
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**
**DEPARTMENT OF INFORMATION SYSTEMS**


# MODUL PRO SLEDOVÁNÍ ZNALOSTÍ UŽIVATELŮ PRO KANBOARD.ORG

**SKILLMATRIX PLUGIN FOR KANBOARD.ORG**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **JAN VÁLKA**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Ing. FRANTIŠEK ŠČUGLÍK, Ph.D.**
**SUPERVISOR**



**BRNO 2019**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

# Zadání bakalářské práce


Student: **Válka Jan**
Program: Informační technologie
Název: **Modul pro sledování znalostí uživatelů pro Kanboard.org**
**Skillmatrix Plugin for Kanboard.org**
Kategorie: Web

Zadání:

1. Seznamte se s nástrojem Kanboard.org a s technologiemi PHP a SQLite.
2. Nastudujte možnosti tvorby vestavných modulů pro Kanboard.org.
3. Navrhněte modul pro zadávání, vizualizaci, a vyhledávání znalostí jednotlivých uživatelů. V návrhu zohledněte možnost dynamicky přidávat množiny znalostí a různé úrovně přístupových práv pro jejich editaci.
4. Po konzultaci s vedoucím navržené řešení implementujte a otestujte. Dosažené výsledky zhodnoťte.

Literatura:

- Dokumentace Kanboard.org [online], Dostupné z: https://docs.kanboard.org/en/latest/
- Welling, L., Thomson, L.: PHP a MySQL Kompletní průvodce vývojáře, CPress, 2017
- Kroenke, D., M., Auer, D. J.: Databáze, CPress, 2014

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Ščuglík František, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 30. října 2018

# Abstrakt

Cílem této práce je vytvoření modulu, pro sledování znalostí stávající uživatelů aplikace Kanboard. Podstatou tohoto modulu je ohodnotit schopnosti zaměstnanců a přidělit jim štítky. Na základě těchto atributů je tu dále možnost filtrovat jednotlivé uživatele. V práci jsem vytvořil nástroj, který ulehčí práci manažerů při rozhodování, které zaměstnance vybrat na ať už nový, či stávající projekt.



# Abstract

The goal of this thesis is to create a plugin for monitoring knowledge of existing Kanboard users. The main goal of this plugin is to evaluate employees' abilities and assign them labels. Based on these attributes, it's possible to filter individual users. In this thesis I have created a tool that will facilitate the process of deciding which employee to assign to either new or current project.



# Klíčová slova

Kanboard, Kanboard modul, PHP, SQLite, Webové technologie, Web aplikace, Web server, Informační systém, Databáze, Uživatelské rozhraní

# Keywords

Kanboard, Kanboard plugin, PHP, SQLite, Web technologies, Web application, Web server, Information system, Databases, User interface


# Citace

VÁLKA, Jan. *Modul pro sledování znalostí uživatelů pro Kanboard.org*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. František Ščuglík, Ph.D.

# Modul pro sledování znalostí uživatelů pro Kanboard.org

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Františka Ščuglíka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Válka
9. května 2019

## Poděkování

Chtěl bych poděkovat panu Ing. Františku Ščuglíkovi, Ph.D za poskytnuté konzultace a poradenství.

# Obsah

# Kapitola 1

# Úvod

Když se podíváte do historie na všechny velké projekty, které lidstvo dokázalo stvořit, vždy mají stejnou strukturu. Na začátku je myšlenka jednoho, či několika lidí. Tato myšlenka roste a začne na sebe nabalovat stále větší počet lidí. Právě v této fázi je nutné dát věcem řád dříve, než nám přeroste přes hlavu. Bez určitých pravidel by systém nefungoval a zkolaboval. Důkazy tohoto jevu můžeme vidět všude kolem nás. Každý mravenec v mraveništi má svoji roli, každá smečka vlků má svoje jasně určená pravidla atp. V každé kolektivní činnosti je vždy někdo, kdo vymýšlí, někdo, kdo řídí a ti, kteří vykonávají úkoly. Tato práce se zaměřuje na jeden ze způsobů, jak činnost řídit. Po představení jednoho konkrétního nástroje, je nástroj obohacen o novou funkcionalitu, která by mohla lidem s řízením pomoci.

Toto téma jsem si vybral z toho důvodu, že řeší konkrétní problém reálného světa. Je spousta zadání, která jsou sice obtížná, ale nakonec se stejně založí do šuplíku a nikdy se nepoužijí. Dále mě zaujala oblast webových aplikací, protože se jedná o oblast, která se za posledních pár let dost změnila a tento trend dál pokračuje. Myšlenka, že vytvoříme jednu aplikaci a uživatelé ji mohou používat bez ohledu na operační systém, či dokonce platformu, je velice atraktivní.

Tato práce se zabývá rozšířením stávající aplikace Kanboard o modul, který by umožňoval sledovat jednotlivé znalosti uživatelů. Využití tohoto rozšíření spočívá v ulehčení výběru potencionálních účastníků na novém, či stávajícím projektu. Pokud manažer přesně ví, jaké lidi potřebuje a co by měli umět, může si je jednoduše vyhledat a ušetřený čas věnovat jiným problémům.

V první kapitole je vysvětlen princip řízení metodiky Kanban, způsob instalace současného programu Kanboard a popis použitých nástrojů. Další dvě kapitoly pojednávají o návrhu rozšíření a posléze o jeho implementaci. Na závěr je popsán způsob testování a shrnutí dosažených výsledků.

# Kapitola 2

# Současná aplikace Kanboard

Dříve, než začnu popisovat ovládání a funkcionalitu stávající aplikace, je potřeba si vyjasnit co je metodika `Kanban`.

## 2.1 Metodika Kanban

Slovo Kanban znamená v japonském jazyce „vývěsný štítek“. Tento koncept byl vytvořen v padesátých letech minulého století firmou Toyota, avšak proslavil se až v sedmdesátých letech, během globální recese, kdy bylo pro firmy životně důležité snížit plýtvání materiály a snížit ceny. Jeho autor, Taiičiho Óno byl prý inspirován americkými supermarkety [4], kde zákazník:

- dostane co požaduje
- v čase kdy to požaduje
- množství, které požaduje

Aplikováním této myšlenky došlo k:

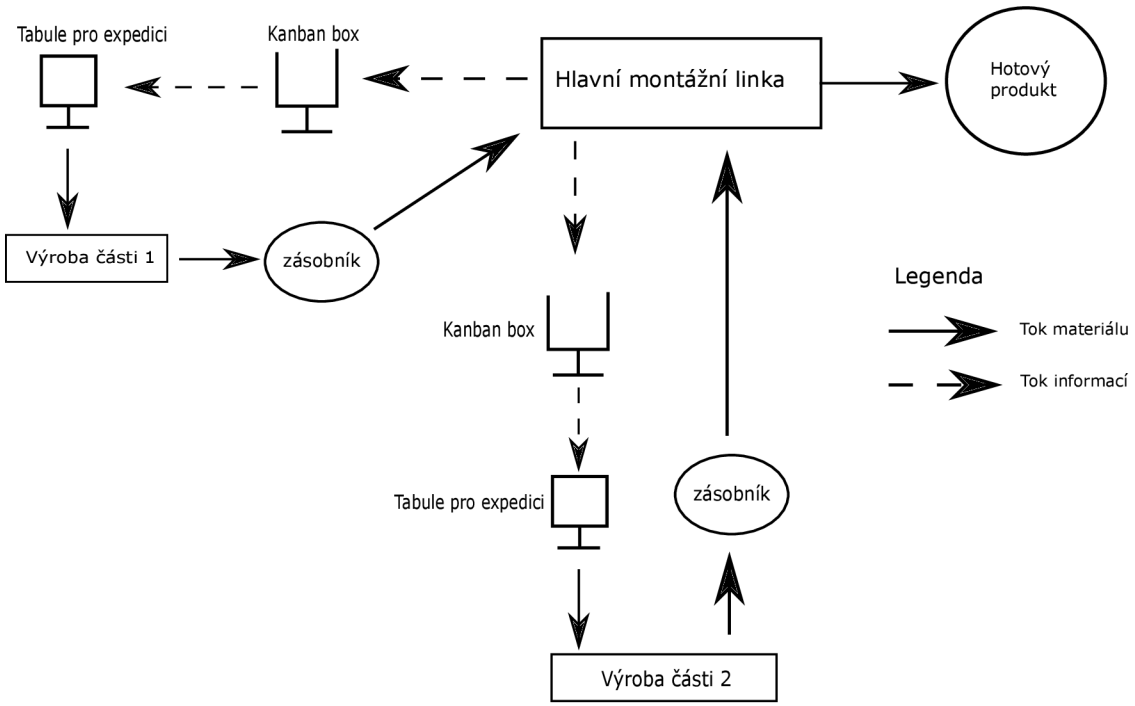
- redukci skladů a produkčního cyklu
- zrychlení předávání informací ve výrobě
- zvýšení produktivity

V systému Kanban, jsou informace ohledně jednotlivých částí výroby reprezentovány pomocí karet. Tyto karty kolují systémem, předávají informace, a tím dochází k řízení výroby. Tradiční řízení výroby je založen na tzv. „push“ konceptu. Tento způsob řízení uvažuje s konstantním počtem výrobků. Naskladní se určitý počet materiálu, z toho se vytvoří určitý počet polotovarů a tak dále, až vznikne konkrétní produkt. Pokud však někde dojde k problému, například zpoždění v některém kroku, nebo nízký odběr zboží, reaguje tento systém na nové požadavky velice pomalu. Systém Kanban je tzv. „pull“ systém. Na začátku je vznešen požadavek na konkrétní množství produktu. Od shora dolů se tento požadavek propaguje a mění na množinu karet, kde každá representuje konkrétní informaci pro daný výrobní celek. Tímto způsoben nedochází ke zbytečnému vytížení skladů, zvyšuje se kvalita výrobků a rychleji se reaguje na případné změny ve výrobě. Kanban metodiky jsou v dnešní době používány v celé řadě odvětví, nejen v automobilovém průmyslu. Například výrobci zubních kartáčků Oral-B se implementací toho systému podařilo odstranit problémy spojené s balením jejich produktů [7].

## Jednoduchý Kanban model

Pro lepší pochopitelnost slouží následující model 2.1.

Jedná se o jednoduchou továrnu na bonbóny. Od zákazníka přijde požadavek na nějaké množství konkrétních bonbónů. Řekněme sto melounových bonbónů. Tento požadavek se rozdělí na jednotlivé karty. Potřebujeme nějaké želé a obaly. Každá karta obsahuje konkrétní informace pro daný úsek výroby. Tyto karty se pošlou na pracoviště. Pracovník si na kartě přečte kolik polotovarů je potřeba vyrobit a dá se do práce. Na konci dne máme požadované množství produktu, v požadovaný čas bez zbytečných přebytků.

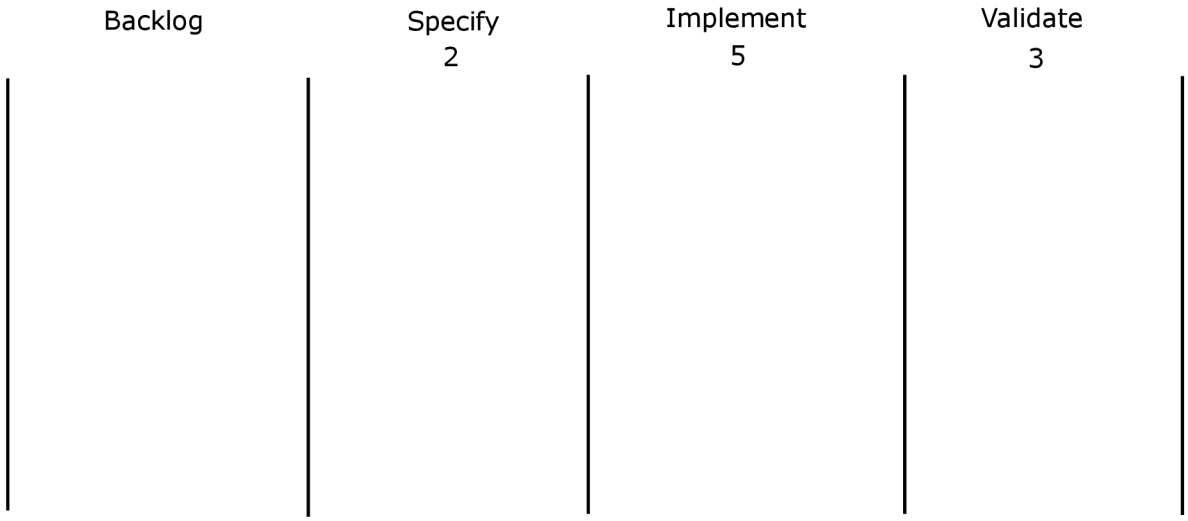



Obrázek 2.1: Schéma jednoduché továrny

## Použití kanbanu pro vývoj softwaru

Pokud si myslíte, že Kanban lze použít jen ve výrobních továrnách, jste na omylu. Tento přístup lze také uplatnit při vývoji softwaru. Jedna z firem, kde je Kanban používán, je například Microsoft [3].

Uspořádání a manipulace s kartami je však poněkud odlišná. Karty už necestují po továrně, ale jsou na jednom místě. Tyto karty jsou rozmístěny do jednotlivých sloupců. Každý sloupec udává stav rozpracovanosti jednotlivých úkolů. Nejjednodušší tabulka Kanban obsahuje sloupce tři. Nějaký zásobník pro nové úkoly (anglicky Backlog), dále sloupec s úkoly, na kterých se zrovna pracuje (anglicky In Progress) a poslední, kde jsou umístěny dokončené úkoly. Samozřejmě sloupců si můžeme zvolit libovolné množství, případně je dále dělit. Jelikož se jedná o tzv. „pull“ systém, začíná se s přesunem karet vždy z části nejvíce vpravo. Pro ukázku jsem vytvořil následující příklad Kanban tabule 2.2.




Obrázek 2.2: Příklad `Kanban` tabule pro vývoj softwaru

Vidíme zde sloupec `backlog`. Jedná se o zásobník, kam přichází nové úkoly či zakázky. Dále je zde sloupec `specify`. V tomto kroce se úkol rozdělí na množinu podúkolů tak, aby každá nová karta měla zhruba stejnou časovou náročnost. Ve sloupci `implement` dochází k samotné implementaci podúkolu. Po úspěšné implementaci se karta přemisťuje do sloupce `validate`, kde dochází k testování části. Pokud validace proběhla v pořádku, byl úkol splněn a karta odchází ze systému. Příklad použití může být následující. Na začátku dne se přijde k tabuli. Jelikož se začíná částí nejvíce vpravo, manažer se zeptá, jestli některá karta z validace je již hotová. Pokud ano, dají se pryč a přechází se na sloupec implementace. Opět pokud je nějaké položka již implementována, přesune se do sloupce validate. Tímto způsobem se pokračuje dál, až se projdou všechny sloupce. Celý proces netrvá více než pár minut a každý zaměstnanec jasně ví, na čem má pracovat.

Na obrázku 2.2 jste si mohli všimnout čísel u jednotlivých sloupců. Jedná se o limit, který určuje kolik rozpracovaných položek může být v každém sloupci. Tento limit má dvě klíčové role:

- limituje počet položek, které ovlivní změny v požadavcích zákazníka, změny návrhu atp.
- přizpůsobí rychlost vývoje nejpomalejšímu kroku

Vývoj produktu nikdy nemůže být rychlejší než jeho nejpomalejší krok, proto je tedy zbytečné, aby ostatní položky přibývaly, když některá část nestíhá. Jen by se hromadily. Limit by měl být stanoven tak, aby tento nejpomalejší krok nijak neomezoval produkci, například nasazením největším počtem pracovníků.

Pokud nastane situace, že jeden sloupec má vše splněno a čeká na druhý, patrně nastal nějaký problém a můžeme mu nějak pomoci. Přitom není nutné, aby byli zaměstnanci v dané problematice experti, i taková banální věc jako dojít týmu pro oběd může ušetřit spoustu času. Zaměstnanci jsou jeden tým, kteří mají společný cíl, a to předat zákazníkovy kvalitní produkt ve stanoveném čase.

Může se postupem času ukázat, že v některém kroku dochází ke zdržení častěji, v takovém případě musí dojít k analýze kořene problému. Možná nejsou počáteční úkoly rozděleny na dostatečně malé podúkoly, možná má některý zaměstnanec problém s návrhem atp.

## 2.2 Kanboard

Nyní už víme, co Kanban je a k čemu je lze použít, představíme si tedy aplikaci Kanboard.

Kanboard je „opensourcová“ implementace systému Kanban. Tato aplikace je pod licencí MIT[1] a je volně dostupná ke stažení pro každého na adrese Kanboard.org[2]. Jelikož se jedná o webovou aplikaci, mohou tuto aplikaci používat uživatelé bez ohledu na operační systém.

Uživatelé této aplikace si vytváří projekty a dále jim tato aplikace slouží k jejich řízení. Projekty mohou být soukromé (na těchto projektech pracuje uživatel sám), či společné pro více uživatelů. Každý projekt obsahuje úkoly, které se musí splnit a tyto úkoly se přemísťují do sloupců, které indikují současný stav. Aplikace nabízí celou řadu užitečných funkcí jako například:

- limitování rozpracovaných úkolů
- vytváření štítků pro jednotlivé typy úkolů
- vyhledávání konkrétních úkolů
- přidělování úkolů konkrétním uživatelům
- vkládání komentářů k vyřešení problémů u jednotlivých úkolů
- uživatelské role a týmy
- vedení statistik

Mezi hlavní přednosti aplikace patří jednoduché ovládání, kdy uživatelé mohou jednoduše přetahovat pomocí myší úkoly mezi jednotlivými sloupci.

V současné době je aplikace přeložena do více jak třiceti jazyků, tudíž lze tuto aplikaci používat i bez znalosti angličtiny.

### Instalace aplikace Kanboard

Jelikož je aplikace napsaná v jazyce PHP, je potřeba mít tento jazyk nainstalovaný. Většina moderních distribucí operačních systémů disponuje verzí 7.3 (v době psaní je tato verze poslední), avšak postačí verze 5.6. Dále je potřeba instalací balíčků `gd`, `mbstring`. Plný seznam všech závislostí najdete v dokumentaci Kanboardu [1].

Pokud používáte Windows 10 a nechce se Vám řešit nastavování správných cest, vřele doporučuji si nainstalovat linuxovou příkazovou řádku. Ušetří Vám to spoustu problémů a „workflow“ je pak stejný, jak na linuxových systémech.

Instalace aplikace Kanboard je velice jednoduchá. Pokud máte již nainstalovaný webserver stačí pouze nakopírovat složku kanboard. Aplikace je pak dostupná na adrese:

```
https://adresavasehoserveru/kanboard/
```

Plnohodnotný webserver však pro spuštění není nutností. Od verze 5.4 poskytuje PHP svůj vlastní vestavěný webserver. Výkon sice není optimální, ale pro vývoj či vyzkoušení je více než dostačující. Pro spuštění aplikace tímto způsobem se stačí navigovat do složky kanboard, kde se nachází soubor `index.html`. Poté zavoláte v příkazovém okně php s přepínačem `-S` a patřičnou adresou. Příklad může vypadat následovně:

[1] https://cs.wikipedia.org/wiki/Licence_MIT
[2] https://kanboard.org/

```
php -S localhost:8000
```

Aplikace pak bude dostupná na adrese `http://localhost:8000`.

Mezi další oblíbené způsoby instalace patří použití VirtualBoxu[3] s linuxovým obrazem, či Docker[4].

### Instalace modulů do aplikace Kanboard

Instalace modulů do aplikace Kanboard je přímočarý proces. Stačí pouze nakopírovat složku obsahující modul do složky plugins. Důležité je, aby se složka modulu jmenovala stejně, jako modul samotný a začínala velkým písmenem. Výsledná stromová struktura může vypadat následovně:

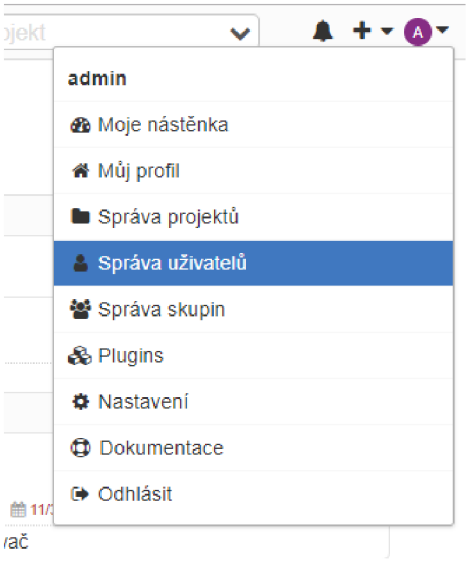
```
kanboard
├── app
├── assets
├── data
├── libs
├── plugins
│   ├── PluginA
│   └── PluginB
│       ├── Assets
│       └── Controller
│           └── ...
├── vendor
└── ...
```

## Používání aplikace

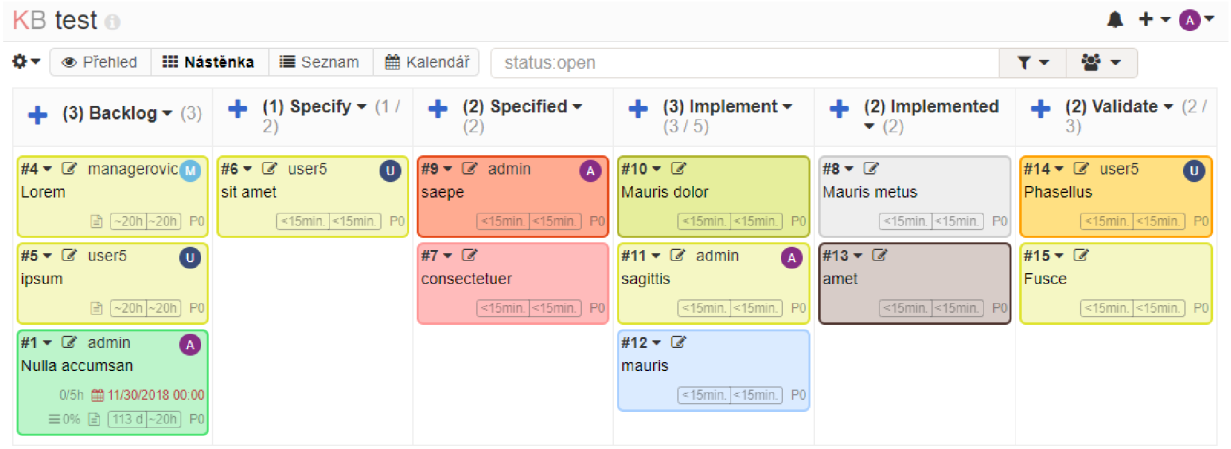
Po úspěšném nainstalování a otevření aplikace Vás uvítá obrazovka s přihlašovacími údaji. Pokud jste spustili aplikaci poprvé, existuje jen jeden uživatel a to `admin`. Přihlašovací jméno i heslo je `admin`. V pravém horním rohu aplikace pak můžete vytvořit další uživatele viz 2.3. Jakmile máte vytvořené uživatele, můžete je rozřadit do týmů. Stačí vytvořit nový tým a přidat jednotlivé uživatele. Takto vytvořené týmy lze pak přiřazovat na projekty. Vytvářet týmy sice není nutnost (na projekt se dají přiřadit i uživatelé bez týmů), ale přidávat projektu uživatele po jednom, není komfortní.

Nyní na řadu přichází již samotné projekty. Při vytváření nového projektu máte na výběr ze dvou možností, buďto soukromý, či běžný projekt. Soukromý je pouze pro daného uživatele a nemůže do něj přidávat další členy. Jakmile si vytvoříte projekt, je čas mu nastavit požadované parametry, jako například počet a jména sloupců, či dokonce další horizontální linky tzv. „swimlanes“. Nové položky můžete vytvářet pomocí tlačítka plus u každého sloupce, nově vytvořený úkol se tak rovnou umístí na správné místo. Dále v projektu můžete vytvářet štítky pro upřesnění kategorie úkolu. Tyto štítky pak můžete přidávat kartám při vytváření, či editaci. Příklad vytvořeného projektu můžete vidět na obrázku 2.4.

[3] https://www.virtualbox.org/
[4] https://www.docker.com/




Obrázek 2.3: Menu správy aplikace




Obrázek 2.4: Příklad tabule projektu

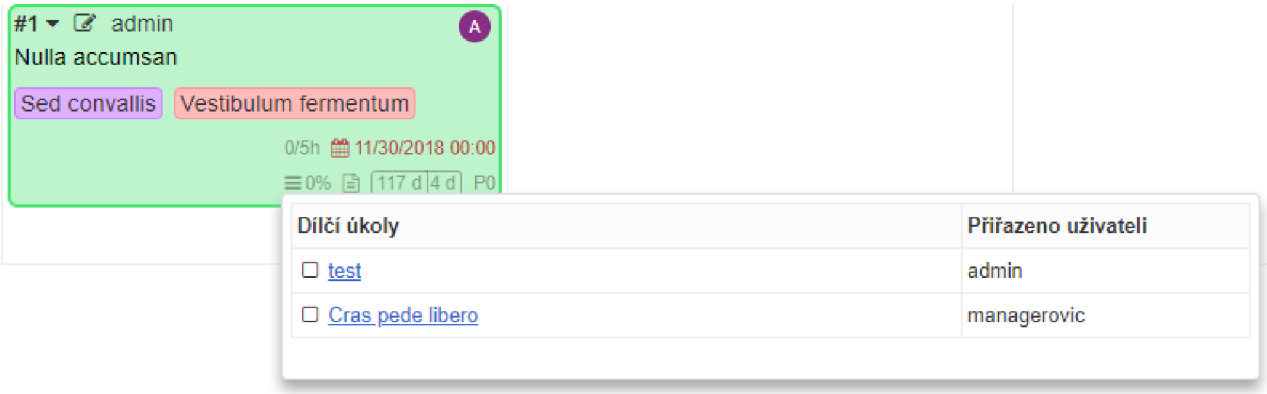
Uvedený příklad projektu má etapy vývoje rozděleny do šesti sloupců. Každý sloupec má své jméno a v závorce počet položek.

Dále některé sloupce mají nastavené limity. Například sloupec implement je limitovaný na pět položek. Pokud by tento limit byl překročen, upozorní uživatele červeným rozsvícením viz 2.5.


Obrázek 2.5: Příklad tabule projektu

Každá karta má následující atributy:

- jméno
- popis
- jméno uživatele, kterému je přiřazena
- přiřazené štítky
- odhadovaný čas na dokončení
- čas strávený v současném sloupci
- termín na dokončení
- seznam dílčích úkolů




Obrázek 2.6: Karta úkolu

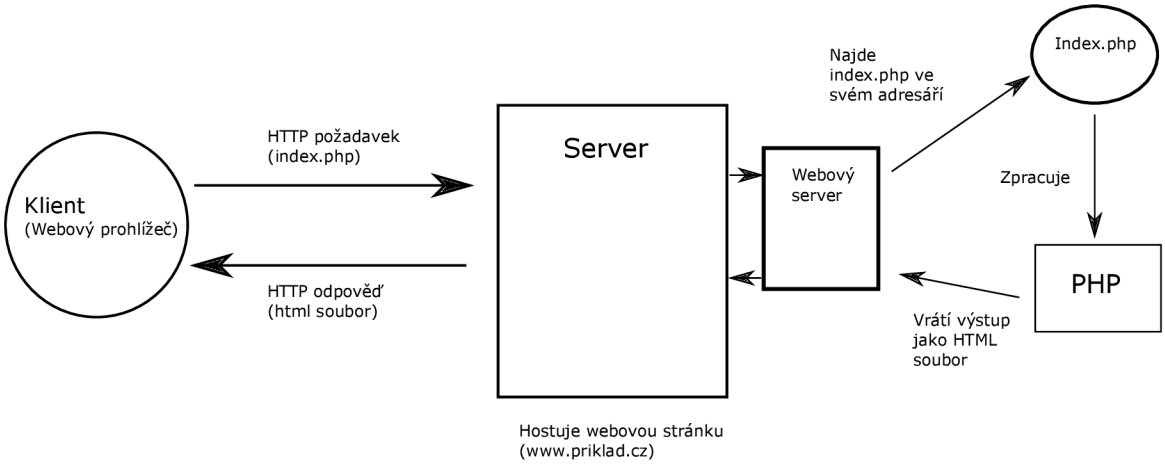
Jakmile člen týmu kartu vypracuje, přetáhne ji myší do následujícího sloupce tzv. „drag and drop“.

## 2.3 Použité technologie a implementace aplikace

Aplikace Kanboard je tzv. webová aplikace. Tyto aplikace jsou dostupné z webového prohlížeče a uživateli se jeví jako běžné internetové stránky. V dnešní době je pro uživatele velice obtížné rozeznat, jestli se jedná o běžnou webovou stránku, nebo webovou aplikaci. V následujících sekcích budou popsány nástroje, které jsou používány stávající aplikací.

### 2.3.1 Webový server

Webové aplikace fungují na principu klient-server. Klient (nejběžněji webový prohlížeč) pošle serveru požadavek. Server tento požadavek, zpracuje a klientovy odpoví. Webserver je počítač na kterém běží služba, která má na starost správu těchto požadavků. Mezi nejpoužívanější web serverový software patří v současné době Apache a Ngix[5].




Obrázek 2.7: Schéma komunikace mezi klientem a webovým serverem

Ke komunikaci se pro webové služby používá protokol `HTTP` [2]. Protokol běží na aplikační vrstvě L7. Zprávy protokolu jsou rozděleny na dvě části: Hlavičku a tělo. Hlavička zprávy klienta vypadá následovně:

```
METHOD resource HTTP/VERSION
Host: hostname
```

Na začátku je vždy typ použité metody. Nejčastěji používané metody jsou `GET` a `POST`. Metoda `GET` se používá pro běžné načítání obsahu stránky, nebo odesílání malého počtu dat (do 512B). `POST` je použita v případě, pokud chceme na server odeslat větší objem dat, nebo v případě, kdy není vhodné zobrazovat poslaná data v URL (například u formulářů). Resource je ta část adresy za lomítkem. Dále následuje verze HTTP protokolu a hostname. Hostname je adresa stránky, na kterou se snažíme připojit. Dále mohou být v hlavičce další informace od klienta, jako například, který prohlížeč uživatel používá a jakou sadu znaků podporuje.

[5] https://w3techs.com/technologies/overview/web_server/all

Webserver na tento dotaz odpoví zprávou se stavovým kódem. Odpovědi se dle stavového kódu dělí následovně:

- 2xx úspěch
- 3xx problém při přesměrování
- 4xx problém při vyřizování požadavku
- 5xx interní chyba serveru

### 2.3.2 Programovací jazyky

Webová aplikace Kanboard je primárně napsaná v jazyce `PHP`. Pro animace a vykreslování vyskakovacích oken je použita javascriptová knihovna `jQuery`[6].

#### PHP

Jedná se o skriptovací, objektově orientovaný, programovací jazyk určený především k vytváření dynamických, internetových stránek (původně „Personal Home Page“). V současné době se jedná o nejpoužívanější skriptovací jazyk na straně serverů[7]. Při použití PHP jsou veškeré výpočty prováděny na straně serveru (tzv. backend) a uživateli je ukázán pouze výsledek. Jedním z jeho poznávacích znaků je syntax. A to fakt, že každá proměnná začíná znakem "$". To, jestli je to dobře nebo ne, nechám na čtenáři.

PHP je slabě typovaný, dynamický jazyk. Tento směr byl kdysi velice populární. V poslední době se však od tohoto trendu ustupuje (například populární TypeScript[8] pro javascript). V PHP tak přibylo tzv. „napovídání typů“ (anglicky „type hinting“). Nejdříve ve verzi 5, která umožnila ošetření parametrů, aby byly typu objekt, pole, rozhraní nebo návrat funkce (anglicky „callback functions“). Ve verzi 7 se tato možnost rozšířila i na proměnné skalárního typu jako například `integer` nebo `string`. Oproti jiným, objektově orientovaným jazykům (například `C++`), lze dědit pouze jednu třídu.

V době, kdy snad každý měsíc přijde na svět nový, „revoluční“ javascriptový framework, se jedná o poměrně starý webový programovací jazyk (rok založení 1995). Tento fakt přináší celou řadu výhod, ale i nevýhod. Mezi výhody bych zařadil:

- množství funkcí ve standartní knihovně
- rozsáhlá dokumentace
- podpora u hostujících služeb
- velké množství projektů a kódů

Nevýhody z mého pohledu jsou:

- nekonzistence pojmenování funkcí
- nekonzistentní pořadí parametrů ve funkcích
- na odladění si musíte postačit s příkazem `echo`
- zabezpečení

[6] https://jquery.com/
[7] https://w3techs.com/technologies/overview/programming_language/all
[8] https://www.typescriptlang.org/

### PDO

Od verze 5.1 přibyla do PHP knihovna PDO. Jedná se o knihovnu pro komunikaci s databázemi. V současné době se všechny databázové transakce řeší právě touto knihovnou, nebo knihovnou z ní odvozenou viz 3.3.3. Mezi hlavní důvody, proč používat PDO patří:

- bezpečnost
- jedno rozhraní pro více databázových systémů
- objektově založené

Na začátku si vytvoříte instanci třídy PDO (či třídy, která třídu PDO dědí) a dále pracujete s tímto objektem. Výroky se nejdříve musí připravit metodou `prepare()` a až poté je možné vykonat metodou `execute()`.

### SQLite

SQLite je relační databázový systém napsaný v jazyce C. Na rozdíl od tradičních databázových systémů, které jsou založeny na principu klient-server se jedná pouze o knihovnu, která se stane součástí programu. V současné době se jedná o nejrozšířenější databázový systém na světě[9], nacházející se v každém mobilním telefonu, ve většině počítačů a webových prohlížečů.

SQLite vyhovuje `ACID` [5] požadavkům na databáze. V SQLite je celá databáze uložena jako jeden soubor. Databáze je tak snadno přenositelná. Co se příkazů týče, podporuje všechny běžně používané funkce relačních databázových systémů. SQLite není s výjimkou primárního klíče silně typovaný jazyk. Pokud máte pole typu NUMERIC, můžete do něj zapsat jak řetězec `"1234"` tak číslo `1234`, převod je uskutečněn za Vás. Podporovány jsou následující datové typy:

- `TEXT`
- `NUMERIC`
- `INTEGER`
- `REAL`
- `NONE`

Pokud byste tedy potřebovali datový typ `BOOLEAN` nebo `DATE`, musíte si ho vytvořit sami, například jako `INTEGER`.

[9] https://sqlite.org/mostdeployed.html

# Kapitola 3

# Návrh a implementace modulu SkillMatrix

Tato kapitola je rozdělena do tří sekcí ve stejném pořadí, v jakém probíhal vývoj. Nejprve bylo potřeba nastudovat, jak se moduly pro Kanboard vytváří, dále vytvoření návrhu aplikace a posléze implementace.

## 3.1 Způsob vytváření modulů

Jedna z hlavní části tvorby této práce bylo nastudovat, jak se moduly vytváří. Na toto téma je v dokumentaci Kanboardu [1] věnována kapitola „Plugin development“. Níže popíšu ty části, které jsem při práci použil.

### 3.1.1 Adresářová struktura

Moduly pro Kanboard mají pevně danou adresářovou strukturu. Tato struktura je zobrazena viz níže.

```
plugins
└── JménoModulu
    ├── Asset
    ├── Controller
    ├── LICENSE
    ├── Locale
    │   └── cs_CZ
    ├── Model
    ├── Plugin.php
    ├── README.md
    ├── Schema
    ├── Template
    └── Test
```

Jelikož je tato struktura pevně daná nemusíte ji vytvářet ručně. Pro tento účel existuje nástroj `cookiecutter`. Pro instalaci tohoto nástroje můžete použít příkaz:

```
pip install -U cookiecutter
```

Po úspěšné instalaci spustíte nástroj příkazem:

```
cookiecutter gh:kanboard/cookiecutter-plugin
```

Nástroj se Vás zeptá na jméno modulu, který budete vytvářet, jméno autora a další údaje popisující modul. Pokud některé údaje nechcete zadávat, nebo ještě přesně nevíte tyto informace, můžete krok přeskočit. Tyto údaje se dají doplnit později. Jakmile máte vygenerovanou adresářovou strukturu, musí být složka umístěna na správném místě viz 2.2 a modul musí být registrován, bez těchto dvou kroků Kanboard váš modul nepozná.

### 3.1.2 Registrace modulu

Pro registrování modulu, slouží soubor `Plugin.php`. Tento soubor musí obsahovat každý modul. Pokud jste použili nástroj `cookiecutter` je tento soubor pro Vás již připraven. Tento soubor obsahuje třídu `Plugin`, která dědí od třídy `Base` a obsahuje následující metody:

- `initialize()`
- `onStartup()`
- `getPluginName()`
- `getPluginDescription()`
- `getPluginAuthor()`
- `getPluginVersion()`
- `getPluginHomepage()`

Je možné do této třídy přidat další metody s pevně daným názvem, avšak u svého modulu jsem tuto možnost nevyužil.

Můžete si povšimnout, že v `get` metodách jsou informace, které jste zadávali do nástroje `cookiecutter`.

Pro nás však jediná důležitá metoda je metoda `initialize()`. Jakmile do aplikace Kanboard přidáte váš modul, je to právě tato metoda, kterou zavolá, a ve které se dozví všechny podstatné informace.

### 3.1.3 Překlad modulu

Jelikož ne všichni uživatelé mluví stejným jazykem, je vhodné modul přeložit. V době psaní této práce, modul podporuje jazyky angličtina a čeština.

Pro překlad aplikace slouží globální funkce `t($key)` a soubor `translations.php`. Tento soubor se musí nacházet ve složce `Locale`, v podsložce se jménem odpovídající zkratce daného jazyka. Pro češtinu tedy složka cs_CZ, pro francouzštinu fr_FR atp.

Soubor `translations.php` obsahuje asociativní pole překladů ve tvaru: `klíč => hodnota`. Příklad tohoto pole je uveden níže.

```
<?php

return array(
  'Update value' => 'Aktualizovat hodnotu',
  'Remove tag' => 'Odebrat štítek',
  'Remove skill' => 'Odebrat schopnost',
  ...
)
```

Výpis 3.1: Příklad asociativního pole překladů

Zpravidla je klíčem anglická fráze a hodnotou požadovaný překlad. Použití funkce je následující:

```
t('neco');
```

Ve výsledku se uživateli zobrazí překlad této fráze, nebo anglický termín, pokud tento klíč nebyl nalezen ve Vašem souboru překladů, ani v hlavním.

### 3.1.4 Vlastní cesty

Jestliže vytváříte modul, který pouze nemodifikuje stávající funkce, ale i přidává vlastní, je nutné si vytvořit nový adresový prostor. Musíte si tedy vytvořit vlastní ovladač tzv. controller, který stávající aplikace bude používat pro zpracování Vašich akcí. Tento nový zdrojový soubor se musí nacházet ve složce `Controller`. Nově vytvořená třída musí dědit od původní třídy `Basecontroller`. Dále je nutné uvést v registračním souboru údaj o existenci tohoto souboru. Provede se to následovně:

```
$this->route->addRoute('/moje/vlastni/cesta', 'mujController',
                       'mojeAkce', 'mujModul');
```

Nyní už Kanboard ví, kterou funkci zavolat pokud se uživatel dostane na adresu

```
/moje/vlastni/cesta
```

### 3.1.5 Registrování nových, pomocných tříd

V programu Kanboard existuje celá řada pomocných tříd tzv. helper tříd, které obsahují užitečné metody. Pokud však nenajdete metody, které hledáte, je tu možnost udělat si helper vlastní. Vaše pomocná třída vždy dědí od `Base` třídy a je umístěna ve složce Helper. V souboru `Plugin.php` je pak registrována následujícím způsobem:

```
$this->helper->register('mujHelper',
                '\Kanboard\Plugin\JménoMéhoPluginu\Helper\MujHelper');
```

Nyní můžete vaše metody volat ve kterékoliv části kódu příkazem:

```
$this->helper->mujHelper->mojeFunkce();
```

Jedinou výjimkou jsou šablony, kde se volání provádí následovně:

```
$this->mujHelper->mojeFunkce();
```

### 3.1.6 Šablony

Šablony tzv. templaty jsou podstatnou částí každé webové aplikace. V těchto šablonách se nachází veškeré uživatelské rozhraní. Vše, co uživatel na obrazovce vidí, bylo vygenerováno některou šablonou. Každá webová stránka se skládá z celé řady prvků. Pokud tyto prvky začnete počítat a třídit zjistíte, že se často opakují. Bylo by tedy zbytečné tyto části psát znovu a znovu. Stejně je tomu i u šablon.

Každá šablona se skládá z komponent. Komponenta se může skládat z jednoduchého prvku jako tlačítko, nebo z více prvků například tabulka, či dokonce celá část stránky může být jeden komponent. Tyto komponenty pak stačí v požadované části kódu zavolat. Tímto způsobem nejen redukujeme počet řádků kódu, což napomáhá k přehlednosti a udržitelnosti kódu, ale i zaručíme, že jednotlivé části jsou napříč stránkami konzistentní.

V Kanboardu je několik pomocných tříd ke generování prvků. Při mé práci se ukázaly jako nejužitečnější následující třídy:

- `ModalHelper`
- `UrlHelper`
- `UserHelper`
- `FormHelper`

Pokud tedy chceme v šabloně umístit tlačítko s odkazem můžeme použit tento příklad:

```
<?= $this->modal->mediumButton('ikona',t('Jméno tlačítka'),
                               'JmenoControlleru',
                               'jmenoAkceControlleru',
                               array('plugin' => 'JménoPluginu'))?>
```

Tato metoda zbytek práce odvede za nás.

Můžeme si vytvářet i vlastní komponenty a šablony. Pro takto vytvořené soubory slouží složka `Template`. Ve funkcích se Vaše vytvořené šablony vykreslují následujícím příkazem.

```
$this->response->html($this->render(
  'JmenoModulu:jmenoSouboru',
    array(
          'plugin' => 'SkillMatrix',
          'klic1' => 'Hodnota1',
          ...
      )))
```

Pokud chcete znovu využít stávající šablonu, např. dashboard, slouží k tomu tento příkaz:

```
$this->response->html($this->helper->layout->dashboard(
  'JmenoModulu:jmenoSouboru',
  array(
          'plugin' => 'SkillMatrix',
          'klic1' => 'Hodnota1',
          ...
      )))
```

Volání Vašich komponent v šablonách pak vypadá následovně.

```
<?= $this->render('JmenoModulu:jmenoSouboru', array(
          'plugin' => 'SkillMatrix',
          'klic1' => 'Hodnota1',
          ...
      )) ?>
```

### 3.1.7 Migrace databází

Aplikace Kanboard automaticky řeší migrace databází za Vás. Pokud tedy Váš modul potřebuje k funkci pozměnit stávající databázi, je tento proces velice přímočarý. Vaše schéma umístěte do složky `Schema` a název musí mít stejný jako databázový systém, pro něž příkazy píšete např. `Sqlite.php` pro sqlite atp. Vnitřní struktura souboru poté vypadá následovně:

```
<?php

namespace Kanboard\Plugin\MujPlugin\Schema;

const VERSION = 1;

function version_1($pdo)
{
  $pdo->exec('CREATE TABLE IF NOT EXISTS tabulka (
  "id" INTEGER PRIMARY KEY,
  "project_id" INTEGER NOT NULL,
  "neco" TEXT,
  FOREIGN KEY(project_id) REFERENCES projects(id) ON DELETE CASCADE
  )');
}
```

Konstanta `VERSION` obsahuje číslo poslední verze úpravy databáze. Pokud tedy zjistíte, že je potřeba databázi upravit, stačí si vytvořit novou funkci `version_X` a konstantu `VERSION` nastavit na novu hodnotu `X`.

Veškeré příkazy relační databáze jsou řešeny knihovnou `PDO` viz 2.3.2.

## 3.2 Návrh modulu

Ať už člověk řeší jakýkoliv problém, vždy musí začít návrhem. Bez této fáze, kdy se řešitel podívá na celý obrázek z vyšší perspektivy a rozdělí problém na jednotlivé části, by se vývoj ani tohoto modulu neobešel. Dobrý návrh dokáže ušetřit spoustu práce navíc, která by vznikla přehlédnutím některých faktů.

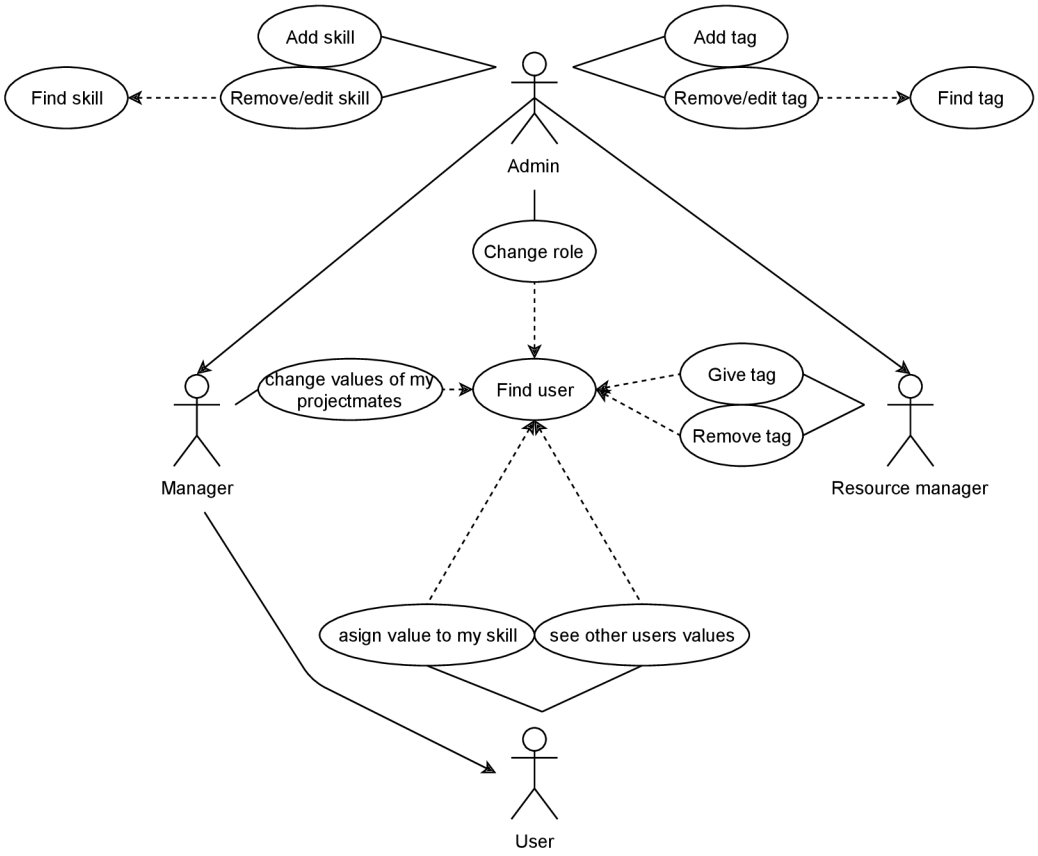
### 3.2.1 Diagram případu použití

Pro přečtení požadavků na vytvářený modul, jsem se nejdříve zamyslel nad tím, jak který uživatel bude modul používat. Přece jenom administrátor bude mít trochu jiné možnosti a povinnosti, než běžný uživatel, a proto by tento fakt měl návrh modulu reflektovat. Původní návrh modulu vypadal následovně. Jak vidíte na obrázku 3.1 tento návrh počítal se čtyřmi pevně danými rolemi.

- administrátor
- manažer
- manažer zdrojů
- běžný uživatel

Role administrátora je spravovat modul, tudíž po nainstalování potřebuje mít možnost přidat, smazat, či jinak editovat schopnosti a štítky tak, aby vyhovovaly dané oblasti nasazení aplikace. Další krok je přiřazení rolí uživatelům. Dále má přístup ke všem funkcím ostatních uživatelů pro případ, že by někde nastal problém.

Manažer má na starost správu hodnot, které si uživatelé přidělili u schopností. Pokud si uživatel přidělí hodnotu, která dle názoru manažera je v rozporu se skutečností, má




Obrázek 3.1: Původní diagram použití

možnost tuto hodnotu pozměnit. Každý manažer však má tuto pravomoc pouze u uživatelů, se kterými on sám pracuje na projektech. Nemělo by smysl, aby manažer mohl hodnotit uživatele, se kterými nepracoval.

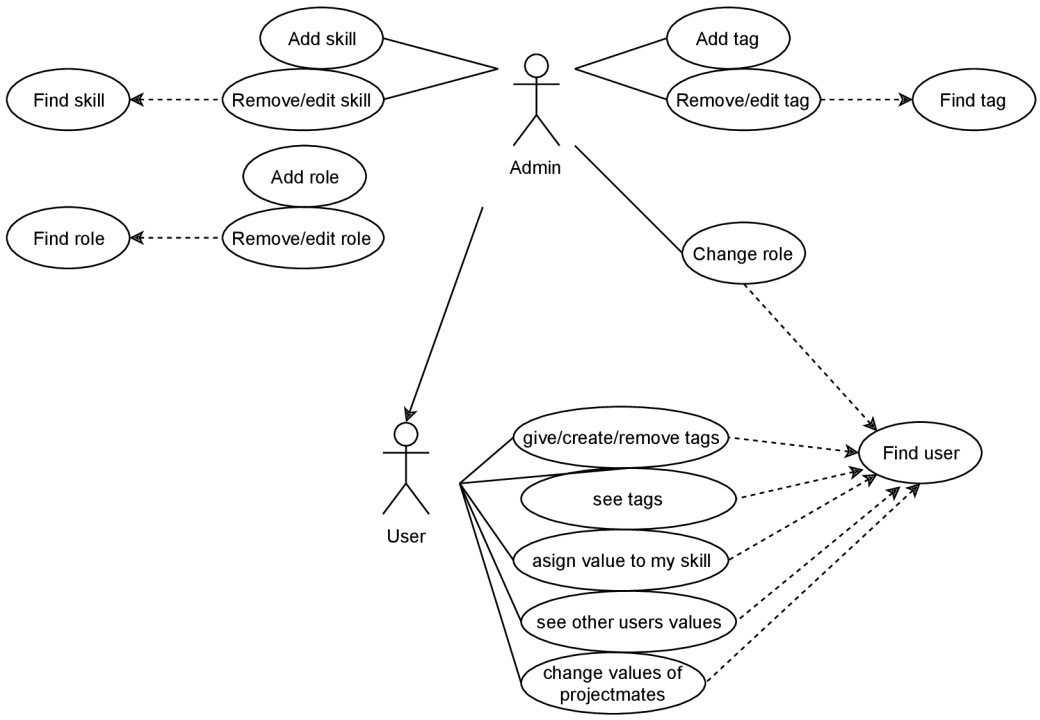
Manažer zdrojů má možnost přiřazovat uživatelům štítky. Pokud štítek doposud neexistuje, tak i vytvářet nové štítky. Tyto štítky slouží jako dodatečné informace, které můžou dopomoci k rozhodování, který uživatel by byl vhodný na ať už nový, či stávající projekt.

Běžný uživatel pouze vidí hodnoty schopností uživatelů a měnit může pouze svoje hodnoty.

Po implementaci těchto rolí, otestování a konzultaci, se však ukázalo, že pevně určené role nejsou ideálním řešením problému. Za zmínku stojí následující nedostatky řešení:

- Jak předat informaci administrátorovi, která role má jakou pravomoc?
- Co když administrátorovi nebude toto roztřídění vyhovovat?

Proto se zrodil nový návrh aplikace, který tyto problémy odstranil.




Obrázek 3.2: Nový a zároveň současný diagram použití

Tento model počítá pouze se dvěma pevnými rolemi, administrátorem a běžným uživatelem. Tyto role mají stejnou pravomoc jako v předchozím modelu tzn. administrátor má veškerou pravomoc, kdežto uživatel může editovat pouze své hodnoty schopností a o existenci štítů neví. Nově si však administrátor může vytvořit svoje role. Pokud tedy chce vytvořit roli, která může editovat schopnosti všech uživatelů, ale například nevidět štítky, má tuto možnost.

Dále je tu možnost editovat stávající role. Jestliže se tedy administrátor rozhodne, že uživatelé by měli vidět, jaké mají přiřazeny štítky, ale zároveň nechce, aby je měnili, ta možnost tu je. Práva jsou rozdělena do dvou kategorií a každá kategorie obsahuje tři úrovně.

Práva schopností:

- může editovat hodnoty schopností kohokoliv
- může editovat hodnoty schopností projektových spolupracovníků
- může editovat pouze sám sebe

Práva štítků:

- může přidávat štítky
- může pouze vidět štítky
- nevidí štítky

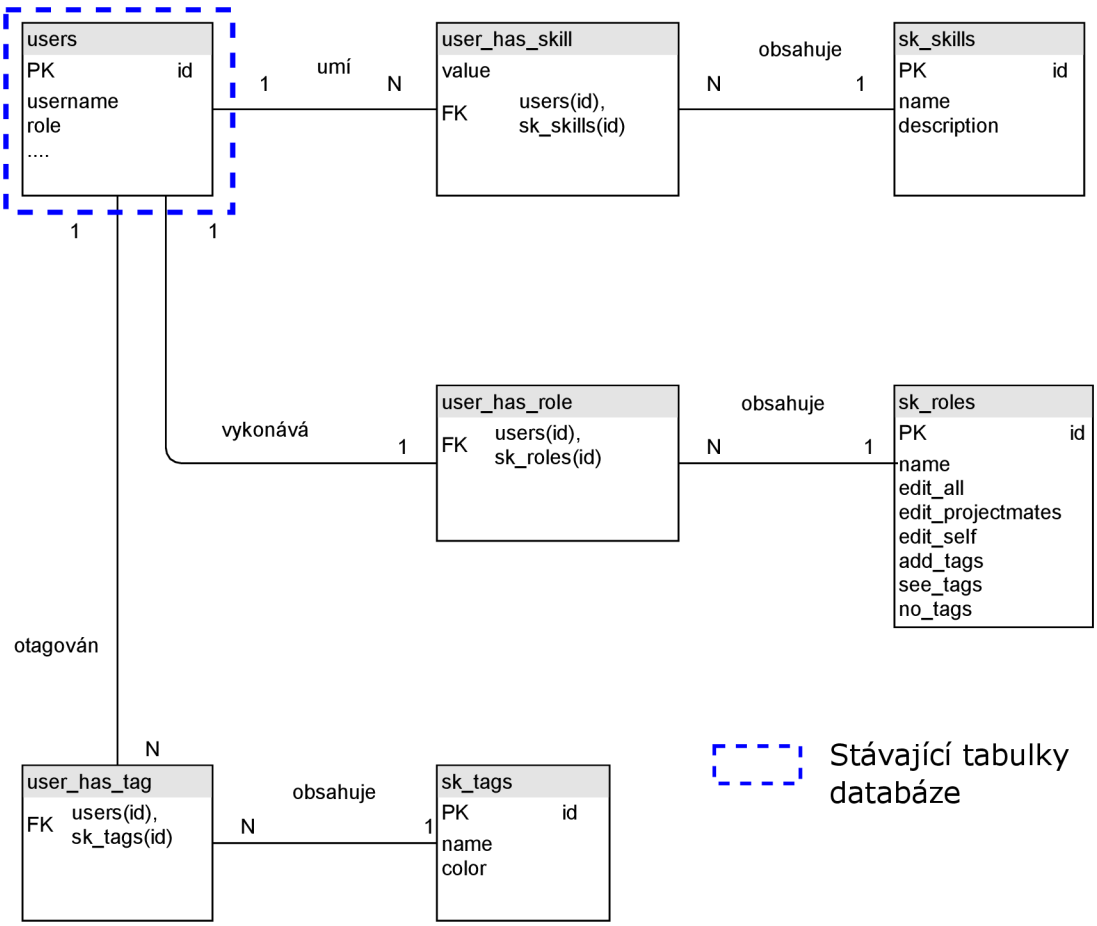
Z těchto množin posléze administrátor vybere požadovanou kombinaci práv.

### 3.2.2 ER diagram

Stávající databáze aplikace obsahuje celou řadu tabulek a jejich seznam najde v příloze 1. Po nastudování, která tabulka slouží k čemu, jsem došel k závěru, že k úspěšnému provázání tohoto modulu a stávající aplikace je potřeba pouze jedna stávající tabulka, a to tabulka se jménem `users`. Tato tabulka obsahuje dvacetšest dimenzí, v modulu byly však využity pouze tyto čtyři:

- `id`
- `name`
- `username`
- `role`

Schéma 3.3 znázorňuje strukturu nově vytvořených tabulek potřebných ke správnému chodu mého modulu.




Obrázek 3.3: ER diagram rozšíření stávající databáze

Každá schopnost, role i štítek má vlastní tabulky s požadovanými dimenzemi. Mezi tabulkami schopnosti a users je vztah N:N, jelikož každý uživatel může mít několik schopností a každá schopnost může mít několik uživatelů. Z toho důvodu bylo potřeba vytvořit pomocnou tabulku `user_has_skill` aby se vazba N:N převedla na dvě vazby 1:N a N:1.

Stejný postup byl aplikován i na tabulku tags. Každá dimenze `name` byla ošetřena vlastností `UNIQUE`. Využitím této vlastnosti bylo docíleno zabránění případu, kdy dvě, nebo více rolí, schopností či štítků mělo shodné jméno. Při vytváření vazeb u tabulky rolí byla situace poněkud odlišná. Rolí sice může být několik, ale každý uživatel má pouze jednu roli. Mezi tabulkami users a `user_has_role` je tedy vztah 1:1. Tento vztah je možný vyřešit přidáním stávající tabulce users nový atribut a tudíž by se tabulka `user_has_role` zrušila. Pro tuto možnost jsem se však nerozhodl z toho důvodu, že nechci nijak měnit schéma stávající aplikace.

## 3.3 Implementace

Po nastudování způsobu vytváření modulů a provedeném návrhu, nastoupila na řadu část implementace. V registračním souboru `Plugin.php` jsem přidal vlastní controller a nastavil ho na cestu `/skillmatrix`.

```
public function initialize()
{
  $this->route->addRoute('/skillmatrix', 'SkillMatrixController',
                         'load', 'SkillMatrix');
  ...
}
```

Dále jsem rozšířil současnou šablonu dashboardu o tlačítko Skillmatrix, které se odkazuje na cestu výše vytvořenou.

```
public function initialize()
{
  ...
  $this->template->hook->attach('template:dashboard:sidebar',
                                'SkillMatrix:dashboard/sidebar');
  ...
}
```

V dalším kroku jsem obohatil stávající množiny CSS pravidel a javascriptových funkcí o své vlastní.

```
public function initialize()
{
  ...
  $this->hook->on('template:layout:css',
                  array('template' =>
                  'plugins/SkillMatrix/Assets/css/custom.css'));
  $this->hook->on('template:layout:js',
                  array('template' =>
                  'plugins/SkillMatrix/Assets/js/functions.js'));
  ...
}
```

V posledním kroku jsem zaregistroval své dvě pomocné třídy `skillDatabaseHelper` a `skillFunctionsHelper`.

```
public function initialize()
```

```
{
  ...
  $this->helper->register('skillDatabaseHelper',
            '\Kanboard\Plugin\SkillMatrix\Helper\SkillDatabaseHelper');
  $this->helper->register('skillFunctionsHelper',
            '\Kanboard\Plugin\SkillMatrix\Helper\SkillFunctionsHelper');
}
```

Těmito příkazy jsem zaregistroval všechny soubory, nutné k registrování. V následujících podsekcích vysvětlím implementaci těchto částí. Pro případ, že by někdo chtěl více proniknout do funkčnosti kódu, jsem každou část řádně okomentoval tak, aby bylo možné vygenerovat dokumentaci nástrojem PHPdoc[1]. Programátor tak dostane kompletní seznam tříd a metod s popisem významu, vstupních parametrů a návratových typů.

### 3.3.1 Ovladač modulu

Metody této třídy by se daly rozdělit na čtyři kategorie:

- sekce administrátora
- sekce uživatele
- vykreslení modalu
- vykonání operace nad databází

Po kliknutí na tlačítko SkillMatrix viz 4.1 nebo zadáním adresy /skillmatrix se aktivuje metoda `load()` třídy `SkillMatrixController`. Metoda `load()` rozhodne o dalším kroku. Pokud je uživatel administrátorem modulu, invokuje se metoda první sekce tzn. `adminUserSection()`. V opačném případě metoda `userSection()`.

#### Sekce administrátora

Část administrátora byla pro přehlednost rozdělena do čtyř sekcí viz 4.2. Každá sekce sdružuje operace nad konkrétním typem dat. Algoritmus těchto sekcí je možno popsat následovně:

```
načti řetězec vyhledávání
načti informace o uživateli
načti záhlaví tabulky
načti informace daného typu pro vytvoření řádků tabulky

pokud řetězec vyhledávání není prázdný{
  vyfiltruj položky záhlaví tabulky
  vyfiltruj informace k vytvoření řádků tabulky
}
předej všechny načtené, případně vyfiltrované informace odpovídajícímu
pohledu
```

[1] http://docs.phpdoc.org/

### Sekce uživatele

Pro uživatele je v controlleru vyhrazena metoda `userSection()`. Její algoritmus je následující:

```
načti řetězec vyhledávání
načti informace o uživateli
načti záhlaví tabulky

pokud uživatel může editovat spolupracovníky{
  načti seznam těchto lidí
}
načti seznam uživatelů a jejich schopností
seřaď tento seznam
načti seznam uživatelů a jejich štítků

pokud řetězec vyhledávání není prázdný{
  vyfiltruj položky záhlaví tabulky
  vyfiltruj informace k vytvoření řádků tabulky
}
předej všechny načtené, případně vyfiltrované informace pohledu
userView
```

### 3.3.2 Šablony

Z důvodu většího počtu souborů byla složka `Template` rozdělena na následující podsložky:

```
Template
├── dashboard
├── modal
├── table
└── view
```

Složka `dashboard` obsahuje pouze jeden soubor, a to soubor `sidebar.php`. Tento soubor obsahuje pouze tlačítko Skillmatrix. Toto tlačítko se přidává do původní šablony `sidebar` viz 3.3.

Další složky budou probrány podrobněji.

#### modal

V této složce jsou umístěny všechny „vyskakující okna“ tzv. modaly. Obsah složky je následující:

```
modal
├── assignTagModal.php
├── editSkillValueModal.php
├── changeColorTagModal.php
├── changeDescriptionSkillModal.php
├── changeRoleModal.php
├── changeRoleRights.php
├── removeModal.php
└── renameModal.php
```

Každý soubor obsahuje konkrétní modal pro konkrétní funkci. Jelikož akce přejmenování a odstranění prvků jsou si velice podobné, sdružil jsem je vždy do jednoho souboru, tedy do souborů `removeModal.php` a `renameModal.php`.

Samotná implementace těchto modalů je velice jednoduchá. Vždy se jedná o prostou HTML kostru, která obsahuje pár funkcí pro vypsání hodnoty proměnné, či volání helper funkce pro vytvoření požadovaného elementu.

### table

V modulu jsou používány následující komponenty tabulek:

```
table
├── adminRoleTable.php
├── adminSkillTable.php
├── adminTagTable.php
├── adminUserTable.php
├── resourceManagerTable.php
└── userAndManagerTable.php
```

První čtyři tabulky, jsou použity v jednotlivých sekcí administrátora.

Tabulka `resourceManagerTable.php` je použita v případě, kdy uživatel má právo vidět či editovat štítky. Ve zkratce by se dalo fungování tabulky zapsat následujícím pseudokódem:

```
<table>
<thead>
<?php foreach ($hlavickaTabulky): ?>
    <td>vypiš jméno schopnosti a její tootlip</td>
<?php endforeach ?>
</thead>
<?php foreach ($poleUzivateluSeSchopnostmi): ?>
     <tr>
       <?php foreach ($hlavickaTabulky): ?>
         <td>
           <?php if záznam existuje): ?>
             vypiš jeho hodnotu
           <?php else: ?>
             vypiš znak '-'
         </td>
     </tr>
     <tr>
       <td>
         <?php if uzivatel může editovat štítky): ?>
           vypiš mu tlačítko pro přidávání
       </td>
       <td>
         <?php foreach ($poleUzivateluSeStritky): ?>
           <span>
             <?= $jmenoTagu ?>
             <?php if uzivatel může editovat štítky): ?>
               vypiš mu tlačitko pro odstranění štítku
           </span>
```

```
            </td>
          </tr>
      <?php endforeach ?>
  </table>
```

Tabulka `userAndManagerTable.php` je ochuzena o část řešící štítky.

### view

Každá obrazovka, na které se uživatel nachází, je řešena jako samostatný pohled tzv. view. Stejně jako v 3.3.1, je část administrátor rozdělena na čtyři pohledy. Obsah složky je proto následující:

```
view
├── adminsViewRole.php
├── adminsViewSkill.php
├── adminsViewTag.php
├── adminsViewUser.php
└── userView.php
```

Každý pohled se skládá z několika komponent. Většinou se jedná o tyto části:

- tlačítko pro vytvoření nového záznamu
- vyhledávací oblast
- tabulka s požadovanými informacemi

## CSS a javascript

Na rozdíl od jiných webových frameworků, kde každá komponenta má možnost vlastních kaskádových stylů (např. Angular[2]), je v této aplikaci jeden CSS soubor, který se aplikuje pro všechny komponenty. Bylo tedy potřebné vhodně zvolit názvy CSS tříd, aby nedošlo k předefinování současných a tím pozměněním současného vzhledu aplikace. K mému vlastnímu překvapení nakonec nebyla potřeba velkého množství nových CSS pravidel. Jediná pravidla, která jsem v mém modulu potřeboval a nebyla součástí původní množiny pravidel se týkala vzhledu štítků a drobné úpravy vzhledu tabulky.

Javascriptové funkce jsem nakonec použil pouze čtyři, kdy tři z nich jsou pouze kosmetického rázu. Ta čtvrtá, která výraznějším způsobem ovlivňuje funkcionalitu, byla použita u šablon tabulek `adminUserTable.php` a `resourceManagerTable.php`. Tato funkce vypadá následovně:

```
$(document).on("click", ".showTagButton", function () {
  var rowToShow = $(this).data("row_id");
  var rowToHighlight = (rowToShow * 2) +1;
  $("#table tr:eq("+rowToHighlight+") td").toggleClass("is-light");
  $('#'+rowToShow).toggle("fast","swing");
})
```

Tato funkce slouží k odhalování oblasti dedikované pro štítky. Uživateli se tak po kliknutí tato oblast rozbalí a po opětovném kliknutí opět schová. Toto řešení jsem zvolil z důvodu jednoduchosti implementace a z předpokladu, že počty záznamů nedosahují stovek. Pokud

[2] https://angular.io/

by se opravdu jednalo o stovky uživatelů a každý by měl desítky štítků, pravděpodobně bych zvolil řešení nástrojem AJAX[3].

### 3.3.3 Pomocné třídy

#### SkillDatabaseHelper

Pomocná třída `SkillDatabaseHelper` slouží k operacím s nově vytvořenými tabulkami relační databáze. Veškeré operace `SELECT`, `INSERT`, `UPDATE` a `REMOVE` nad tabulkami SkillMatrix modulu se nacházejí právě zde. Pro implementaci těchto operací jsem zvolil knihovnu `PicoDB`[4]. Jedná se o rozšíření knihovny PDO 2.3.2. Knihovna `PicoDB` podporuje databázové systémy: SQLite 2.3.2, MS-SQL[5], MySQL[6] a PostgreSQL[7].

Mezi hlavní výhody knihovny PicoDB patří jednoduchý syntax a nezávislost na zvoleném databázovém systému. V současné době modul SkillMatrix podporuje pouze databázový systém SQLite 2.3.2, avšak pokud by došlo k rozšíření podpory o např. MySQL, stávající kód by byl stále funkční. Jediné, co by bylo třeba doplnit, je soubor s vytvořením tabulek pro daný systém do složky `Schema` viz 3.1.7.

V příkladech níže jsou uvedeny SQLite příkazy a jeho PicoDB ekvivalenty.

```
SELECT * from mojeTabulka WHERE sloupec1=1
```

```
$this->db->table('mojeTabulka')->eq('sloupec1',1)->findAll();
```

```
SELECT jmeno from mojeTabulka WHERE ID>1
```

```
$this->db->table('mojeTabulka')->gt('id',1)->findOneColumn('jmeno');
```

```
UDPATE mojetabulka SET plat = 25000 WHERE vek = 35 AND jmeno='Radek'
```

```
$this->db->table('moje')->eq('vek', 35)
                        ->eq('jmeno', 'Radek')
                        ->update(array('plat' => 25000));
```

#### SkillFunctionsHelper

V této třídě jsou umístěny funkce, které pracují s již načtenými daty z databáze. Jedná se o metody potřebné k vyhledávání viz 3.3.4. Tyto metody aplikují na vstupní pole hodnot pravidla obsažená v řetězci `$searchQuery`. Každá metoda je určena k aplikování jiné části pravidel. Použití kombinací těchto metod je v programu pokryto veškeré vyhledávání.

### 3.3.4 Vyhledávání

S narůstajícím počtem dat vzniká potřeba tyto data nějakým způsobem filtrovat. Bylo tedy za potřeby implementovat funkci vyhledávání, pro komfortnější uživatelskou obsluhu. Implementace této funkce byla již nastíněna viz 3.3.3. Jedná se o několik funkcí, kde každá řeší jinou uživatelem zadanou podmínku. Výsledek z jedné funkce se použije na vstupu následující funkce, až je docíleno požadovaného výsledku. Takto je například implementováno vyhledávání pro běžného uživatele:

[3] https://api.jquery.com/category/ajax/
[4] https://github.com/elvanto/picodb
[5] https://www.microsoft.com/cs-cz/sql-server/sql-server-2017
[6] https://www.mysql.com/
[7] https://www.postgresql.org/

```
$tableHead = $this->helper->skillFunctionsHelper
                ->getFilteredTableHead($searchQuery,$tableHead);
$userSkillArray = $this->helper->skillFunctionsHelper
                ->getFilteredUsersBySkillValue($searchQuery,
                                                $userSkillArray);
$userSkillArray = $this->helper->skillFunctionsHelper
                ->getFilteredUsersByName($searchQuery,$user['name']
                                        ,$userSkillArray);
if ($rights['no_tags'] != 1){
    $userSkillArray = $this->helper->skillFunctionsHelper
                ->getFilteredUsersByTag($searchQuery,
                                        $userSkillArray);
}
```

Výpis 3.2: Implementace vyhledávání běžného uživatele

Můžete si povšimnout, že v uvedeném příkladě se nachází podmínka. Tato podmínka řeší příklad, že by uživatel, který nemá přidělené právo pro zobrazení štítků, chtěl podle nich vyhledávat. Mohlo by tak dojít k tomu, že by uživatelé zkoušeli, jestli nemají náhodou přiděleny štítky `XYZ`. Toto chování není žádoucí.

### Syntaxe vyhledávání

Při vytváření syntaxe vyhledávání jsem se držel stávajícího vzoru. Ten je následující:

```
atribut : hodnota
```

Lze zadat několik těchto konstrukcí najednou, v takovém případě jsou od sebe odděleny mezerou.

```
assigne:me due:tomorrow
```

V případě, že by některá hodnota byla více slovná, je nutné ji dát do uvozovek. Například takto:

```
assigne:"Petr Novák" due:tomorrow
```

V modulu SkillMatrix lze vyhledávat na základě následujících atributů:

- `user`
- `skill`
- `tag`

Hodnoty pro atribut `user` můžou být uživatelské přihlašovací jméno, jméno uživatele, nebo klíčové slovo `me`, toto slovo dosadí jméno aktuálního uživatele. Vyhledávání schopností lze možné agregovat. Lze tedy vyhledat jen ty uživatele, kteří mají u požadované schopnosti `X` hodnotu `Y`. V současné době jsou podporovány následující matematické operace:

- `>`
- `<`
- `>=`

- `<=`
- `=`

Výsledný příkaz, může vypadat například následovně:

```
skill:skill1>=2 skill:skill2>3
```

Vzhledem k tomu, že se tyto hodnoty pohybují v rozmezí 1-5, nepovažoval jsem za podstatné imlementovat několik podmínek pro jednu schopnost. Následující výraz je tedy neplatný:

```
skill:skill1>1 skill:skill1<4
```

Administrátor má množinu atributů, podle kterých lze vyhledávat, rozšířenou o následující, v závislosti v jaké sekci se právě nachází:

- id
- role
- color

Význam atributu `id` se liší v závislosti na sekci, v které se administrátor právě nachází. Pokud se vyskytuje v sekci schopností, slouží tento atribut k nalezení požadované schopnosti. V sekci štítků se tento atribut vztahuje na štítky atp. Pomocí atributu `role`, si může administrátor vyfiltrovat uživatele na základě jejich přidělených rolí. Položka `color` se týká sekce štítků. Administrátor si tak může například příkazem:

```
color:"dark grey"
```

vyfiltrovat všechny štítky tmavě šedé barvy.

# Kapitola 4

# Uživatelské rozhraní

Dobře udělané uživatelské rozhraní je v dnešní době klíčová komponenta úspěchu produktu. Nezáleží na tom, jak je Vaše aplikace revoluční a kolik funkcí obsahuje, pokud ji uživatel nebude umět používat, tak je k ničemu. Navíc žijeme v době, kdy na snad každý problém existuje celá řada řešení, takže Váš potencionální uživatel přejde ke konkurenci.

Cílem mého řešení uživatelského rozhraní pro modul SkillMatrix bylo vytvořit systém, který:

- je lehce pochopitelný
- nemění současné umístění ovládacích prvků
- pokračuje v duchu současného vzhledu
- snaží se omezit počet kliknutí na minimum

Lehká pochopitelnost je primárním účelem uživatelské rozhraní. Čím méně času stráví uživatel zjišťováním, které tlačítko vlastě provádí jakou akci, a kde že to tlačítko vlastně je, tím méně bude uživatel frustrován a tím více může věnovat času tomu, k čemu aplikace slouží.

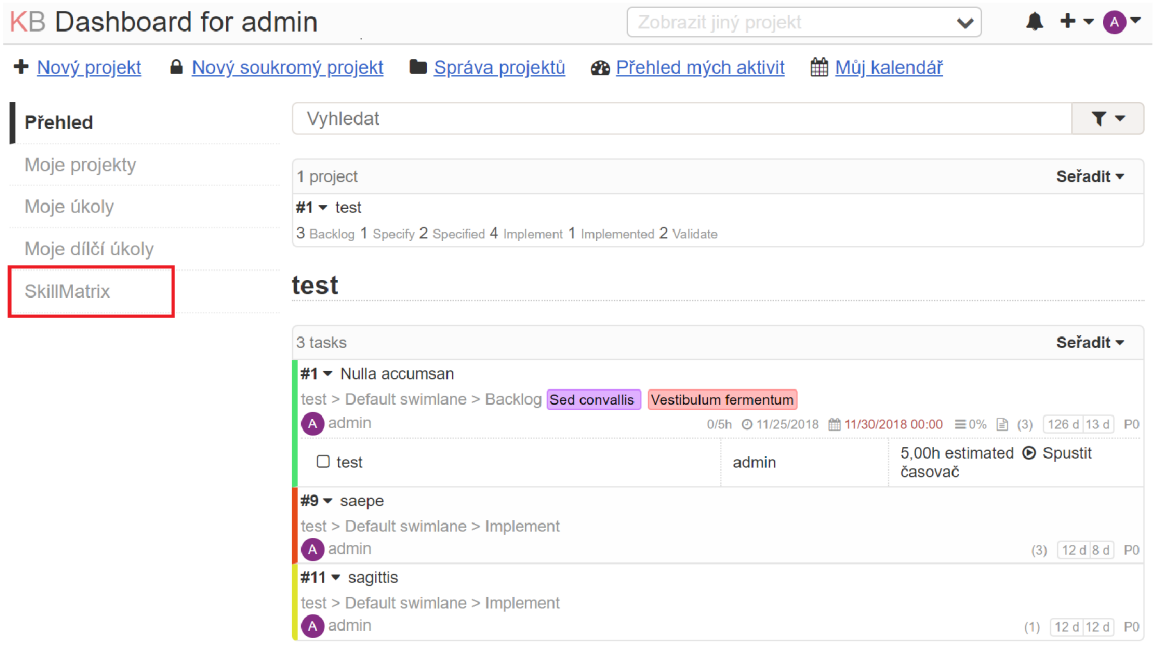
Zachování původního vzhledu a navázání na tento styl bylo nutné z důvodu, že se jedná o modul. Tento modul nemusí být přítomen od samotného začátku nasazení aplikace. Uživatelé mohou být již z dřívějšího používaní aplikace zvyklí na určitý vzhled a rozmístění prvků a kdyby toto celé bylo změněno jen kvůli tomu, že byl přidán nový modul, který třeba daný uživatel použije jen občas, asi by je to nepotěšilo. Při navrhování uživatelského rozhraní tohoto modulu jsem si vzpomněl na název knížky pana Steva Kruga „Nenuť mě přemýšlet“ [6]. Pokud rozmístím na obrazovku prvky, mělo by být na první pohled zřejmé, na který se dá kliknout, a jaké činnosti se týká.

Dále jsem se snažil minimalizovat počet kroků nutných k provedení operace. V celé aplikaci platí, že pokud chci změnit nějakou hodnotu (a jsem ve správné sekci), měly by k tomu stačit tři kroky. První pro vybrání hodnoty, druhý pro vybrání nové hodnoty a třetí pro potvrzení. Ve většině běžných případů odpovídá počet kroků počtu kliknutí. Pokud některý element slouží k destruktivní akci, vždy nejdříve vyběhne varovné okno, kde uživatel potvrdí tuto operaci kliknutím na červené tlačítko s odpovídajícím textem.

### Hlavní obrazovka

Při vstupu do aplikace se uživatel dostane na tzv. dashboard. Je to jakýsi rozcestník, kde se dále uživatel rozhoduje, kterou funkci vlastně chce použít. Proto jsem již zde umístil

tlačítko mého modulu 4.1. Pokud uživatel má jasný cíl, co chce udělat, neměl by se muset dlouze proklikávat, než se dostane do požadované sekce.



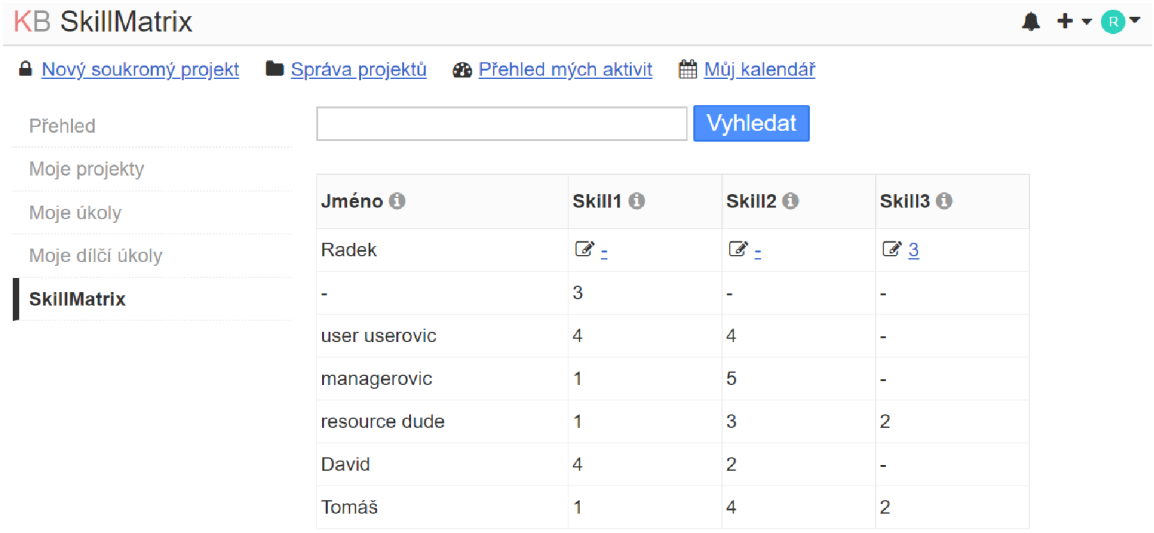

Obrázek 4.1: Hlavní obrazovka s přidaným tlačítkem pro modul SkillMatrix

Dále je tato kapitola rozdělena na dvě podkapitoly. První se týká uživatelského rozhraní běžného uživatele a druhá rozhraním pro administrátora.

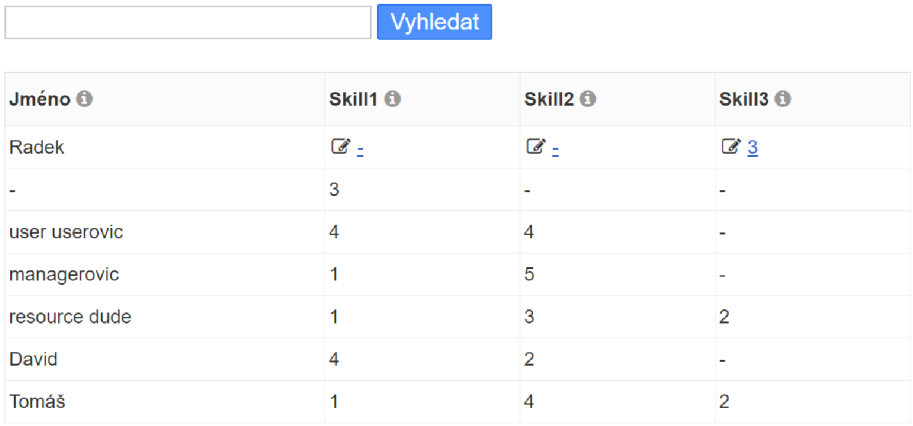
## 4.1 Uživatelské rozhraní běžného uživatele

Po kliknutí na tlačítko SkillMatrix se uživatel přemístí do hlavní sekce modulu, viz obrázek 4.2. Tato sekce vypadá velice podobně jako dashboard, pouze se změnila hlavní oblast pro obsah a nadpis. Tímto je docíleno, že i když se do této sekce uživatel dostane poprvé, náhodou, nebo „ze zvědavosti“, stále ví, kde se nachází a případně, jak se dostat zpět.

Nyní se zaměřím na část s nově zobrazeným obsahem. Jak je vidět na obrázku 4.3, je zde vyhledávání a tabulka. Hlavičku tabulky tvoří sloupec se jménem uživatele a další sloupce, každý odpovídající jedné schopnosti. Řádky tabulky dále tvoří jednotlivý uživatelé aplikace. Prvním záznamem v tabulce je pro přehlednost vždy současný uživatel. Každý uživatel má dále číselné ohodnocení u jednotlivých schopností. Jedná se o stupnici 1 až 5, kde 5 je nejlepší možná hodnota (uživatel je v této oblasti expertem) a 1 je pravý opak (uživatel tuto oblast vůbec neovládá). Pokud si uživatel nějakou schopnost ještě neohodnotil (nebo to neudělal někdo jiný), je tento údaj proškrtnutý.

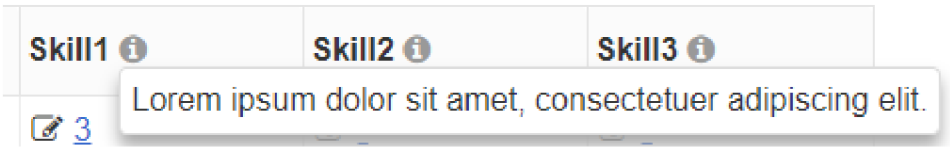



Obrázek 4.2: Hlavní část modulu




Obrázek 4.3: Tabulka uživatelů

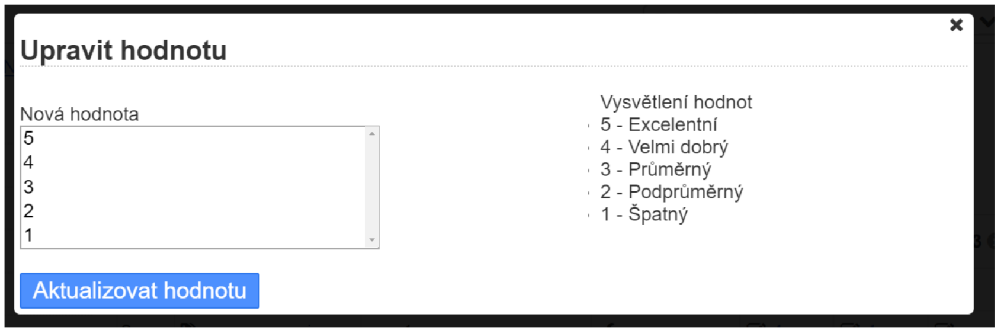
Pokud by uživatel náhodou nevěděl, co si pod názvem některé schopnosti představit, po najetím myši na ikonku vedle schopnosti se mu zobrazí detailnější popis viz obrázek 4.4.




Obrázek 4.4: Detailní popis schopností

Dále si můžete všimnout na obrázku 4.2, že ne všechny hodnoty vypadají stejně. Některé vypadají „klikatelně“. Po jejich kliknutí se zobrazí uživateli nové okno tzv. modal viz obrázek 4.5. V pravé části se uživatel dozví, co která hodnota znamená, v levé části si ji kliknutím vybere a ve spodní části se nachází tlačítko pro uložení nové hodnoty. Pokud uživatel má

v kontextu tohoto modulu dostatečnou pravomoc viz sekce 3.2, může editovat i hodnoty ostatních uživatelů.




Obrázek 4.5: Modal zobrazující výběr nové hodnoty pro schopnost

V mém modulu nemají uživatelé pouze schopnosti, ale i štítky tzv. tagy. Obrázek 4.6 ukazuje, jak vypadá tabulka, pokud uživatel může vidět, popřípadě editovat štítky. Jak jde vidět z obrázku, oproti první tabulce, jsou zde jména uživatelů „klikatelná“ a mají u sebe ikonku štítku. Po jejich kliknutí se rozbalí nová sekce obsahující štítky a jejich ovládací prvky.

| Jméno | Skill1 | Skill2 | Skill3 |
|---|---|---|---|
| managerovic | 1 | 5 | - |
| user userovic | 4 | 4 | - |
| - | 3 | - | - |
| resource dude | 1 | 3 | 2 |
| David | 4 | 2 | - |
| Radek | - | - | 3 |
| Tomáš | 1 | 4 | 2 |

Obrázek 4.6: Tabulka v případě, kdy uživatel může vidět, či dokonce editovat štítky

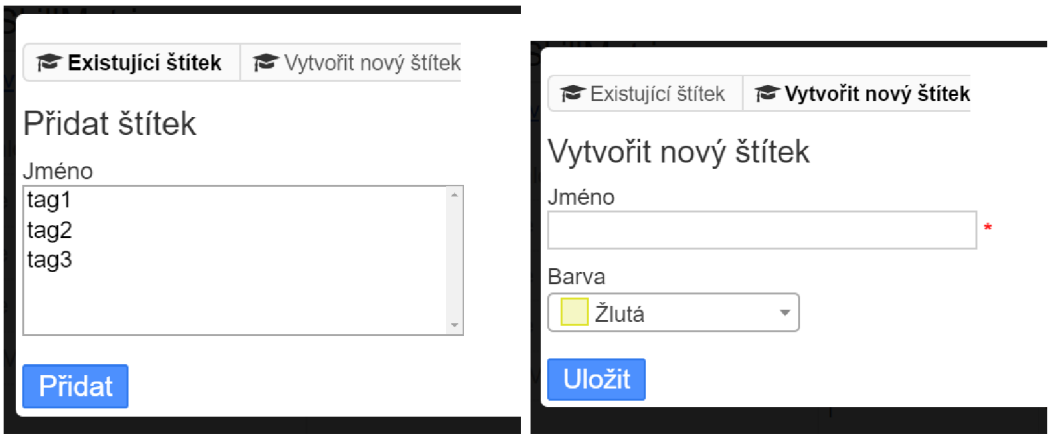
Na obrázku 4.7 je zobrazeno porovnání těchto sekcí, kde uživatel může tagy pouze vidět, nebo i upravovat. Kliknutím na křížek u daného štítku se objeví výstražné okno s dotazem, jestli uživatel chce opravdu štítek odebrat a po potvrzení se uživateli odebere.


Obrázek 4.7: Levé zobrazení odpovídá případu, kdy uživatel může editovat štítky, v pravém zobrazení štítky pouze vidí

Tlačítko s ikonkou plus a názvem „Přidat štítek“ slouží k přidávání štítků. Po kliknutí se zobrazí nové okno rozdělené na dvě záložky.

První záložka slouží pro případ, že uživatel chce přidělit štítek, který již existuje. Stačí pouze vybrat ze seznamu možností a potvrdit volbu.

Druhá záložka slouží pro případ, že uživatel požadovaný štítek v nabídce nenašel, a proto potřebuje vytvořit nový. Nachází se zde prvek pro zadání jména nového štítku a seznam barev pro zvolení. Po zadání informací a potvrzení se štítek vytvoří a rovnou přidá uživateli. Nemusí se tak uživatel proklikávat, aby štítek vytvořil a pak znovu proklikávat, aby ho našel v nabídce existujících a zvolil.




Obrázek 4.8: Levý obrázek zobrazuje možnost přidání stávajícího štítku, pravý možnost vytvoření nové štítku a následné přidělení

## 4.2 Uživatelské rozhraní administrátora

Původní návrh této sekce byl velice odlišný od zobrazení běžného uživatele. Pro každou akci existoval box, ve kterém byly dva elementy. První pro zvolení současné hodnoty některého záznamu (například jméno schopnosti) a druhý pro zadání nové hodnoty (například nové jméno schopnosti). Toto řešení bylo výhodné z pohledu vývojáře. Bylo snadno implementovatelné a velice rychle se na něm dalo testovat, které funkce backendu už fungují a které se ještě musí dodělat. Z pohledu uživatele to však byla noční můra obsahující nejen tyto problémy:

- Jak mám znát id uživatele, či ostatních záznamů, když nevidím databázi?
- Co když chci změnit několik záznamů konkrétního uživatele?
- Proč nevidím všechny atributy hledaného záznamu najednou?
- Jak rychle poznám, jestli se změny provedly?
- Kde že byl ten box, který hledám?

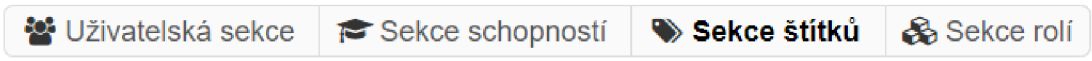
Po několika dalších pokusech nakonec vznikla verze dnešní.

Dnešní vzhled sekce je rozdělen na 4 záložky:

- uživatelská sekce
- sekce schopností
- sekce štítků

- sekce rolí

Prvek záložek je vidět na obrázku 4.9 a jeho vzhled byl zvolen tak, aby vypadal stejně, jako záložky uživateli známé z jiných částí programu a aby každá možnost měla napovídající ikonku. Dále pro zachování konzistence mají jednotlivé sekce obdobný vzhled. Sekce budu dále popisovat ve stejném pořadí, jak jsou vyobrazeny v programu.




Obrázek 4.9: Záložky sekcí administrátora

### Uživatelská sekce

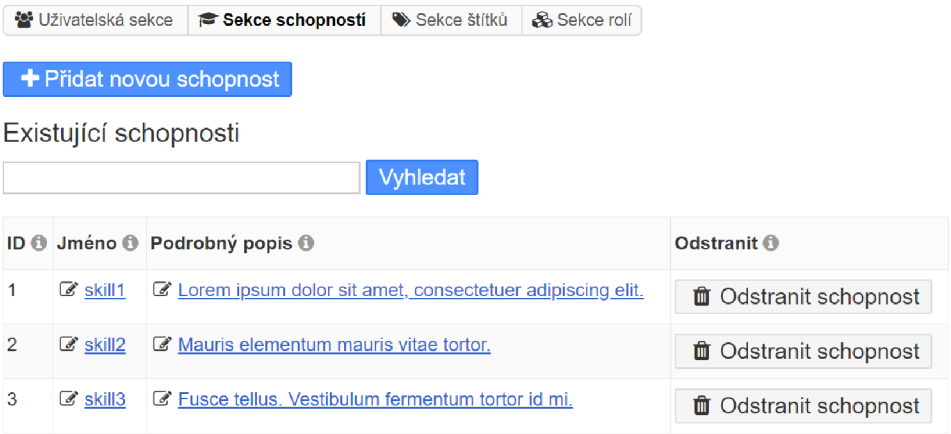
Tato sekce vypadá velice podobně, jako sekce běžného uživatele. Opět je zde vyhledávání a tabulka téměř identického vzhledu. Rozdíl je v tom, že administrátor modulu má v tabulce více sloupců a veškerou pravomoc. Přibyly zde sloupce jako id uživatele, jeho uživatelské jméno a role. Administrátor může měnit všechny hodnoty atributů uživatelů v rámci modulu. Jedná se o následující operace:

- měnit hodnoty u schopností
- přidávat, případně odebírat uživatelům štítky
- měnit role uživatelů

Změna uživatelské role se provede kliknutím na roli daného uživatele, vybráním nové role v seznamu a následném potvrzením.

### Sekce schopností

Jak jde vidět na obrázku 4.10, oproti uživatelské sekci zde přibylo nové tlačítko. Toto tlačítko slouží k vytvoření nové schopnosti. Po jeho kliknutí se rozbalí oblast obsahující všechny elementy nutné k této operaci.




Obrázek 4.10: Sekce schopností

V tabulce je pak možné měnit jméno a podrobný popis schopností, případně požadovanou schopnost z databáze odstranit.

### Sekce štítků

Sekce štítků je vedena ve stejném duchu jako sekce schopností. Obsahuje tlačítko pro vytvoření nového štítku, vyhledávání pro případ, že by štítků bylo velké množství a tabulku štítků. Tabulka umožňuje změnu jména a změnu barvy štítku.

### Sekce rolí

Poslední částí uživatelského rozhraní administrátora je sekce obsahující operace týkajících se rolí. Opět se zde vyskytuje tlačítko pro vytvoření nové role a tabulka obsahující název role, jejich pravomoc a tlačítko na případné odstranění. Z ohledem na malý počet kombinací všech práv u rolí jsem do této sekce nepřidával komponentu vyhledávání.

# Kapitola 5

# Testování

Testování je podstatnou částí vývoje každého produktu. Je jedno, jestli je váš produkt psací pero, automobil či software. Bez testování se neodhalí případné nedostatky produktu. Proto je vhodné testovat v průběhu, aby případná změna ovlivnila co nejmenší počet souvislostí. Je důležité si uvědomit, že přes veškeré testování, může výsledek stále obsahovat chyby. Nejde totiž zaručit, že produkt je bezchybný. Pouze jde zaručit, že produkt neobsahuje chyby z testované množiny. Čím je tato množina obsáhlejší, tím je větší pravděpodobnost, že výsledek bude v pořádku.

V následující sekci je popsán způsob vytváření automatických testů, který byl použit při vývoji tohoto modulu.

## 5.1 Automatizované testy

Činnost testování aplikací je ze své podstaty repetitivní. Při každé přidané funkcionalitě je nejen potřeba ověřit její správná funkčnost, ale i zjistit, jestli náhodou nedošlo k poškození původního celku. Proto je tedy nutné otestovat nejen novou část programu, ale i stávající. A jelikož se tento proces neustále opakuje, je vhodné tuto činnost zautomatizovat.

Aplikace Kanboard používá k automatizaci testů „framework“ `PHPUnit`[1]. Tento nástroj není obsažen ve standartní instalaci PHP, a proto si ho musí uživatel sám doinstalovat. Buďto budete postupovat dle dokumentace a opíšete tři řádky do konzole, nebo stačí balíček `phpunit` stáhnout pomocí správce balíčků, pokud používáte linux. Správnou konfiguraci nástroje si můžete ověřit zadáním následujícího příkazu ve složce, kde máte umístněnou instalaci Kanboardu.

```
phpunit -c tests/units.sqlite.xml
```

V době psaní této práce má aplikace Kanboard napsáno 1073 testů.

Zde bych chtěl upozornit na možné problémy. První problém je ten, že pokud Kanboard stáhnete jako zip, nestáhnou se potřebné složky s testy a to i přesto, že budete stahovat poslední vezi z hlavní větve githubu. Jediné řešení je si onu větev naklonovat příkazem:

```
git clone https://github.com/kanboard/kanboard
```

Druhý problém se týká samotného nástroje `PHPUnit`. Ve verzi 6 došlo u tohoto nástroje ke změně způsobu načítání závislostí bez ohledu na zpětnou kompatibilitu. V současné verzi 8 tento problém stále přetrvává a jelikož byly původní testy napsány dříve, jedná se

[1] https://phpunit.de/

o problém. Naskytují se dvě možná řešení. Buďto změníte načítání problematických částí kódu ve zdrojových souborech testů, nebo použijete některou předchozí verzi (já použil verzi 5).

### Vytváření automatických testů pro modul

Veškeré Vámi vytvořené testy by měly být umístěny ve složce `Test`. Dále je vhodné, pokud je aplikace rozsáhlejší, je rozdělit stejně jako zdrojové soubory. Struktura adresáře obsahující testy pro modul `SkillMatrix` vypadá následovně:

```
plugins
└── SkillMatrix
    ├── ...
    ├── Test
    │   ├── Helper
    │   │   ├── SkillDatabaseHelperTest.php
    │   │   └── SkillFunctionsHelperTest.php
    │   └── PluginTest.php
    └── ..
```

Spuštění těchto testů provedete následujícím příkazem:

```
phpunit -c tests/units.sqlite.xml plugins/SkillMatrix/Test/
```

Tyto testy jsou zaměřeny na ověření správné funkčnosti pomocných tříd `SkillDatabaseHelper` a `SkillFunctionsHelper`.

Na začátku každého testu se vytvoří tzv. container. Jednotlivé operace se dále provádí nad tímto objektem. Po inicializaci objektu dojde k vytvoření nové databáze. Databáze obsahuje stejné údaje, jako kdybyste program poprvé spustili. Je zde tedy pouze jeden uživatel (administrátor) a hodnoty, které modul vytváří při nainstalování viz 3.1.7. Použitím tohoto kontejneru je dosaženo skrytí současného stavu databáze. Testy proto nijak neovlivní stávající databázi. I kdyby v některém testu došlo k chybě, po skončení testů se nic nestane.

Princip testovacích funkcí je velice jednoduchý. Nejprve provedete testovanou operaci a dále sledujete její návratovou hodnotu. Když se shoduje se vzorem, proběhlo vše v pořádku. Pokud ke shodě nedošlo, nastal někde problém. Další testování pak ukáže, zda se chyba nachází v programu či v testu.

Jednoduchá testovací funkce vypadá následovně:

```
public function testGetUserRole(){
  $skillDatabaseHelper = new SkillDatabaseHelper($this->container);
  $this->assertSame('app-admin', $skillDatabaseHelper->getSkillUserRole(1));
  $this->assertSame(null, $skillDatabaseHelper->getSkillUserRole(2));
  $this->assertSame(null, $skillDatabaseHelper->getSkillUserRole(42));
}
```

Výpis 5.1: Otestování role administrátora

V současné době je pro modul `SkillMatrix` napsáno 14 testů. Tyto testy jsou napsány tak, aby vždy ověřovaly jednu konkrétní sadu operací.

Jelikož jsem autorem jak testů tak i programu, používal jsem při vytváření testů metodu bílé skříňky tzv. „whitebox“. Při této metodě je tvůrci testů známa implementace jednotlivých funkcí. Může se tedy zaměřit na komplexní části implementace daných operací. Navíc může při potížích snadno odhalit, kde se stala chyba.

## 5.2 Konečné testování modulu

Pro konečné testování modulu jsem vytvořil testovací množinu uživatelů. Vytvořená množina obsahovala sedm uživatelů. Tito uživatelé byly rozděleni do týmů a přiděleni na projekty tak, aby výsledek simuloval reálné použití. Každý uživatel měl dále svoji roli v rámci `SkillMatrix` modulu a na základě těchto rolí, se ověřoval správný chod aplikace.

# Kapitola 6

# Závěr

Cílem této práce bylo vytvoření modulu pro sledování znalostí uživatelů pro Kanboard.org. Tento cíl práce byl splněn v celém rozsahu. Došlo jak k nastudování nutných bodů, tak i k návrhu a implementaci. Výsledný modul umožnuje dynamické přidávání schopností a hodnocení uživatelů. Editace těchto hodnot je podmíněna dostatečnými přístupovými právy. To, který uživatel má jaká přístupová práva, je odvozeno od uživatelských rolí v rámci modulu. Role modulu je možno dynamicky vytvářet, mazat, či jinak editovat. Podle hodnot těchto atributů je pak možné jednotlivé uživatele vyhledávat. Dále v modulu existují štítky, které jsou uživatelům přidělovány. Štítky poskytují jednoduché informace a i zde je možnost, uživatele podle nich hledat. Konečný vzhled i chování modulu je vytvořeno tak, aby splynulo se stávající aplikací a nepůsobilo rušivým dojmem. Modul je navržen a implementován tak, aby nezasahoval do stávající databáze a nedošlo tak ke konfliktu s ostatními rozšířeními. K ověření správného chování byla použita kombinace automatických testů a manuálních operací z pohledu uživatele. Myslím si, že výsledek je nejen funkční, ale i z uživatelského hlediska použitelný.

Na začátku práce jsem netušil, jakým způsobem je možné řídit větší projekty (u školních projektů o dvou, třech lidech tento problém nikdy nenastal). Proto jsem rád, že jsem se díky této práci seznámil s přístupem Kanban a naučil se i něco jiného, než další programování.

V budoucnu by se tato práce dala rozšířit o analytiku. V současné době si uživatelé vkládají vlastní hodnocení, popřípadě ho mění manažer, pokud s hodnotou není spokojen. Jsou to však subjektivní hodnoty. Mohlo by tedy být zajímavé generovat tyto hodnoty na základě dříve splněných projektů a zpětné vazby. Největší problém bych viděl v získání dat z dlouhodobého používaní.

# Literatura


[1] Kanboard dokumentace. [online].
URL https://docs.kanboard.org/en/latest/

[2] Berners-Lee, T.; Fielding, R.; Frystyk, H.: Hypertext Transfer Protocol – HTTP/1.0. online.
URL https://tools.ietf.org/html/rfc1945

[3] Brechner, E.: *Agile Project Management with Kanban*. Developer Best Practices, Microsoft Press, první vydání, 2015, ISBN 978-0735698956.

[4] Huang, C.-C.; Kusiak, A.: Overview of Kanban systems. 1996.
URL https://pdfs.semanticscholar.org/d1d9/d4fc00a195f6bdd44357ada54605a8d09deb.pdf

[5] Kroenke, D. M.; Auer, D. J.: *Databáze*. CPress, 2014.

[6] Krug, S.: *Don't Make Me Think: A Common Sense Approach to Web Usability, 2nd Edition*. New Riders Press, druhé vydání, 2005, ISBN 0321344758,9780321344755.

[7] Naik, M. R.; Kumar, E. V.; Goud, B.: Electronic Kanban System. 2013.
URL https://pdfs.semanticscholar.org/0005/95dd378b6d5bcd4bc1bc68a01cd8638fc851.pdf

| Jméno |
|---|
| action_has_params |
| actions |
| column_has_move_restrictions |
| column_has_restrictions |
| columns |
| comments |
| currencies |
| custom_filters |
| groups |
| invites |
| last_logins |
| links |
| password_reset |
| plugin_schema_versions |
| predefined_task_descriptions |
| projectn_activities |
| project_daily_column_stats |
| project_daily_stats |
| project_has_categories |
| project_has_filters |
| project has_groups |
| project_has_metadata |
| project_has_notification_types |
| project_has_roles |
| project_has_users |
| project_role_has_restrictions |
| projects |
| remember_me |
| sessions |
| settings_has_users |
| subtask_time_tracking |
| subtasks |
| swimlanes |
| tags |
| task_has_external_links |
| task_has_files |
| task_has_links |
| task_has_metadata |
| task_has_tags |
| tasks |
| transitions |
| user_has_notifications |
| user_has_unread_notifications |
| users |

Tabulka 1: Seznam všech tabulek stávající databáze