

3.2 Princip fungování

Po spuštění aplikace se nejprve načítají data ze souborů. Tyto soubory obsahují potřebná data pro chod aplikace. Jsou automaticky ukládány při připojení k databázi. Nejsou-li všechna tato data nalezena, není možné aplikaci spustit. V tom okamžiku je uživatel vyzván, aby byla tato data stažena z databáze. Poté dojde k načítání souboru konfigurace, obsahující potřebné informace o připojení k databázi. Nejsou-li k dispozici, zobrazí se dialogové okno, kde uživatel vyplní potřebné údaje připojení k databázi a vytvoří se tak soubor konfigurace. Následuje stažení potřebných dat pomocí údajů nalezených v konfiguračním souboru.

Všechna potřebná data jsou uložena do jednotlivých souborů v binární podobě. Proběhne-li vše v pořádku, další spuštění aplikace nebude vyžadovat přihlášení k databázi.

Před spuštěním grafického prostředí je jméno přihlášeného uživatele a aktuální datum nastaveno jako název okna. Zároveň je načítán soubor obsahující informace o tom, zůstal-li aktivní některý úkol. Jsou-li data nalezena, dojde k dopočítání odpracované doby dle dnešního data a data, kdy byl úkol započat. Po dopočítání se uživateli zobrazí informační zpráva o úkolu, který byl doposud aktivním.

Než dojde k naplnění tabulky, obsahující uživatelovy projekty a úkoly, proběhne nahrání souboru logu odpovídající aktuálnímu datu. Pokud takový soubor existuje, znamená to, že za daný den už máme odpracovanou dobu. Tento odpracovaný čas přiřazen k jednotlivým úkolům. Poté proběhne naplnění tabulky, která obsahuje načtená data ze souboru úkolů a projektů a případně souboru s logy.

Po těchto uvedených úkonech je teprve možné aplikaci využívat. Jak aplikaci správně ovládat je popsáno v kapitole 5 - *uživatelská příručka*. Pro spuštění časoměřiče je potřeba zvolit možnost *Start*. S tím proběhne několik operací. Aktivuje se vlákno, počítající odpracovaný čas. Jsou zapsány údaje spuštěného úkolu do souboru, aby bylo zajištěno, že nedojde ke ztrátě dat. Je vytvořen nový záznam logu, který je ihned i uložen. Pro daný úkol je nastavena hodnota stavu na „X“, to bude zobrazeno v tabulce, aby měl uživatel přehled o tom, jaký úkol je vykonáván.

Při volbě *Stop* jsou vlákna naopak deaktivována. Dojde k zastavení počítání odpracované doby, dojde k aktualizaci logu a uložení potřebných souborů. Stav úkolu bude změněn na řetězec o žádném znaku, v tabulce nebude tedy nic zobrazeno. V poslední fázi bude vymazán obsah souboru s probíhajícím úkolem.

V souboru s logy je mimo jiné uložena hodnota, říkající, je-li uvedený záznam nahrán v databázi. Toho je využíváno již při naplnění tabulky. Ve sloupci čas jsou uváděny součty odpracovaných časů jen těch záznamů, které nejsou v databázi. Nahrává-li se úkol do databáze, je vždy odeslán veškerý odpracovaný čas daného dne, který ještě není v databázi. Proběhne-li úspěšné nahrání dat do databáze, v tabulce bude ve sloupci „čas [min]“ zobrazena nula.

Možnost *upravit* umožňuje editovat odpracovaný čas. Kladná hodna čas přidá, záporná odebere. Každá taková změna bude uložena také do souboru s logy. Hodnota logu „čas_od“

i „čas_do“ je nastavena na „00:00“. Z těchto záznamů lze tedy poznat, který čas byl takto přidán.

Při vypnutí aplikace je načítán opět soubor obsahující údaje o aktivním měření úkolu. Jsou-li data nalezena, uživateli je zobrazeno dialogové okno, kde má na výběr ukončit měření úkolu, nebo nechat běžet.

3.3 Architektura

Nejlepší způsob, jak aplikaci naprogramovat je rozdělení zdrojového kódu do vrstev. Tyto vrstvy jsou navzájem propojené. Ale spodní vrstva nesmí komunikovat s vrstvou nadřazenou. Nejpoužívanější architekturou je v současné době „Třívrstvá architektura“. Uživatel používající daný program komunikuje pouze s první vrstvou, nazývanou „prezenční“. To je to, co uživatel vidí. Po ní následuje „aplikační vrstva“, kde je umístěna veškerá logika aplikace. Tato vrstva není nikterak závislá na předchozí. Posledním článkem architektury je „datová vrstva“, také nazývána jako „databázová“. Ta zajišťuje práci s daty, jako je například čtení hodnot. Tento způsob má své výhody. Například nový vzhled programu je spojen pouze se změnou vrstvy „prezenční“, ostatní zůstanou beze změny a nezmění se tím funkčnost aplikace. Na obrázku 10 je zobrazena architektura, která byla použita při implementaci aplikace. [14]

3.3.1 Datová vrstva

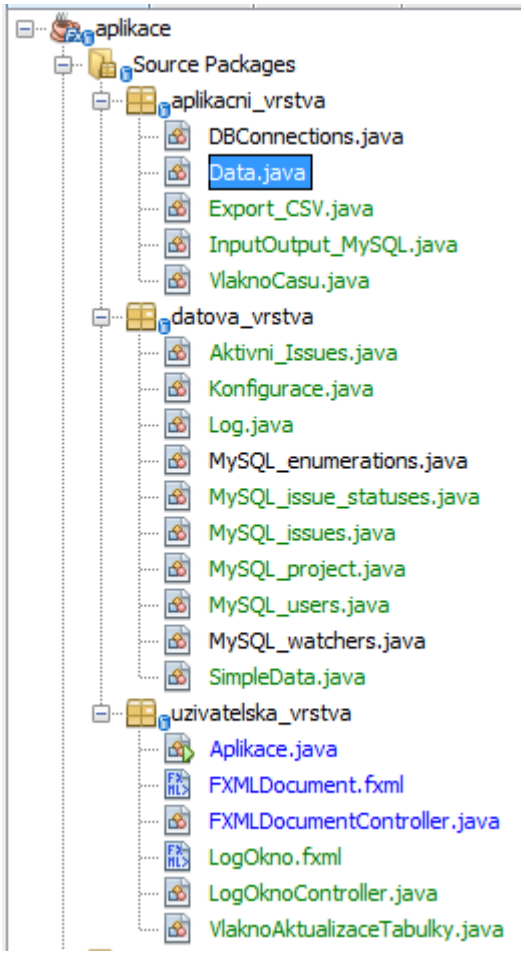
Datovou vrstvu nalezneme pod označením *datova_vrstva*. Obsahuje všechny typy dat, kterých se v programu využívá. Jedna třída symbolizuje právě jeden konkrétní typ. Tyto třídy obsahují konstruktor, privátní proměnné a metody čtení a zápisu. Třídy, které je nutné ukládat pro zajištění off-line režimu, mají implementováno rozhraní “Serializable“. To umožňuje data těchto tříd správně ukládat do souboru, případně načítat ze souboru.

3.3.2 Aplikační vrstva

V balíčku *aplikacni_vrstva* se nachází veškerá logika aplikace. Třída *DBConnections.java* se stará o komunikaci s databází. Potřebné parametry jsou uloženy při první spuštění do souboru a jsou instancemi třídy *Konfigurace.java*. Nejobsáhlejší třídou je *Data.java*, která se stará o veškeré načítání, ukládání, mazání, či přidávání dat. *Export_CSV.java* umožňuje exportovat záznam logů do souboru CSV. Jednou z hlavních tříd je *InputOutput_MySQL.java*, ta se stará o načítání všech dat do databáze, ta jsou předávána do metod třídy *Data.java,* kde jsou následně ukládána do souborů. Pro počítání odpracované doby a nasledného ukládání dat se stará třída *VlaknoCasu.java*. Instance této třídy se stává „podprocesem“.

3.3.3 Uživatelská vrstva

Třídy obstarávající grafické uživatelské prostředí jsou obsaženy v balíčku *uzivatelska_vrstva*. Komponenty JavyFX volají metody těchto tříd. Metoda spouštějící samotnou aplikaci je implementována v třídě *Aplikace.java*. Vzhled hlavního okna a jeho komponenty je uložen v třídě *FXMLDocument.fxml*. Metody ovládající tyto komponenty jsou obsahem *FXMLDocumentController.java*. Pro možnost přehledu odpracovaného času na jednotlivých úkolech je využita třída *LogOknoController.java* spolu se třídou *LogOkno.fxml*. Poslední třídou aplikace je *VlaknoAktualizaceTabulky.java*, která je také potomkem třídy *Thread*. Vlákno je aktivováno se spuštěním úkolu a zaniká zastavením. Obstarává aktualizaci tabulky, aby byly uvedené údaje validní. Tato činnost probíhá každou minutu.



Obrázek 10: Přehled použitých balíčků a tříd aplikace

3.4 Paměťová datová struktura

Jako datová struktura je využíván ArrayList. Jde o dynamické pole, které má neměnnou délku. Ve chvíli, kdy je tato délka překročena, dojde k vytvoření nového pole dvojnásobné

velikosti a prvky jsou do tohoto pole překopírovány. V případě nevyužití délky pole je vytvořeno pole menší a prvky jsou do něj opět zkopírovány. Tato struktura je již implementována v jazyce Java. Její použití je tedy velmi jednoduché. Pomocí metody `add()` přidává prvky na konec pole. Prvky si uchovávají svou pozici v poli. K těm se přistupuje právě prostřednictvím indexů. První prvek se nachází na indexu 0 a poslední na indexu velikost pole zmenšené o jedna. Nalezení prvku spočívá v procházení celého pole a porovnávání hodnoty s hledanou. Mimo již zmíněnou metodu poskytuje řadu jiných. Mezi nejpoužívanější patří `size()`, která vrací počet prvků v poli. Metoda `get(index)` vrací objekt uložený na tomto indexu, kde index je kladné celé číslo. Pole lze vymazat pomocí metody `clear()`.

Jednotlivé tabulky databáze systému Redmine mají odpovídající třídu v aplikaci. Vlastnosti z tabulky jsou definovány jako privátní proměnné dané třídy. Tabulka "issues" odpovídá třídě *MySQL_issues.java*, "projects" odpovídá *MySQL_project.java*. Na stejném principu jsou vytvořené i ostatní třídy začínajícím názvem *MySQL*.

Data těchto tříd jsou uchovávány v ArrayListech ve třídě *Data.java*. Metody obsluhující daná data jsou obsaženy právě v této třídě.

Hlavní třídou aplikace, obstarávající celý běh, je *FXMLDocumentController.java*. V této třídě se nachází objekt třídy *Data.java*, *InputOutput_MySQL.java*, *LogOknoController.java*, *VlaknoAktualizaceTabulky.java*, *SimpleData.java*. Objekty těchto typů nejsou v žádné jiné třídě používané. Pro zjištění a záznam odpracovaného času obsahuje daná třída i objekty, zapsané v *ArrayListu*, typu Log.java.

Třída *Data.java* obsahuje již výše zmíněné třídy začínajícím názvem *MySQL*. Dále obsahuje i objekt typu *VlaknoCasu.java*. Toto vlákno má na starosti počítání odpracované doby.

3.5 Souborová datová struktura

Jednou z hlavních podmínek aplikace je možné používání i v off-line režimu. Načtená data musí být tedy uložena v paměti počítače. V této aplikaci je to vyřešeno tím, že každý potřebný typ získaný z databáze je uložený v binárním souboru. Binární soubor je zvolen z mnoha důvodů. Poskytuje velmi rychlé načítání a ukládání dat. Nelze jednoduše přepsat obsah a způsobit tak problém s následným spuštěním aplikace. Obsah těchto dat není potřeba číst jiným programem. V neposlední řadě je velikost těchto souborů velmi malá a nezabírá tak zbytečné místo v počítači. Uložení úkolů do souboru je vidět na příkladu 7.

Příklad 7

```
FileOutputStream file;

        try {
            file = new FileOutputStream("issues");
            file.flush();
            ObjectOutputStream oos = new ObjectOutputStream(file);
            oos.writeObject(listIssues);
            file.close();
        } catch (IOException ex) {
            Logger.getLogger(Data.class.getName()).log(Level.SEVERE,  null,
ex);
        }
```

Binárních souborů aplikace využívá i pro zápis odpracovaných časů, tedy logů. Tyto soubory jsou ukládány ve složce „log“. Zobrazení obsahu těchto souborů je umožněno prostřednictvím aplikace. V položce *Soubor* je potřeba zvolit možnost *Log* a v levém dolním rohu kliknout na tlačítko *Načíst jiný*. Zobrazí se dialogové okno, kde si lze vybrat, který den zobrazit. Tyto data nejsou nikdy vymazána, uživatel se tak může podívat, co dělal například minulý pátek. Obsah těchto souborů může být pro uživatele z mnoha důvodů zajímavý. A proto se v tomtéž okně nalézá možnost *Exportovat*. Aktuálně načtený log se překopíruje do souboru CSV. Tento typ souboru lze otevřít v tabulkovém, případně textovém editoru a s údaji dále pracovat.

3.6 Jiné možnosti ukládání dat

Požadavek chodu aplikace bez připojení k databázi, znamená uložení potřebných dat. Asi nejlepším řešením je soubor a databáze. V této aplikaci byl zvolen soubor, neboť práce s ním je jednodušší, než vytváření lokální databáze na PC. Nyní se vyskytly další možnosti, jaký typ souboru použít. Po dlouhém uvážení byl zvolen binární soubor a to ze dvou hlavních důvodů. Prvním bylo, že čtení a zápis dat je daleko rychlejší, než s ostatními typy. Druhým důvodem bylo, že obsah těchto souborů není potřebné číst odjinud, než z aplikace. Binární soubor nelze správně přečíst z běžných aplikací počítače.

4 IMPLEMENTACE

Tato kapitola přibližuje vybrané implementační části. Při psaní zdrojového kódu byl kladen důraz na to, aby při rozšíření funkcí aplikace, bylo patrné, co který kód obstarává. Proto byly v hojné míře využívány dokumentační komentáře popisující danou funkci kódů. Při vytváření aplikace byla uživatelská vrstva vytvořena jako poslední. Pro testování byla použitá jiná třída, která vypisovala výsledky do konzole.

4.1 Třída Data.java

Veškerá data, která byla načtená z databáze, jsou obsaženy v privátních proměnných dynamického pole ArrayList<> ve třídě *Data.java*. Jsou zde obsaženy metody pracující se soubory, jak načítání, tak ukládání dat. Nalézají se zde všechny metody operující s daty aplikace. Metody přidání nových prvků kontrolují, zda-li prvek již není obsažen v daném poli, aby nedošlo k redundaci dat. Ukázka metody pro přidání úkolu je znázorněna na příkladu 8.

Příklad 8

```
public void pridejIssues(MySQL_issues issues) {
        boolean nalezeno = false;
        for (int i = 0; i < listIssues.size(); i++) {
        int pom= listIssues.get(i).getId().compareTo(issues.getId());
//kontrola nalezení úkolu
            if (pom == 0) { //pom nabývá hodnoty „0" právě tehdy, když byl
úkol nalezen
                nalezeno = true; //hodnota oznamující nalezení prvku
            }
        }
        if (!nalezeno) {
            listIssues.add(issues); //úkol je přidán
        }
    }
```

4.2 Třída InputOutput_MySQL.java

Veškeré metody, které se starají o načítání dat z databáze, jsou obsaženy ve třídě *InputOutput_MySQL.java*. Získaná data jsou předávány do výše zmíněné třídy *Data.java*, odkud jsou nejprve získána konfigurační data pro připojení k databázi. Při připojení k databázi dochází k aktualizaci záznamu "last_login_on" tabulky "users" na aktuální datum. Tento postup je znázorněn v ukázce příkladu 9. Jako ukázka načítání v příkladu 10, je použita

tabulka “enumerations“. Tato data jsou používána pro upřesnění aktivity činnosti při nahrávání odpracovaného času do databáze.

Příklad 9

```
Calendar kalendar;
kalendar = kalendar.getInstance();
stmt.executeUpdate("Update users set last_login_on='" +
formatData.format(kalendar.getTime()) + "' where id= '" + idUzivatele +
"';");
```

Příklad 10

```
try {   // načítání enummerations
   Statement stmt = con.createStatement();
// “con“ je instance třídy “Connection“, která je vytvořena při vytvoření
spojení s databází

   ResultSet rs = stmt.executeQuery("Select id, name,type from
enumerations where type='TimeEntryActivity';");//executeUpdate - nahrání
  while (rs.next()) { //Jsou nalezena další data?
  data.pridejEnumerations(new
MySQL_enumerations(rs.getString(1),rs.getString(2),rs.getString(3)));
//předání získaných dat do třídy „Data.java“
            }
        } catch (SQLException ex) {//nastane-li chyba bude použita
následující vyjímka

Logger.getLogger(InputOutput_MySQL.class.getName()).log(Level.SEVERE,
null, ex);
        }
```

4.3 Třída FXMLDocumentController.java

Nejobsáhlejší třídou celé aplikace je *FXMLDocumentController.java*. Obsahuje více než 1500 řádků kódu, který se stará o ovládání celé aplikace. Jednotlivé prvky GUI vyvolávají metody této třídy. Názvy komponent GUI musí být shodné v projektu a JavaFX Scene Builder. V příkladu 11 je znázorněna metoda pro vytvoření vlastního úkolu.

Příklad 11

```
@FXML
    private void pridejUkol() {
        JTextField nazev = new JTextField();
        Object[] message = {
            "Název úkolu:", nazev};

        int option = JOptionPane.showConfirmDialog(null, message, "Přidání nového úkolu", JOptionPane.OK_OPTION);
        if (option == JOptionPane.OK_OPTION && !nazev.getText().isEmpty()) {
            // předání potřebných dat do třídy „Data.java“
            // pro přidání vlastních úkolů disponuje třída „MySQL_issues“ druhým zjednodušeným konstruktorem
            data.pridejMujUkol(new MySQL_issues(nazev.getText()));
            //překreslení tabulky, aby byl zobrazen i nově přidaný úkol
            naplnTabulku();
        }
         // úkol musí mít název, jinak nebude přidán, docházelo by k nejednoznačnosti úkolů
        if (nazev.getText().isEmpty()) {
            JOptionPane.showMessageDialog(null, "Nelze přidat nepojmenovaný úkol.");
        }
    }
```

4.4 Problémy během implementace a jejich řešení

První problém nastal, když probíhalo aktivní měření odpracovaného času a počítač byl vypnut. Vlákno, které se staralo o zvyšování odpracované doby, tudíž zaniklo s vypnutím počítače. Při následném spuštění počítače odpracovaná doba neodpovídala době od startu měření času. Požadavek byl takový, aby se dopočítávala odpracovaná doba i za podmínek, že se PC spustí o několik dní později. Tento problém byl vyřešen tím, že pokud se spustí práce na určitém úkolu, uloží se stav tohoto úkolu do binárního souboru. V tomto stavu je uloženo id úkolu, čas a datum spuštění. Je-li časoměřič řádně zastaven prostřednictvím aplikace, obsah tohoto souboru je vymazán. Při každém spuštění aplikace se kontroluje, zda uvedený soubor obsahuje data. Pokud neobsahuje, aplikace pokračuje v dalších instrukcích. Pokud jsou ale data nalezena, probíhá dopočítávání odpracované doby. Tento algoritmus porovnává uvedený datum spuštění s aktuálním datem PC. Pokud je datum shodné, dopočítává rozdíl časů

spuštění úkolu a aktuálního času PC. Když však datum není shodné, nejprve je dopočítán čas za první den, kdy byl úkol spuštěn. Dalšími kroky je zvýšení nalezeného data o jeden den a porovnání, rovná-li se tento den dnešnímu dni. Pokud ano, je dopočítán rozdíl času od 00:00, tedy půlnoci, do aktuálního času PC. Je-li datum odlišné, odpracovaná doba pro tento den je nastavena na hodnotu 1440, což odpovídá počtu minut za jeden den. Dojde opět ke zvýšení data a cyklus probíhá znovu a ověřuje, je-li datum shodné s aktuálním datem PC. Toto řešení pro každý nalezený den vytváří příslušný soubor logu a zapisuje do něj příslušné hodnoty. V případě prvního testovaného dne, je již tento soubor logu vytvořen, ten tedy pouze aktualizuje.

Podobný problém nastává, když počítač přejde do režimu spánku. Aplikace tedy není vypnuta, ale vlákno, které se stará o počítání odpracovaného času nemůže probíhat. Tento problém je vyřešen tak, že aplikace testuje, je-li toto vlákno ve stavu démona. Tento stav říká, že vlákno nebylo řádně ukončeno a je v paměti počítače, ale nevykonává svou činnost. Aplikace tedy kontroluje, je-li toto vlákno v tomto stavu. Pokud ano, dojde k dopočítání odpracované doby dle aktuálního času a času, kdy byl úkol spuštěn.

Dalším problémem bylo, když nastane půlnoc a aplikace aktivně počítá odpracovanou dobu. Aplikace je spuštěna s údajem aktuálního dne nacházející se v názvu okna. Toto překreslení není možné bez restartování aplikace. Tabulka se každou minutu synchronizuje. Ovšem nastal-li den jiný, neměla požadované data a aplikace přestala správně fungovat. Tento problém byl implementován do vlákna, které se stará o aktualizaci tabulky. Hlídá aktuální čas a je-li tento čas rovný 23:59, aplikaci vypne a uloží spuštěný soubor, aby nedošlo ke ztrátě dat. Při opětovném spuštění aplikace se použije již řešený problém dopočítávání odpracované doby. Toto řešení poskytuje správná data, ale nutí uživatele aplikaci opět spustit, chce-li běžící úkol zastavit.

Dle požadavků bylo nutné do aplikace zakomponovat změnu přihlašovacích údajů k databázi. Předpokládá se, že aplikace bude sloužit jednomu uživateli a k této změně bude docházet jen velmi výjimečně. Uložené binární soubory nesoucí data databáze budou vymazány, neboť nový uživatel bude mít jiné úkoly. Avšak soubory s logy zůstávají zachovány napořád, z důvodů možného přihlášení původního uživatele. Tyto soubory obsahují citlivá data a jejich obsah by měl být přístupný pouze uživateli, který je vytvořil. Každý tento soubor nese jméno začínající číslem „id“ uživatele, které je nalezeno v databázi a je tudíž jedinečné, následuje řetězec „LOG“ a datum, pro který se tento soubor vztahuje. Řešení, jak zabránit

neoprávněnému uživateli k otevření je učiněno pomocí masky. Ta je implementována do dialogového výběrového okna načítání logu. Jsou zobrazovány tedy pouze ty soubory, jejichž název odpovídá „id“ aktuálně přihlášeného uživatele a následným řetězcem „LOG“.

4.5 Testování

Aplikace byla podrobena detailnímu testování chodu aplikace při nečekaných podmínkách. V této kapitole budou přiblížené vybrané scénáře testování. Scénáře obsahují sady postupů ovládání aplikace. Při běhu vybraného scénáře se sleduje, jak se aplikace chová. Doběhne-li scénář až do konce, neznamená to, že je vše správné. Kontrolují se ještě výsledná data, odpovídají-li očekávanému výsledku. Existuje tedy pravidlo, čím více scénářů, tím bezchybnější aplikace. Nelze však nikdy zaručit naprostou funkčnost bez jediné chyby. [15]

4.5.1 Scénář vypnutí aplikace

Spuštění aplikace – vybrání úkolu – spuštění časoměřiče – vypnutí aplikace – po nějaké době opětovné zapnutí.

Tento test proběhl zcela bez ztráty dat. Po opětovném zapnutí byl načten soubor, který obsahoval informace o neukončeném úkolu. Odpracovaný čas byl tedy dopočítán od data startu do aktuálního času PC.

4.5.2 Scénář režimu spánku

Spuštění aplikace – vybrání úkolu – spuštění časoměřiče – přechod PC do režimu spánku – po nějaké době probuzení PC.

Tento test se rovněž obešel bez ztráty dat. Po probuzení PC došlo k tomu, že vlákno, které se staralo o aktualizaci času, se stalo démonem. Tento stav vlákna je hlídán metodou *run()* třídy *VlaknoCasu.java*. I nyní došlo ke správnému dopočítání času.

4.5.3 Scénář zapnutí PC o několik dní později

Spuštění aplikace – vybrání úkolu – spuštění časoměřiče – vypnutí PC – zapnutí PC o tři dny později.

Požadavkem bylo, aby nebyl ztracen žádný čas. A to ani v případě, že uživatel může pracovat i několik hodin v kuse. Řešení tohoto scénáře je nejobsáhlejší metodou aplikace. Až po několikanásobném procházení toho scénáře, došlo ke správnému započítání odpracované doby. Musí se zde kontrolovat datum započaté práce a to následně navyšovat, až do aktuálního dne. Za každý den se správně vytvořil soubor logu a v něm správná hodnota odpovídající počtu minut za den.

4.5.4 Scénář půlnoci

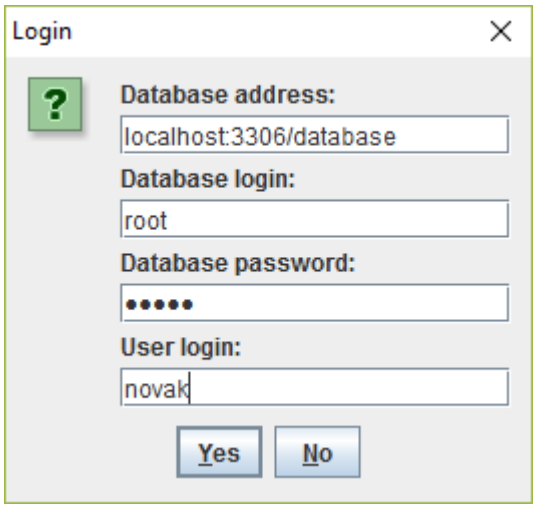
Spuštění aplikace – vybrání úkolu – spuštění časoměřiče – nastání půlnoci.

Z důvodu možnosti práce i přes noc, bylo nutné prověřit i tento scénář. Řešení této situace v aplikaci spočívá ve vypnutí aplikace v čase 23:59. Bylo implementováno mnoho řešení, jak daný problém vyřešit. Každý z nich měl ale své chyby. Plynulý chod aplikace nebyl možný z mnoha důvodů. Prvním z nich byla změna názvu okna aplikace, která nese informaci o aktuálním dnu. Tento text není dynamický, ale statický a tedy nelze ho měnit za chodu aplikace. Hlavním příčinou, proč aplikace nemohla pokračovat dále, byla práce s logem aktuálního dne. Do tohoto souboru se ukládá odpracovaný čas a z něj se nahrávají data potřebná k vykreslení tabulky. Přechod na nový den by znamenal, uložení tohoto souboru a vytvoření nového s aktuálně spuštěným úkolem. Použité řešení, vypnutí aplikace, není zcela uživatelsky přívětivé, ale nedojde k žádné ztrátě dat. V tomto případě je řešení postaveno na *scénáři zapnutí PC o několik dní později*.

5 UŽIVATELSKÁ PŘÍRUČKA

5.1 První spuštění

Pro úspěšný chod aplikace je nutné mít balíček *mysql-connector-java-5.1.39-bin.jar* ve složce „lib“. Poté je možné spustit soubor *aplikace.jar*. Při každém spuštění se kontroluje, zda byla nalezena data, tedy uložené binární soubory. První spuštění tedy nelze provést bez připojení k databázi. Zobrazí se dialogové okno (viz Obrázek 11) *Login*, kde je potřeba zadat údaje pro připojení do databáze. Jsou-li údaje správné, dojde k načtení dat z databáze a jejich uložení do souborů. Při dalším spuštění se čtou data již z těchto souborů.

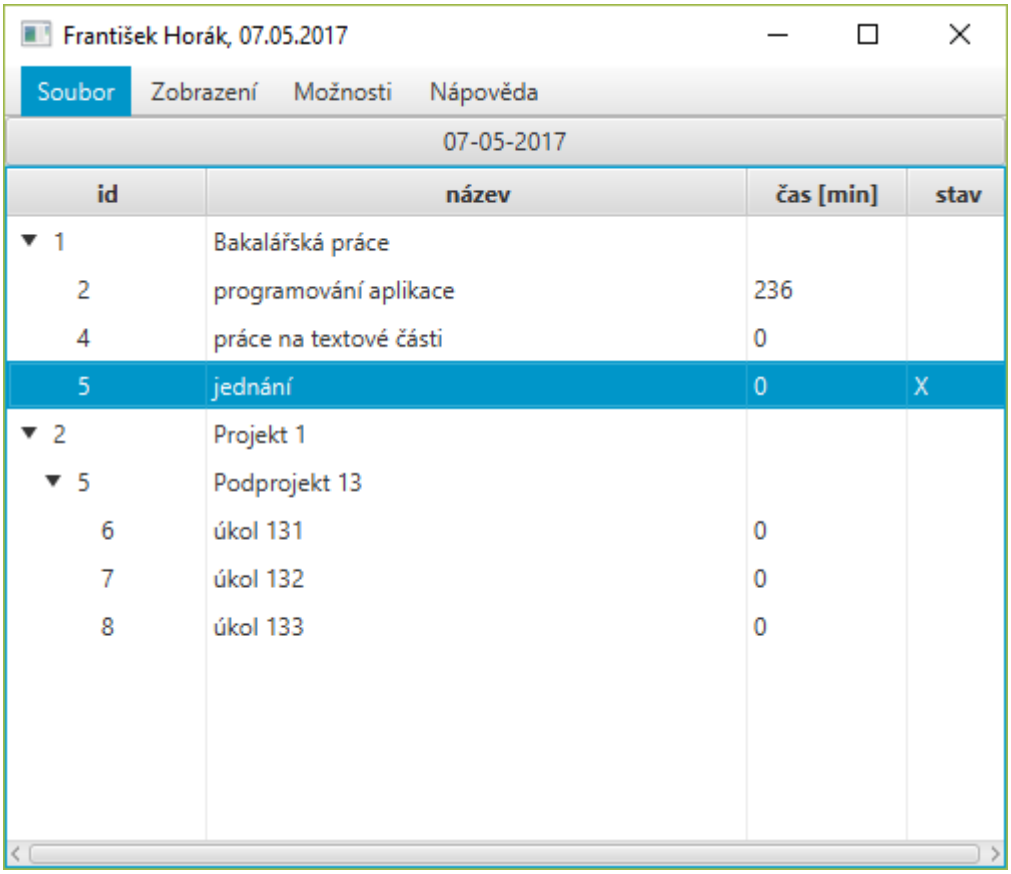


Obrázek 11: Login – přihlašovací okno aplikace

5.2 Orientace v aplikaci

Při úspěšném spojení s databází se nám zobrazí okno znázorněné na obrázku č. 12. Pro kontrolu je v titulku aplikace zobrazeno uživatelovo jméno, příjmení a aktuální datum. Největší komponentou, je tabulka, kde se zobrazují uživatelovy projekty a úkoly. První sloupec zobrazuje *id* patřičného projektu, potažmo úkolu. Má-li projekt podprojekt, či úkol, zobrazí se malá černá šipka, která umožňuje po kliknutí daný strom rozbalit, či skrýt. Dalším sloupcem je *název*, ten zobrazuje jméno daného úkolu, či projektu. Třetí hodnotou v tabulce je *čas*, zde se zobrazuje odpracovaná doba na úkolu. Poslední sloupec signalizuje, zda je daný úkol spuštěn. Je-li úkol aktivní, v tomto sloupci se zobrazí hodnota „X“. Z obrázku č. 12 můžeme vyčíst, že na úkolu „programování aplikace“ je odpracováno 236 minut. Běžícím úkolem je *jednání*. Nad touto tabulkou se nachází tlačítko, kde je zobrazen datum. Je-li dnešního dne, lze na úkolech pracovat. Po kliknutí na toto tlačítko lze v zobrazeném dialogovém okně vybrat jiný den a zobrazit tak jeho odpracované časy. Tato možnost slouží

především pro kontrolu, je-li odeslán odpracovaný čas z vybraného dne do databáze. Po opětovném kliknutí na toto tlačítko se opět načte aktuální den.



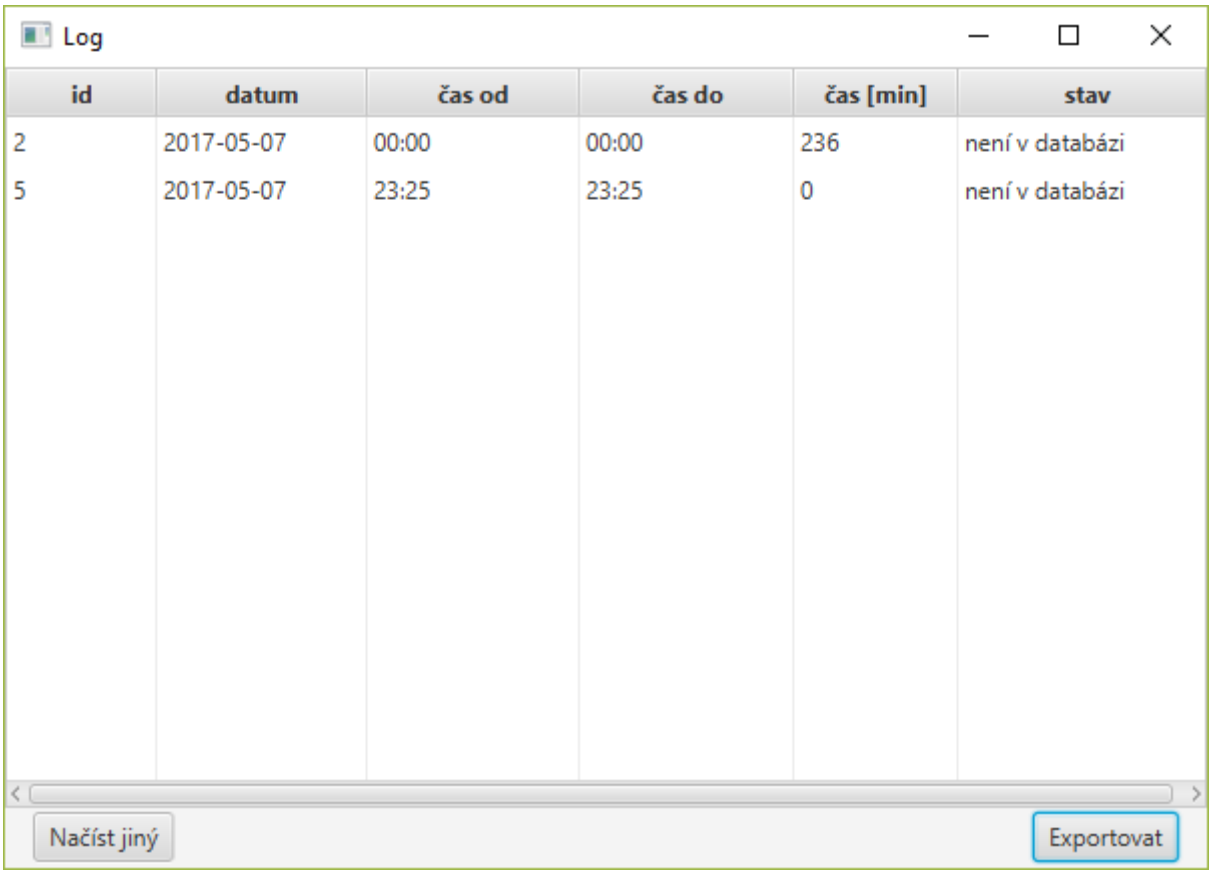
Obrázek 12: Hlavní okno aplikace

5.3 Hlavní menu

5.3.1 Soubor

Nejvíce možností, jak s aplikací pracovat je umožněno prostřednictvím hlavního menu. V záložce *Soubor* se nachází možnost zobrazit logy, tedy odpracované časy, volbou *Log*. Po kliknutí se otevře nové okno (viz obrázek 13). V tabulce jsou zobrazeny údaje o dnešních odpracovaných časech. Lze zkontrolovat, zda je tento čas již nahrán v databázi (sloupec *stav*) a od kdy do kdy byl úkol vykonáván. Po kliknutí na tlačítko *Načíst jiný* se otevře dialogové okno, kdy je možné vybrat jiný den a zobrazit v tabulce pak jeho odpracované časy. Aktuálně zobrazený den je možné exportovat do souboru CSV pomocí tlačítka *Export*. V záložce *Soubor,* v hlavním okně aplikace, se nachází volba *Aktualizovat z TIM*, tato možnost nahraje nová data z databáze. Není nutné se opětovně přihlašovat k databázi, neboť při prvním startu aplikace byl uložen konfigurační soubor. Změnit přihlašovací údaje k databázi je umožněno

prostřednictvím další položky *Nastavení*. Veškerá data budou smazána a nahrazena novými. Jediné, co zůstane zachováno, jsou soubory log, tedy ty soubory, kde je uveden odpracovaný čas. Poslední možností je volba *Konec*, která ukončí aplikaci. Pokud je aktivní nějaký úkol, je zobrazeno dialogové okno a lze vybrat, zda má daný úkol nadále běžet, či naopak.



Obrázek 13: Okno logů

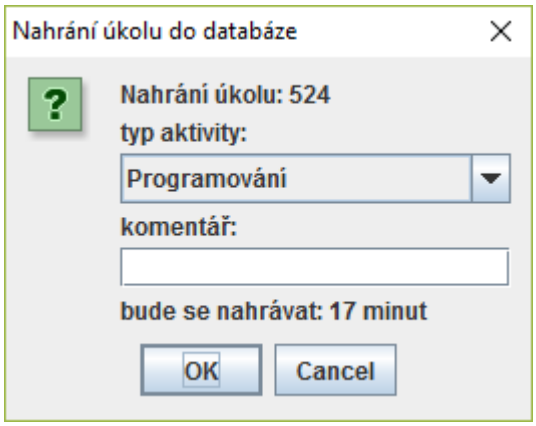
5.3.2 Zobrazení

V položce *Zobrazení* se lze vybrat, které úkoly se mají zobrazit v tabulce. Defaultně je nastaveno zobrazení mých a sledovaných úkolů. Možnost *Uzavřené úkoly* zobrazuje úkoly, které jsou již uzavřené. Kliknutím na patřičnou položku se úkoly zobrazí, či nezobrazí. Tuto funkčnost je dobré využít při přeplněné a nepřehledné tabulce.

5.3.3 Možnosti

Další volbou z hlavního menu jsou *Možnosti*, veškeré tyto funkce se skrývají i pod pravým tlačítkem myši při kliknutí na tabulku. První nabízená možnost je *Start*, pomocí níž je spuštěn časoměřič na aktuálně vybraném úkolu z tabulky. Pro kontrolu se ve sloupci „stav“ v tabulce zobrazí hodnota „X“. Nyní se sloupec „čas (min)“ bude aktualizovat každou minutu. Běžící

úkol lze zastavit další volbou z této nabídky *Stop*. Ukončí se počítání úkolu a ze sloupce „stav“ zmizí hodnota „X“. Funkce *Start* a *Stop* jsou navíc implementovány i jako dvojklik levým tlačítkem myši na daný úkol. Neběží-li žádný úkol, vybraný úkol se spustí, běží-li některý úkol, bude zastaven. Funkce „Upravit“ umožňuje editovat odpracovaný čas. Uživatel pak manuálně zapíše čas, například když si zapomněl úkol spustit. Je možné zadat čas i záporný, zapomenul-li uživatel naopak úkol vypnout. Výsledný čas bude zobrazen v tabulce. Volba *Nahrání do databáze* slouží k nahrání odpracovaného času do databáze. Je nutné být v tuto chvíli připojen k internetu a mít přístup k databázovému serveru. V dialogovém oknu, znázorněným na obrázku č. 14, uživatel vybere, jaký typ aktivity na daném úkolu prováděl. Má možnost k odpracovanému času přidat komentář, který bude uložen do databáze. Nahrává se vždy aktuálně odpracovaný čas na daném úkolu, bez ohledu na to, kolikrát uživatel daný úkol spustil a vypnul. Po nahrání času do databáze se v kolonce *čas [min]* zobrazí hodnota „0“. Pod čarou v nabídce se skrývají dvě položky *Přidat úkol* a *Odebrat úkol*. První volbou má uživatel možnost přidat si vlastní úkol a na něm sledovat odpracovaný čas. Druhou volbou tento úkol zase odstraní. Tyto dvě volby slouží pouze pro uživatele aplikace, jako pomocné sledování jiných úkolů. Tyto úkoly nelze nahrát do databáze. Jejich čas je také zaznamenán v souboru s logy. Pro tyto úkoly je automaticky vytvořen nový projekt *Moje projekty* a přidané úkoly jsou jeho potomky.



Obrázek 14: Dialog nahrání úkolu do databáze

5.3.4 Nápověda

V poslední položce hlavního menu *Nápověda* uživatel nalezne pod volbou *O aplikaci* informace o aplikaci. V případě potíží, názorů, či připomínek je možné nalézt zde e-mail autora aplikace a kontaktovat jej. Mimo jiné zde naleznete i údaj o verzi aplikace, kterou právě používáte.

5.4 Otestování aplikace bez přístupu k databázi Redmine

Otestování aplikace je umožněno dvěma způsoby. První možností je nainstalování lokální databáze MySQL do počítače. Do nově vytvořené databáze je potřeba nahrát obsah souboru *sqlStruktura.sql*, který se nachází ve složce *použití s databází* na disku CD. Při spuštění *aplikace.jar* je nutné vyplnit informace o databázi a login uživatele z tabulky "users". Je důležité, aby byl ve složce *lib* soubor *mysql-connector-java-5.1.39-bin.jar*. Druhou možností je spuštění aplikace bez instalace databáze do PC. Tato možnost však neumožňuje použití funkcí *nahrát do databáze* a *aktualizovat z TIM*. Pro tuto možnost se na CD nachází složka *použití bez databáze*. K vyzkoušení aplikace pak stačí spustit soubor *aplikace.jar*.

6 ZÁVĚR

Tato bakalářská práce byla vytvořena za účelem implementace nového nástroje pro sledování a záznam odpracovaného času. V úvodu byly popsány důvody vzniku této bakalářské práce. Dále pak byl přiblížen systém Redmine, jehož klientem je nově vytvořená aplikace. V následující kapitole byly představeny vybrané nalezené aplikace, které však nesplňují požadavky zadavatele. Tyto požadavky byly představeny v kapitole následující. V dalších kapitolách byly stručně popsány použité technologie. Nejobsáhlejší kapitolou byl popis samotného návrhu aplikace a vývoje aplikace. Byly uvedeny ukázky vybraných částí zdrojového kódu a řešení vyskytlých problému během implementace. V poslední kapitole byla popsána uživatelská příručka aplikace.

Stěžejní částí této aplikace bylo řešení nečekaných událostí stavu počítače a dopočítávání odpracované doby za více dnů. Nejobtížnější postupy řešení problémů byly popsány v kapitole 4.4 *Problémy během implementace a jejich řešení*.

V bakalářské práci byly splněny veškeré zadané požadavky aplikace. V budoucnu by mohla být tato aplikace nadále vyvíjena. Aplikace byla napsána v přehledném zdrojovém kódu, díky tomu lze snadno rozšířit její funkce a možnosti. Při drobné úpravě by byla aplikace schopná spolupracovat i s jiným databázovým serverem. Aplikace již nyní implementuje všechny požadované funkcionality a je tak připravena pro rutinní nasazení v praxi.

7 POUŽITÁ LITERATURA

[1] Redmine. *Redmine* [online]. 2006 [cit. 2017-05-10]. Dostupné z: http://www.redmine.org/projects/redmine/wiki

[2] Third Party Tools. *Redmine* [online]. 2006 [cit. 2017-05-10]. Dostupné z: http://www.redmine.org/projects/redmine/wiki/ThirdPartyTools

[3] Zkuste Toggl a vemte čas do vlastních rukou | Blog SEO Linhart.cz. *Blog SEO Linhart.cz | Novinky z projektů provozovaných SEO Linhart s.r.o.* [online]. 2014 [cit. 10.05.2017]. Dostupné z: http://blog.seolinhart.cz/zkuste-toggl-a-vemte-cas-do-vlastnich-rukou/

[4] Time Tracker - Timesheet – Aplikace pro Android ve službě Google Play. *Google Play*. [online]. Copyright © 2017 Google [cit. 10.05.2017]. Dostupné z: https://play.google.com/store/apps/details?id=ch.gridvision.pbtm.androidtimerecorder&hl=cs

[5] R0B. Kapow – program na evidenci ztráveného času na projektu či pracovní doby. *Manjaro Linux CZ | MANJARO* [online]. 2013 [cit. 2017-05-10]. Dostupné z: http://www.manjaro.cz/kapow-program-na-evidenci-ztraveneho-casu-na-projektu-ci-pracovni-doby/

[6] KLAUS, Josef. TimeEdition: Měřte si čas, který strávíte prací. *Živě* [online]. 2013 [cit. 2017-05-10]. Dostupné z: http://www.zive.cz/clanky/timeedition-merte-si-cas-ktery-stravite-praci/sc-3-a-167690/default.aspx

[7] MySQL databáze - český manuál. *Junext* [online]. 2002 [cit. 2017-05-10]. Dostupné z: http://www.junext.net/mysql/

[8] OMKAR, Alok. JavaFX and MySQL Sample Illustration. *365: A program for a day.*. [online]. 2013 [cit. 2017-05-10]. Dostupné z: http://365programperday.blogspot.cz/2013/07/javafx-and-mysql-sample-illustration.html

[9] What Is JavaFX? *Oracle* [online]. 2017 [cit. 2017-05-10]. Dostupné z: http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

[10] JavaFX Scene Builder. *Oracle* [online]. 2017 [cit. 2017-05-10]. Dostupné z: http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html

[11] Processes and Threads. *Oracle* [online]. 2015 [cit. 2017-05-10]. Dostupné z: https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html

[12] MySQL Connector/J » 5.1.39. *MVNRepository* [online]. 2016 [cit. 2017-05-10]. Dostupné z: https://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.39

[13] MySQL Connector/J 5.1 Developer Guide. *Oracle* [online]. 2017 [cit. 2017-05-10]. Dostupné z: https://dev.mysql.com/doc/connector-j/5.1/en/

[14] DRESLER, Robert. Vícevrstvé architektury aplikací. *Robert Dresler* [online]. 2011 [cit. 2017-05-10]. Dostupné z: http://www.robertdresler.cz/2011/04/vicevrstve-architektury-aplikaci.html

[15] ČERMÁK, Miroslav. Testovací scénář. *CleverAndSmart* [online]. 2009 [cit. 2017-05-10]. Dostupné z: http://www.cleverandsmart.cz/testovaci-scenar/