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.

8. ledna 2020

Poděkování

Děkuji všem a za vše. Nevíte-li, co sem napsat, příkaz odstraňte.

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 8. ledna 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 několika větách shrňte obsah a přínos této práce v češtině. Po přečtení abstraktu by měl mít čtenář dost informací pro rozhodnutí, zda chce Vaši práci číst.

Klíčová slova REST API, aplikace, Java.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords REST API, application, Java.

Obsah

Seznam obrázků

Úvod

Vykazování slouží k běžné pracovní činnosti. Ve většině firem stačí pouhé vykázání pracovní doby zaměstnanců, ale existují odvětví, které potřebují vést detailní vykazování času. Příkladem jsou firmy působící v IT odvětví. Vývoj softwaru je problematické a proto vznikl komplikovaný projekt management se spoustou nástrojů, které pomáhají s řízením projektu.

IT firmy, pokud nevyvíjí svůj vlastní produkt, často vytvářejí software na zakázku. Obvykle mají v jednu chvíli více zakázek pro různé zákazníky a pro firmu je kritické, aby evidovala správný počet odpracovaných hodin, které pak fakturuje zákazníkovi.

Existuje ale i další způsob ekonomické činnosti; firma může zaměstnávat certifikované specialisty, kteří poskytují podporu a údržbu. Toto není typická doména projektového řízení. I když je samozřejmě důležité nasadit projektové řízení, není nutné pracovat se všemi nástroji. Důležité je hlídat plnění smluv a správně alokovat lidské zdroje.

Problémem většiny dnešních vykazovacích systémů je, že buď velmi drahé nebo velmi složité se spoustou nepotřebných funkcí. Cílem této práce je porovnat jednotlivé existující nástroje a jejich cenu. Dále se v této práci podíváme na srovnání návratnosti mezi vývojem vlastního jednoduchého řešení a placením již existujícího vykazovacího systému.

Poslední částí je návrh a implementace jednoduchého systému na vykazování práce.

V této práci bych chtěl porovnat běžné aplikace. Naimplementovat jednoduché řešení v REST API.

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.

Dále bude obsahovat popis vnitřních procesů, organizační strukturu a jak firma využije reportovací systém.

1.1 Definice malé IT společnosti

První věcí, kterou je potřeba stanovit je, co to vlastně je malá IT firma. Podniky se dají dělit z velkého množství hledisek. Zde je uvedeno několik nejpoužívanějších příkladů [1]:

- podle právní formy
  - podniky jednotlivce – tj. živnosti, podnikání je 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 [2]
  1.
     - veřejný – poskytování veřejných služeb, financováno z veřejných rozpočtů [3]
     - 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 společnost 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, poskytování podpory softwaru, údržbou a poradenstvím, ale nevyvíjí vlastní software, ať již např. ve formě krabicového softwaru nebo webové aplikace. Zaměstnanci jsou outsourcováni k zákazníkům, kteří potřebují odbornou podporu nebo poradenství third-party softwaru nebo technologie, např. Oracle databáze nebo ERP systému SAP. Tento software sice vyvíjí a licencuje vydavatel, ale zavádění, podporu a customizaci přenechávají svým partnerům. Zákazníkům modelové firmy se nevyplatí zaměstnávat odborníky, protože pro ně nemají tolik práce a jsou drazí.

Modelová společnost má reálný obraz ve společnosti, která je partnerem SAPu. Firma poskytuje služby v podporu pro několik středních a velkých

zákazníků. Customizing je další běžná věc - lze zadat doprogramování funkcí, které nejsou v SAPu. Svým zákazníků také pomáhá s 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. Zároveň se může stát, že je jeden zaměstnanec přidělen na více projektů.

V takto malé firmě je management pouze pár lidí, majitel, obchodní zástupce a projekt manager.

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. Všichni zúčastněný by měli vědět o cílech projektu.

Většina projektů v IT je specifická. Ani ne tak svým rozsahem nebo pracností, ale potřebou evidovat čas. Většina lidí, potažmo firem, nemá pouze jednu zakázku, ale více. Proto vzniká nutnost evidovat pracnost jednotlivých úkolů, projektů.

Některé 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. Firma pak přidělí svého zaměstnance. Takový člověk se pak může po zbylé tři dny věnovat jinému projektu. Důležité je pro management 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 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 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 velmi obtížné sehnat odborníky na systém SAP, který využívá velké množství firem. Celý podnikatelský plán je získávat projekty pro své zaměstnance. 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. Tyto úpravy se nedějí každý den a jsou spíše nárazové, proto firmy dávají přednost outsourcingu před zaměstnáváním vlastních specialistů.

Projekt management je v těchto případech zredukován na evidenci hodin.

Kapitola 2

Popis a srovnání vykazovacích systému

V této části textu se budu zabývat srovnáním existujících nástrojů. Některé z uvedených programů jsou přímo určeny pro vykazování času. Ostatní služby jsou spíše nástroje na projektový management nebo celofiremní informační systém. Následující seznam není úplný, přesto může obsahovat zajímavé nástroje, které mohou splňovat potřeby malé firmy.

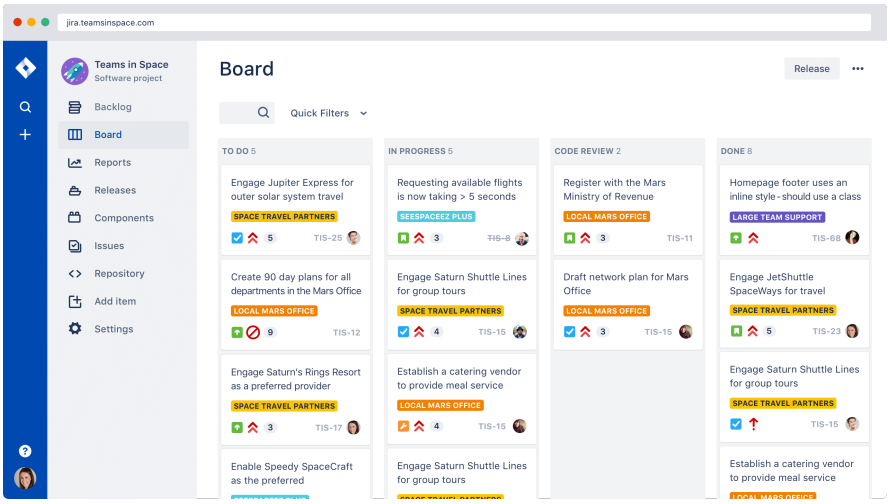
2.1 Co by měl všechno umět

Pro účely modelové firmy musí aplikace naplňovat základní funkcionalitu popsanou níže:

- Evidence zaměstnanců.
- Evidence různých projektů.
- Evidence lidí na projektu.
- Schopnost vykazovat čas na přiřazené projekty.
- Bezpečnost

Očekávám, že každý nástroj bude splňovat první požadavek. Každý zaměstnance musí mít vlastní přihlašovací údaje, aby mohl systém využívat. U uživatelů by se mělo evidovat několik základních atributů jako je např. jméno a příjmení, datum narození, fotografie nebo kontaktní informace, tedy ideálně data pro backoffice případně mzdové informace. Taková evidence pak může sloužit i jako jednoduchý adresář zaměstnanců.

Evidence projektů znamená být schopen zobrazit všechny projekty, kterých se firma účastní. Ideálně ješ projekt vlastní objekt, který může agregovat doplňující informace.



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

Další požadavek říká, že je nutné pracovat s přiřazením zaměstnanců k projektu, jinak je aplikace pouhá evidence projektů. Na jednom projektu pracuje více zaměstnanců a naopak, zaměstnanci mohou být přiřazeni na více různých projektů. Aplikace musí být schopná zobrazit zaměstnanci všechny jeho aktivní projekty a managementu zobrazit projekt s přiřazenými zaměstnanci.

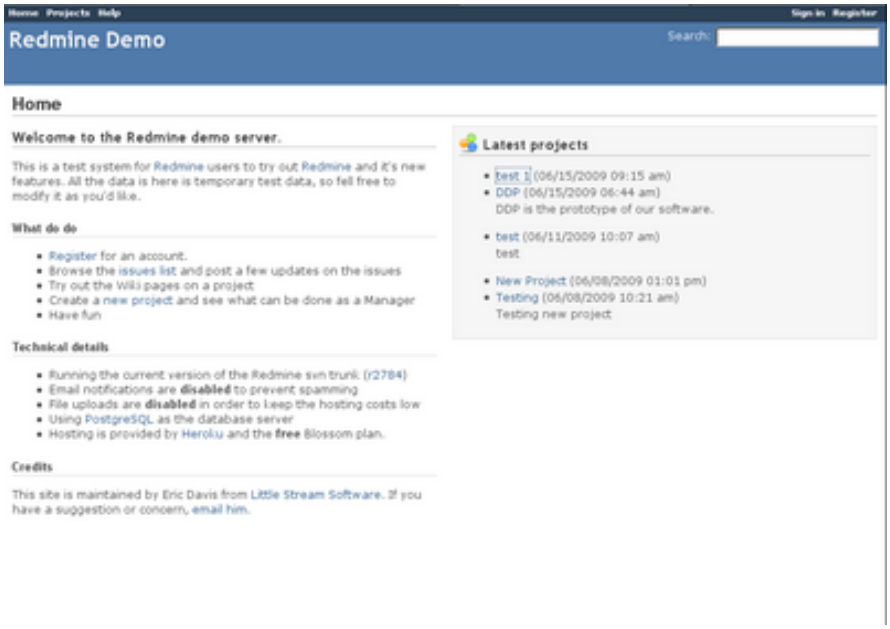
Poslední požadavek je schopnost vykázat odpracovaný čas. Výkaz musí obsahovat informace o datu a počtu odpracovaných hodin. Aplikace by měla s těmito hodnotami pracovat a ukazovat počet vykázaných hodin zaměstnanců na jednotlivých projektech v shrnutí užitečném zaměstnanci nebo managementu. Takový report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

2.1.1 Jira

Jira[4] [5] je proprietární nástroj pro agilní softwarový vývoj. Jira je naprogramována v Javě a je dostupná přes webový prohlížeč nebo mobilní aplikaci pro iOS a Android.

Jira je nástroj pro správu projektů. Základní kamenem jsou úkoly (tasks), které obsahují množství různých políček jako je popis, poznámky, tagy, komentáře nebo důležitost. Úkolům lze dále přiřadit osoby, které se issue zajímají. Volitelně se určuje fáze, ve které se úkol nachází (in progress, testing, done, atd.). Zaměstnanci vykazují čas na tyto úkoly. Výkazy se pak tvoří na základě těchto logů.

Jira exceluje především v podpoře agilního vývoje softwaru. Jednotlivé tasky se přiřazují do sprintů, týmy mají možnost vytvářet dashboardy s vlastním workflow. Issue se může sdružovat do sprintů, dashboardů nabo backlodů.



Obrázek 2.2: Ukázka domovské stránky software Redmine.

Bohužel 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 $za měsíc.

2.1.2 Redmine

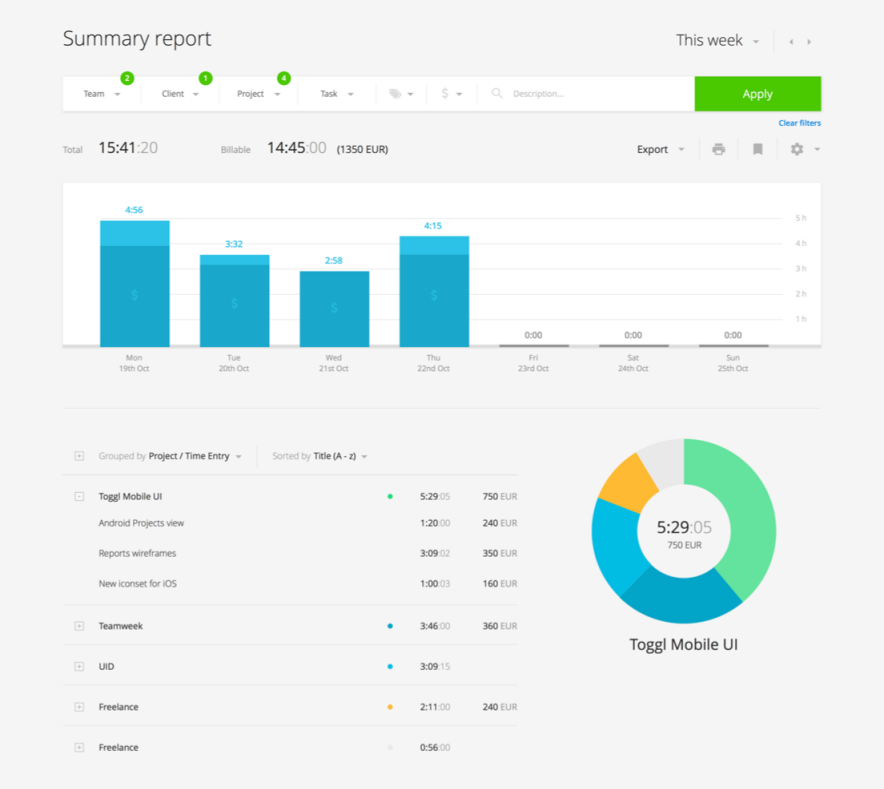
Redmine je webový nástroj pro projektový management a issue tracking systém. Uživatelů umožňuje vytvářet projekty nebo vykazovat čas. Součástí Redmine je vlastní wiki, který může být využit pro sdílení důležitých informací pro celou firmu nebo pouze pro určitý projekt.

Redmine se podobá softwaru Jira, ale nemá takovou podporu v agilní metodice.

Webová aplikace je napsána v Ruby on Rails a licencována v GNU. Základní verze je tedy zdarma, ale lze nakoupit některé pokročilé plug-iny.

2.1.3 Bugzilla

Bugzilla je serverové řešení původně vyvinuté pro sledování chyb (bug tracking), ale lze ho využít i pro projektový management.



Obrázek 2.3: Ukázka výkazu ze software Toggl.

2.1.4 Toggl

Toggl 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, údaj, ke kterému můžeme přistupovat jako ke výkazu, timesheetu. 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. Rozhraní stopek je uživatelsky přívětivé.

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

Kapitola 3

Analýza

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

Při návrhu funkčních a nefunkčních požadavků vycházím ze zadání diplomové práce.

3.1 Popis aplikace

Popisovaná aplikací je vykazovací systém a jejím hlavním přínosem je usnadnit evidence výkazů práce na různých projektech. V každém vykazovacím systému budu zkoušet jednoduché úkony, abych zjistil, jestli testovaný software vyhovuje našim use case.

3.2 Požadavky na aplikaci

3.3 Procesy

V této kapitole jsou popsány procesy, které bude aplikace podporovat. Převážně se jedná o vykazování, správu uživatelů a tvoření reportu. Reportem myslím zprávu o projektu nebo zaměstnanci, která bude především obsahovat počet vykázaných hodin.

Následuje výčet jednotlivých procesů:

1. Evidence odpracovaných hodin zaměstnanců na jednotlivé projekty.
2. Report vykázaných hodin na projetku.
3. Report vykázaných hodin zaměstnance.

3.3.1 Vykázání času na projektu

Člověk, který je již přiřazený na projekt, musí mít možnost vykázání času na projekt. Uživatel musí zadat datum a čas, který na projektu strávil. Další informace, např. typ práce nebo krátká poznámka, jsou vítané, ale nejsou nezbytné.

3.4 Funkční požadavky

Funkční požadavky specifikují konkrétní funkce aplikace.

1. Jednoduchá evidence zaměstnanců a jejich dovedností.
2. Jednoduchá evidence projektů.
3. Přiřazení zaměstnance na jeden nebo více projekt.
4. Vykazování odpracovaných hodin.
5. Jednoduché plánování vytíženost zaměstnanců.
6. Autentizace uživatele systému.
7. Rozlišení uživatelských rolí.

3.4.1 Založení osoby/projektu

Uživatel bude moci založit novou osobu/projekt do systému. Ideálně by nemělo být možné zakládat bez administrátorských práv. Dále je v tomto use casu vyžadována úprava položky (nemělo by být možné upravovat vše, ale naopak uživatel by mohl změnit informace o položkách, které mu patří). Posledním úkonem, který by měl systém zvládnout, je smazání již vytvořené položky. CRUD

3.4.2 Přiřazení osoby k projektu

Jedním ze základních požadavků je také možnost přiřazení lidí na projekt. Položka osoba a projekt by již měly být přítomny v systému. Dalším požadavkem je, aby bylo možné přiřadit více různých lidí na stejný projekt nebo stejného člověka na různé projekty. Opět by se mělo rozlišit, kdo je oprávněný tento úkon udělat.

3.4.3 Vykazování na projekt

3.5 Nefunkční požadavky

Nefunkční požadavky specifikují vlastnosti a omezující podmínky aplikace.

1. Aplikace podporuje rozhraní REST API.
2. Podpora více uživatelů
3. Bezpečnost

3.6 Mapování procesů na funkční požadavky - případy použití

změnit název případně odstranit. jak používá aplikaci běžný uživatel, maanger, administrátor

3.7 Uživatelské role

Aplikace rozlišuje několik rolí v systému: uživatel, manager a administrátor. Nepřihlášený uživatel nemá přístup do aplikace a nesmí zobrazovat ani měnit data v aplikaci.

3.7.1 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 opravovat vlastní výkazy. Dále je uživateli umožněno zobrazit svůj výkaz za celý měsíc, kde bude vidět, kolik hodin vykázal.

1. Správa vlastních údajů.
2. Vykazování.
3. Report za měsíc.

3.7.2 Manager

Management má práva k dalším činnostem: správa projektů, přiřazení osoby k projektu, zobrazení reportů a filtrovat zaměstnance podle kompetencí. Dále může mít manager přístup do stejných částí jako uživatel.

Tato role je zodpovědná za správu projektů.

1. Vytvořit projekt.
2. Přiřadit osobu k projektu.
3. Zobrazit report projektu.
4. Získat doporučení na základě kompetencí.

3.7.3 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.

1. Přiřadit role.

Kapitola 4

Návrh

Cílem je navrhnout a implementovat REST API. Při návrhu aplikace jsem vycházel z funkčních a nefunkčních požadavků.

4.1 Návrh API

Z nefunkčních požadavků je určeno použitý REST API jako rozhraní aplikace. REST je zkratkou pro anglické Representation State Transfer.

4.1.1 Pricipy RESTu

REST navrhl a popsal Roy Riedling ve své dizertační práce v roce 2000.

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.

4.1.2 OpenAPI

OpenAPI (dříve označovaný jako Swagger) označuje několik podobných věcí. Za prvé je to doporučení jak navrhovat přehledné a funkční REST rozhraní, ve kterém je méně chyb. 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 querry parametry, zabezpečení nebo popis HTTP chybových kódů. Většinu ze zmíněných věcí lze okomentovat, a tak zjednodušit prací programátorům, kteří využívají popsané rozhraní.

Za druhé je to sada nástrojů, která umožňuje generovat dokumentaci do technologie HTML spustitelnou ve webovém prohlížeči. Dále se využívá ke

generování kódu především pro data transfer třídy (DTO) nebo třídy, které obsluhují jednotlivé požadavky. Generátor existuje 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.

4.2 Návrh vstupů a výstupů

4.3 Návrh zabezpečení a rolí v systému

do zabezpečení patří i logování v aplikaci, kdz víme, co kdo kdy dělal

Kapitola 5

Implementace a technologie

V této kapitole bych se rád věnoval vybraným technologiím, které lze využít pro vývoj webové aplikace. Ne nutně jsou všechny použity, protože se nutně nemusejí hodit pro tento projekt. To ale neznamená, že se musí takto rozhodnout každý.

Tento seznam není kompletní a soustřeďuje se především na technologie spojené s jazykem Java. Tento směr jsem si vybral, protože je vhodný pro vývoj REST API a mám v něm největší zkušenosti.

5.1 Použité technologie

5.1.1 Java

Java je programovací jazyk, který vznikl v 90. letech. V současné obě je to jeden z nejpopulárnějších jazyků pro vývoj programů, např. je to druhý nejpoužívanější programovací jazyk na githubu [**?**]. Na rozdíl od ostatních programovacích jazyků, které jsou navrženy, aby se zkompilovaly do strojového kódu, nebo byly intepretovány ze zdrojového kódu během v runtime, 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.

5.1.1.1 Spring Framework

Framework 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 dodat sám.

Spring Framework je jedním z nejdůležitějších a nejrozšířenějších Java frameworků. V jeho první verzi umožňoval pouze funkci depency injection, dnes je platforma pro všechny Java aplikace, od malých projektů po aplikace pro operační systémy až pro webové aplikace. Spring je sestaven z modulů, které poskytují různou funkcionalitu např. jako Aspect-oriented programming, Data access, Transaction management, Model–view–controller, Authentication and authorization, Messaging, and Testing. Všechny projekty lze nalézt na spring.io/projects.

5.1.1.2 Spring boot

Spring Boot je projekt pro usnadnění a urychlení vývoje spring aplikací. Výhoda pro vývojáře je, že se nemusí starat o verze knihoven a částečně je naprogramovaná konfigurace, která je u všech projektů stejná.

convection over configuration okomentovat

5.1.2 Databáze

Databáze je nedílnou součástí většiny moderních aplikací. Slouží především k perzistentnímu uložení dat.

V dnešní době si programátoři mohou vybrat mezi dvěma technologie databází:

- relační (SQL) databáze,
- NoSQL databáze.

Mezi SQL a NoSQL databází je několik rozdílů.

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

5.1.3 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¡T,ID¿. 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.

Další výhoda, která ulehčuje vývoj, se jmenuje

5.1.4 Maven

Maven je nástroj pro správu a automatizaci sestavení (tzv. buildu) softwaru. První verze byla vydána 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, které chceme využít. Jeho hlavní konfigurační XML soubor se nachází v kořenovém adresáři aplikace a popisuje jednotlivé operace, které jsou nad aplikací prováděny.

Alternativou pro Maven Gradle nebo Ant.

5.1.5 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.

5.2 Dokumentace, javadoc, swagger

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 neumožnit. Spring framework pracuje s modulem Security, který poskytuje nástroje pro autentizaci a autorizaci.

```
@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 5.1: Implementace rozhraní UserDetailsService v Spring Security.

5.3.1 Autentizace

Nejprve jsem 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 serverové 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.

Poté jsem musel vytvořit a nakonfigurovat úložiště s uživateli, které obsahuje uživatelská jména, hesla a role v systému. Toho jsem docílil vytvořením třídy, která splňuje rozhraní UserDetailsService s metodou loadUserByUsername5.1, 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.

```
@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 5.2: Ukázka použití anotace @PreAuthorize v Spring Security.

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

Kapitola 6

Testování

Testování je v dnešní době důležitou částí vývoje. Každý složitější softwarový produkt může obsahovat chyb, které lze odstranit správným testováním, a zajistit co nejvyšší kvalitu produktu.

Chyby je nutné odhalit co nejdříve, protože s pozdější fází vývoje stoupá cena opravy chyby.

Různé technologie byly použity pro testy různých částí softwaru. Pro unitové, česky jednotkové, testování byl použit JUnit framework. Pro testování výstupních objektů byla použita třída MockMvc.

K testování softwaru jako celku byl využit Postman, který je volně dostupný nástroj na posílání HTTP requestů.

Kapitola 7

Nasazení

7.1 Docker

Docker je open-source nástroj, umožňující vývoj a distribuci software v balíčku. 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

V rámci této práce jsem vytvořil analýzu návratnosti.

Nejprve musím odhadnout náklady na vývoj aplikace. Dále se chci zaměřit na výpočet návratnosti investice.

Je důležité zmínit, že pro plnou funkčnost aplikace je zapotřebí dopracovat frontendovou část, která se stará o uživatelskou přívětivost a zobrazování dat. Její pracnost odhaduji o něco více než jakou má backendová část.

Do výsledného srovnání bych měl započítat i náklady na hosting, případně na cenu domény nebo bezpečnostních certifikátů. Nakonec jsem se rozhodl tyto náklady zanedbat, protože jejich velikost je ve srovnání s náklady na vývoj zanedbatelná.

8.1 Odhad ceny aplikace

Popisovanou aplikace jsem v návrhu rozdělil na dvě části: backend a fronted. Z implementace backendové části programu mám přibližný odhad pracnosti. Vývojem jsem strávil přibližně hodin. Aplikace ale nepovažuji za plně hotovou, protože se můžou objevit nové požadavky nebo chyby, které jsou potřeba dokončit. Další částí, kterou chci započítat, je podpora frontendového vývoje. Kompletní dokončení backendu může trvat ještě další měsíc.

Frontendová část je vyznačuje vyšší pracností zejména kvůli tvorbě uživatelského prostředí. Pro pokrytí co největšího počtu uživatelů je nejpravděpodobnější vývoj webové aplikace. Doba vývoje může trvat přibližně 4-5 měsíců.

Cena programátora je v rozsahu od 4000 do 4500 podle zkušenosti za jeden den. Vývoj backendu trvá přibližně 3-4 měsíce, tedy 60-80 MD. Náklady na celou backendovou část jsou 240 000 až 320 000 Kč při platu 4000 pro jednoho programátora. 270 000 až 360 000 při platu 4500 za den. Náklady na frontend počítám stejně.

Celková odhadnutá cena je 560 000 při optimistickým odhadu.

Roční výnos investice počítám jako ušetřené náklady na nákup obdobné aplikace - 50 zaměstnanců je 500 dolarů na měsíc, 6 000 dolarů na rok je 136 271,70 Kč. Roční úspora je přibližně 130 000 Kč.

8.2 Doba návratnosti

Doba (počet let), za kterou peněžní příjmy z investice vyrovnají počáteční kapitálový výdaj na investici.

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

kde IN je celková investice a CF je roční peněžní tok. Při optimistickém odhadu to je:

$$\frac{560000}{130000} \cong 4,3 \tag{8.2}$$

8.3 Návratnost investice

To je v případě optimistického odhadu po pěti letech:

$$\frac{650000 - 560000}{560000} \approx 16\% \tag{8.3}$$

Kapitola 9

Zefektivnění manažerských procesů

Kapitola 10

Možná další vylepšení

- vylepšení zabezpečení
- export reportů
- plánování/report po dnech

Závěr

Doplňte závěr.

Literatura

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

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

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

[4] Atlassian: Jira Software. Dostupné z: `https://www.atlassian.com/software/jira`

[5] Atlassian: Jira Software. `https://www.atlassian.com/software/jira`.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

ERP Enterprise Resource Planning

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
exe ..... adresář se spustitelnou formou implementace
src
    impl ..... zdrojové kódy implementace
    thesis ..... zdrojová forma práce ve formátu LaTeX
text ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
```