FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Vykazovací systém pro menší firmy v IT
Student:	Bc. Michal Kocánek
Vedoucí:	Ing. David Buchtela, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2020/21

Pokyny pro vypracování

Cílem práce je analýza a implementace systému pro vykazování práce. Aplikace bude řešit evidenci odpracovaných hodin na projektech a bude umožňovat základní podporu pro management, tj. zobrazení vytíženosti zaměstnanců a jednoduché plánování budoucího vytížení. Aplikace by měla zohledňovat různé dovednosti zaměstnanců, tedy měla by doporučit vhodné lidi na různé projekty.

1. Vypracujte analýzu uživatelských požadavků a existujících řešení vykazovacích systémů.
2. Proveďte návrh a implementaci backendové části v REST API.
3. Popište zefektivnění manažerských procesů, zpracujte studii ekonomické návratnosti vlastní implementace oproti placení již existující aplikace.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 28. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Report System for Small IT Companies

Bc. Michal Kocánek

Katedra softwarového inženýrství
Vedoucí práce: Ing. David Buchtela, Ph.D.

28. května 2020

Poděkování

Rád bych poděkoval své rodině, přítelkyni a přátelům za podporu při celém mém studiu. Poděkování rovněž patří vedoucímu práce Ing. David Buchtela, Ph.D., který se mé práce ujal.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 28. května 2020 .....

Odkaz na tuto práci

Kocánek, Michal. *Report System for Small IT Companies*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

V diplomové práci se zabývám problémem vykazování času v IT firmách. Nejprve vysvětlím proces vykazování a provedu průzkum existujících vykazovacích systémů. Dále se věnuji analýze, návrhu a implementaci Java Spring aplikace. Nakonec hodnotím návratnost investice do vývoje vlastní vykazovací aplikace.

Klíčová slova Java aplikace, REST API, vykazování času, pracovní výkaz, hodnocení návratnosti.

Abstract

Goal of this thesis is to develop an information system for time reporting. Firstly I evaluate the process and review existing applications. Then I analyze, design and implement Java Spring application. Lastly I calculate return on investment of developing custom application instead of purchasing existing software.

Keywords Java application, REST API, time reporting, timesheet, return on investment.

Obsah

Úvod **1**

1 Definice společnosti **3**
1.1 Definice malé IT společnosti 3
1.2 Project management 5

2 Popis a srovnání vykazovacích systému **7**
2.1 Požadované funkcionality 7

3 Analýza **13**
3.1 Procesy 13
3.2 Funkční požadavky 14
3.3 Nefunkční požadavky 16
3.4 Uživatelské role a oprávnění 16
3.5 Model případů užití 17

4 Návrh **23**
4.1 Architektura 23
4.2 Výběr programovacího jazyka 23
4.3 Výběr frameworku 24
4.4 Výběr databáze 24
4.5 Návrh API 25
4.6 Model 27

5 Implementace a technologie **29**
5.1 Použité technologie 29
5.2 Implementace REST API 33
5.3 Zabezpečení 34

6 Testování **37**

6.1 Jednotkové testování 38
6.2 Funkční testování 38

7 Nasazení **39**
7.1 Docker 40

8 Srovnání návratnosti, ekonomické zhodnocení **41**
8.1 Odhad ceny aplikace 41
8.2 Ekonomická návratnost investice 42

9 Zefektivnění manažerských procesů **47**
9.1 Doporučení zaměstnanců 47
9.2 Reportování práce 48

Závěr **49**

Literatura **51**

A Seznam použitých zkratek **57**

B Obsah přiloženého CD **59**

Seznam obrázků

2.1 Ukázka software Jira 8
2.2 Plugin Tempo 9
2.3 Ukázka software Redmine 10
2.4 Ukázka software Toggl 11

3.1 Use Cases 21
3.2 Use Cases 22

4.1 Architektura aplikace 24
4.2 Class Diagram 28

8.1 NPV 45
8.2 NPV 46

Seznam tabulek

4.1 URL zdroje 26

8.1 Citlivostní analýza nákladů 42
8.2 Návratnost investice 43
8.3 NPV 44

Úvod

Vykazování činnosti je nedílnou součástí běžného pracovního chodu firmy. Zatímco ve většině firem stačí pouhé vykázání pracovní doby jednotlivých zaměstnanců, existují odvětví, které potřebují více detailní systémy vykazování. Jedním z příkladem jsou firmy působící v IT odvětví. Tyto firmy, které se zabývají vývojem softwaru, potřebují ke svému fungování komplexní project management, který jim pomůže s úspěšným řízením jednotlivých projektů.

IT firmy, pokud nevyvíjejí svůj vlastní produkt, často vytvářejí software na zakázku. Je proto obvyklé, že takové firmy mají v jednu chvíli více zakázek pro různé zákazníky, což vytváří potřebu pro pečlivou evidenci odpracovaných hodin na jednotlivých zakázkách, které pak firma fakturuje zákazníkovi. K tomu právě může sloužit vhodně navržený vykazovací systém.

Další využití vykazovacího systému může být řízení outsourcingu (vyčlenění služeb, procesů nebo zdrojů a činností mimo organizaci formou dlouhodobého smluvního vztahu [1]), který je poměrně typický právě pro IT odvětví. IT firmy často uzavírají smlouvy s certifikovanými specialisty, které mohou najímat jak na pravidelnou spolupráci, tak třeba na konkrétní projekt. Vykazovací systém může usnadnit hlídání plnění smluv a také správnou alokaci lidských zdrojů.

V práci nejprve popíši, jak taková malá IT firma vypadá, jak vypadá modelová firma, pro kterou bude aplikace navržena. Druhá kapitola bude věnována popisu již existujících vykazovacích systémů, zhodnocení jejich výhod a nevýhod. Následuje analýza navrhované aplikace, ve které jsou popsány požadavky kladené na aplikaci a také je zde nastíněno její používání. Další kapitoly se zabývají samotným návrhem aplikace, její implementací, vybranými technologiemi, testováním a nasazením. V poslední části je uvedeno srovnání návratnosti, ekonomické zhodnocení a celkové zefektivnění manažerských procesů.

Kapitola 1

Definice společnosti

Před popisem a definicí vykazovacích systémů musím definovat malou IT firmu. V této kapitole nejdříve popíši definici společnosti a jaké existuje jejich dělení. Dále upřesním jakou modelovou společností se bude tato diplomová práce zabývat.

1.1 Definice malé IT společnosti

První věcí, kterou potřebuji stanovit, je, jak vypadá malá IT firma. Podniky lze dělit z mnoha hledisek. Zde je uvedeno několik nejpoužívanějších příkladů [2]:

- podle právní formy
  - podniky jednotlivce – tj. živnosti, kdy je podnikání upraveno živnostenským zákonem
  - obchodní společnosti – řídí se obchodním zákonem
    - osobní společnosti
    - kapitálové společnosti
  - družstva
  - státní podniky
- podle hlavního cíle
  - ziskové
  - neziskové
- podle působnosti
- podle sektorů [3]

1.
   - veřejný – poskytování veřejných služeb, financováno z veřejných rozpočtů [4]
   - soukromý
   - smíšený
2.
   - primární – tj. prvovýroba, zahrnuje odvětví, které přeměnují primární zdroje na suroviny (např. zemědělství, těžba)
   - sekundární – tj. průmysl a výroba, zahrnuje odvětví, které přeměňují suroviny na výrobky či zboží (např. stavebnictví, elektrotechnika)
   - terciální – tj. služby, zahrnuje veškerou lidskou činnost, jejíž podstatou je poskytování služeb (např. zdravotnictví, informační a komunikační služby)
   - kvaternární – tj. věda a výzkum

- podle velikosti
  - mikropodniky
  - malé podniky
  - střední podniky
  - velké podniky

Dle předchozích rozdělení lze uvažovanou malou IT společnost definovat jako firmu spadající do soukromého terciálního sektoru, která má mezi 10 – 50 zaměstnanci a její roční obrat nepřesáhne 10 mil. EUR či její bilanční suma nepřesáhne 10 mil. EUR. Modelová společnost, pro jejíž potřeby je program vyvíjen, je blíže popsána v následující podkapitole.

1.1.1 Modelové společnost

Modelová firma se zabývá IT službami, poskytováním podpory softwaru, údržbou a poradenstvím, ale většinou nevyvíjí vlastní software ve formě např. krabicového softwaru, webové aplikace nebo zakázkového vývoje.

První příklad, který zde uvedu, je společnost, která se specializuje na outsourcováni vlastních zaměstnanců k zákazníkům, kteří potřebují odbornou podporu nebo poradenství third-party softwaru nebo technologie, např. Oracle databáze [5] nebo ERP systému SAP [6]. Tento software sice vyvíjí a licencuje vydavatel, ale zavádění, podporu a customizaci přenechávají svým partnerům. Zákazníkům modelové firmy se většinou nevyplatí zaměstnávat odborníky přes tyto technologie, protože by je nedokázali plně využít, nemají žádné IT oddělení nebo by náklady na zaměstnání specialisty převýšili náklady, které zaplatí využitím služeb jiné firmě.

Jiný příklad je firma, která pomáhá živnostníkům s hledáním zakázek a projektů. Firma bude mít více různých zakázníků, kterým dodává vývojáře v různých programovacích jazycích na doplnění vlastní kapacity. Například může shánět projekty pro deset Java vývojářů a deset JavaScript programátorů. Většinou se podaří získat projekt pro jednoho či dva vývojáře, a tak musí rozdělit své vývojáře na více projektů k různým zákazníkům. Živnostníci fakturují svoji práci naší firmě a ta fakturuje zákazníkovi.

Obě výše popsané firmy mají stejný problém: potřebují vědět kolik hodin jejich zaměstnanci odpracovali hodin pro jakého zákazníka. Informaci o počtu odpracovaných hodin pak firma využívá pro tvorbu faktur.

Modelová společnost má reálný obraz ve společnosti, která je partnerem SAPu. Firma poskytuje podporu pro několik středních a velkých zákazníků, kteří potřebují doplnit své IT oddělení, případně úplně outsourcují helpdesk. Customizing je další běžná věc – doprogramování funkcí, které nejsou přítomny v SAPu. Svým zákazníků také pomáhájí s bezproblémovým přechodem na nové verze nebo nové technologie.

Ve firmě pracuje kolem 30 zaměstnanců, kteří jsou přidělovány na projekty pro zákazníky. Na jeden projekt je obvykle přiřazeno více zaměstnanců v různých kompetencích, ale zakázník také může chtít někoho na částečný úvazek. Zároveň se může stát, že je jeden zaměstnanec přidělen na více projektů. Zaměstnanci musejí vykazovat na správný projekt a výkaz pak slouží managementu pro fakturaci zákazníkovi a peněžitou odměnu pro zaměstnance.

V takto malé firmě je management pouze několik lidí, většinou majitel, obchodní zástupce a projekt manager, kteří mají na starost běh firmy, alokaci lidí na projekty, oslovování potencionálních zákazníků a získávání nových zakázek.

1.2 Project management

Projektový management (project management) je o tom jak správně určit cíle a jakým způsobem se dosáhnou. K úspěšnému cíli je také zapotřebí vědět o přidělených prostředcích a všichni, kteří se podílejí na projektu, by měli vědět o jejich cílech.

Projekty v IT jsou specifické ani ne tak svým rozsahem nebo pracností, ale potřebou evidovat čas. Většina lidí nebo firem nemá pouze jednu zakázku, ale má jich více.

Některé IT firmy dokonce nemají vlastní projekty ve smyslu, že by dodávali celý produkt nebo software na zakázku. Taková firmu zprostředkovává zakázky svým zaměstnancům a ve smlouvě se zaváže, že bude dodávat pracovní sílu určitém objemu po časovou dobu, například 2 dny v týdnu po dobu šesti měsíců nebo provozují zákaznickou podporu a zákazník jim platí podle počtu vyřízených dotazů. Firmy pak přidělují své zaměstnance k projektům podle nasmlouvaných hodin.

Jeden zaměstnanec pak pracuje dva dny na projektu pro jednoho zákazníka a tři dny se věnuje jinému projektu. Pro management je důležité vidět a pracovat s časovou možností svých zaměstnanců. Často tímto způsobem funguje údržba nebo customizing softwaru. Pro zákazníka je to výhodnější, protože nemusí platit zaměstnance, kterého nemůže plně vytížit.

K podpoře projektového managementu existuje velké množství různých nástrojů. V této práci bych se chtěl výhradně zaměřit na problém vykazování času.

Se stále navyšujícím se počtem technologií je obtížné, dokonce i pro velké mezinárodní firmy, mít zaměstnance, kteří ovládají všechny technologie. Do projektu často vstupuje více stran, tak aby zajistil pokrytí všech oblastí od analytiků přes UI/UX odborníků až po vývojáře a testery.

Firmy se můžou dokonce zaměřit na zajištění odborníků pro některé důležité softwarové systémy, např. na složité ERP nebo CRM systémy. V dnešní době je velmi obtížné sehnat specialistu na systém SAP, který využívá velké množství firem. Společnost si koupí licenci systému SAP, ale nasazení a nastavení softwaru se rozhodne přenechat externímu dodavateli. Případně stejná firma po nějaké době používání zjistí, že SAP nezvládá pokrýt její specifický proces a zaplatí si doprogramování takové funkcionality. Tyto úpravy se nedějí každý den a jsou spíše nárazové, proto firmy dávají přednost outsourcingu před zaměstnáváním vlastních specialistů.

Kapitola 2

Popis a srovnání vykazovacích systému

V této části textu se budu zabývat srovnáním existujících nástrojů, které podporují vykazování času. Některé z uvedených programů jsou přímo určeny pro vykazování, ale ostatní služby jsou spíše nástroje na projektový management nebo celofiremní informační systém.

Základem analýzy stávajících řešení jsou zkušnosti s jejich používáním. To se skládá z vytvoření projektu a vykazování na něho. Dalším zdrojem informací o produktech budou webové stránky firem zodpovědných za jejich vývoj.

Následující seznam není úplný, přesto obsahuje zajímavé nástroje, které mohou splňovat potřeby modelové firmy.

2.1 Požadované funkcionality

Pro účely modelové firmy by aplikace měla splňovat tyto funkce:

- Evidence zaměstnanců.
- Evidence různých projektů.
- Evidence lidí na projektu.
- Vykazování času na přiřazené projekty.
- Oddělení rolí.

Očekávám, že každý nástroj bude splňovat první požadavek. Informace o uživateli by měla obsahovat několik základních atributů, jako je např. jméno a příjmení, datum narození, fotografie nebo kontaktní informace. Ideálně bude aplikace schraňovat všechna data důležitá pro management. Taková evidence pak může sloužit nejen jako jednoduchý adresář zaměstnanců, ale i jako výchozí

aplikace pro základní správu mezd. Každý uživatel musí mít vlastní přihlašovací údaje, aby mohl systém využívat a byl identifikovatelný v systému.

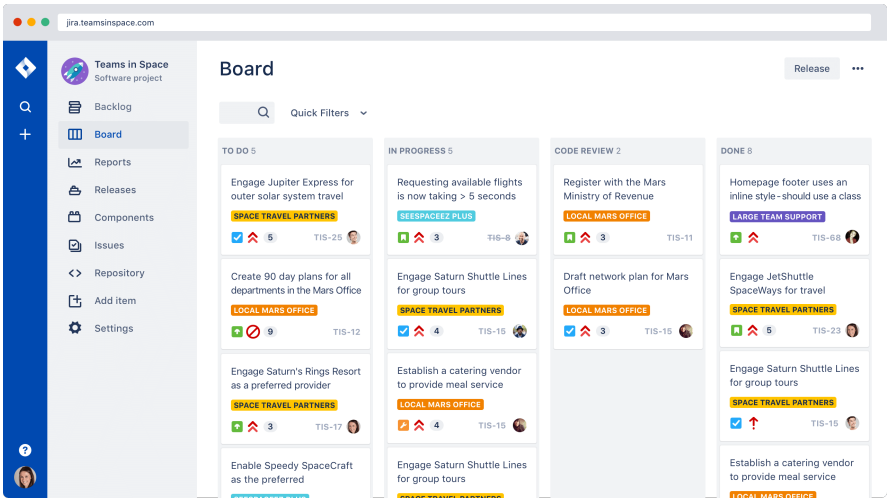
Evidence projektů je podobná evidenci zaměstnanců. Aplikace musí být schopna zobrazit všechny projekty, kterých se firma účastní nebo účastnila, spolu s klíčovými údaji o projektu.

Další požadavek říká, že je nutné pracovat s alokací zaměstnance a projektu. Na jednom projektu pracuje více zaměstnanců a naopak zaměstnanci mohou být přiřazeni na více různých projektů. Aplikace má být schopná zobrazit zaměstnanci všechny jeho aktivní projekty a managementu zobrazit projekt s přiřazenými zaměstnanci.

Předposlední požadavek je schopnost vykázat odpracovaný čas. Výkaz musí obsahovat informace o datu a počtu odpracovaných hodin. Aplikace by měla s těmito hodnotami pracovat a ukazovat počet vykázaných hodin zaměstnanců na jednotlivých projektech v shrnutí užitečném zaměstnanci nebo managementu. Takový report může mít např. formu týdenního nebo měsíčního výkazu.

Rozdělení rolí znamená, že aplikace pracuje s různými pozicemi ve firmě a management má více práv než zaměstnanci. Dále chci zabránit, aby uživatelé vykazovali na projekty, které jim nejsou přiřazené, nebo vykazovali za jiné uživatele. Samozřejmostí by mělo být soukromí dat, tedy aby jiný uživatel nemohl zobrazit data někoho jiného bez oprávnění.

2.1.1 Jira



Obrázek 2.1: Ukázka software Jira. Na obrázku je možné vidět příklad boardu a specifické úkoly rozdělené do několika sloupců.

Jira[7] je proprietární nástroj pro agilní softwarový vývoj. Aplikace je vy-

tvořená v jazyce Java a je dostupná přes webový prohlížeč nebo mobilní aplikaci pro iOS a Android.

Jira je především nástroj pro správu projektů. Základním kamenem projektu jsou úkoly (v systému naývané jako issues), které obsahují množství různých informací jako např. název úkolu, podrobný popis nebo štítky (tagy). Úkolům lze přiřadit osoby, které ho vykonávají. Volitelně se mohou k jednotlivým úkolům nastavit různé fáze, ve kterém se úkol aktuálně nachází. Konkrétní fáze úkolu v softwarové vývoji mohou například být: analýza, vývoj, testování, hotovo, atd. Uživatelé mohou také přidávat komentáře k úkolům s bližšími informacemi.

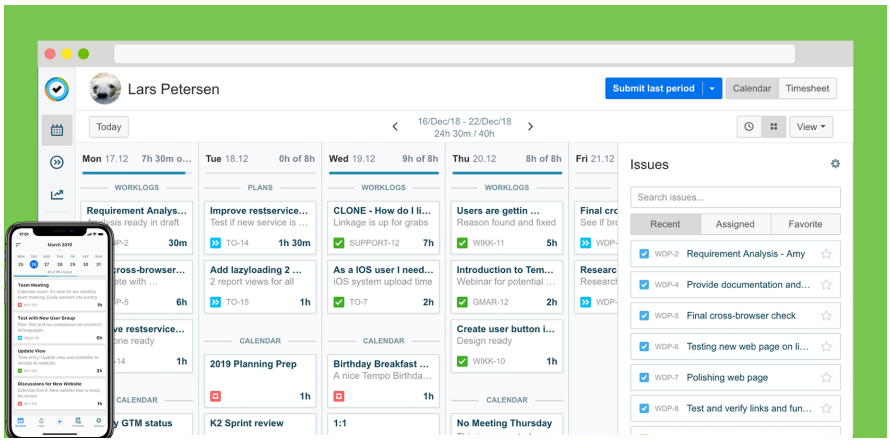
Zaměstnanci vykazují čas na obrazovce s detailem jednotlivých úkoly, výkazy se pak pojí s jedním konkrétním úkolem. Jira bohužel v základní verzi není nijak zvlášt uživatelsky přívětivá pro vykazování práce. Uživatel nemá možnost získat souhrný výkaz, a tak neví, kolik hodin již odpracoval. Odpracované hodiny musí počítat u jednotlivých úkolů. Tento problém řeší Tempo plugin 2.2 [8], který je ale placený.

Jira exceluje především v podpoře agilního vývoje softwaru. Jednotlivé úkoly se přiřazují do sprintů, týmy mají možnost vytvářet dashboardy s vlastním workflow. Na druhou stranu je Jira velice komplexní nástroj, jehož osvojení trvá nějakou dobu. Jira je také poměrně výpočetně náročný program a jeho provozování stojí nemalé prostředky z hlediska lidských zdrojů, které se starají o její bezproblémový chod.

Cena se odvíjí od počtu aktivních uživatelů. Čím více lidí používá jiru, tím je systém nákladnější, ale také je nižší cena pro jednoho uživatele. Při nákupu 50 licencí je cena 350 dolarů za měsíc.

Výhody:

- Uživatelsky přívětivé.



Obrázek 2.2: Tempo plugin, který zjednodušuje vykazování.

- Webová aplikace.
- Podpora agilní metodiky.

Nevýhody:

- Náročná na hosting.
- V základní podobě je vykazování nepraktické.

2.1.2 Redmine

Obrázek 2.3: Ukázka domovské stránky software Redmine.

Redmine [9] je webový nástroj pro projektový management a issue tracking systém. Součástí Redmine je také vlastní wiki stránka, která může být využitelná pro sdílení důležitých informací a novinek ve frimě. Webová aplikace je napsána v Ruby on Rails a licencována v GNU. Základní verze je tedy zdarma, ale lze nakoupit některé pokročilé rozšíření.

V Redmine mohu snadno evidovat zaměstnance a projekty. K projektu lze rovněž přidat zaměstnance a konkrétní úkoly, které mohou představovat kalendářní měsice. Zaměstanci pak vykazují na úkoly a management tím získá lepší přehled o práci na projektu.

Webová aplikace je vzhledově mírně zastaralá, ale nechybí zde žádné potřebné funkce. Nevýhodou pro firmu může být náročnější hosting aplikace.

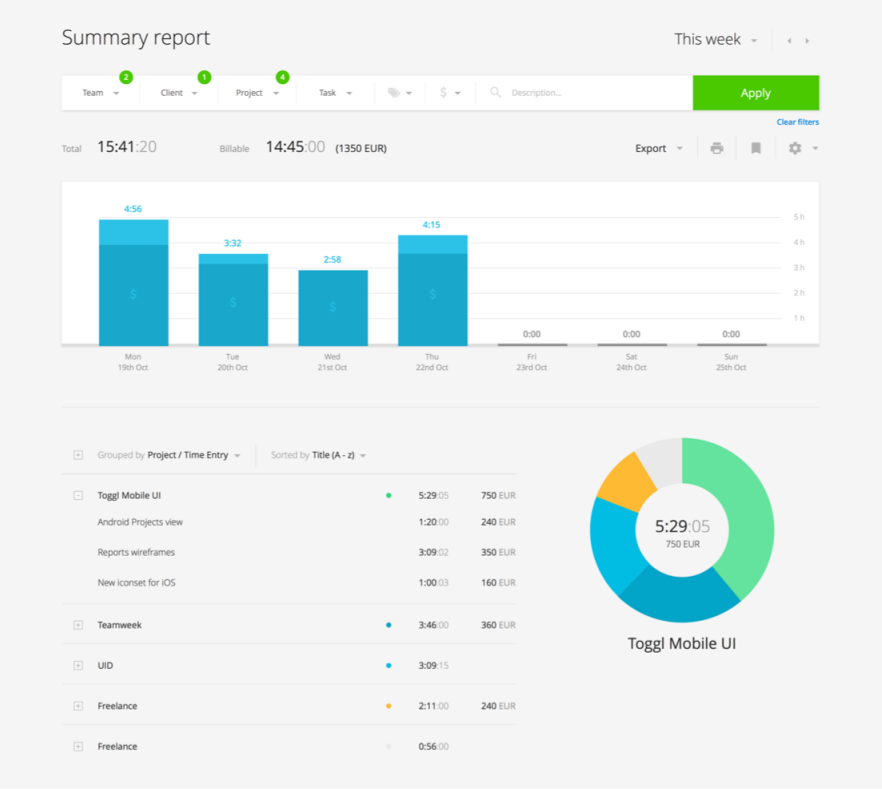
Výhody:

- Webová aplikace.
- Open source.

Nevýhody:

- Náročné na hosting.

2.1.3 Toggl



Obrázek 2.4: Ukázka výkazu ze software Toggl.

Toggl [10] je jednoduchý nástroj pro měření času. Lze si ho představit jako chytré stopky, které můžeme podle potřeby spustit a vypnout. K naměřenému času se pak napíše činnost a přiřadit k projektu. Toggl existuje ve formě webové aplikace v prohlížeči, ale také v desktopové verzi pro Windows, Mac a Linux.

Po přihlášení do službu lze spustit stopky a kdykoliv je zastavit. Po zastavení se vytvoří časový úsek, ke kterému můžeme přistupovat jako k pracovnímu výkazu. Dále můžeme specifikovat provedenou činnost (pojmenovat výkaz) a volitelně přiřadit k projektu. Nástroj detekuje nečinnost pokud ho

uživatel zapomněl vypnout. Vložit časový úsek lze provést i manuálně, nemusí se spouštět stopky.

Pokud se výkazy budou poctivě přiřazovat k projektu, tak Toggl umožňuje filtrovat všechny výkazy projektu ve specifikovaném časovém období, tím vznikne souhrný výkaz.

Toggl má několik úrovní placení. Pokud ho chceme využít ve firemním prostředí, cena začíná na 10 dolarech za člena týmu na měsíc. Pro základní individuální použití je Toggl zdarma.

Výhody:

- Webová aplikace.
- Multiplatformní aplikace pro měření času.

Nevýhody:

- Cena.
- Složitější vytváření přehledu odpracovaných hodin.

Kapitola 3

Analýza

Tato kapitola se zabývá analýzou domény navrhované aplikace. Uvedu zde požadavky kladené na aplikaci a způsob, jakým bude s aplikací zacházeno. Poznatky z kapitoly budou základem pro návrh v následující kapitole.

Popisovaná aplikací je vykazovací systém a jejím hlavním přínosem je usnadnit evidenci výkazů práce na různých projektech. Součástí analýzy je převážně backendová část, která pracuje zejména s logikou programu a souvisejícími daty.

Druhá část aplikace, nazývaná frontend, slouží k zobrazení dat a interakci s uživateli. V této kapitole neuvádím žádné návrhy obrazovek nebo diskuzi o požadavcích na UI/UX.

3.1 Procesy

Zde stručně popíši procesy, které bude aplikace podporovat. Převážně se jedná o správu projektů a uživatelů a proces vykazování práce. Dále stručně popíši tvorbu přehledu vykázaných hodin.

3.1.1 Evidence zaměstnanců a projektů

Nejprve, před možností evidence hodin, musí aplikace umožnit přidávat nové projekty nebo zaměstanance do systému. Akci bude moci vykonat pouze uživatel s většími právy v systému. Cílem aplikace není shromažďovat velký objem dat, ale získat základní informace o zaměstanci nebo projektu. U zaměstanců se navíc bude evidovat seznam dovedností. Aplikace umožní filtrovat zaměstnance podle dovedností a usnadní tím hledání vhodných lidí na projekt podle specifických znalostí.

Úprava položek je funkce, která by neměla v aplikaci chybět. Projekt by mohl být upravován pouze managementem, na druhou stranu by systém měl podporovat editaci vlastního uživatelského profilu.

Poslední část procesu je vymazání starých a nepotřebných položek, které zůstaly aktivní.

3.1.2 Přiřazení zaměstnance na projekt

Přiřazení zaměstnance na projekt je další důležitý proces, který bude aplikace podporovat. Management bude mít díky aplikace přehled o všech zaměstnancích a jejich přidělených projektech. V malých firmách rozhoduje o přiřazení na projekty majitel nebo projektový manažer a aplikace by to měla respektovat tím, že umožní proces vykonat pouze uživatelům s touto rolí.

Uživatel, který je přidělený na projekt, získává pravomoc na tento projekt vykazovat.

3.1.3 Vykazování práce

Zaměstnanec, který je již přiřazený na projekt, má možnost v systému vykazovat čas strávený prací na projektu. Uživatel musí alespoň zadat datum a počet odpracovaných hodin do systému, jinak nebude výkaz aplikací přijat. Ke každému výkazu lze připojit poznámku, čeho se daný výkaz týkal.

Typ činnosti je další důležitá informace, která by neměla v aplikaci chybět. Nemusí se vždy jednat o pracovní výkaz, ale aplikaci můžeme využít k evidování počtu vyčerpané dovolené nebo nemocenské.

3.1.4 Tvorba přehledu práce

Aplikace umožní vytvořit zprávu odpracovaných hodin za uvedené období, nejčastěji za kalendářní měsíc. Management tímto získá ucelenou informaci o tom, kdo na jakém projektu odpracoval kolik hodin. Zaměstnavateli poskytne přehled vykázaných hodin na projektu podklad k fakturaci zákazníka.

Zaměstnanec si také v aplikaci může nechat vypsat počet svých odpracovaných hodin. Osoba samostatně výdělečně činná bude podle výpisu fakturovat zaměstnavateli. Zaměstnance může tento přehled informovat o přibližné velikosti jeho příští mzdy.

3.2 Funkční požadavky

Funkční požadavky specifikují konkrétní funkce aplikace z pohledu uživatele. Následuje seznam funkcí, které budou v aplikaci dostupné uživateli.

Při návrhu funkčních požadavků částečně vycházím ze zadání diplomové práce.

3.2.1 Správa uživatelů

F01 Registrace
Registrace uživatele bude možná. Uživatel si musí zvolit unikátní uživatelské jméno.

F02 Ověření uživatele
Aplikace je schopná ověřit uživatele pomocí uživatelského jména a hesla.

F03 Změna hesla
Aplikace umožní uživatelům měnit svá hesla.

F04 Editace údajů
Uživatelé budou moci upravit údaje o sobě.

F05 Kompetence
Aplikace bude u osob evidovat jejich kompetence. K jednomu člověku lze přiřadit více dovedností.

3.2.2 Správa projektů

F06 Založení a editace projektu
Vedení společnosti bude moci vytvářet a upravovat projekty evidované v aplikaci. Zaměstnanec nemá v systému právo zakládat nebo měnit informace o projektech.

F07 Přidání uživatele na projekt
Dalším požadavkem je možnost přiřazení lidí na projekt. Zaměstnanec a projekt by již měly být přítomny v systému. Aplikace by mělá být schopná přiřadit více lidí na stejný projekt nebo stejného člověka na různé projekty. Pouze management je oprávněný provést tento úkon.

F08 Zobrazení přiřazených projektů
Uživatel bude moci zobrazit projekty, které mu byly přiřazeny nadřízeným.

F09 Plánování vytíženosti
Aplikace bude zobrazovat časové vytížení zaměstnanců, zejména počátek a konec projektů.

3.2.3 Vykazování

F10 Vykazování
Uživatel může vykazovat pouze na přiřazené projekty.

F11 Zobrazit vlastní vykázané hodiny
Aplikace bude vytvářet reporty pro uživatele. Uživatel bude moci specifikovat časový úsek pro zobrazení vykázaných hodin.

F12 Editace a mazaní vlastních výkazů
Uživatel bude moci upravit nebo případně odstranit vlastní výkazy. Cizí výkazy bude moci editovat nebo mazat pouze manager.

F13 Schvalování výkazů
Nadřízený bude moci schvalovat výkazy.

3.3 Nefunkční požadavky

Nefunkční požadavky specifikují vlastnosti a omezující podmínky aplikace.

Při návrhu nefunkčních požadavků částečně vycházím ze zadání diplomové práce.

N1 REST API
Aplikace bude komunikovat skrz rozhraní REST API.

N2 Persistentní uložení dat
Aplikace bude podporovat uložení dat do databáze.

N3 Podpora více uživatelů
Více uživatelů bude moci komunikovat s aplikací najednou.

N4 Bezpečnost
Aplikace bude poskytovat základní ochranu dat uživatelů a bude rozlišovat různé role v systému.

3.4 Uživatelské role a oprávnění

Aplikace musí rozlišovat několik rolí, aby správně určovala práva v systému. Základní uživatelská role je pro zaměstnance. Manager bude mít v systému větší práva než zaměstnanec a role s největšími právy bude náležet administrátorovi. Nepřihlášený uživatel nemá přístup do aplikace a nesmí zobrazovat ani měnit data v aplikaci.

3.4.1 Uživatel

Běžným uživatelem aplikace je zaměstnanec, který má přístup ke svému účtu a k vykazování práce. Vykazovat lze pouze na projekty, které jsou mu přiděleny. Uživatel je schopný měnit své údaje v systému a upravovat vlastní výkazy. Nakonec je uživateli umožněno zobrazit svůj výkaz za celý měsíc, kde bude vidět, kolik hodin vykázal.

3.4.2 Manager

Management má stejná práva jako obyčejný zaměstnanec a několik zodpovědností navíc. Manager může v aplikaci spravovat data o projektech a má přístup ke všech výkazům v aplikaci.

3.4.3 Administrátor

Administrátor je speciální role správce, který je pověřen technickou stránkou aplikace. Jeho úkolem je zejména správa uživatelů a zodpovídá za vytváření uživatelských účtů a správnost jejich rolí. Administrátor má všechny práva jako uživatel a manager dohromady plus možnost zakládat a dále spravovat účty uživatelů.

3.5 Model případů užití

Případy užití představují ucelené jednotky funkčnosti, které systém poskytuje. Rumbaugh [11] je definuje jako „specifikaci posloupnosti akcí, včetně proměnných a chybových posloupností, které může systém, podsystém nebo třída vykonávat prostřednictvím interakce s externími objekty za účelem poskytnutí služby přinášející nějakou hodnotu“.

3.5.1 Aktéři

Aktéři představují role přidělené objektům, které využívají popisovaný systém. Při hledání aktérů se zjišťuje kdo nebo co systém používá a jak s ním komunikuje.

V aplikaci jsem definoval několik rolí odvozených od pozic v modelové společnosti.

Uživatel
: Jedná se o obyčejného uživatele aplikace bez větších práv v systému. V modelové firmě to většinou je zaměstnanec. Tento aktér se blíží systémové roli uživatele.

Manager
: Role reprezentuje vedení společnosti a má zvýšená práva pro správu projektů a zaměstnanců.

Administrátor
: Reprezentuje uživatelskou roli administrátora, který má nejvyšší práva v aplikaci.

3.5.2 Správa Uživatelů

Use Case I: **Registrace**

Aktoři: Administrátor

1. Administrátor vyplní jméno, příjmení a uživatelské jméno.
2. Aplikace vytvoří nového uživatele a vygeneruje jeho první heslo.

Use Case II: **Smazání uživatele**

Aktoři: Administrátor

1. Administrátor vybere uživatele ke smazání.
2. Administrátor odešle žádost o vymazání uživatele za systému.
3. Aplikace smaže uživatele.

Use Case III: **Přihlášení**

Aktoři: Uživatel, Manager, Administrátor

1. Uživatel vyplní své přihlašovací údaje.
2. Aplikace ověří správnost odeslaným údajů.

Use Case IV: **Úprava osobních údajů**

Aktoři: Uživatel

1. Uživatel upraví své osobní údaje, např. jméno, příjmení, kontaktní údaje nebo své dovednosti. Uživatel odešle požadavek na uložení.
2. Aplikace změny uloží.

Use Case V: **Změna hesla**

Aktoři: Uživatel, Manager

1. Uživatel vyplní nové heslo a odešle ho aplikaci.
2. Aplikace uživateli uloží nové heslo.

Use Case VI: **Vyhledat podle dovedností**

Aktoři: Manager

1. Management vybere dovednost podle které chce vyhledávat.
2. Aplikace zobrazí všechny zaměstnance s danou dovedností.

Use Case VII: **Zobrazení vytíženosti**

Aktoři: Manager

1. Management vybere zaměstnance, u kterého chce zobrazit vytíženost.
2. Aplikace zobrazí vytíženost zaměstance.

3.5.3 Správa projektů

Use Case VIII: **Přidat projekt**

Aktoři: Manager

1. Management chce přidat nový projekt.
2. Management vyplní údaje o projektu jako je název, zkratku nebo poznámku.
3. Management poté nový projekt odešle k uložení.
4. Aplikace úloží nový projekt.

Use Case IX: **Úprava projektu**

Aktoři: Manager

1. Management zobrazí seznam projektů a vybere ten, který chce upravit.
2. Management upráví detaily projektu.
3. Management poté uloží změny.
4. Aplikace uloží změny upraveného projektu.

Use Case X: **Přiřazení zaměstnance na projekt**

Aktoři: Manager

1. Management vybere projekt, ke kterému chce přiřadit zaměstnance, ze seznamu všech projektů.
2. Management zvolí zaměstnance. Volitelně může management vyplnit informace o délce práce na projektu tím, že vyplní datum zahájení a ukončení.
3. Aplikace uloží informaci o přiřazení zaměstnance na projekt.

Use Case XI: **Odstranění zaměstance z projektu**

Aktoři: Manager

1. Management vybere projekt.
2. Management zvolí zaměstnance, kterého chce odstranit z projektu.
3. Aplikace odstraní vazbu mezi zaměstnancem a projektem.

Use Case XII: **Odstranění projetku**

Aktoři: Administrátor

1. Administrátor vybere projekt, který chce smazat, ze seznamu projektů.
2. Administrátor odešle žádost o vymazání.
3. Aplikace smaže projekt.

3.5.4 Vykazování

Use Case XIII: **Vykazování na projekt**

Aktoři: Uživatel

1. Uživatel zvolí projekt, na který chce vykázat.
2. Uživatel vyplní údaje o výkazu, zejména datum a počet odpracovaných hodin. Uživatel poté výkaz uloží.
3. Aplikace uloží nový výkaz.

Alternativně

1. Aplikace zobrazí kalendář a Uživatel zvolí den, na který chce vykázat.
2. Uživatel vyplní údaje o výkazu, zejména projekt a počet odpracovaných hodin. Uživatel poté výkaz uloží.
3. Aplikace uloží nový výkaz.

Use Case XIV: **Editace výkazu**

Aktéři: Uživatel

1. Uživatel zvolí výkaz, který chce upravit.
2. Uživatel upraví detaily výkazy, např. projekt, datum nebo počet odpracovaných hodin.
3. Uživatel změněný výkaz uloží.
4. Aplikace uloží změny.

Use Case XV: **Smazání výkazu**

Aktéři: Uživatel

1. Uživatel zvolí výkaz, který chce odstranit z aplikace.
2. Uživatel vybraný výkaz odstraní.
3. Aplikace vymaže výkaz.

Use Case XVI: **Report projektu**

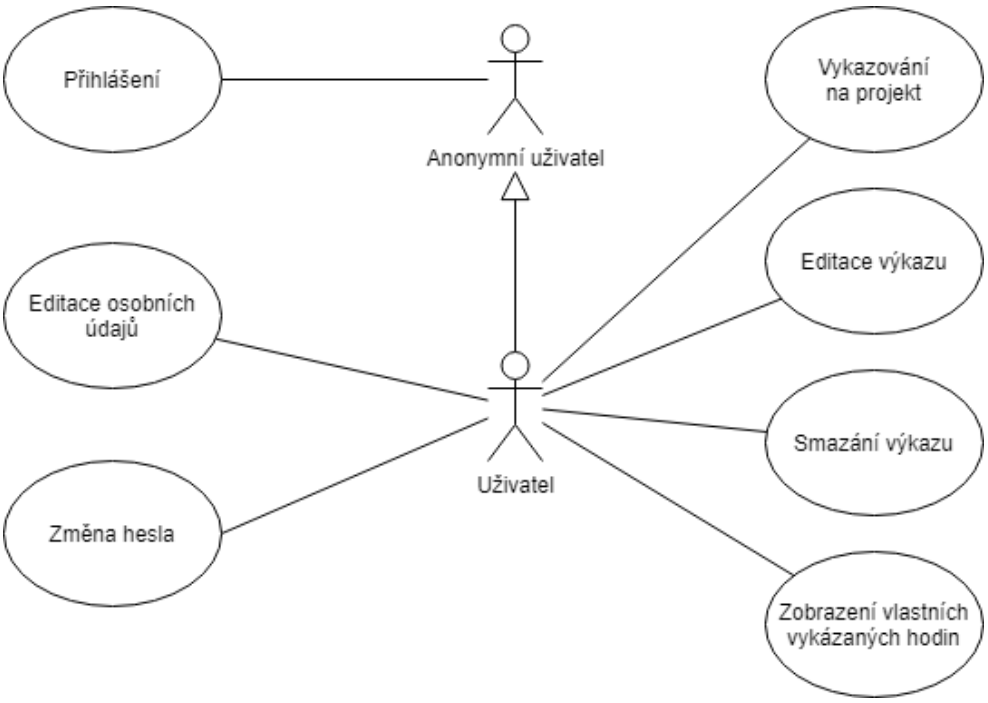
Aktéři: Manager

1. Management zvolí projekt a období, které se týká výkazů.
2. Aplikace zobrazí všechny výkazy, které se týkají projektu v dotazovaném období a součet všech hodin.

Use Case XVII: **Report zaměstnance**

Aktéři: Uživatel, Manager

1. Uživatel zvolí období.



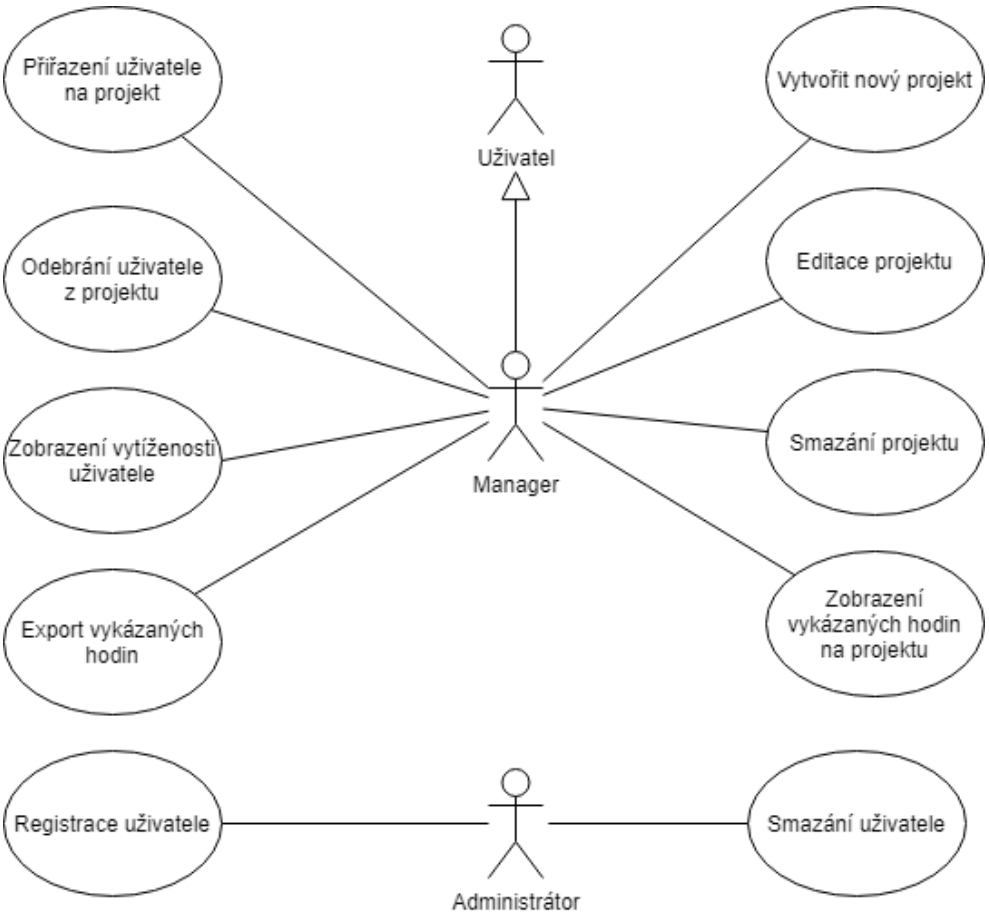
Obrázek 3.1: Use case Uživatel.

2. Aplikace zobrazí přehled všech výkazů z požadovaného období a součet hodin.

Use Case XVIII: **Export vykázaných hodin**

Aktéři: Uživatel, Manager

1. Uživatel zvolí období.
2. Aplikace exportuje výkaz.



Obrázek 3.2: Use case Manager a Administrátor.

Kapitola 4

Návrh

V této kapitole se věnuji návrhu aplikace a vycházím z funkčních a nefunkčních požadavků, které jsou popsány v předcházející kapitole.

4.1 Architektura

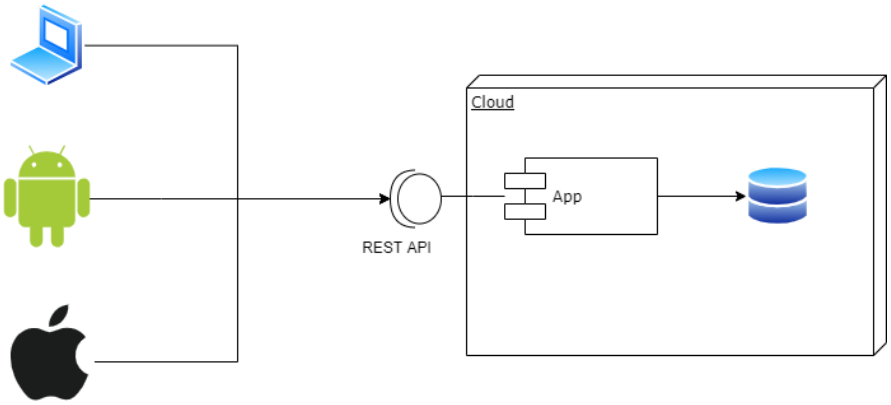
Architektura celé aplikace je třívrstvá a tvoří ji vrstva prezenční, aplikační a databázová 4.1.

Aplikační a datová vrstva aplikace bude nasazena na veřejném cloudovém serveru. Aplikační logiku plánuji zabalit do kontejneru pomocí technologie Docker [12] a výsledný kontejner spustit na některém cloudové službě. Výhodou zvoleného postupu je, že nemusím aplikaci nasazovat na žádný konkrétní aplikační server jako je např. Apache Tomcat [13]. Na stejné cloudové službě plánuji uložit data pomocí některé databáze. S tímto serverem bude klient komunikovat prostřednictvím REST API.

Prezenční vrstvu tvoří klient, který zprostředkuje uživatelské rozhraní. Klient může být webová aplikace nebo mobilní aplikace pro iOS a Android a slouží čistě jako zobrazovací a ovládací prostředek, kde se nevyskytuje žádná aplikační logika. Někdy se prezenční vrstva může označit pojmem frontend a nejčastěji je vytvořen v JavaScriptu za použití některého známého frameworku např. Angular [14], React [15] nebo Vue.js [16].

4.2 Výběr programovacího jazyka

Existuje velký výběr programovacích jazyků, které jsou více méně vhodné pro tvorbu webové aplikace. Některé jsou velmi populární a často se nacházejí vysoko v žebříčkách jazyků pro web. Příkladem může být Java [17], Python [18] nebo PHP [19] pro backend a JavaScript s HTML a CSS pro frontend, z méně využívaných můžu jmenovat Ruby, Rust nebo Go [20, 21, 22].



Obrázek 4.1: Návrh architektury aplikace.

Při výběru jazyka beru v potaz hlavně moji osobní preferenci a zvolil jsem Javu, protože jsem již získal určitou znalost jazyka.

Mezi další kritéria určitě patří kvalita dokumentace a frameworků pro tvorbu webových aplikací. Java je také více bezpečná než Python a PHP, ale tvorba kodu může zabrat více času [23].

4.3 Výběr frameworku

Pracovat pouze s programovacím jazykem bez žádného frameworku je možné pro menší projekty, ale pro účely mojí diplomové práce jsem si uvědomil nutnost použití některého fameworku. Při výběru jsem posoudil několik kritérií: jestli splňuje funkční a nefunkční požadavky z analýzy, kvalitu dokumentace, podporu komunity a v neposlední řadě množství existujících knihoven.

Pro Javu existuje několik webových frameworků [24, 25], ale do užšího výběru jsem zařadil Spring Framework [26] a Play Framework [27], které podle mě nejlépe splňují podmínky na aplikaci. Nakonec jsem se rozhodl pro Spring Framework, protože je často využíváný, existují pro něj spousty návodů a mám s ním osobní zkušenost.

4.4 Výběr databáze

Databázové systémy jsou nedílnou součástí většiny moderních aplikací, protože poskutují perzistentní uložení dat, se kterými uživatelé pracují.

V dnešní době existují dvě dominantní databázové technologie mezi kterými jsem se rozhodoval:

- SQL databáze.
- NoSQL databáze.

SQL databáze, také označovány jako relační, jsou nejběžnější a nejrozšířenější databázové systémy. Data jsou zde strukturovány do tabulek a řádků. Bohužel tento přístup není praktický použitelný pro big data, protože ty jsou ze své podstaty nestrukturovaná. Přestože SQL databáze dokáží pojmout velké množství dat, je jejich kapacita omezená a škálovatelnost je problematická.

NoSQL databáze vznikly aby řešili některé nedostatky SQL databází, zejména škálovatelnost a uložení nestrukturovaných dat. Na druhou stranu NoSQL databáze ztrácí ACID vlastnost, která mimojiné zaručuje konzistentnost dat.

Některé rozdíly jsou posány v tabulce níže [28]:

	SQL	NoSQL
ACID	Ano	Ne
SQL	Ano	Ne
OLTP	Částečně	Ano
Škálovatelnost	Ne	Ano
Distribuované	Ne	Ano

Ve své práci jsem se rozhodl pro MongoDB [29], která je jedním z mnoha NoSQL databází. Hlavním důvodem pro tento typ místo klasické SQL databáze bylo urychlení vývoje. MongoDB je document based a proto není potřeba dopředu definovat žádné schéma. To zjednoduše vývoj zejména při změně doménového modelu, protože se nemusí aktualizovat změny v databázových tabulkách a následně migrovat data.

4.5 Návrh API

4.5.1 Pricipy RESTu

REST je softwarový návrhový vzor, který vytvořil a popsal Roy Riedling ve své dizertační práce v roce 2000 [30]. Webová aplikace, která splňuje definici REST API, se nazývá RESTful. Takovou aplikaci lze popsat pomocí zdrojů, které obsahuje, a operací nad nimi. Rozhraní umožňuje různým klientům (např. mobilním aplikacím nebo webovým prohlížečům) vytvářet nebo upravovat data.

Základní vlastnosti aplikace jsou vyjádřeny v resources, které mají unikátní identifikátor (v tomto případě unikátní URL). REST definuje následující operace nad daty: Create, Read, Update a Delete (CRUD). Resource může nabývat různých podob, např. XML, HTML, JSON. Všechny operace jsou bezestavové (stateless) a proto každý požadavek musí obsahovat informace potřebné k jeho vykonání. REST je zkratkou pro anglické Representation State Transfer.

Server se při žádosti o zdroj řídí dvěmi věcmi: unikátním identifikátorem zdrojem ve formě URL, kterým říkáme o co máme zájem a operací nad uvedeným zdrojem specifikovanou HTTP metodou.

4.5.2 OpenAPI

OpenAPI, také označovaný jako Swagger, pomáhá s popisem REST API. Obsahuje doporuční doporučení jak navrhovat přehledné a funkční REST rozhraní. Popis všech dostupných resources, aplikovatelných metod nebo objektů, ať už na vstupu nebo výstupu, se nachází ve formátu YAML nebo JSON. Mezi další věci, které se definují v OpenAPI, jsou parametry, zabezpečení nebo popis HTTP návratových kódů. Většinu ze zmíněných věcí lze komentovat, a tak zjednodušit prací programátorům, kteří využívají popsané rozhraní.

Z takto vytvořeného popisu API lze vygenerovat HTML dokumentaci spustitelnou ve webovém prohlížeči. Dále se využívá ke generování kódu především pro data transfer třídy. Generátor existuje pro většinu programovacích jazyků.

Při návrhu rozhraní jsem využil nástroj OpenAPI, který standardizuje popis REST endpointů. Vytvořil jsem popisující YAML soubor, který jsem využil k vygenerování dokumentace.

4.5.3 Návrh URL

URL je řetězec znaků, který ve webové architektuře slouží k lokalizaci unikátních zdrojů informací. Příkladem zmíněného zdroje může být být HTML stránka, CSS dokument, obrázek, atd. Dotazy na URL zdroj jsou vyžizováný webovými servery a jejich správce je zodpovědný za správnost adresy URL a odpovídajícího zdroje. URL obsahuje několik důležitých částí pro fyzickou lokalizaci zdrojů – URL adresa začíná protokolem (http nebo https), následuje doménové jméno, číslo portu, cesta ke zdroji a nepovinné parametry [31]. Adresa může vypadat například následovně:

```
http://www.domain.com:80/project/report?from=01-01-2020&to=31-01-2020
```

REST API využívá URL k lokalizaci objektů v aplikaci. API poskytuje rozhraní k několika objektům: uživatel, projekt, výkaz. Každý z uvedených zdrojů bude mít unikátní adresu.

Tabulka 4.1: Tabulka ukazuje návrh URL adres v aplikaci.

URL	HTTP metoda	Parametr
/persons	GET, POST	
/persons/id	GET, PUT, DELETE	
/projects	GET, POST	
/projects/id	GET, PUT, DELETE	
/reports/person/{id}	GET	from, to
/reports/projects/{id}	GET	from, to

Další část návrhu API patří definici vstupních a výstupních objektů, které se nacházejí v těle http metody. Objekt uživatele, který bude aplikace vracet

```
{
        "id": "5e8eed25d4c72409e41d7bf9",
        "name": "Michal",
        "username": "michal",
        "competenceSet": [
                {
                        "name": "Mongo"
                },
                {
                        "name": "Java"
                }
        ]
}
```

Ukázka kódu 4.1: Objekt Uživatele ve formátu JSON.

```
{
        "name": "Michal",
        "projects": [
                {
                        "id": "5e8f1f7ed4c7244e2089f238"
                        "name": "Important project",
                        "hoursWorked": 16.0
                }
        ],
        "totalHoursWorked": 16.0
}
```

Ukázka kódu 4.2: Ukázka souhrnu pro uživatele.

v metodě GET, bude vypadat přibližně jako v ukázce 4.1. Všechny objekty z návrhu tříd mají svojí reprezentaci ve vstupních a výstupních objektech.

V další ukázce 4.2 je zobrazen návrh, jak by mohl vypadat souhrn výkazů.

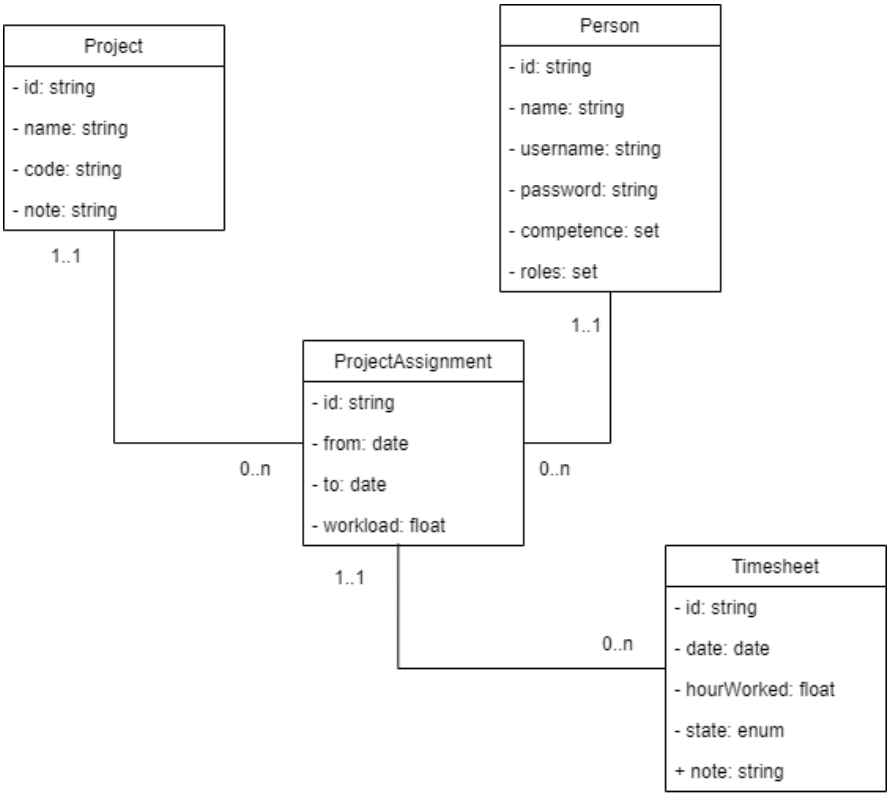
4.6 Model

Ke kvalitnímu návrhu aplikace patří doménový model aplikace. Na následujícím diagramu 4.2 je znázorněný diagram modelu tříd odpovídající logice programu.

Při vytváření modelu jsem úmyslně vynechal implementační detaily a pomocné třídy a soustředil se na zachycení vztahů mezi třídami. Každá entita má id, které ji identifikuje v databázi.

Entita Person a Project jsou nejvýše postavené v projektu. Person obsahuje několik zajímavých atributů: username a password jsou důležité pro přihlašování do aplikace, roles pomáhají při autorizaci uživatele v systému a naposledy competence obsahují informaci o dovednostech.

ProjectAssignment představuje přiřazení zaměstance na projekt. Atributy from a to informují o období, kdy je zaměstnanec přiřazený k projektu, jsou důležité pro výpočet vytíženosti.



Obrázek 4.2: Class Diagram

Timesheet představuje výkaz, obsahuje informace kdy a kolik hodin se odpracovalo. State reprezentuje stav výkazu, jestli je odevzdaný, schválený nebo odmítnutý.

Kapitola 5

Implementace a technologie

V této kapitole bych se rád věnoval vybraným technologiím, které jsem využil pro vývoj webové aplikace. Tento seznam není kompletní a soustřeďuje se především na technologie spojené s jazykem Java.

5.1 Použité technologie

5.1.1 Java

Java [17] je programovací jazyk, který vznikl v 90. letech ve společnosti Sun Microsystems. V současné době je to jeden z nejpopulárnějších jazyků pro vývoj server-side aplikací a standard pro vývoj aplikcí pro operační systém Android [32]. Další důvodem dnešní popularity je historie Java platformy, protože existuje velké množství dokumentace, knihoven a frameworků zaměřené na aplikace pro různé oblasti podnikání včetně eshopů a bankovních systémů.

Na rozdíl od ostatních programovacích jazyků, které jsou navrženy, aby se zkompilovaly do strojového kódu, nebo byly intepretovány ze zdrojového kódu v běhu aplikace, je Java navržena tak, aby se zkompilovala do bytecodu, který se poté spouští v Java Virtual Machine (JVM).

Java byla vyvinuta Jamesem Goslingem a jeho spolupracovníky na počátku 90. let ve firmě Sun Microsystems. Java se dělí na verze, které se vydávají. Od první verze, která vyšla roku 1995, proběhlo několik změn v důležitých verzích a byla přidána spousta různých funkcionalit. Ve verzi 1.8 přibylo několik důležitých novinek – např. přidáním lambda funkcí, které částečně umožňují funkcionální programování, lepší způsob kontroly null poiteru. Poslední verzí je Java 12 (JDK 12) vydaná 19. května 2019.

5.1.2 Spring Framework

Spring Framework [26] reprezentuje společný design a částečnou implementaci pro řešení problému nebo skupiny problémů. Framework je ze své podstaty

neúplný a programátor musí doplnit chybějící části, které pak definují chování aplikace. Jinými slovy framework poskytne implementaci pro problémy, které se vyskytují ve všech aplikacích, ale business logiku musí programátor vypracovat sám.

Spring Framework je jedním z nejpoužívanějších a nejrozšířenějších frameworků v Javě. Ze začátku primárně obsahoval funkci depency injection, ale dnes je to běžná platforma pro všechny druhy aplikací od malých projektů přes webové služby až po podnikové systémy.

Spring se skládá z modulů, které poskytují různou funkcionalitu, zmíním např. Aspect-oriented programming, Data access, Transaction management, Model–view–controller, Security, Messaging, and Testing. Programátor si vybere všechny potřebné knihovny.

Všechny projekty lze nalézt na spring.io/projects.

5.1.3 Spring Boot

Spring Boot [33] je projekt pro usnadnění a zrychlení vývoje Spring aplikací. Poskytuje základní nastavení a programátor nemusí věnovat tolik času konfigurací věcí, které jsou pro všechny projekty stejné.

Spring Boot poskytuje platformu pro vývoj samostatných Java aplikací vhodných pro firemní systémy. Vývojáři se mohou soustředit na logiku aplikace bez nutnosti konfigurovat celý Springový kontext.

Mezi další vlastnosti patří [33]:

- Vytvoření samostatné Spring aplikace.
- Aplikační server je přítomný pro běh aplikace.
- Poskytuje konfiguraci pro knihovny aplikace.

Pro využití výhod knihovny Spring Boot jsem musel nakonfigurovat Maven 5.1.

5.1.4 MapStruct

MapStruct [34] je nástroj, který generuje kod pro mapování mezi dvěmi Java objekty. Výhodou nástroje je, že využívá metody a je tedy rychlý a typovaný. V práce jsem MapStruct použil k mapování objektů z REST API na objekty se kterými pracuje aplikace. Takto snadno pak odděluji jednotlivé vrstvy od sebe.

5.1.5 MongoDB

MongoDB [29] je document based NoSQL databáze. Místo běžného schématu tabulka a řádek u běžného relační databáze, Mongo používá kolekce a dokumenty.

```
<project>
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.1.4.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>cz.thesis</groupId>
        <artifactId>timesheet</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>timesheet</name>
        <description>Project for diploma thesis</description>

        <properties>
                <java.version>1.8</java.version>
                <org.mapstruct.version>1.3.1.Final</org.mapstruct.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-rest</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-security</artifactId>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
        </build>
</project>
```


Ukázka kódu 5.1: Konfigurační soubor pro Maven.

```
public interface IMapper {

        PersonDto personToPersonDto(Person p);
        Person personDtoToPerson(PersonDto p);

        PersonDetailDto personToPersonDetailDto(Person p);
        Person personDetailDtoToPerson(PersonDetailDto p);

        ProjectDto projectToProjectDto(Project p);
        Project projectDtoToProject(ProjectDto p);

        TimesheetDto timesheetToTimesheetDto(Timesheet t);
        Timesheet timesheetDtoToTimesheet(TimesheetDto t);
}
```

Ukázka kódu 5.2: Definice pro Mapstruct.

Dokument je podobný řádku ve smyslu nejmenší formy dat, kterou lze do relační databáze vložit, je dokument. Dokument obsahuje políčka a jeho strukturu tvoří dvojice klíč hodnota podobný formátu JSON. Na rozdíl od řádku v relační databázi, dokument nemusí obsahovat stejné položky, některé mohou chybět, jiné zase přebývat, dokument akorát musí obsahovat unikátní id. Dokumenty se vkládají do kolekcí a jsou víceméně ekvivalentní s tabulkou v relační databázi.

5.1.6 Přístup k datům

Pro komunikaci s databází využívám již existující knihovny pro Spring, která se jmenuje Spring Data. Tato knihovna vznikla pro usnadnění přístupu k různým databázovým technologiím a jejím cílem je poskytnou konzistentní rozhraní pro práce s daty.

Spring Data je spíše rozhraní než použitelná implementace komunikace mezi aplikací a databází. Pro přístup k datům uloženým v MongoDB jsem využil třídu MongoRepository [35]. Při využití této knihovny mám k dispozici metody pro manipulaci s daty: ukládání, modifikace a mazání objektů spolu s hledáním podle ID nebo jiných atributů.

5.1.7 Maven

Maven je nástroj pro správu a automatizaci sestavení (tzv. buildu) softwaru. První verze byla vydána již v roce 2006. Maven je primárně určený pro Javu, ale je možné ho využít i v jiných programovacích jazycích.

Maven pomáhá ve dvou oblastech: popis buildu a externích závislosti software. Můžeme například specifikovat verzi Javy, název aplikace nebo knihovny,

```
public interface IPersonRepository
        extends MongoRepository<Person, String> {

    List<Person> findAllByCompetenceSet(Competence competence);

    Optional<Person> findByUsername(String username);

}
```

Ukázka kódu 5.3: Implementace rozhraní MongoRepository.

které chceme využít.

Jeho hlavní konfigurační XML soubor se nachází v kořenovém adresáři aplikace a popisuje jednotlivé operace, které jsou nad aplikací prováděny. Jednotlivé konfigurační soubory od sebe mohou dokonce dědit a tím vzniká přehledná konfigurace rozdělená na části. Některé části se dají využít ve více projektech a nedochází tak k opakování kódu.

Alternativou pro Maven je Gradle nebo Ant.

5.1.8 GIT

Git je distribuovaný systém sdr, který je používám pro týmovou spolupráci při vývoji software a správu verzí. Git vytvořil Linus Torvalds a první verze vyšla v roce 2005.

Git je v dnešní době velmi populární nástroj pro sdílení a verzování softwaru. Existují lokální a vzdálená verze repozitáře. Programátor nejprve pracuje ve své nesdílené lokální verze a dokončené změny odesílá na server. Gitlab a Github jsou příklady specializovaných stránek pro umístění vzdáleného repozitáře.

Protože jsem na projketu pracoval sám, tak jsem Git využil především jako prostředek k zálohování rozpracovaného softwaru.

5.2 Implementace REST API

Spring framework podporuje vytváření RESTových služeb pomocí web modulu.

Pro vytvoření REST API nejprve musí programátor vytvořit objekt, který bude obsluhovat jednotlivé URL a přidat anotaci @RestController na úroveň třídy. Takto Spring pozná, že třída je kontroler v architektuře Springu, a navíc specifikuje, že se jedná o REST API a podle toho bude serializovat a deserializovat návratové typy a parametry metod.

Přesná hodnota URL, kterou metoda obsluhuje, se specifikuje anotací, která obsahuje informaci o http metodě a URL.

```
@RestController
@RequestMapping("/persons")
public class PersonController {

        private final IPersonService service;

        public PersonController(IPersonService service) {
                this.service = service;
        }

        @GetMapping(produces = "application/json")
        @PreAuthorize("hasAnyRole('ADMIN', 'MANAGER', 'USER')")
        public List<PersonDto> getAll() {
                return service.getAll();
        }

        @GetMapping(value = "/{id}", produces = "application/json")
        @PreAuthorize("hasAnyRole('ADMIN', 'MANAGER')
         or #id == authentication.name")
        public PersonDetailDto getById(@PathVariable String id) {
                return service.getById(id);
        }
}
```

Ukázka kódu 5.4: REST API.

5.3 Zabezpečení

Zabezpečení aplikace se primárně skládá ze dvou činností: autentizace a autorizace. Autentizace spočívá v ověření uživatele, tedy že je tím, za koho vydává. To se nejčastěji děje pomocí uživatelského jména a hesla. Autorizace znamená přidělení oprávnění uživateli, tj. schopnost aplikace poznat, že uživatel má práva vykonat nějakou činnost (například smazat objekt) a pokud je nemá, tak mu tuto operaci zakázat. Spring framework pracuje s modulem Security, který poskytuje nástroje pro autentizaci a autorizaci.

5.3.1 Autentizace

Nejprve jsem se musel rozhodnout, který způsob autorizace bude aplikace podporovat. Nakonec jsem se rozhodl pro Basic access authentication, který posílá údaje pro přihlášení v hlavičce HTTP. Výhodou je snadná konfigurace na severové straně. Na druhou stranu je velikou nevýhodou nezašifrování přihlašovacích údajů a je snadné odposlechnutí uživatelského jména a hesla útočníkem. Mezi další používané metody patří OAuth protokol.

```
GET / HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Authorization: Basic YWRtaW46YWRtaW4=
```

Ukázka kódu 5.5: Přihlašování pomocí Basic access authorization.

```
@Component
public class MongoUserDetailsService implements UserDetailsService {

  private final IPersonRepository usersRepository;

  public MongoUserDetailsService(IPersonRepository usersRepository) {
        this.usersRepository = usersRepository;
  }

  @Override
  public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException {
        Person p = usersRepository.findByUsername(s)
                .orElseThrow(() -> {throw new UsernameNotFoundException("User not found");});
        List<GrantedAuthority> roles = p.getRoles().stream()
                .map((role) -> new SimpleGrantedAuthority("ROLE_" + role.toUpperCase()))
                .collect(Collectors.toList());
        return buildUser(p, roles);
  }

  private UserDetails buildUser(Person user, List<GrantedAuthority> authorities) {
        return new User(user.getId(), user.getPassword(), authorities);
  }
}
```

Ukázka kódu 5.6: Implementace rozhraní UserDetailsService v Spring Security.

Poté jsem musel vytvořit a nakonfigurovat úložistě s uživateli, které obsahuje uživatelská jména, hesla a role v systému. Toho jsem docílil vytvořením třídy, která splňuje rozhraní UserDetailsService s metodou loadUserByUsername5.6, která vrátí přihlášeného uživatele nebo vyhodí výjimku pokud přihlášení neproběhlo správně. Naposledy jsem musel nakonfigurovat Spring Security, aby využíval novou třídu pro ověřování přihlašovacích údajů.

5.3.2 Autorizace

Druhou částí zabezpečení je kontrola přístupu. Ta může fungovat několika způsoby: zabezpečíme URL nebo přístup do metod. Rozhodl jsem se pro druhou část, protože mi dává větší kontrolu nad zabezpečením přístupu. V aplikaci jsem navrhl tři bezpečnostní role, které definují role uživatele: ROLE_USER, ROLE_MANAGER, ROLE_ADMIN.

Spring Security řeší kontrolu oprávnění před spuštěním funkce pomocí anotací. Anotace @PreAuthorize zabraňuje vykonání kódu pokud nemá splněna

```
@GetMapping(value = "/{id}", produces = "application/json")
@PreAuthorize("hasAnyRole('ADMIN', 'MANAGER') or #id == authentication.name")
public PersonDto getById(@PathVariable String id) {
        return service.getById(id);
}
```

Ukázka kódu 5.7: Ukázka použití anotace @PreAuthorize v Spring Security.

podmínka pro vstup.

Kapitola 6

Testování

Testování je důležitou částí vývoje softwaru, protože každý složitější produkt obsahuje chyby, které lze odstranit správným testováním, a tím zajistit co nejvyšší kvalitu produktu. Chyby je nutné odhalit co nejdříve, protože s pozdější fází vývoje stoupá cena opravy chyby.

Software lze testovat několika různými metodami v průběhu celého životního cyklu. Testy je možné kategorivat např. podle [36] takto:

- **Unit testy:** Testuje se nejmenší jednotky softwaru. Aby testy byly přesné, musí tester znát jejich implementaci a funkci. V praxi jednotkové testy píší programátoři, aby otestovali funkčnost vlastní práce. Unit testy se hojně vyžívají během Continuous Integration v rámci automatizovaného testování [37].
- **Integrační testy:** Integrační testy slouží k testování spolupráce několika modulů aplikace. Jednotlivé moduly mohou samy o sobě fungovat bezchybně, ale jejich integrace může přinést chyby.
- **Systémové testy:** Cílem systémových testů je vyhodnotit specifikace systému. Obvykle je software pouze část většího počítačového systému, který se v této části testuje jako jeden celek.
- **Akceptační testy:** Akceptační testy většinou provádí zákazník, aby se ujistil, že dodaný produkt splňuje všechny funkční a nefunkční požadavky a naplňuje jeho představy o kvalitě.

Další rozdělení techniky testů je podle znalosti systému [36]. Rozlišujeme black box, metodu testování na základě znalostí požadavků a specifikace, ale bez znalosti implementace. Na opačné straně je white box testování, které vychází ze pochopení architektury, interních struktu a vnitřní implementace.

V rámci testování vyvíjeného systému jsem se zaměřil na white box testování. Pro otestování celkové funkčnosti jsem naopak využil přístupu black box.

6.1 Jednotkové testování

Pro jednotkové testování jsem využil několik knihoven pro různé části kódu.

Pro testování metod a modulů jsem zvolil knihovnu JUnit 5 [38], kterou Spring integruje a testy se dokáží spustit najednou. JUnit je open source testovací framework pro Javu.

Všechny metody, které obsahují testy, jsou označeny anotací @Test. V těchto metodách se porovnávají očekávané hodnoty s hodnotami, které vrátí testovaná metoda. Tyto testy jsou spouštěné pokaždé, když se sestavuje aplikace, aby se předešlo přidání nových chyb do aplikace.

Kromě testování veřejných metod jsem testoval chování kontrolerů. K tomu lze použít např. knihovnu MockMvc [39]. Chtěl jsem především zajistit, aby kontrolery správně odpovídali na definované URL. Testy také ověřují jeslti kontrolery správně přijímá vstupní objekty a porovnává odpovědi s očekávaným výstupem.

6.2 Funkční testování

K funkčnímu testování jsem využil aplikaci Postman [40], která zjednodušuje vývoj a testování různých druhů API. Postman zejména zlehčuje posílání dotazů rohraní REST, SOAP a GraphQL.

Při funkčním testování je důležité, aby jednotlivá volání probíhala v určitém pořadí, např. nejprve musí být v systému vytvořený zaměstnanec, aby mohl být přiřazen na projekt a mohl začít vykazovat.

Pro testování jsem si vytvořil několik vzorových dotazů, které jsem použil k volání REST API. Tyto dotazy jsem poté poskládal do kolekcí, které Postman umožňuje automatizovaně posílat a tím vlastně vzniká testovací scénář.

Všechny kolekce jsou k dispozici v příloze.

Kapitola 7

Nasazení

Nasazení softwaru na webový hosting je v dnešní době možné uskutečnit několika způsoby.

Základní postup nasazení Java webové aplikace, který byl běžný před několika lety, je tento: zdrojový kód zkompilovat do webového archivu (war soubor) a do něho se přidají ostatní soubory důležité pro aplikaci (HTML a CSS, XML a jiné soubory obsahující konfigurace). Výsledný war soubor je poté nasazen na cílový aplikační server např. pomocí SSH. War soubor se zkopíruje do konkrétního adresáře, který je na serveru k dispozici. Jakmile webový server zjistí přítomnost archivu v tomoto adresáři, automaticky se spustí proces k nasazení aplikace.

V současné době je pro Java aplikace, ale i pro jiné programovací jazyky, k dispozici alternativa. Aplikace se může zabalit do univerzálního kontejneru, který se pak nasadí na konkrétním serveru. Výhodou tohoto řešení je např. [41, 42]:

- Přenositelnost – Kontejner je lehce přenositelný mezi různými platfomami a webovými cloudy.
- Efektivity – Využití méně výpočetního výkonu než při klasické virtualizaci.
- Stardadtizace – Programátor vytvoří aplikace na svém pracovním počítači a poté vytvoří kontejner, který oběží stejně na růnych platformách.
- Bezpečnost – Aplikace beží ve svém vlastním sandboxu, který omezuje komunikaci mezi různými kontejnery.
- Jednoduchá škálovatelnost – Vytvoření více stejných kontejnerů škáluje aplikace, která je pak schopná obsloužit více dotazů najednou.

Na druhou stranu je zvyšuje komplexita, protože se programátor musí seznámit s novým konceptem.

```
FROM azul/zulu-openjdk-alpine:8
ENV PORT 8080
EXPOSE 8080
COPY target/timesheet-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java","-jar","app.jar"]
```

Ukázka kódu 7.1: Použitá konfigurace v Dockeru.

Docker je nástroj, který jsem zvolil, pro kontejnerizaci aplikace.

Pro nasazení softwaru jsem si musel vybrat veřejný cloud. Rozhodoval jsem se mezi službami Microsoftu, Amazonu a Googlu.

7.1 Docker

Docker je open-source nástroj, který zjednodušuje vytvaření, nasazení a běh aplikace na serveru pomocí kontejnerů. Kontejner umožňuje vývoj a distribuci softwaru zabalený spolu se vších, co je potřeba k běhu aplikace. Díky tomu se programátor může spolehnout na fakt, že aplikace poběží stejně na jiném fyzické nebo virtuálním zařízení nehledě na různé nastavení systému.

Na Docker lze pohlížet jako na formu virtualizace, ale na rozdíl od klasického virtuálního systému, na kterém běží operační systém.

Rozhraní kontejnerů je jednotné pro platformy Linux, Windows a macOS. Tudíž lze na tuto technologii nahlížet jako na odlehčenou virtualizaci. Služba vznikla v roce 2013, ale rychle získává na popularitě.

Technologie dockeru ulehčuje nasazení aplikace na webový hosting. Programátorům stačí definovat Dockerfile neboli soubor se závislostmy. Docker dokáže tento soubor přečíst a na serveru naistalovat závislosti, nakonfigurovat proměnné a spustit aplikaci s definovaným příkazem a parametry.

Kapitola 8

Srovnání návratnosti, ekonomické zhodnocení

Tato kapitola se zabývá ekonomickou stránkou aplikace. Nejprve se pokusím vytvořit odhad ceny aplikace jako funkčního celku, za předpokladu, že bude přístupná přes webový prohlížeč. Další část se bude zabývat ekonomickou návratností investice a nastíní několik různých scénářů.

8.1 Odhad ceny aplikace

Odhad rozsahu ceny aplikace je náročná disciplína, kde každý může lehce udělat chybu. Méně než třetina softwarových projektů skončí včas a dodrží plánovaný rozpočet [43, 44].

Nejprve jsem se rozhodl získat odhad pracnosti podle plánované velikosti aplikace [45]. Jednou z možností, jak odhadnout velikost (a tudíž pracnost) aplikace je dle počtu obrazovek. V aplikaci jsem identifikoval deset různých obrazovek a to:

1. Seznam projektů
2. Seznam zaměstnanců
3. Detailní pohled na projekt
4. Detailní pohled na zaměstance
5. Doporučení zaměstnace na projekt
6. Přiřazení zaměstnance na projekt
7. Vytvoření výkazu
8. Upravení výkazu

9. Přehled odpracovaných hodin zaměstnance
10. Přehled odpracovaných hodin na projektu

Dle tohoto odhadu se dá mluvit o malé až středně velké aplikace. Celosvětově se cena takovéto aplikace pohybuje mezi 1 250 000 až 2 500 000 Kč. Lze ale předpokládat, že v České republice se bude pravděpodobně jednat o mnohem menší částku, protože české mzdy jsou menší, než s jakými počítá průzkum, který je převážně zaměřen na americký trh [46, 47, 48].

Průměrná cena jednoho dne práce programátora na českém trhu je přibližně 10 000 Kč [49]. Studie tuto cenu získala analýzou smluv, částka tedy reflektuje cenu, kterou lze získat na trhu služeb. Cena za programátora může být i nižší, pokud je zaměstnaný ve firme nebo pokud se jedná o přímého kontraktora. V takovém případě lze vycházet z průměrného platu programátora v České republice, který je zhruba 52 500 Kč [50].

V následující tabulce 8.1 je provedena citlivostní analýza nákladů. Lze sledovat, jak se náklady mění s platem programátora a jeho „výkonností“ tj. tím, za jak dlouho zvládne aplikaci udělat. Do výsledného srovnání bych měl započítat i náklady na hosting, případně cenu domény nebo bezpečnostních certifikátů. Vzhledem k tomu, že tyto položky jsou zanedbatelné v porovnání s cenou vývoje, rozhodl jsem se je v rámci zjednodušení zanedbat.

Tabulka 8.1: Citlivostní analýza vstupních nákladů v závislosti na době vývoje a ceně za měsíc vývoje.

	Cena programátora na měsíc v Kč			
Počet měsíců	50 000	100 000	150 000	200 000
3	150 000	300 000	450 000	600 000
6	300 000	600 000	900 000	1 200 000
9	450 000	900 000	1 350 000	1 800 000
12	600 000	1 200 000	1 800 000	2 400 000

8.2 Ekonomická návratnost investice

Investice znamená vložené finanční prostředky s cílem finačního zisku v budoucnosti. Cíl investice ale nemusí být přímo navázán na výnos nebo úrok, ale může být ve formě zvýšení konkurenceschopnosti a tržního postavení.

Techniky hodnocení návratnosti investic se používají pro zhodnocení zdali a jak rychle se finační prostředky vrátí, tudíž jestli se daná investice vyplatí. Tyto techniky se běžně dělí do dvou částí: statické a dynamické [51].

Statické metody se používají hlavně pro hodnocení krátkodobých investičních projektů, protože neberou v úvahu časovou hodnotu peněz. Naopak dynamické metody berou v úvahu časovou hodnotu peněz a všechny peněžní toky jsou diskontovány na současnou hodnotu.

Roční výnos investice počítám jako ušetřené náklady na nákup jiné vykazovací aplikace. Průměrná cena nástroje pro projektový management je kolem deseti dolarů za uživatele na měsíc [52]. Finanční úspora při 50 zaměstnancích je 500 dolarů na měsíc, z toho vychází roční úspora přibližně 130 000 Kč.

Při počítání návratnosti je rovněž důležité vhodně zvolit životnost projektu. Aplikaci by mohla modelová firma využívat minimálně 10 let, proto v příkladech počítám s životností investice 10 let. Pro zjednodušení výpočtu neuvažuji náklady na budoucí údržbu softwaru.

8.2.1 Návratnost investice

Doba návratnosti investice je důležitý ukazatel pro každého, kdo zvažuje určitou investici bez ohledů na výši dané investice. Nejjednodušším způsobem, jak ohodnotit dobu návratnosti je tzv. prostá doba návratnosti. Aby se investice vyplatila, prostá doba návratnosti by měla být kratší než polovina životnosti investice [53].

ROI vypočítám podle vzorce:

$$ROI = \frac{CF}{IN} \tag{8.1}$$

kde IN je celková investice a CF je roční peněžní tok.

Jak je ze vzorce vidět, jedná se o velice zjednodušený způsob, který počítá s nediskontovaným peněžním tokem a uvažuje peněžní tok pro každý rok identický. V následující tabulce 8.3 je vidět za jakých podmínek se investice vyplatí za předpokladu, že ji porovnáváme jen z hlediska prosté doby návratnosti.

Tabulka 8.2: Tabulka zobrazuje návratnost investice.

	Cena programátora na měsíc v Kč			
Počet měsíců	50 000	100 000	150 000	200 000
3	1,2	2,3	3,5	4,6
6	2,3	4,6	6,9	9,2
9	3,5	6,9	10,4	13,8
12	4,6	9,2	13,8	18,5

8.2.2 Současná čistá hodnota - Net Present Value

Čistá současná hodnota, zkráceně NPV, je metoda, která vyjadřuje rozdíl mezi současnou a budoucí hodnotou peněz. NPV se používá k plánování rozpočtu a vyhodnocení výnosnosti investice. Hlavní výhodou tohoto kritéria je zahrnutí faktoru času.

Diskontovaná hodnota představuje budoucí hodnotu vyjádřenou v dnešní hodnotě, protože současná hodnota peněz je vyšší než budoucí. To se děje ze především ze dvou důvodů – jednak hodnota peněz kvůli inflaci klesá a také

se volné peníze mohou bezrizikově investovat například na spořící účet bank. Proto se u NPV budoucí peněžní toky diskontují. Jiný pohled na diskontovou míru je možný v podobě ušlého zisku, který bych získal, kdybych peníze investoval jiným způsobem, např. investicí do peněžního trhu nebo modernějšího vybavení. Určení výše diskontu není jednoznačné a každý investor si diskontovou míru může určit v jiné hladině. Diskont jsem se rozhodl určit jako 8 %.

NPV se vypočítá podle vzorce s jednorázovou investicí [54]:

$$NPV = \sum_{t=1}^{n} \frac{CF_t}{(1+i)^t} - IN \tag{8.2}$$

kde:
CF_t = Peněžní toky během jednoho roku t
i = Diskontní míra
n = Životnost investice v letech
IN = Počáteční investice

Tabulka 8.3: Porovnání NPV rozdílných počátečných investic.

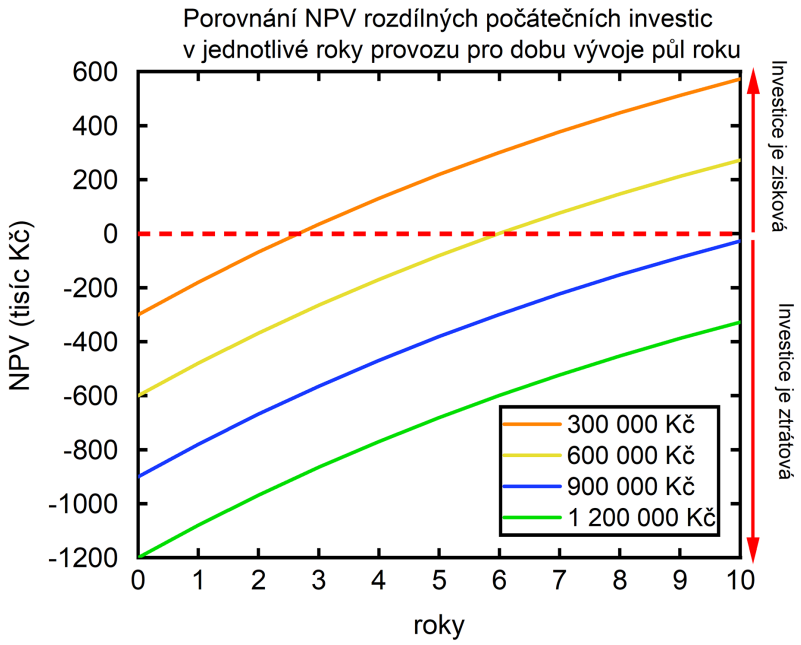
	Cena programátora na měsíc v Kč			
Počet měsíců	50 000	100 000	150 000	200 000
3	722 311	572 311	422 311	272 311
6	572 311	272 311	-27 689	-327 689
9	422 311	-27 689	-477 689	-927 689
12	272 311	-327 689	-927 689	-1 527 689

V následujících grafech uvádím příklady NPV v jednotlivých rocích a závislost NPV na diskontní míře pro scénáře, ve kterých vývoj aplikace trvá půl roku, což osobně považuji za nejpřesnější odhad doby vývoje.

Na prvním grafu 8.1 lze vidět v jakém roce se investice stane ziskovou pro různou počáteční investice. Lze vidět, že při platu programátora 50 000 Kč za měsíc (tj. počátečný investici 300 000 Kč) se investice stane ziskovou již v průběhu třetího roku provozu aplikace, zatímco pokud bude plat programátora vyšší než 150 000 Kč za měsíc, tak se ziskovou během života aplikace nestane nikdy.

Druhý graf 8.2 znázorňuje závislot NPV na diskontní míře. Z tohoto grafu je možné určit IRR neboli vnitřní výnosové procento. IRR zjistíme jako:

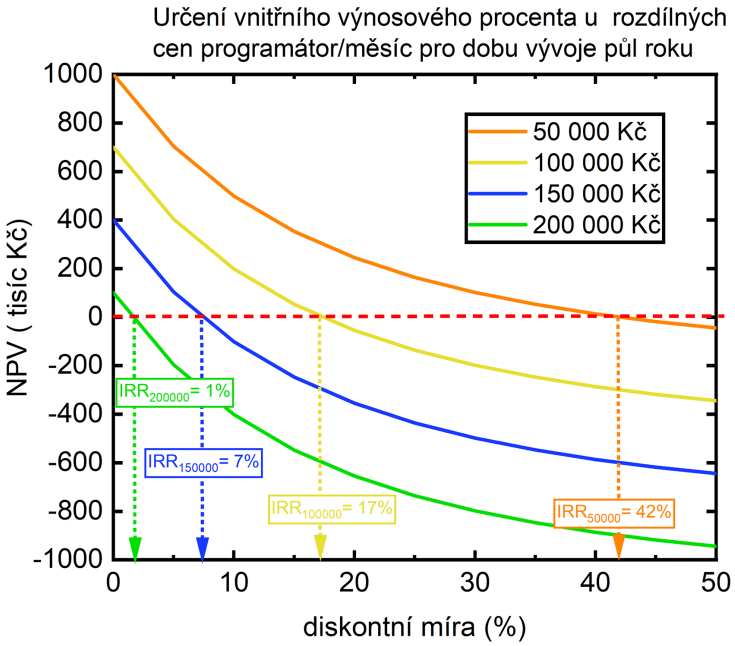
$$\sum_{t=1}^{n} \frac{CF_t}{(1+IRR)^t} - IN = 0 \tag{8.3}$$



Obrázek 8.1: Graf návratnosti investice vypočtený pomocí metody NPV

Vnitřní výnosové procento je hodnota diskontní míry, při které je NPV rovno 0. Grafické znázornění lze vidět na grafu 8.2. Investice podle IRR lze posuzovat tak, že čím je procento vyšší, tím je investice výhodnější.

Vyšší NPV mi v budoucnu přinese vyšší cash flow a vyšší IRR mi zajistí lepší zhodnocení investice. Pokud by se u porovnání dvou investic stalo, že jedna bude mít lepší NPV, ale druhá bude mít vyšší IRR, může si firma vybrat jestli preferuje vyšší budoucí peněžní toky nebo lepší zhodnocení projektu.



Obrázek 8.2: Graf návratnosti investice vypočtený pomocí metody NPV

Kapitola 9

Zefektivnění manažerských procesů

Aplikace je navržena tak, aby v ideálním případě, byla podpůrným nástrojem pro procesní řízení firmy. Tomu napomáhá jednak tím, že může doporučit zaměstnance na jednotlivé projekty na základě jejich dovedností a také jejich vytíženosti. Dále systém podporuje snadnější fakturování práce tím, že zobrazuje odpracované hodiny na každém z projektů.

9.1 Doporučení zaměstnanců

Jak již bylo řečeno, jednou z funkcí navrhované aplikace je doporučení zaměstnanců. V praxi to může fungovat tak, že vedoucí pracovník si v aplikaci zobrazí všechny své zaměstnance, kteří ovládají určitou dovednost, např. programuje v jazyce Java. Poté může zjistit, jak je daný zaměstnanec vytížen v nejbližších měsících. U všech uživatelů lze uvést jejich vytíženost při přiřazení, např. zaměstnanec A bude na nějakém konkrétním projektu pracovat od 1. 6. do 31. 12. na plný úvazek. Informace se zobrazí při dotazu na vytížení zaměstnance. Manager například zjistí, že danému zaměstnanci na konci příštího měsíce končí projekt a bude ho moci zařadit do výběrového řízení jiného projektu, nebo může být zaměstnanec plně vytížený na několik dalších měsíců a není proto vhodným kandidátem a musí vybrat někoho jiného.

Podmínkou správného doporučení (a tudíž i správného fungování aplikace) je pravidelné doplňování aktuálních informací. To může mít na starost jak manager, tak někdo ze zaměstnanců. Je pravděpodobné, že každý zaměstnanec ovládá více různých dovedností. V širším významu můžeme dovednost nahradit například získanou certifikací nebo znalostí cizího světového jazyka.

Toto řešení může značně usnadit práci projektovým manažerům, vedoucím týmů apod., kteří nemusejí ztrácet čas tvořením si vlastních seznamů zaměstnanců, jejich dovedností a množství zadané práce. Pokud firma takovou aplikací dis-

ponuje, stačí si ji otevřít a všechny užitečné informace jsou přehledně na jednom místě.

9.2 Reportování práce

Aplikace generuje výkazy práce za specifické období, což ulehčuje práci jak managementu tak zaměstnancům. Nadřízení, kteří fakturují odpracované hodiny celé firmy či týmu zákazníkovi, potřebují získat jednoduchý a ucelený výkaz.

Například modelová společnost fakturuje zákazníkovi jednou měsíčně projekt, na kterém pracují dva zaměstnanci. Management poté, co oba zaměstnanci dokončí vykazování za měsíc, získá ze systému počet hodin, které vyfakturuje zákazníkovi.

Na druhé straně, zaměstnanec má zaevidovaný počet hodin a může si snadno vypočítat mzdu.

Závěr

Cílem práce bylo analyzovat, navrhnout a implementovat jednoduchý vykazovací systém, který by napomáhal chodu v malé IT firmě.

Nejprve jsem definoval společnost a popsal modelovou IT firmu. V další kapitole jsem shrnul a porovnal existující systémy, které se zabývají vykazováním času, a snažil jsem se co nejlépe pochopit tuto problematiku. Na základě vlastních znalostí a srovnání ostatních vykazovacích systému jsem vypracoval podklady k analýze aplikace.

V další části práce jsem analyzoval požadavky na aplikaci. Zde jsem vypracoval podrobnější popis funkčních požadavků a příkladů použití. Ze své analýzy jsem poté vycházel v kapitole o návrhu aplikace. V rámci návrhu jsem nejdříve musel zvolit přijatelné technologie, které jsem následně využil pro vybudování systému. Během analýzy a návrhu jsem vycházel ze zadání diplomové práce a do řešení jsem se snažil zahrnout všechny požadované funkce a vlastnosti aplikace.

Implementační část práce pak navazuje na návrh a více zde popisuji využité technologie a některé postupy při tvorbě aplikace. Zvolil jsem programovací jazyk Java a Spring Framework pro realizaci webové aplikace, jako datové úložiště jsem si vybral databázi MongoDB. Výstupem je spustitelná aplikace, která poskytuje aplikační rozhraní REST API.

Po implementace jsem aplikaci otestoval pomocí automatizovaných testů a rovněž manuálně vyzkoušel její funkčnost. Celou aplikaci jsem poté zabalil do kontejneru pomocí nástroje Docker a nyní je připravená k nasazení na webový hosting. V budoucnu bych chtěl aplikaci nasadit na veřejnou cloudovou službu.

Dalším cílem práce bylo zhodnotit ekonomickou návratnost projektu. Nejprve jsem se pokusil odhadnout náklady na vývoj aplikace pomocí citlivostní analýzy nákladů a poté jsem metodou NPV vypočetl výnosnost. Vývoj vlastního softwaru, místo kupování licence jiného programu, se pro firmu vyplatí v případě, že cena za prgramátora nepřesánhne 50 000 Kč měsíčně nebo vývoj nebude trvat déle než 3 měsíce. Jediným dalším pro firmu přijatelným scénářem se ukázal vývoj trvající půl roku s platem programátora 100 000 Kč za měsíc.

Všechny další kombinace byly ztrátové. Nakonec jsem posoudil zlepšení firemních procesů, které se aplikace snaží podporovat.

Cíle práce mohu označit za dosažené a zadání za splněné, přestože aplikace není plně dokončená. Snažil jsem se aplikaci naprogramovat do využitelné verze. Technologie zvolené pro implementaci považuji za vhodně vybrané. V průběhu práce jsem nenarazil na žádný problém, který by nebylo možné vyřešit. Při vývoji a psaní diplomové práce jsem se seznámil s novými frameworky a technologiemi, které mi pomohou v budoucím profesním životě.

Literatura

[1] *Sektory trhu [online]*. 2018, [cit. 2020-05-25]. Dostupné z: https://managementmania.com/cs/outsourcing

[2] Hyršlová, J.; Klečka, J.: *Ekonomika podniku*. Vysoká škola ekonomie a managementu, 2008, ISBN 978-80-86730-36-3.

[3] *Sektory trhu [online]*. 2016, [cit. 2020-01-03]. Dostupné z: https://managementmania.com/cs/sektory-trhu

[4] Darden, S.: *Veřejný sektor (Public Sector) [online]*. 2017, [cit. 2020-01-02]. Dostupné z: https://managementmania.com/cs/verejny-sektor

[5] Oracle Database. [cit. 2020-05-25]. Dostupné z: https://www.oracle.com/cz/database/

[6] *What is SAP Software? [online]*. 2020, [cit. 2020-05-25]. Dostupné z: https://www.cbronline.com/what-is/what-is-sap-software-4912451/

[7] Atlassian: Jira Software. [cit. 2020-05-25]. Dostupné z: https://www.atlassian.com/software/jira

[8] Tempo Timesheets. [cit. 2020-05-25]. Dostupné z: https://marketplace.atlassian.com/apps/6572/tempo-timesheets-time-tracking-report?hosting=cloud&tab=overview

[9] Redmine. [cit. 2020-05-25]. Dostupné z: https://www.redmine.org/

[10] Toggl. [cit. 2020-05-25]. Dostupné z: https://toggl.com/

[11] Rumbaugh, J.; Jacobson, I.; Booch, G.: *The Unified Modelling Language Reference Manual*. 2005. Dostupné z: http://www.temida.si/~bojan/IPIT_2014/literatura/UML_Reference_Manual.pdf

[12] Docker Inc.: Docker.

[13] Apache Tomcat. [cit. 2020-05-25]. Dostupné z: http://tomcat.apache.org/

[14] Angular. [cit. 2020-05-25]. Dostupné z: https://angular.io/

[15] React. [cit. 2020-05-25]. Dostupné z: https://reactjs.org/

[16] Vue.js. [cit. 2020-05-25]. Dostupné z: https://vuejs.org/

[17] Oracle Corporation: Java. [cit. 2020-05-25]. Dostupné z: https://openjdk.java.net/projects/jdk/

[18] Python Foundation: Python. [cit. 2020-05-25]. Dostupné z: https://www.python.org/

[19] PHP. [cit. 2020-05-25]. Dostupné z: https://www.php.net/

[20] Franklin, C.: *10 Hot Programming Languages To Build Web Apps [online]*. 2016, [cit. 2020-04-30]. Dostupné z: https://www.informationweek.com/software/10-hot-programming-languages-to-build-web-apps/d/d-id/1327471

[21] Shiotsu, Y.: *Web Development 101: Top Web Development Languages to Learn in 2018 [online]*. 2017, [cit. 2020-04-30]. Dostupné z: https://www.upwork.com/blog/2017/11/top-web-development-languages-2018/

[22] James, N.: *8 Best Programming Languages to Develop an Ecommerce Website in 2017 [online]*. 2017, [cit. 2020-04-30]. Dostupné z: https://www.webecommercepros.com/best-programming-language-ecommerce-website-development-2017

[23] Hornostaiev, M.: *Java, Python, and PHP: Which is Better for Server Backends? [online]*. 2019, [cit. 2020-04-30]. Dostupné z: https://erminesoft.com/java-python-and-php-whichis-better-for-server-backends/

[24] T., M.: *10 of the Most Popular Java Frameworks of 2020 [online]*. 2020, [cit. 2020-05-01]. Dostupné z: https://stackify.com/10-of-the-most-popular-java-frameworks-of-2020/

[25] Shankar, R.: *10 Best Java Frameworks to Use in 2020 [online]*. 2020, [cit. 2020-05-01]. Dostupné z: https://hackr.io/blog/java-frameworks

[26] Spring Framework. [cit. 2020-05-01]. Dostupné z: https://spring.io/

[27] Play Framework. [cit. 2020-05-01]. Dostupné z: https://www.playframework.com/

[28] Binani, S.; Gutti, A.; Upadhyay, S.: *SQL vs. NoSQL vs. NewSQL - A Comparative Study*. 2016. Dostupné z: https://pdfs.semanticscholar.org/9d92/d600ff1a4bec346b6ca3fe6d8bf9677294b2.pdf

[29] MongoDB, Inc.: MongoDB. [cit. 2020-05-25]. Dostupné z: https://www.mongodb.com/

[30] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[31] Mozzila: *What is a URL? [online]*. 2019, [cit. 2020-05-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL

[32] Githut. [cit. 2020-04-23]. Dostupné z: https://githut.info/

[33] Spring boot. [cit. 2020-05-23]. Dostupné z: https://spring.io/projects/spring-boot

[34] Mapstruct. [cit. 2020-05-23]. Dostupné z: https://mapstruct.org/

[35] Interface MongoRepository. [cit. 2020-05-23]. Dostupné z: https://docs.spring.io/spring-data/mongodb/docs/current/api/org/springframework/data/mongodb/repository/MongoRepository.html

[36] COPELAND, L.: *A Practitioner's Guide to Software Test Design*. Artech House Publishers, 2004, ISBN ISBN 1-58053-791-X.

[37] Sacolick, I.: *What is CI/CD? Continuous integration and continuous delivery explained [online]*. 2020, [cit. 2020-05-15]. Dostupné z: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

[38] JUnit 5. [cit. 2020-04-25]. Dostupné z: https://junit.org/junit5/

[39] MockMvc. [cit. 2020-04-27]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/test/web/servlet/MockMvc.html

[40] Postman. [cit. 2020-04-25]. Dostupné z: https://www.postman.com/

[41] Peppas, N.: *The Benefits Of Containerization [online]*. 2019, [cit. 2020-05-05]. Dostupné z: https://www.liquidweb.com/kb/the-benefits-of-containerization/

[42] Gandhi, R.; Szmrecsanyi, P.: *The Benefits of Containerization and What It Means for You [online]*. 2019, [cit. 2020-05-05]. Dostupné z: https://www.ibm.com/cloud/blog/the-benefits-of-containerization-and-what-it-means-for-you

[43] SuperOkay: *10 reasons why software development projects fail [online]*. 2017, [cit. 2020-05-11]. Dostupné z: https://medium.com/superokay/10-reasons-why-software-development-projects-fail-7200e7c9ae2e

[44] Burger, R.: *20 Surprising Project Management Statistics [online]*. 2016, [cit. 2020-05-11]. Dostupné z: https://blog.capterra.com/surprising-project-management-statistics/

[45] Ostroverkh, Y.: *A Step-by-Step Guide on Calculating the Software Development Cost [online]*. 2018, [cit. 2020-05-11]. Dostupné z: https://diceus.com/how-to-estimate-software-development-costs/

[46] O., K.: *The Costs of Software Development: Challenges and Ready-Made Estimations [online]*. 2019, [cit. 2020-05-11]. Dostupné z: https://www.cleveroad.com/blog/software-development-costs

[47] Kurilo, A.: *What is the Cost of Software Development? [online]*. 2020, [cit. 2020-05-11]. Dostupné z: https://www.business.com/articles/the-cost-of-software-development/

[48] Flackett, J.: *How much does it cost to build a software application? [online]*. 2015, [cit. 2020-05-11]. Dostupné z: https://www.linkedin.com/pulse/how-much-does-cost-build-software-application-dr-john-flackett

[49] Bruckner, T.: Přehled obvyklých cen ICT prací [online]. 2018, [cit. 2020-05-11]. Dostupné z: https://www.mvcr.cz/soubor/prehled-cen-ict-praci-25-5-2018.aspx

[50] Platy v kategorii: Informační technologie [online]. [cit. 2020-05-11]. Dostupné z: https://www.platy.cz/platy/informacni-technologie

[51] Techniky hodnocení investic (investičních variant) [online]. 2017, [cit. 2020-05-11]. Dostupné z: https://managementmania.com/cs/techniky-hodnoceni-investic

[52] Pardo-Bunte, M.: How Much Does Project Management Software Cost? 2020 Pricing Guide [online]. 2020, [cit. 2020-05-11]. Dostupné z: https://www.betterbuys.com/project-management/project-management-pricing-guide/

[53] Tomáš, J.: *Ekonomika podniku II. [online]*. 2012, [cit. 2020-05-11]. Dostupné z: https://www.vsem.cz/data/data/sis-texty/studijni-texty-bc/st_ep_epII_tomas2.pdf

[54] Zikmund, M.: *Hodnocení investic: Čistá současná hodnota (NPV) stručně a jasně [online]*. 2010, [cit. 2020-05-15]. Dostupné z: http://www.businessvize.cz/rizeni-a-optimalizace/hodnoceni-investic-cista-soucasna-hodnota-npv-strucne-a-jasne

Příloha **A**

Seznam použitých zkratek

IT Informační technologie

OSVČ Osoba samostatně výdělečně činná

UI/UX User Interface, User Experience

GUI Graphical user interface

XML Extensible markup language

REST Representation State Transfer

SOAP Simple Object Access Protocol

HTML Hypertext Markup Language

CSS Cascading Style Sheets

JSON JavaScript Object Notation

API Application Programming Interface

ERP Enterprise Resource Planning

HTTP Hypertext Transfer Protocol

JVM Java Virtual Machine

JDK Java Development Kit

ACID Atomicity, Consistency, Isolation, Durability

SQL Structured Query Language

OLTP Online Transaction Processing

PHP Hypertext Preprocessor

NVP Net Present Value

IRR Internal Rate of Return

Příloha **B**

Obsah přiloženého CD

Vhodným způsobem vizualizujte obsah přiloženého média. Lze použít balíček `dirtree` a vytvořit např. následující výstup (adresáře src a text s příslušným obsahem jsou *povinné*):

```
readme.txt ..... stručný popis obsahu CD
src ..... zdrojové kódy implementace
text ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
    thesis ..... zdrojová forma práce ve formátu LaTeX
```