České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Aplikace Skill Matrix

Bakalářská práce

Marek Landa

Studijní program: Otevřená informatika
Studijní obor: Software
Vedoucí práce: Ing. Božena Mannová, Ph.D.

Praha, Srpen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Landa** Jméno: **Marek** Osobní číslo: **434796**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Aplikace Skill Matrix

Název bakalářské práce anglicky:

Skill Matrix Application

Pokyny pro vypracování:

Navrhněte a implementujte aplikaci Skill Matrix, podle specifických požadavků zadavatele. Skill Matrix slouží k zaznamenávání a vyhledávání v seznamech znalostí pracovníků firmy, které mohou sloužit k přidělování pracovníků na jednotlivé projekty firmy.
1) Seznamte se s problematikou a analyzujte existující systémy poskytující podobnou funkcionalitu.
2) Na základě vyhodnocení těchto poznatků, specifikujte požadavky na funkcionalitu navrhovaného systému s přihlédnutím k požadavkům zadavatele.
3) Seznamte se s technologiemi potřebnými pro vytvoření aplikace.
4) Navrhněte a implementujte aplikaci.
6) Otestujte aplikaci včetně uživatelských testů a výsledky vyhodnoťte.
7) Při zpracování využívejte vhodně prostředky SI.

Seznam doporučené literatury:

[1] Pressmann R. S.: Software Enegineering,
[2] https://www.analyticsinhr.com/blog/create-skills-matrix-competency-matrix/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Božena Mannová, Ph.D., kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2020** Termín odevzdání bakalářské práce: **14.08.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Božena Mannová, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, Srpen 2020

.....
Marek Landa

Abstrakt

Skill Matrix je nástroj, který slouží k evidenci znalostí a dovedností zaměstnanců. Tento nástroj umožňuje nejen ukládání dat o schopnostech, ale umožňuje i vyhledávání v nich. Konkrétní implementace nástroje Skill Matrix může být provedena různými způsoby, například v MS Excel. Cílem této práce je navrhnout implementaci nástroje Skill Matrix, která bude lépe vyhovovat stanoveným požadavkům a dokáže nahradit stávající nevyhovující řešení založené na tabulkovém procesoru.

Součástí práce je analýza požadavků zadavatele a návrh vhodného technologického řešení. Práce obsahuje funkční aplikaci, která je doplněna jak o automatické testy, tak i o testy uživatelské.

Klíčová slova: skill matrix, webová aplikace, Java, Angular, Spring, SQL databáze

Abstract

Skill Matrix is a tool for records of employees' skills and capabilities. Such tool enables storage of the data and searching through them as well. Implementation of Skill Matrix can be done in various ways, for example by using MS Excel. The goal of this thesis is to design and implement Skill Matrix application according to provided specifications. Such application will be a replacement for an existing insufficient solution which is implemented by a spreadsheet.

Tasks performed during work on this thesis include specification analysis, application design and selection of suitable technologies. The application deploys automatic tests. In the late stage of development we held usability tests with real users.

Keywords: skill matrix, web application, Java, Angular, Spring, SQL database

Poděkování

Chtěl bych poděkovat mé vedoucí práce Ing. Boženě Mannové Ph.D. za trpělivost a pomoc při tvorbě této bakalářské práce. Dále bych rád poděkoval všem, kteří mi věnovali čas při odborných konzultacích. V neposlední řadě děkuji i rodině a kolegům za podporu a zázemí.

Seznam obrázků

Seznam zkratek

ACID atomicity, consistency, isolation, durability. 10

AOP Aspektově orientované programování. xi, 16

BASE basically available, soft state, eventual consistency. 10

CI Continuous Integration. 12

DI Dependency Injection. 15

DTO Data Transfer Object. xiii, 17

ERP Enterprise Resource Planning. 1

HR Human Resources. 1, 5

HTML HyperText Markup Language. 14

IoC Inversion of Control. 15

JDK Java Development Kit. 9

JPA Java Persistance API. 17

JPQL Java Persistance Query Language. xi, 16

JSON JavaScript Object Notation. 11

NoSQL Not only SQL. 10

POJO Plain Old Java Object. 17

REST Representational state transfer. 11

SOAP Simple Object Access Protocol. 11

SQL Structured Query Language. xi, 10, 16, 17

VIM Vi IMproved. 18

XML Extensible Markup Language. 11

Obsah

Kapitola 1

Úvod

Skill Matrix[24] je nástroj sloužící k zaznamenávání a vyhledávání v seznamech znalostí pracovníků společností. Má často formu jednoduché Excel tabulky nebo je v lepším případě součástí ERP nebo HR systémů. Pokud jsou ale potřeby společnosti specifické nebo jsou potřeba podrobné metody analýzy a vizualizace dat, je nutné mít Skill Matrix vytvořenou na míru.

V této práci řeším návrh a implementaci řešení podle specifických požadavků firmy. Úkolem je vybrat vhodné technologie a provést realizaci jejímž výstupem bude řešení na míru ve formě webové aplikace. Nedílnou součástí implementace je také automatické testování a následné uživatelské testy.

1.1 Popis problematiky Skill Matrix

Skill Matrix je ve většině případů tabulka zaznamenávající úroveň znalostí jednotlivých zaměstnanců. Může sloužit k vhodnému přerozdělování volných kapacit na nově vznikající projekty nebo na projekty, jež vyžadují zapojení dalších lidí. V našem případě bude Skill Matrix využívána nejenom k přehledu znalostí zaměstanců, ale i k zobrazení vývoje jejich znalostí a pomůže tak sledovat odborný růst zaměstnance.

Skill Matrix také může sloužit ke zjištění toho, které znalosti v kterém týmu chybí a dát prostor pro doplnění těchto znalostí ať už přidáním dalšího člena do týmu, interním školením nebo jiným způsobem.

V tomto případě pracuji se zadáním od nejmenované pražské firmy, která Skill Matrix používá, ale současná implementace není dostačující a přináší mnoho problémů. Není tak možné naplno využít data, která se ve Skill Matrix nachází. Proto se zadavatel rozhodl zadat požadavky na novou aplikaci, která by se lépe přizpůsobila konkrétním potřebám

firmy.

Kapitola 2

Původní řešení

Prvním krokem je shrnutí původního řešení zadavatele, které není technicky vyhovující. Informace o současném řešení mi také pomohly s návrhem aplikace a její implementací jelikož je tak možné se vyvarovat problémům, které v řešení byly.

2.1 Použité technologie

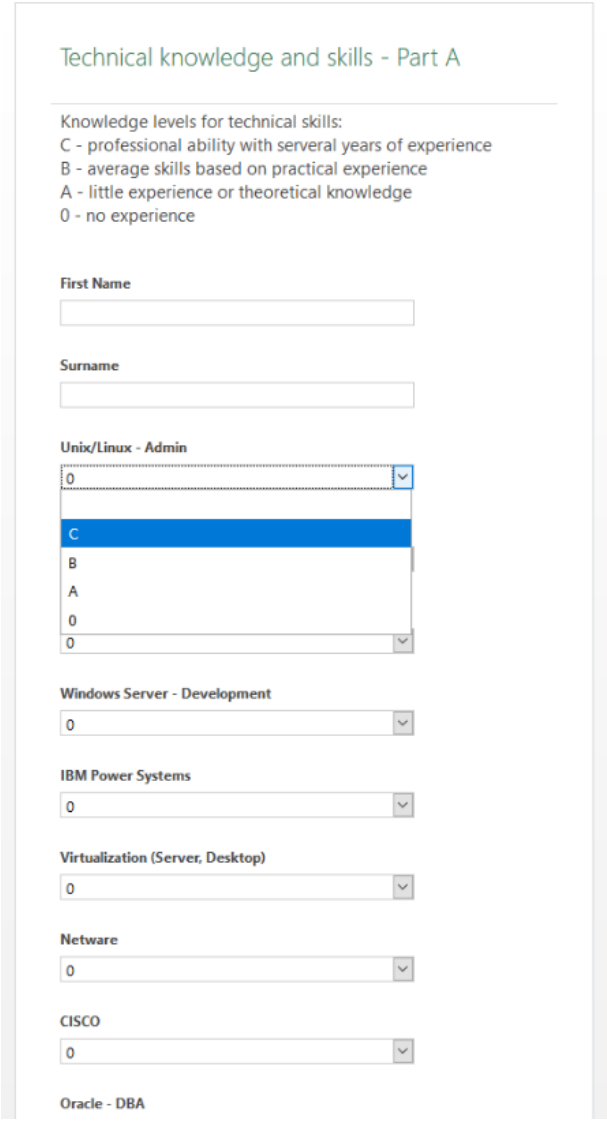
V době zadání této práce se v rámci firmy pro získání informací o odborných znalostech zaměstanců používá technologie Microsoft Forms [13]. Jedná se o platformu, která umožňuje tvorbu formulářů. V tomto případě do předem připraveného formuláře zaměstanec vyplní své jméno a dále u předdefinovaných znalostí vybere jednu z následujících možností:

- C - profesionální zkušenost s několikaletou praxí
- B - průměrná zkušenost na základě aktivního používání znalosti
- A - malá zkušenost nebo teoretická znalost
- 0 - žádná zkušenost

Data zadaná do těchto formulářů jsou dále uložena do formy tabulky Microsoft Excel. V tomto formátu je pak možné je filtrovat a vyhledávat v nich.

2.2 Nedostatky původního řešení

Řešení, které firma v současné době používá je jednoduché a umožňuje sběr informací o zkušenostech zaměstnanců. Je také možné informace zobrazit ve formě tabulky. Zároveň má ale současné řešení nedostatky, jako například:



Obrázek 2.1: Původní řešení pomocí Microsoft Forms

	A	B	C	D	E
1	Firstname	Surname	Unix/Linux - Admin	Windows Server - Development	IBM Power Systems
2	Marek	Landa	B	A	0
3	Petr	Omacka	C	0	B

Obrázek 2.2: Ukázka Excel souboru obsahující původní data

- Zaměstanec musí vyplnit několik těchto formulářů.
- Zadaná data je poté potřeba ručně spojit přes zadané jméno zaměstnance.
- Ztrácí se přehled o aktuálnosti dat, neexistuje nic jako pravidelná kontrola.
- Je potřeba ručně odstranit údaje o zaměstnancích, kteří již ve firmě nepracují.
- Člověk zpracovávající daná data si musí pamatovat jakou úroveň znalostí která zkratka značí.
- Chybí stromová struktura znalostí (kategorie, podkategorie, jednotlivé znalosti).
- Chybí informace o délce praxe s použitím dané technologie/znalosti.
- Chybí informace o vývoji úrovně znalosti jednotlivých zaměstnanců.
- Přístup k vyplněním formulářům má pouze oddělení HR, které dále poskytuje požadovaná data jednotlivým vedoucím.

Vzhledem k těmto negativním stránkám byly specifikovány požadavky pro novou aplikaci, která by měla stávající řešení nahradit a poskytnout větší množství funkcí. Zároveň by také měly být odstraněny existující nedostatky.

2.3 Dostupná obdobná řešení

Jediným nalezeným podobným produktem byla aplikace *Advanced Skill Matrix* [18]. Toto řešení je placené a neobsahuje některé z požadovaných funkcionalit. Sice řešení obsahuje i některé funkcionality navíc, nicméně ty nejsou potřebné, tudíž pro naše použití nevyvažují vyšší cenu produktu.

Kapitola 3

Specifikace zadání

Zjišťování podrobností zadání probíhalo rozhovorem se zainteresovanými osobami (tj. budoucími uživateli). Jedním z důležitých specifik implementované aplikace je dostupnost aplikace z webového rozhraní. Webová aplikace byla upřednostněna hlavně kvůli snadnému přístupu a jelikož není nutné pro použití instalace žádného software. Jediný požadavek je webový prohlížeč.

Dalším důležitým aspektem jsou role uživatelů. Je potřeba naimplementovat role od administrátorské úrovně až po řadové zaměstnance, kteří mohou pouze upravovat svůj profil. Podle přiřazené role budou mít jednotliví uživatelé různé pravomoce a budou moct upravovat různá data. Jedná se o jasné zlepšení oproti současnému stavu, kdy má k datům přístup pouze pár uživatelů, kteří je musí ručně distribuovat dál.

Vzhledem k tomu, že původní řešení neumožňovalo strukturovat informace o znalostech, je potřeba naimplementovat rozdělení znalostí do kategorií a podkategorií.

Použití vhodné databáze pro ukládání dat je dalším krokem, který je potřeba realizovat a to včetně jejího návrhu. Dále aplikace musí mít vhodné řešení logování a hlášení chybových a provozních stavů.

Kapitola 4

Analýza a návrh

Během analýzy byly zpracovány a vyhodnoceny sesbírané požadavky od zadavatele. Na základě analýzy byl dále vytvořen návrh aplikace. Jedním z kroků návrhu byl i výběr konkrétních technologií, které byly použity při samotné implementaci.

4.1 Případy užití

Standardní uživatel může provádět následující akce:

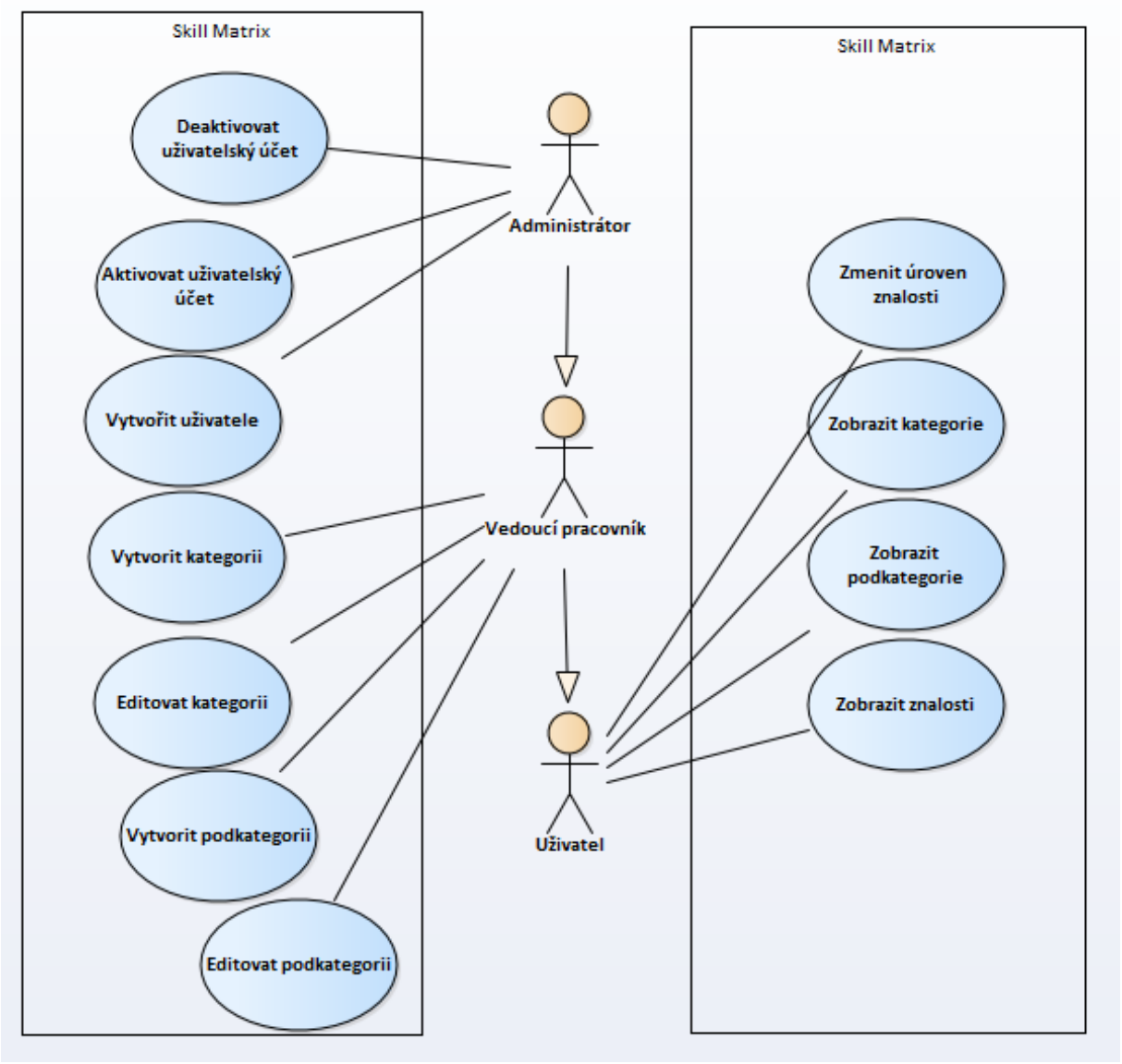
- zobrazit znalosti
- upravit úroveň znalosti
- zobrazit kategorie
- zobrazit podkategorie

Uživatel vedoucí role může provádět všechny výše uvedené akce a navíc tyto další:

- vyhledávat uživatele podle znalostí
- vytvářet a editovat kategorie
- vytvářet a editovat podkategorie

Administrátor může provádět všechny výše uvedené akce a navíc tyto další:

- přidávat uživatele
- aktivovat a deaktivovat uživatele
- prohlížet detaily jednotlivých entit v databázi



Obrázek 4.1: Diagram případů užití

Výběr použitých technologií je klíčovým krokem při implementaci webové aplikace. U použitých technologií bylo dbáno především na 2 hlavní kritéria:

- Údržba systému.
- Provozní náklady.

Při výběru technologií jsem se tedy přiklaňoval k těm, které jsou ve firmě již používané, jelikož umožňují jednodušší údržbu. Dalším kritériem pro výběr je nutnost pořízení placené licence. Upřednostňuji technologie, které nevyžadují placené licence k provozu.

4.2 Serverová část

V rámci zadavatelské firmy se většina systémů vyvíjí v jazycích Java, C++ a Python. Používají se i další technologie, ale pouze okrajově. Pro serverovou část jsem vybral Java Springboot. Jeho velkou výhodou je, že má vestavěný aplikační server Tomcat [16]. Díky tomu odpadá nutnost výběru aplikačního serveru.

Dále byla pro vývoj zvolena OpenJDK [14]. Oproti Oracle JDK jsem ji zvolil, protože není zpoplatněno její použití. Java Development Kit je produktem Oracle Corporation, který obsahuje soubor základních nástrojů pro vývoj aplikací pro platformu Java. V tomto případě by se sice jednalo o malý počet zakoupených licencí, protože server obsluhující webovou část by zatím vyžadoval pouze malé množství souběžně běžících instancí. I přesto jsem se rozhodl pro neplacenou variantu.

4.2.1 Nástroje pro build aplikace

Mezi v současnosti nejvíce používané build tooly dnes patří především Maven [2] a Gradle [6].

Maven je historicky starší technologie, nabízí širokou škálu pluginů pro jednotlivé kroky buildu. Ty v Mavenu představují seřazený list, následující krok vždy závisí na seznamu předchozích kroků.

Oproti tomu Gradle reprezentuje jednotlivé kroky v orientovaném acyklickém grafu. Toto dovoluje jednodušší inkrementální build, neboli lze přebuildit pouze části které se změnily a na nich závislé následovníky.

I přes tyto výhody jsem se rozhodl použít Maven pro jeho větší rozšířenost a jednoduchost.

4.3 Databáze

Jednou z klíčových částí aplikace je databáze, jelikož hlavním cílem je ukládání, správa a filtrování dat. Existuje několik druhů databází a mnoho nástrojů, které je možné pro implementaci použít.

4.3.1 Relační vs nerelační databáze

Výběr vhodné databáze se řídí mimo jiné také jejím modelem. Existuje několik typů modelů, kterých se databáze mohou držet. Každý z nich je potom vhodný pro jiné aplikace.

ACID model [8] je koncept, který se snaží dosáhnout určitých vlastností u databázových transakcí. Tyto vlastnosti jsou:

- Atomicity - Transakce se mohou skládat z vícero dílčích příkazů, atomicita nám zajišťuje, že se buď provede celá transakce nebo žádná její část.
- Consistency - Pokud by transakce způsobila nekonzistenci dat, pak je zastavena a data jsou vrácena do původního stavu.
- Isolation - Jednotlivé transkace jsou od sebe oddělěné, pokud by se měly vykonat 2 transakce najednou, navzájem se neovlivní.
- Durability - Jakmile je transakce dokončena, její výsledek je zaznamenán trvale a nemůže se ztratit ani při výpadku databáze.

ACID model implementují relační (SQL) databáze. Ty jsou vhodné pro aplikace, kde jsou důležité vztahy mezi jednotlivými entitami. V našem případě chceme spojovat více tabulek dohromady a zároveň nebudeme operovat nad velkými daty (Big Data). Do relační databáze můžeme efektivně vkládat data a vyhledávat v nich nebo je aktualizovat, což jsou akce, které jsou pro naši aplikaci klíčové. Relační databáze ale umožňuje i další rozšířené funkcionality.

NoSQL databáze se místo ACID modelu drží BASE modelu:

- Basically Available: Zaručuje odpověď na dotaz, i kdyby měla být chybná.
- Soft state: Stav systému se může v průběhu jeho existence změnit bez zásahu uživatele.
- Eventual consistency: Systém dosáhne konzistentního stavu v určitém časovém horizontu.

BASE model je vhodný pro velké objemy dat. Nerelační (NoSQL) databáze bohužel nedosahují plného ACID modelu, tudíž jsou pro nás nevhodné. Zvolil jsem tedy relační databázi.

Při výběru konkrétní databáze jsem nejprve vyloučil komerční řešení jako například Oracle DB. Vybíral jsem tedy z několika opensource řešení, v užším výběru byl jak PostgreSQL [15], tak MariaDB [12]. Obě řešení vyhovovala mým požadavkům, nakonec jsem zvolil PostgreSQL. Tato databáze je určena v mém případě pro použití v produkčním nebo testovacím prostředí.

Pro vývojové prostředí byla použita H2 databáze [7] s persistencí na disku. Persistence na disku umožňuje zachovávat data mezi jednotlivými spuštěními apikace bez nutnosti spouštění databáze samostatně.

Pro automatické testy byla použita H2 databáze s persistencí v paměti. Persistence v paměti způsobí smazání databáze včetně dat při každém novém spuštění testů. Předešlé testy tak neovlivní výsledek dalších testů.

4.4 Klientská část

Pro klientskou část aplikace jsem se rozhodoval především mezi JavaScriptovými frameworky Angular [1], React [17] a Vue [23]. V současné době se jedna o nejpopulárnější frameworky s podobným počtem uživatelů [5]. Nakonec jsem zvolil Angular. Především kvůli jeho komponentové architektuře a podpoře Dependency Injection (popis dále v sekci 5.4.1).

4.5 REST-API

Pro komunikaci mezi serverovou a klientskou částí bylo použito REST-API. Representational state transfer (REST) je architektura rozhraní, navržená pro distribuované prostředí. RESTful Web services umožňují přistupovat a operovat nad poskytovanými daty pomocí jednotného a předdefinovaného seznamu bezstavových operací.

Bezstavovost nám umožňuje oddělení implementace klientské části od serverové, jelikož odpověď serveru není závislá na předchozí interakci s klientem. To je mimo jiné jeden z důvodů proč bylo použito právě REST-API.

Další alternativou by bylo použití SOAP (Simple Object Access Protocol). SOAP využívá výměny zpráv ve formátu XML přes síť. Protože používá XML místo JSON formátu, je potřeba zapisovat více dat a zpracování zpráv je náročnější [21]. SOAP neumožňuje bezstavovost což je vlastnost, kterou vyžaduji.

Pro dokumentaci vytvořeného REST-API byl použit nástroj Swagger [3]. Swagger je opensource framework pro návrh, tvorbu a dokumentaci REST-API.

4.6 Verzování

Verzování je užitečný nástroj při vývoji softwaru, přináší řadu výhod jako jsou:

- možnost nezávisle pracovat na stejné části projektu pro více vývojářů
- tvorba zálohy starších verzí
- přehled změn - snáze lze zjistit co se od předchozí verze změnilo, případně co mohlo způsobit novou chybu

Pro verzování byl použit Git, oproti další hlavní používané technologii SVN (Subversion) nabízí Git podporu vytváření commitů offline a lokálních branchí.

Pro hostování Git repozitáře jsem vybral Github [1].

4.7 Continuous integration

Continuous integration (CI) je součástí automatizace kontroly vývoje. V mém případě je použito na provedení a oveření následujících akcí:

- build projektu
- spuštění serverových testů
- spuštění klientských testů

Zvažované CI nástroje:

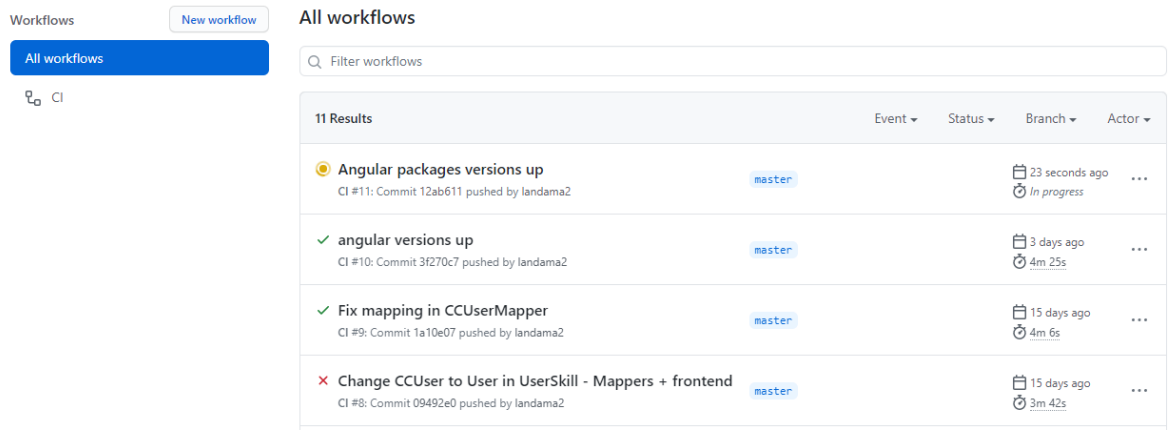
- TeamCity
- Jenkins
- Github Actions

Teamcity je komerční produkt od firmy JetBrains, který může být self-hosted, ale jedná se o produkt, který je výhradně placený. Teamcity jsem tedy zamítnul kvůli neexistující neplacené verzi.

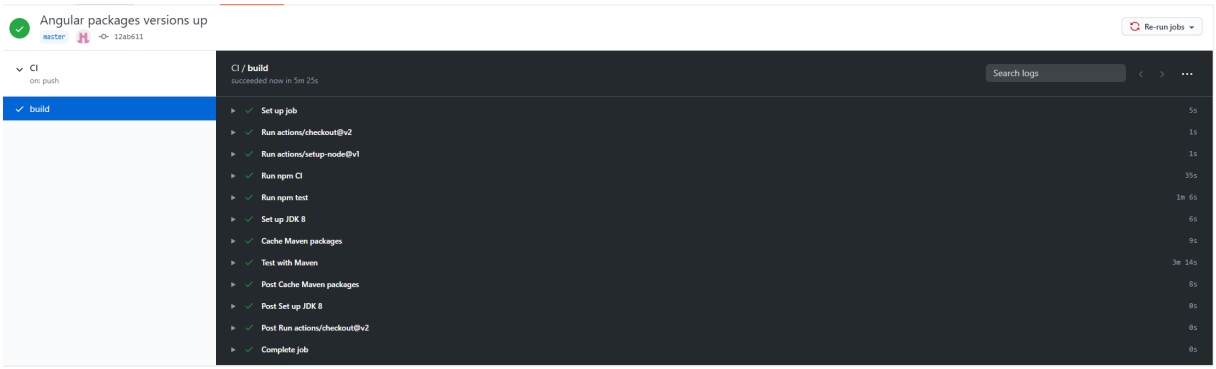
Jenkins je open-source, self-hosted produkt, který je zdarma. Upřednostnil jsem ho v brzké fázi vývoje. Jenkins bohužel vytěžoval výkon stroje, který byl zároveň používán pro vývoj.

Především proto jsem se rozhodl přejít na Github Actions. Poskytuje neplacenou verzi, která je svým rozsahem dostačující pro moje účely. Github Actions se spouští na serverech Githubu což značně uvolnilo lokální zdroje pro vývoj. Ukázka buildů aplikace spuštěných automaticky viz obrázek 4.2. Ukázka jednoho konkrétního buildu včetně jeho součásti viz obrázek 4.3.

[1] https://github.com/landama2/Skill-Matrix-App



Obrázek 4.2: Buildy spuštěné automaticky pomocí Github actions



Obrázek 4.3: Buildu a jeho kroky provedené automaticky pomocí Github actions

Kapitola 5

Implementace

Časově nejnáročnější část této práce spočívala v samotné implementaci. Vybrané technologie byly uvedeny v sekci 4. Nyní se zaměříme na detailnější popis jednotlivých technologií a detaily samotné implementace.

5.1 Generátor JHipster

Pro základ aplikace jsem použil nástroj JHipster, který umožňuje využití mnou vybraných technologií. JHipster je nástroj pro vygenerování minimální funkční aplikace obsahující klientskou i serverovou část. Hlavní výhodou použití generátoru je vygenerování nutného boilerplatu a následující rychlé rozběhnutí aplikace. Jakékoli další úpravy je ale již potřeba provádět ručně. Boilerplate je nutný kus kódu, který se často opakuje s minimálními změnami.

5.2 Angular

Angular je open-source framework pro tvorbu webových aplikací s využitím HTML a Typescriptu. Typescript je nadstavbou JavaScriptu, rozšiřuje jej o statické typování a umožňuje vytváření tříd podle OOP. Typescript se dále kompiluje do čistého JavaScriptu. V produkčním prostředí kompilaci provádí AOT compiler (Ahead Of Time compiler), který provádí kompilaci již při sestavování aplikace. Pro vývoj je použitý JIT (Just in time) kompilátor, to nám umožňuje snazší debugování původního kódu výměnou za snížený výkon[4].

Architektura aplikace, která využívá Angular stojí na několika základních konceptech. Stavební kameny takovéto aplikace jsou NgModules, které poskytují kompilaci kontextu

pro komponenty. Aplikace se pak skládá z několika setů, ve kterých je vždy zkompilován související kód. Každá aplikace musí obsahovat alespoň základní *root* modul, který dále načítá ostatní moduly.

Další výhodou použití Angularu je možnost plné internacionalizace aplikace. Což je použito i v naší aplikaci, podporovaná je v základu česká a anglická mutace.

5.3 Java

Jako hlavní programovací jazyk pro serverovou část byla použita Java. Jedná se o hojně používaný objektově orientovaný jazyk. Hlavní výhodou je množství již existujících frameworků, které mohou být použity.

5.4 Spring

Spring [20] je Javový aplikační framework využívající IoC (Inversion of Control) a DI (Dependency Injection) principů. Umožňuje mimo jiné komunikaci s databází pomocí JPA (Java Persistance API) a autentifikaci pomocí Spring Security.

Springboot [19] je rozšíření Springu. Nabízí jednodušší konfiguraci a spuštění.

5.4.1 Dependency Injection

Dependency Injection nám dovoluje, aby třída nemusela sama vytvářet objekty na kterých závisí. Ty mohou být poskytnuty z venku implementací požadovaného rozhraní. Díky tomuto nemusí být třídy silně provázané a jsou jednodušší na údržbu. Naše třída se tedy poté nemusí měnit pokaždé, když se změní způsob vytvoření požadovavné služby. Což nám následně i zjednodušuje testování, injectované objekty lze nahradit pomocí takzvaných mocků.

Mock object je objekt který se navenek tváří jako požadovaný objekt a komunikuje s okolím pomocí stejných metod jako původní objekt. Na rozdíl od něj, ale postrádá vnitřní logiku a má předdefinované výstupy metod.

5.4.2 Logování

Pro získání informací o akcích prováděných programem a jeho vnitřním stavu je vhodným prostředkem logování.

Spring kromě klasických loggerů nabízí i možnost použití principu Aspektově orientovaného programování (AOP) pro logování při použití každé určené metody bez nutnosti do ní ručně vkládat logovací část. Ukázka takového loggeru viz obrázek 5.1.

```
@Around("applicationPackagePointcut() && springBeanPointcut()")
public Object logAround(ProceedingJoinPoint joinPoint) throws Throwable {
    Logger log = logger(joinPoint);
    if (log.isDebugEnabled()) {
        log.debug("Enter: {}() with argument[s] = {}", joinPoint.getSignature().getName(), Arrays.toString(joinPoint.getArgs()));
    }
    try {
        Object result = joinPoint.proceed();
        if (log.isDebugEnabled()) {
            log.debug("Exit: {}() with result = {}", joinPoint.getSignature().getName(), result);
        }
        return result;
    } catch (IllegalArgumentException e) {
        log.error("Illegal argument: {} in {}()", Arrays.toString(joinPoint.getArgs()), joinPoint.getSignature().getName());
        throw e;
    }
}
```

Obrázek 5.1: AOP logger

5.5 Persistentní vrstva

Persistentni vrstva slouží k oddělení datové a aplikační vrstvy. V tomto případě je datovou vrstvou SQL databáze.

5.5.1 Java Persistance API

Java Persitance API zajišťuje komunikaci s databází pomocí SQL Queries a mapování získaných výsledků na Java objekty.

JPQL

Pro použití Javových tříd přímo v query lze použít JPQL (Java Persistance Query Language), který má podobnou syntaxi jako samotné SQL. Hodí se například na jednoduché získání entit, které obsahují další entity. Pro ukázku použití JPQL viz 5.2.

```
@Query(value = "select distinct skill from Skill skill " +
    "left join fetch skill.category " +
    "left join fetch skill.subCategory",
    countQuery = "select count(distinct skill) from Skill skill")
Page<Skill> findAllWithEagerRelationships(Pageable pageable);
```

Obrázek 5.2: JPQL query

Nicméně stále lze v JPA použít i čisté SQL pomocí tzv. *Native Query* viz. 5.3

```
@Query(value = "SELECT * FROM SKILL " +
    "LEFT JOIN CATEGORY ON CATEGORY.ID = CATEGORY_ID " +
    "LEFT JOIN USER_SKILL ON USER_SKILL.SKILL_ID = SKILL.ID",
    nativeQuery = true)
Page<SkillFullDTO> findAllWithUserSkill(Pageable pageable);
```

Obrázek 5.3: Nativní SQL query

5.5.2 Java Beans, DTO a POJO

Jednotlivé entity v databázi jsou reprezentovány pomocí Java Beans s anotací *@Entity*.

Java Bean je jakákoliv Java třída splňující tyto vlastnosti:

- všechny vlastnosti třídy jsou *private*
- třída obsahuje public konstruktor bez parametrů
- třída implementuje interface *Serializable*

Pro mapování spojení více tabulek byl použit Data Transfer Object (DTO). Data Transfer Object nemusí reprezentovat žádnou konkrétní entitu a může obsahovat všechna potřebná data složená z více databázových tabulek bez nutnosti definovat které části dat je možno vynechat a které ne.

POJO

POJO je čistá datová struktura obsahující pouze instanční proměnné a jejich gettery a settery. Neobsahuje žádnou složitější logiku. V našem případě také implementuje *Serializable*.

5.6 Databáze

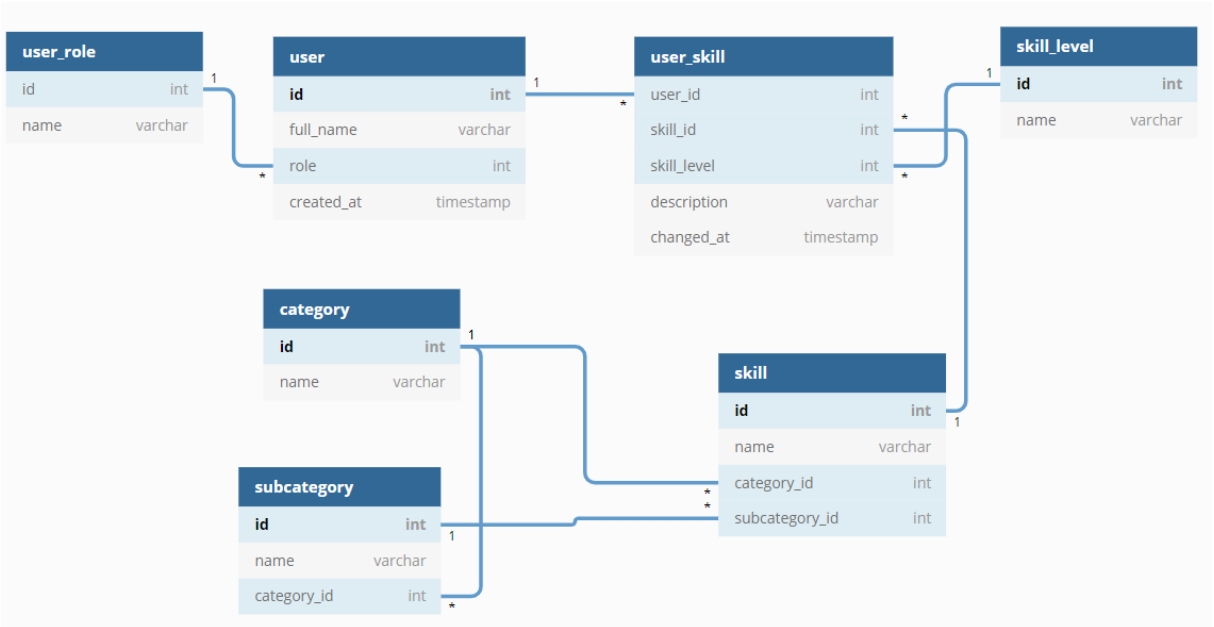
Kromě volby technologií, které jsou použity pro implementaci, je také důležité navrhnout strukturu databáze. Zde představím databázi, která byla navrhnuta pro potřeby aplikace. Ukážu její model a také popis jednotlivých tabulek a vztahy mezi nimi, jelikož se jedná o relační databázi jak jsem uvedl v sekci 4.3.

5.6.1 Model databáze

Vzhledem k tomu, že hlavním cílem aplikace je práce s informacemi o znalostech jednotlivých uživatelů, ústřední entitou databáze je tabulka *skill*.

- *skill* představuje jednotlivou znalost. Příklad jednotlivé znalosti může poté být Bash, VIM, makefile, atd. *skill* patří do *kategorie* nebo *podkategorie*.
- *user_skill* váže tabulky *user* a *skill* a obsahuje textový popis pro případnou doplňující informaci. Tabulka uchovává informaci o času změny, jednotlivé záznamy jsou neměnné pro zachování historie (to nám dále pomůže při vytváření historie znalostí). Dále se váže na *skill_level*.
- *skill_level* neboli úroveň znalosti je číselník obsahující hodnoty (N/A, None, Junior, Middle, Senior). Uchovává informaci o výši odbornosti v dané znalosti.
- *category* představuje kategorii znalosti (například operační systémy).
- *subcategory* představuje podkategorii kategorie znalosti (například Linux, Windows, iOS).
- *user* odpovídá uživateli z hierarchie.

Vizualizace modelu databáze je zobrazena na diagramu 5.4.



Obrázek 5.4: Návrh databáze pro aplikaci Skill Matrix

5.6.2 Role uživatelů

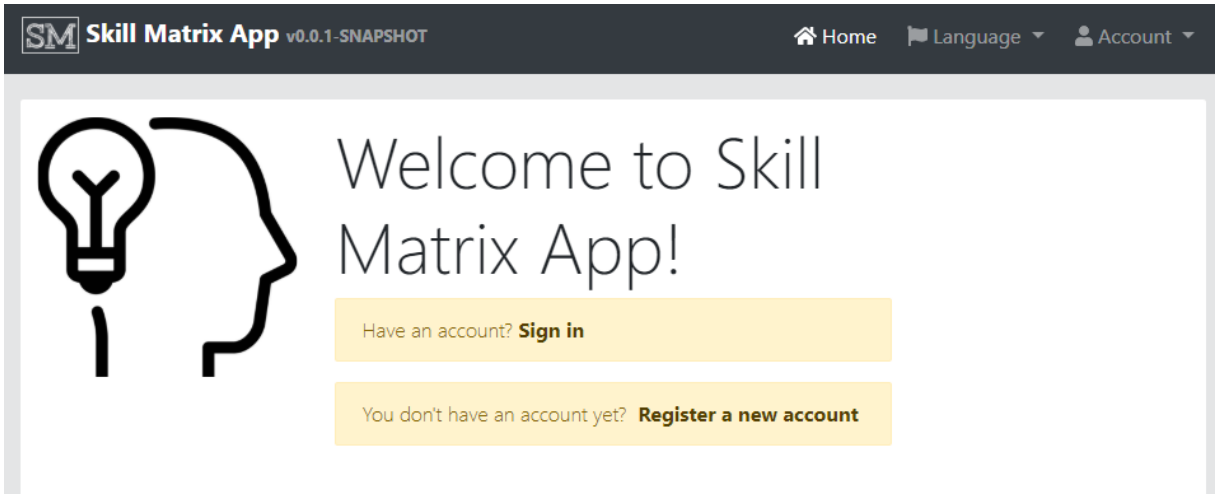
Jedním z důležitých aspektů aplikace je rozdělení jednotlivých uživatelů do různých rolí. Toto rozdělení koresponduje s pracovními pozicemi ve firmě a umožňuje různým uživatelům provádět různé akce. Uživatelé mohou být zařazeni do následujících rolí:

- Administrátor – má přístup k údajům o všech pracovnících, může je číst i měnit, může spravovat práva ostatních uživatelů. Má přístup ke všem entitám v databázi.
- Vedoucí pracovník - má přístup k datům ostatních pracovníků a může spravovat kategorie, podkategorie a znalosti. Může vyhledávat podle požadovaných znalostí.
- Standardní uživatel – může číst a měnit pouze údaje o sobě samém. Role je přiřazena automaticky.

Kapitola 6

Ukázka aplikace

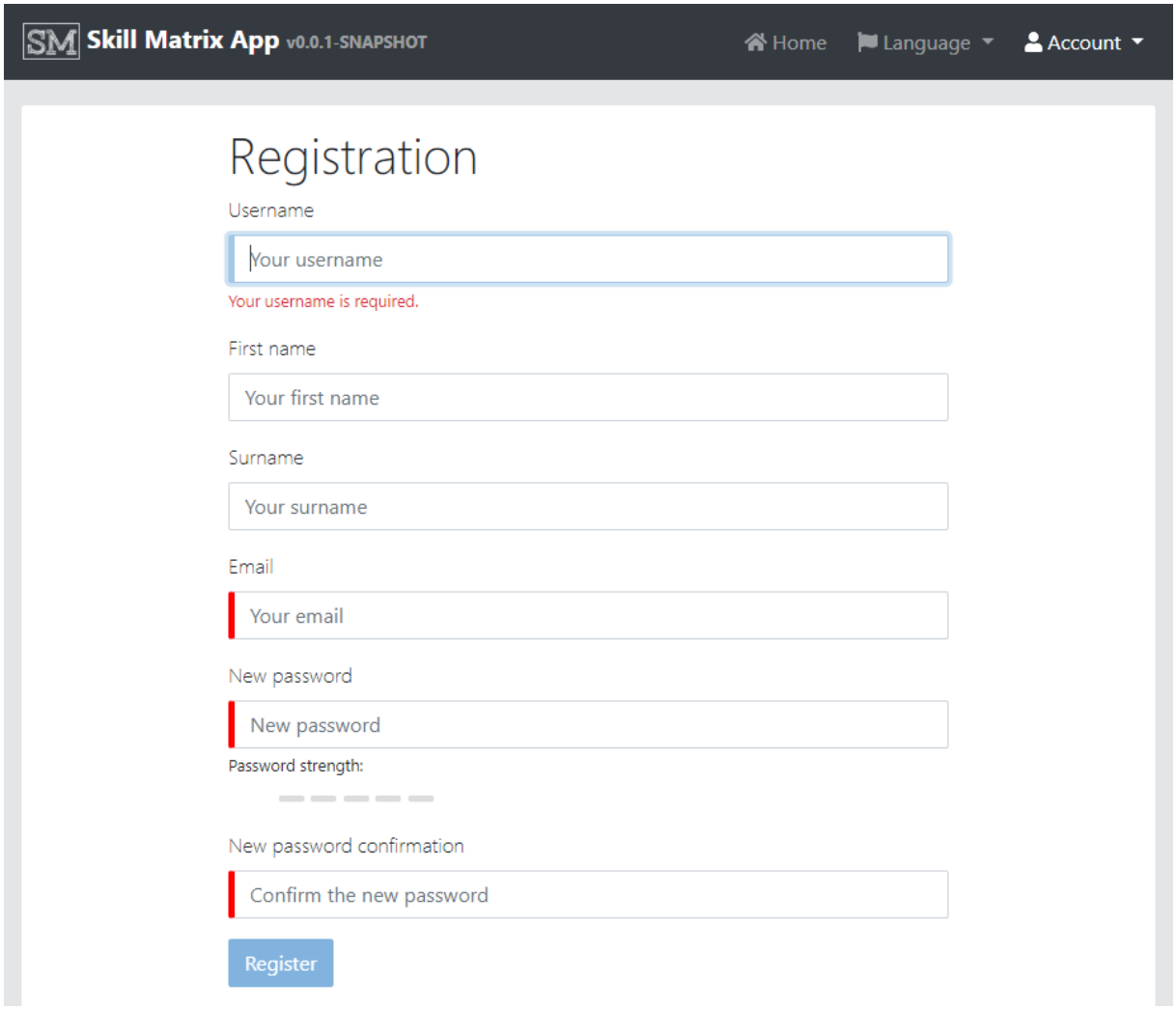
V této sekci bych rád představil finální formu implementované aplikace. Úvodní stranka zobrazena na obrázku 6.1 jednak vítá uživatele v aplikaci a jednak obsahuje užitečné odkazy na registraci a přihlášení. Každý uživatel, který aplikaci chce používat musí být zaregistrován a přihlášen, aby měl přístup do dalších částí aplikace. Ve vrchním menu je odkaz "Home"vedoucí na domovskou stránku. Dalším položkou v menu je "Language", kde je možné přepínat mezi českou a anglickou mutací grafického prostředí. Poslední položkou v menu je "Account", kde jsou odkazy na přihlášení a registraci uživatele.



Obrázek 6.1: Homepage

Při zvolení registrace se uživateli zobrazí registrační formulář, který vidíme na obrázku 6.2. Formulář vyžaduje vyplnení uživatelského jména, křestního jména a příjmení a emailu. Dále si uživatel zvolí a potvrdí heslo. Při zadávání hesla se mění ukazatel pod políčkem, který zobrazuje jak moc je heslo silné. Dole na stránce se nachází tlačítko "Register", kterým se registrace potvrdí.

Při výběru přihlášení na úvodní stránce se otevře přihlašovací dialog, který vidíme na

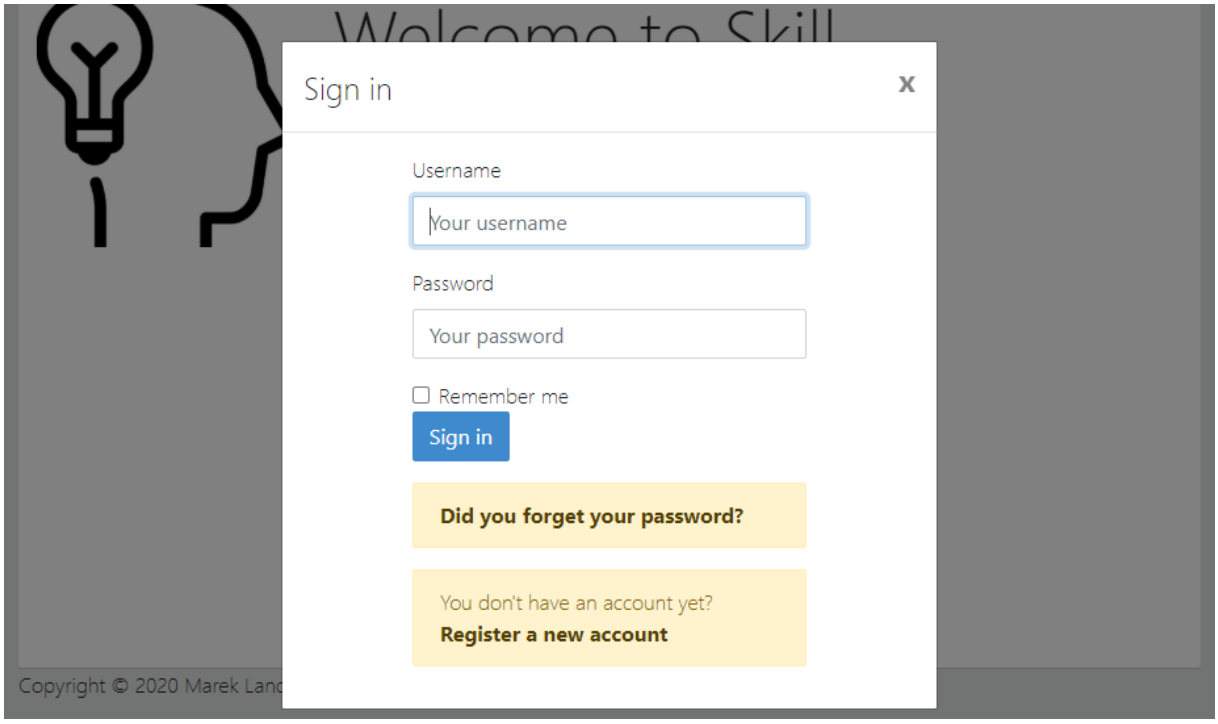


Obrázek 6.2: Registrační formulář

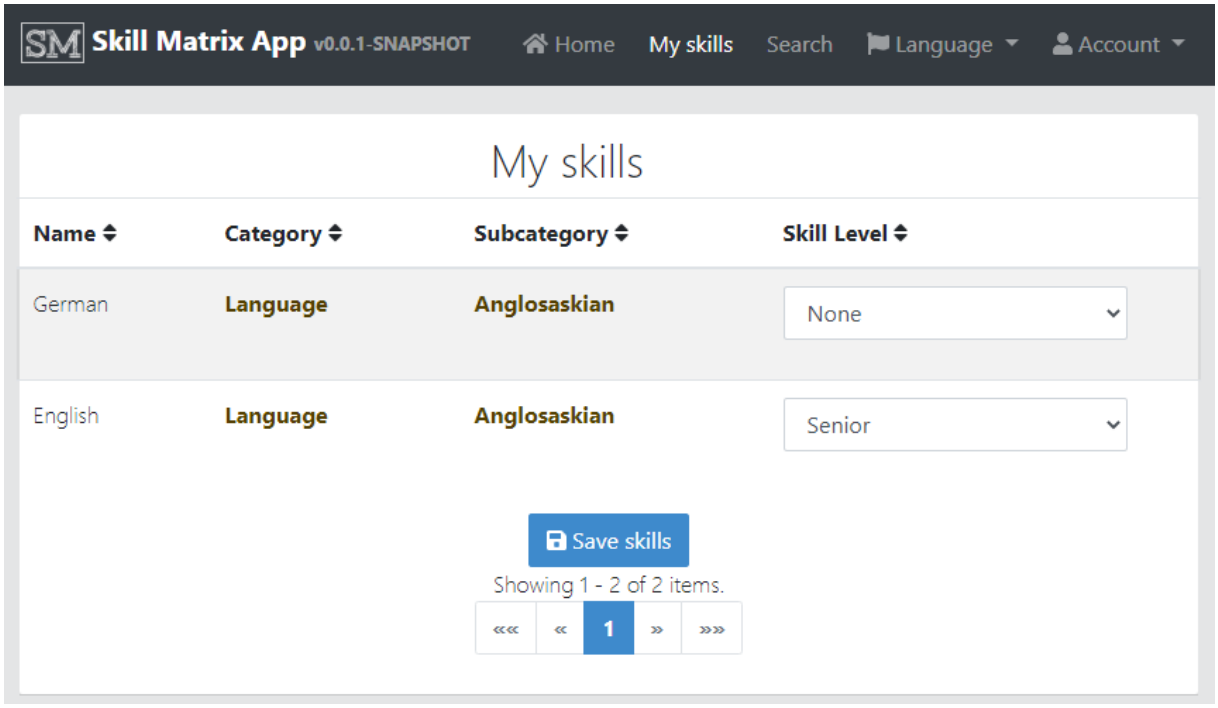
obrázku 6.3. Uživatelé se přihlašují pomocí uživatelského jména a hesla. Také je možné zaškrtnout políčko, které umožní prohlížeči zapamatovat si přihlašovací údaje. V případě, že uživatel zapomene heslo je pod přihlašovacím formulářem odkaz "Did you forget your password?", kde je možné tento problém řešit. Dále je v dialogu odkaz pro registraci v případě, že uživatel ještě nemá vytvořený účet.

V nabídce hlavního menu se nachází položka "My Skills", která uživatele dovede na stránku, kde jsou zobrazeny všechny jeho dovednosti. Tato stránka je vidět na obrázku 6.4. V jednotlivých sloupcích se zobrazuje název znalosti, kategorie, podkategorie a úroveň znalosti. Úroveň jednotlivých znalostí je možné přenastavit a poté uložit tlačítkem "Save skills", které se nachází pod seznamem znalostí.

Pro vedoucí pracovníky je také z hlavní nabídky dostupná stránka na přidávání a úpravu kategorií, která je vidět na obrázku 6.5. Obdobná stránka existuje i pro editaci podkategorií. Na stránce se zobrazí seznam kategorií včetně jejich ID a jména. Každou z



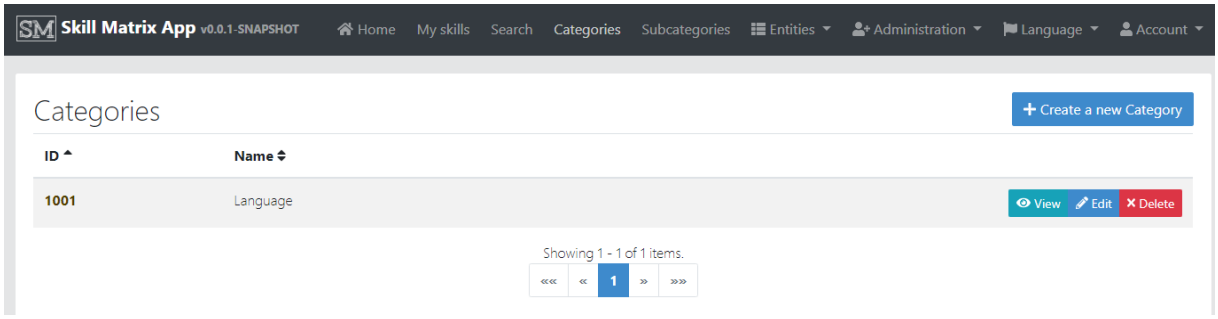
Obrázek 6.3: Přihlašovací formulář



Obrázek 6.4: Moje znalosti

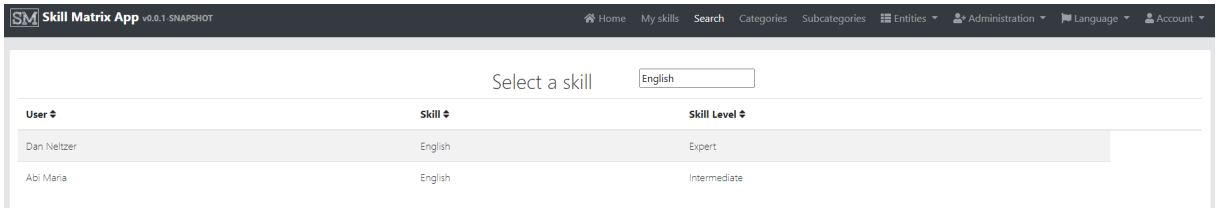
nich je pak možné prohlížet, editovat nebo smazat.

Další důležitou součástí je stránka pro vyhledávání pracovníků podle požadované znalosti, kterou můžeme vidět na obrázku 6.6. Na stránce je pole, kam se vepíše znalost, podle které chce uživatel vyhledávat. Vyhledávací lišta má také funkční autocomplete.



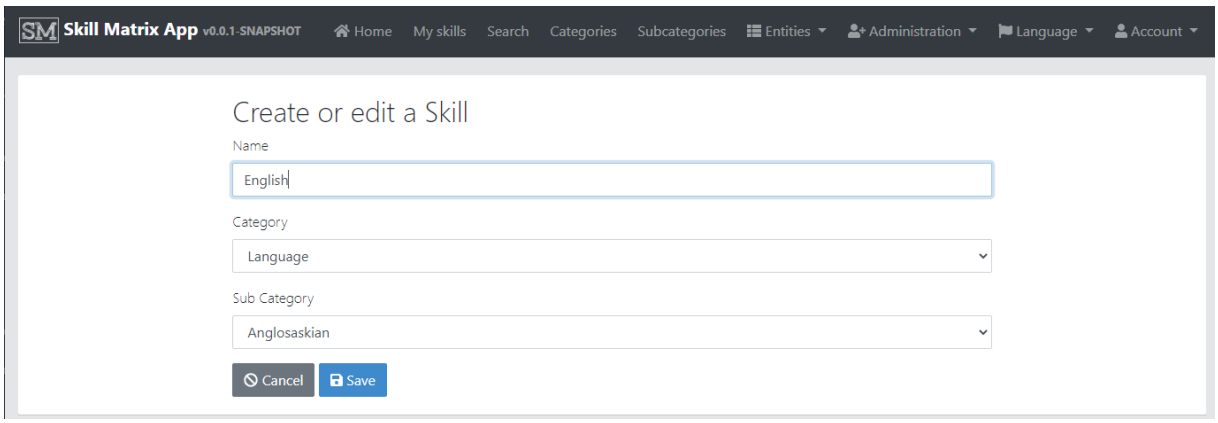
Obrázek 6.5: Kategorie

Vyhledané záznamy je pak možné řadit podle uživatele, názvu znalosti nebo úrovně znalosti.



Obrázek 6.6: Vyhledávání podle znalosti

Na obrázku 6.7 vidíme jak vypadá formulář pro vytvoření nové znalosti.



Obrázek 6.7: Vytvoření nové znalosti

Kapitola 7

Testování

Testování každé aplikace by mělo probíhat ve vícero vrstvách a několikrát za dobu jejího vývoje. V mém případě byly používány během celého vývoje hlavně automatické testy, které slouží především k ověření správné funkčnosti jednotlivých částí aplikace. V pozdní fázi vývoje však proběhly i uživatelské testy. Testy s budoucími uživateli jsou užitečným nástrojem jelikož umožňují odhalit nedostatky opomenuté ať už při návrhu nebo při vývoji.

7.1 Automatické testy

Při automatickém testování je důležité testovat všechny vrstvy aplikace. Je důležité testovat klientskou i serverovou část aplikace.

7.1.1 Jednotkové testy

Jednotkové testy jsou testy na úrovni jednotlivých jednotek (v našem případě tříd a jejich metod). Jednotka je nejmenší testovatelná část. Jejich účelem je ověřit správnou funkci jednotlivých složek softwaru podle očekávaného chování.

Jednotkové testy ověřují zachování základních funkcionalit jednotek při změně kódu. Jsou také jednodušší na výrobu a údržbu než ostatní typy testování.

V mém případě je v serverové vrstvě použit framework *JUnit* [10]. Ukázka viz obrázek 7.1.

Pro test klientské vrstvy byl použit framework *Jest* [9] [22]. Jedná se o standardně používaný testovací framework pro JavaScript vyvinutý firmou Facebook.

```java
@BeforeEach
public void setUp() {
    skillMapper = new SkillMapperImpl();
}

@Test
public void testEntityFromId() {
    Long id = 1L;
    assertThat(skillMapper.fromId(id).getId()).isEqualTo(id);
    assertThat(skillMapper.fromId(null)).isNull();
}
```

Obrázek 7.1: Jednotkový test ve frameworku JUnit

7.1.2 Integrační testy

Integrační testy testují spojení různých částí aplikace, které jsou dohromady testovány jako jeden celek. V našem případě se jedná mimo jiné o testy ukládání a čtení entit z databáze. Ukázka integračního testu komunikace s databází viz obrázek 7.3. Ukázka na klientské části viz obrázek 7.2.

```javascript
describe( name: 'save', fn: () => {
  it( name: 'Should call update service on save for existing entity', fakeAsync( fn: () => {
    // GIVEN
    const entity = new SkillMySuffix( id: 123);
    spyOn(service, method: 'update').and.returnValue(of(new HttpResponse( init: { body: entity })));
    comp.updateForm(entity);
    // WHEN
    comp.save();
    tick(); // simulate async

    // THEN
    expect(service.update).toHaveBeenCalledWith(entity);
    expect(comp.isSaving).toEqual( expected: false);
  }));
});
```

Obrázek 7.2: Jednotkový test ve frameworku Jest

7.2 Uživatelské testy

Uživatelské testování použitelnosti proběhlo v pozdní fázi vývoje aplikace se dvěma testery. Díky načasování bylo možné na základě zpětné vazby aplikaci upravit. Principy

```java
@BeforeEach
public void initTest() { skill = createEntity(em); }

@Test
@Transactional
public void createSkill() throws Exception {
    int databaseSizeBeforeCreate = skillRepository.findAll().size();

    // Create the Skill
    SkillDTO skillDTO = skillMapper.toDto(skill);
    restSkillMockMvc.perform(post( urlTemplate: "/api/skills")
        .contentType(MediaType.APPLICATION_JSON)
        .content(TestUtil.convertObjectToJsonBytes(skillDTO)))
        .andExpect(status().isCreated());

    // Validate the Skill in the database
    List<Skill> skillList = skillRepository.findAll();
    assertThat(skillList).hasSize(databaseSizeBeforeCreate + 1);
    Skill testSkill = skillList.get(skillList.size() - 1);
    assertThat(testSkill.getName()).isEqualTo(DEFAULT_NAME);
}
```

Obrázek 7.3: Integrační test komunikace s databází

uživatelského testování jsem čerpal z *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems* [11]

Testy probíhaly podle dvou krátkých scénářů, které byly navrženy tak, aby pokryly základní funkcionality aplikace. Jedná se o kvalitativní testování, takže není cílem otestovat aplikaci s co nejvíce uživateli, ale získat zpětnou vazbu uživatelů včetně jejich dojmů.

Cílem je získat zpětnou vazbu především na uživatelské rozhraní, ve kterém je potřeba najít určité prvky, ale také na logiku aplikace a její celkovou přehlednost.

7.2.1 Scénaře testování

Vytvořil jsem dva krátké scénáře testování, kde každý z nich pokrývá funkce typické pro danou roli uživatele.

Základní uživatelský scénář bežného zaměstnance

Scénář pro základního uživatele se zaměřuje na registraci nového uživatele a následné přihlášení. Dalším krokem je ověření správnosti zadaných údajů, takže je nutné zobrazit uživatelský profil. Dále uživatel musí zobrazit svoje znalosti a uložit novou úroveň některé ze znalostí.

Kroky scénáře:

- Vytvoření účtu
- Přihlášení
- Ověření správného uložení osobních údajů při registraci
- Zobrazení svých znalostí
- Uložení nové úrovně svých znalostí

Během testování podle prvního scénáře se ukázala především pozitiva uživatelského rozhraní. Registrace i přihlašování do aplikace se zdály být jasné a dobře označené. Vyskytly se však i nedostatky a to ve formě chybějících polí v registračním formuláři. Při registraci je potřeba zadat pouze uživatelské jméno. Jméno a příjmení se musí zadat separátně po přihlášení do aplikace v nastavení uživatelského účtu. Další problém se vyskytnul při zobrazování znalostí uživatele, kdy nebylo plně jasné, že odkaz v menu vede na správnou stránku.

Odhalené chyby:

- Při vytvoření účtu chybí pole na jméno a příjmení, tyto údaje je následně nutno doplnit v nastavení účtu
- Pojmenování sekce znalostí není jasné, uživatel si nebyl jistý jestli se jedná o seznam jeho znalostí

Základní uživatelský scénář vedoucího pracovníka

Scénář pro vedoucího pracovníka se zaměřuje spíše na správu znalostí. Úkoly zahrnují tvorbu nové kategorie a podkategorie znalostí. Dalším úkolem je vyhledávání uživatelů podle znalostí.

Kroky scénáře:

- Vytvoření nové kategorie

- Vytvoření nové podkategorie
- Vytvoření nové znalosti
- Vyhledání zaměstnanců s konkrétní požadovanou znalostí

Jeden objevený nedostatek spočíval v nepřehlednosti menu, kde je potřeba rozkliknout položku "Entities", aby bylo možné dostat se k existujícím kategoriím, podkategoriím a znalostem. S vyhledáváním uživatelů podle znalostí ani s tvorbou nové znalosti nebyl problém.

Odhalené chyby:

- Nepřehlednost menu - umístění položek "Categories", "Subcategories"a "Skills"

7.2.2 Vyhodnocení testování

Na základě proběhlých uživatelských testů jsem vyhodnotil přípomínky a provedl vhodné změny v aplikaci. Především se jednalo o chybějící prvky ve formuláři nebo o špatnou přehlednost hlavní nabídky.

Odhalené chyby:

- Chybějící pole "jméno"a "příjmení"v registračním formuláři
- Struktura hlavní nabídky je nepřehledná a vyžaduje velké množství prokliků
- Sekce znalostí v hlavní nabidce není jasně pojmenovaná

Na základě odhalených chyb byly provedeny následující úpravy:

- Do registračního formuláře jsem přidal pole "Jméno"a pole "Příjmení"
- Pro vedoucí pracovníky a administrátora byly do horní navigační lišty přidány "Kategorie"a "Podkategorie"(oproti původní lokaci v rozklikávacím menu)
- Položka v nabídce "Skills"("Znalosti") byla přejmenována na "My Skills"("Moje znalosti")

Kapitola 8

Budoucí práce

Výsledná aplikace má stále prostor pro zlepšení a pro přidání nových funkcionalit. Jednou z nich je načítání defaultních rolí z Active Directory, což je technologie, která se ve firmě používá a její integrace by tak byla velkým usnadněním.

Dalším užitečným vylepšením by bylo zobrazení historie vývoje znalostí. Databáze je na tohle vylepšení připravená, jelikož automaticky ukládá všechny aktualizace znalostí a je tedy možné vygenerovat posloupnost změn v úrovních znalosti, ale není dokončena integrace do výsledné aplikace.

Dalším krokem je import dat. Import by byl užitečným nástrojem hlavně v případě, kdy by firma měla data ve formě excelové tabulky jako v tomto případě a bylo by tak možné současný stav skill matrix do aplikace rovnou nahrát a dále upravovat v aplikaci. Současně s tím by mohl být implementován i export dat.

Finálním krokem by bylo nasazení aplikace do produkčního prostředí.

Kapitola 9

Závěr

Cílem práce bylo vytvořit aplikaci Skill Matrix podle požadavků zadavatele. Seznámil jsem se s problematikou a požadavky a prozkoumal možnosti použití již existujících řešení.

V rámci návrhu jsem vybral vhodné technologie a nástroje pro serverovou, klientskou i datovou vrstvu aplikace. Dále následovalo seznámení s vybranými technologiemi a samotná implementace. Během vývoje probíhaly automatické testy a v pozdní fázi také kvalitativní uživatelské testy.

Výsledkem práce je webová aplikace Skill Matrix, která byla zpracována podle požadavků zadavatele.

Doufám, že v budoucnu budu mít možnost aplikaci dále doplnit o další funkcionality popsané v kapitole 8. Při práci na tomto projektu jsem získal cenné zkušenosti a rozšířil své znalosti o práci s novými technologiemi se kterými jsem se během studia nesetkal.

Bibliografie

[1] *Angular*. URL: https://angular.io/.

[2] *Apache Maven*. URL: https://maven.apache.org/.

[3] *API Development for Everyone*. URL: https://swagger.io/.

[4] Muhammad Bilal. *What are AOT JIT Compiler in Angular ?* Ún. 2020. URL: https://dev.to/imbilal1/what-are-aot-jit-compiler-in-angular-2-488h.

[5] Shaumik Daityari. *Angular vs React vs Vue: Which Framework to Choose in 2020*. Srp. 2020. URL: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/.

[6] *Gradle Build Tool*. URL: https://gradle.org/.

[7] *H2 Database Engine*. URL: https://www.h2database.com/html/main.html.

[8] Theo Haerder a Andreas Reuter. "Principles of transaction-oriented database recovery". In: *ACM computing surveys (CSUR)* 15.4 (1983), s. 287–317.

[9] *Jest · Delightful JavaScript Testing*. URL: https://jestjs.io/.

[10] "JUnit Testing". In: *Pro NetBeans^{TM} IDE 6 Rich Client Platform Edition*. Berkeley, CA: Apress, 2008, s. 203–216. ISBN: 978-1-4302-0439-8. DOI: 10.1007/978-1-4302-0439-8_9. URL: https://doi.org/10.1007/978-1-4302-0439-8_9.

[11] Steve Krug. *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems*. 1st. USA: New Riders Publishing, 2009. ISBN: 0321657292.

[12] *MariaDB Foundation*. Lis. 2019. URL: https://mariadb.org/.

[13] *Microsoft Forms - easily create surveys, quizzes, and polls*. URL: https://forms.office.com/.

[14] *OpenJDK*. URL: https://openjdk.java.net/.

[15] *PostgreSQL - The World's Most Advanced Open Source Relational Database*. URL: https://www.postgresql.org/.

[16] Apache Tomcat Project. *Apache Tomcat*. URL: http://tomcat.apache.org/.

[17] *React – A JavaScript library for building user interfaces.* URL: https://reactjs.org/.

[18] *Skills Matrix Solution and Capability Framework by ability6.* Květ. 2020. URL: https://ability6.com/advanced-skills-matrix/.

[19] *Spring Boot.* URL: https://spring.io/projects/spring-boot.

[20] *Spring makes Java simple.* URL: https://spring.io/.

[21] Alen Šimec a Magličić. "Comparison of JSON and XML Data Formats". In: 2014.

[22] Mohit Thakkar. "Unit Testing Using Jest". In: *Building React Apps with Server-Side Rendering: Use React, Redux, and Next to Build Full Server-Side Rendering Applications.* Berkeley, CA: Apress, 2020, s. 153–174. ISBN: 978-1-4842-5869-9. DOI: 10.1007/978-1-4842-5869-9_5. URL: https://doi.org/10.1007/978-1-4842-5869-9_5.

[23] *Vue.js - The Progressive JavaScript Framework.* URL: https://vuejs.org/.

[24] Author: Erik van Vulpen. *How to Create a Skills Matrix for Success: Competency Matrix.* Zář. 2019. URL: https://www.analyticsinhr.com/blog/create-skills-matrix-competency-matrix/.