FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Refaktoring a úprava aplikace pro virtualizaci WBS ze systému Youtrack
Student:	Michal Duba
Vedoucí:	Ing. Vlastimil Jinoch
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

1. Seznamte se s původní aplikací.
2. Proveďte optimalizaci původní aplikace zavedením více úrovňového cachování se systematickou invalidací záznamů (to umožní přírůstkové načítání dat z Youtracku, které povede ke snížení zátěže daného systému).
3. Proveďte refaktoring původní aplikace.
4. Vytvořte rest API umožňující vytvoření samostatného frontendu.
5. Dodejte testy pokrývající původní i nově dodané funkcionality.
6. Zaveďte verzování aplikace.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 11. prosince 2020 v Praze.

Bakalářská práce

REFAKTORING A ÚPRAVA APLIKACE PRO VIRTUALIZACI WBS ZE SYSTÉMU YOUTRACK

Michal Duba

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: Ing. Vlastimil Jinoch
6. ledna 2022

Poděkování

Touto cestou bych rád poděkoval Ing. Vlastimilu Jinochovi za ochotu, trpělivost a cenné rady při vedení mé bakalářské práce. Současně bych také rád poděkoval své rodině a přátelům za podporu a pomoc při studiích.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů připřípravě 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. Dáleprohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základěse ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 10. května 2021 .....

Abstrakt

Tato bakalářská práce upravuje, rozšiřuje a optimalizuje existující aplikaci virtualizující WBS ze systému Youtracků. Nejdůležitějším přínosem práce je úspora času uživatelům při využívání této aplikace a možnost aplikaci do budoucna bezpečně a jednoduše rozšířit.

Klíčová slova WBS, Youtrack, víceúrovňové cachování, refaktoring, optimalizace

Abstract

This bachelor thesis modifies, extends and optimizes an existing application virtualizing WBS from Youtracks. The most important benefit of the work is saving users time when using this application and the ability to expand the application safely and easily in the future.

Keywords WBS, Youtrack, multilevel cache, refactoring, optimization

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1
Úvod

Zavedení projektového řízení je v současnosti velmi důležité pro efektivní a úspěšné dokončení většiny různých projektů. Jednou z výhod zavedení projektového řízení je možnost efektivně sledovat postup a náklady k dokončení projektu především vedoucím projektu. K tomu slouží mimo jiné technika WBS, jejímž cílem je rozpad projektu na dílčí činnosti u kterých se pak vedoucímu projektu snadněji odhaduje pracnost, náklady a umožňuje k jednotlivým úkolům přiřazovat členy projektového týmu.
Tato práce bude mít za úkol upravit, rozšířit a zoptimalizovat již existující aplikaci, která vytváří WBS (Work Breakdown Structure) ze systému Youtrack, sloužící k evidenci úkolů, čímž ušetří čas a usnadní práci zejména vedoucímu projektu a jiným uživatelům aplikace. Současná aplikace získává data ze systému Youtrack neefektivně. Zobrazení každé sebemenší změny úkolů, vede ke zbytečně velké zátěži jak aplikace, tak systému youtrack. S touto zátěží je pak úzce spjatá rychlost aplikace. Při libovolné změně v systému Youtrack pak uživatel čeká na zobrazení změn i několik minut v závislosti na počtu projektů a jejich úkolů. Z tohoto důvodu uživatelé aplikace často pracovali s neaktuálním stavem projektu, aby se vyhnuli dlouhému čekání aplikace na zapracování změn ze systému Youtrack a aplikaci aktualizovali, až když to bylo nezbytně nutné. Jedním z hlavních přínosu optimalizace je, že aplikace bude vždy ukazovat aktuální data a tedy již nebude umožněno uživatelům pracovat nad daty neaktuálními.
Sledování stavu projektu je každodenní činnost projektového vedoucího, kterou je potřeba dělat efektivně a proto jsem se rozhodnul pro toto téma.
V práci se budeme dále věnovat úpravě aplikace podle pravidel softwarového inženýrství a řádnému testování, jelikož původní aplikace žádné testy neobsahovala. To povede z dlouhodobého hlediska k lepší údržbě aplikace a jejímu potenciálnímu rozšíření.
Práce je rozdělena na tři části - teoretickou, analýzu a praktickou část. V teoretické části této práce jsou blíže popsány základní koncepty týkající se dané problematiky, například objasnění fungování systému Youtrack. V druhé části, respektive v analýze dochází ke zkoumání původní aplikace a jejich nedostatků. Na základě toho je v práci prezentováno fungování původní aplikace a zvýraznění jejích problematických částí. Rovněž jsou zde definovány hypotézy, které se práce pokusí verifikovat či falzifikovat. V poslední části, části praktické, je rozvedeno řešení zjištěných problému z analytické části. Současně se zde nachází popis fungování aplikace po zavedení patřičných úprav.

Kapitola 2
Cíle práce

Klíčovým cílem práce je seznámit se s původní aplikací k detailnějšímu porozumění jejího fungování, díky kterému dochází k optimálnějšímu zpracování níže zmíněných cílů práce.

Nejvýznamnějším cílem této bakalářské práce je provést optimalizaci v podobě zrychlení fungování původní aplikace zavedením víceúrovňového cachování, a tím ušetřit čas používání aplikace koncových uživatelů.

Práce se následně věnuje dílčím cílům, kterými jsou:

- Refaktoring aplikace.
- Dodání testů pokrývající původní i nově dodané funkcionality.
- Vytvoření REST API.
- Zavedení verzování.

Kapitola 3
Teoretická část

V této části bakalářské práce jsou blíže popsány základní teoretické pojmy pro porozumění zkoumaného tématu. Jsou zde podrobněji vysvětleny kapitoly týkající se hlavního cíle práce - optimalizace aplikace. Například seznámení se s oborem diskrétní matematiky teorie grafů a jejich pojmů, či systémem Youtrack. Okrajově jsou definovány koncepty dílčích cílů. Například verzování či testování.

3.1 Projektové řízení

Projektové řízení představuje proces vedení práce určitého celku, aby dosáhl splnění požadovaných cílů. Postup v projektovém řízení je rozdělen do samostatných po sobě jdoucích kroků. V první řadě je zapotřebí jasně definovat požadavek na projekt, který by měl přesně znázorňovat cíle, kterých se daný projekt snaží dosáhnout. Společně s tím je vhodné si stanovit kritéria, rozsah, dostupné zdroje a například i časovou dotaci vynaloženou na projekt. Po schválení daného projektu zodpovědnými orgány dochází k plánování projektu společně s rozvržením práce v rámci výzkumného týmu a jednotlivých aktivit tak, aby docházelo k vhodné adaptaci projektového plánu na skupinu. Dále je z pohledu projektového řízení velice důležitá konstantní podpora a vedení týmu v průběhu práce na projektu. Na základě této fáze dochází k vyhodnocování vývoje projektu a případným úpravám tak, aby byly splněny původně stanovené požadavky. Po detailní kontrole je projekt ve finální fázi, kdy dochází k jeho hodnocení a následné realizaci. [1]

3.1.1 WBS

WBS (Work Breakdown Structure) představuje mimo jiné součet vedlejších nákladů na úkoly, materiál a další nezbytné prvky v průběhu vývoje projektu. V souvislosti s projektovým řízením představuje tato struktura proces rozdělování projektu na menší komponenty. Jedná se tedy o klíčový prvek při efektivní organizaci a rozdělování příslušné práce daného týmu na jednotlivé lépe zvladatelné úseky. WBS současně poskytuje nezbytný rámec pro detailnější odhad časových nákladů, samotného harmonogramu a dalších komponentů vynaložených na projekt. Tato struktura je obvykle vyvíjena na začátku projektu a předchází jeho podrobnému plánování.[2]

WBS soustředí svou pozornost především na primární produkty a plánované výstupy projektu. Jejím cílem tedy není podílet se na organizaci práce potřebné k výrobě daných produktů. Při vhodném navržení diskutované struktury dochází ke snazšímu přerozdělování práce. Současně dochází k mapování požadavků a hierarchickému rozkladu projektu na jednotlivé fáze, ve kterých je například rozvrženo pracovní úsilí a výkon potřebný k dosažení příslušného cíle. Z pohledu procesu se WBS vyvíjí od konečného cíle a postupně se rozděluje na dílčí komponenty z hlediska

velikosti, pracovní náročnosti, náročnosti určitého systému nebo dalších podobných komponentů. Veškerá práce obsažená ve WBS musí být identifikována, odhadnuta, naplánována a současně i zahrnuta do rozpočtu.[2]

Jak již bylo výše zmíněno, ve struktuře WBS dochází k hierarchickému rozkladu projektu. Toto je ve většině případů rozvrženo v diagramu, který znázorňuje lepší chápání rozsahu práce, která musí být vykonána pro dokončení projektu. [2]

3.2 Teorie Grafů

Teorie grafů je matematická disciplína, která se zabývá vztahy a vlastnostmi struktur, které se nazývají grafy. Graf si lze představit jako množinu objektů a vztahů (vazby) mezi nimi. Jako množinu objektů můžeme například použít množinu křižovatek ve městě a silnice mezi nimi reprezentují jejich vazby. Nyní se již na křižovatky a silnice můžeme dívat, jako na graf na který lze využít známe algoritmy. Například můžeme zjistit nejkratší cestu z jedné křižovatky k druhé, nebo způsob jak navštívíme všechny křižovatky. V následujících kapitolách budou definovány potřebné pojmy z teorie grafů, které se budou v práci vyskytovat, a tedy je nutné je znát.[15]

3.2.1 Graf

▶ **Definice 3.1.** *Graf je uspořádaná dvojice G = (V, E), kde V je množina vrcholů (též uzlů) a E je množina hran – množina (některých) dvouprvkových podmnožin množiny V.[15]*

3.2.2 Podgraf

▶ **Definice 3.2.** *Graf H je podgrafem grafu G, když V(H) ⊆ V(G) a E(H) ⊆ E(G). Tuto skutečnost značíme H ⊆ G.[15]*

3.2.3 Cesta

Cesta je graf, který si můžeme představit, jako posloupnost vrcholů mezi kterými vždy existuje hrana z vrcholu do jeho přímého následníka. Mezi vrcholy existuje právě jedna hrana, a tedy se v této posloupnosti žádné vrcholy neopakují.

▶ **Definice 3.3.** *Nechť $m \geqq 0$.*
Cesta délky m (s m hranami)
Pm je graf ($\{0, \ldots, m\}$, $\{\{i, i + 1\} \mid i \in \{0, \ldots, m - 1\}\}$).
Délka cesty je počet jejích hran.[16]

3.2.4 Kružnice

Kružnice je graf, který si můžeme představit jako uzavřenou cestu. Tedy počáteční a koncový vrchol cesty (grafu) musí být spojené hranou.
Nechť n $\geqq$ 3. Kružnice délky n (s n vrcholy) Cn je graf
($\{1, \ldots, n\}$, $\{\{i, i + 1\} \mid i \in \{1, \ldots, n - 1\}\} \cup \{\{1, n\}\}$)[16]

3.2.5 Souvislost grafu

▶ **Definice 3.4.** *Graf G je souvislý, jestliže v něm pro každé jeho dva vrcholy u, v existuje u-v-cesta. Jinak je G nesouvislý.[16]*

3.2.6 Strom

▶ **Definice 3.5.** *Graf G nazveme stromem, pokud je souvislý a neobsahuje žádnou kružnici (čili je acyklický).[16]*

3.2.7 Zakořeněný strom

▶ **Definice 3.6.** *Zakořeněný strom je strom T, ve kterém je jeden vrchol $r \in V(T)$ označen jako kořen. Leží-li u na (jediné) cestě z v do kořene, pak u je předek v a v je potomek u. Pokud je navíc $\{u, v\} \in E(T)$ hrana, říkáme, že u je otec v a v je syn u. Vrcholy rozdělíme podle vzdálenosti od kořene do hladin: v nulté leží kořen, v první jeho synové, atd. Hloubka vrcholu = jeho vzdálenost od kořene (= číslo hladiny). Důsledek Všechny $v \in V(T) - \{r\}$ jsou potomci kořene r a kořen r je předkem všech ostatních vrcholů.[16]*

3.2.8 Podstrom

▶ **Definice 3.7.** *Podgraf stromu, který je také stromem.[16]*

3.2.9 Větev

▶ **Definice 3.8.** *Každá cesta od kořene k listu.[16]*

3.2.10 List

▶ **Definice 3.9.** *Vrchol ve stromu G nazveme listem, pokud nemá žádného potomka.[16]*

3.3 Youtrack

Systém Youtrack je nástroj využívající se pro projektové řízení. Youtrack mimo jiné umožňuje sledovat projekty a úkoly k jejich dokončení. K jednotlivým úkolům může přiřazovat konkrétní členy, odhadovat pracnost a umožňuje i na úkoly vykazovat odpracovaný čas. [3]

3.3.1 Youtrack Issue

Youtrack definuje issue následovně: *„Issue může znamenat různé věci pro různé lidi. Pro váš tým může issue představovat chybu, úkol nebo závadu. Pro jiný tým, issue reprezentuje například konkrétní funkci produktu a úkoly k jeho dokončení. Sledování stavu problémů je běžnou součástí vývojových týmů. Issue prochází různými fázemi svého životní cyklu. Během těchto fází můžeme měřit a předávat informace o našem pokroku."*[4]

V této práci budeme považovat issue za dílčí činnost, která musí být dokončena pro úspěšné zakončení projektu, do jehož dané issue spadá. Po uzavření všech issue pod daným projektem můžeme prohlásit projekt za dokončený. Issue se považuje za uzavřené, pokud je rozhodnuto, že se na něm již dále nebude pracovat. Nejčastěji z důvodů uzavření je, že podstata issue byla vyřešena, například konkrétní chyba už byla opravena. K uzavření issue může dojít i tehdy, když se založené issue z nějakého důvodu nebude řešit. Může se například jednat o přidání funkcionality, kterou zákazník nakonec nebude chtít přidat a tedy se issue uzavírá nevyřešené.

Otevřený stav[5]:

- Submitted (Předloženo)
- Open (Otevřeno)
- To be discussed (Bude projednáno)
- Reopened (Znovu otevřené)

Uzavřený stav[5]:

- Can't Reproduce (Nelze zreprodukovat)
- Duplicate (Duplicitní)
- Fixed (Vyřešený)
- Won't fix (Nebude se řešit)
- Incomplete (Neúplný)
- Obsolete (Zastaralý)
- Verified (Ověřený)

3.3.2 Obecné Atributy

Každý issue má seznam atributů, které jej popisují. Mezi atributy patří například[6]:

- Název issue
- Autor issue
- Datum vytvoření
- Stav issue
- Množství odpracovaného času
- Odhad času potřebného k dokončení issue

Jedny z nejdůležitějších atributů pro projektového vedoucího jsou atributy, které sledují stav issue z pohledu odpracovaného času a jeho původním odhadem na dokončení. V Youtracku se této funkcionalitě říká time tracking a bude více rozebrána v samostatné kapitole. Youtrack mimo jiné umožňuje vytvořit si i vlastní atributy pro potřeby projektu.

3.3.3 Typ issue

Dalším důležitým atributem issue je jeho typ, který nám lépe umožní strukturovat projekt. Mezi základní typy issue patří například[7]:

- Bug (Issue reprezentující chybu v projektu)
- Feature (Issue reprezentující přidání funkcionality do projektu)
- Task (Issue reprezentující obyčejný úkol)
- Epic (Zastřešující issue, jenž je rodičem pro ostatní issue spadající do podobné kategorie, čímž umožňuje dělit projekt na logické celky a tím jej zpřehlednit)

Youtrack také umožňuje definovat si vlastní typy dle potřeby. V interních projektech se například používá:

- Analysis (Issue zabývající se prací na analýze požadavků)
- Change request (Issue reprezentující rodiče všech issue pro daný projekt)

3.3.4 Vazby

Další klíčová vlastnost issue je možnost jeho provázání s jinými issue pomocí vazeb. „V systému Youtrack lze issue vzájemně propojit pro vyznačení vztahu mezi nimi. Když přidáme vazbu z jednoho issue na druhé, pak se automaticky vytvoří i obrácená vazba. Vazby mezi issue jsou tedy oboustranné. Typy vazeb mají vlastnost určující jejich směr. Dle směru pak můžeme vypozorovat, jaký vztah mezi sebou daná issue mají.“ Nejvýznamnější vazby, které budu v práci zmiňovat, jsou vazby[8]:

- Subtask of
- Parent for

Díky možnosti provázání issue můžeme na množinu všech issue pohlížet jako na graf, kde jednotlivé issue jsou vrcholy grafu a vazby mezi nimi reprezentují hrany grafu.

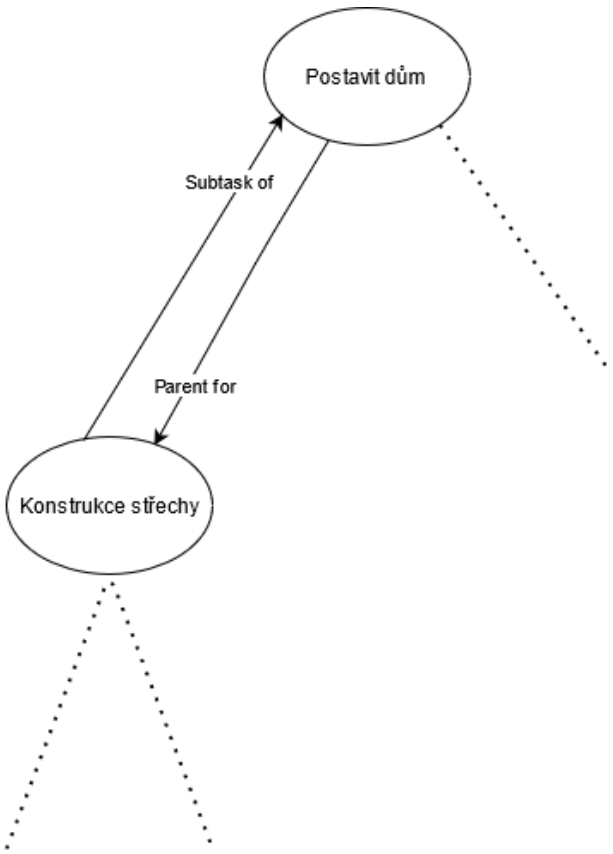
3.3.5 Youtrack projekt

Youtrack projekt reprezentuje kolekci dílčích činností, nazývaných issue, po jejichž uzavření obdržíme předem sjednaný produkt. Dle interních pravidel pro práci na projektu, vždy existuje zastřešující issue, pod který spadají všechny issue pro daný projekt. Z pohledu teorie grafů pak na projekt můžeme koukat jako na zakořeněný strom, kde zastřešující issue reprezentuje kořen stromu a ostatní issue jsou pak jeho potomky. V příkladu projektu se stavbou domu by pak issue Stavba domu reprezentoval již zmíněný kořen stromu a stal by se tak zastřešujícím issue pro tento projekt a neměl žádného rodiče. Vazba Parent for pak bude reprezentovat předka issue a vazba Subtask of jeho potomka.[9]

3.3.6 Time tracking

Funkce time trackingu umožňuje projektovému týmu vykazovat odpracovaný čas na jednotlivá issue v projektu, díky čemuž můžeme sledovat, jak byli jednotlivé úkoly, případně klienti, náročné. Pomocí této funkce můžeme i porovnávat původní odhad pracnosti se skutečným odpracovaným časem.[10]

Obrázek 3.1 Reprezentace vazeb mezi issue



Zdroj: vlastní tvorba.

3.3.7 Work item

Work item slouží k zaznamenávání informace o množství stráveného času na konkrétním issue. Issue může mít k sobě navázáno více work itemů od více různých lidí. Work item může také obsahovat informaci o typu odvedené práce jako je například testování nebo implementace.[11]

Pro využité funkcionality time trackingu je potřeba nakonfigurovat v youtracku atributy, které budou reprezentovat odhad pracnosti a odpracovaný čas. Tyto atributy jsou interně pojmenované Estimation a Spent time a tedy budeme v práci používat zejména tyto názvy.
[11] Spent time je atribut, který pouze uchovává celkový odpracovaný čas na daném issue a dá se navýšit přidáním work itemů (vykázáním pracnosti) na konkrétní issue. Atribut Estimation slouží jako odhad pracnosti pro issue. Tento atribut se může v čase měnit. Například v průběhu práce na issue najdeme jednoduší řešení, se kterým se při odhadování nepočítalo, a tedy můžeme odhad snížit. Stejným způsobem můžeme v průběhu narazit na komplikace a odhad navýšit. Po uzavření issue se atribut estimation automaticky vyplní hodnotou spent time. [11]

Budeme-li používat pouze tyto dva atributy, tak přijdeme o důležitou hodnotu a to je původní odhad pracnosti. Je málo pravděpodobné, aby původní odhad přesně odpovídal realitě, zejména u složitějších problémů. Tento problém je důležitý vyřešit zejména pro projekty typu FTFP (fixed time fixed price), jelikož díky uchování informace o původním odhadu můžeme zjistit, jestli na konkrétním issue netrávíme více času (tedy prodělváme) a projektový vedoucí může kontaktovat člena týmu odpovědného za dané issue a mohou prodiskutovat, proč bylo potřeba pracnost změnit a problém spolu vyřešit. Bez existence atributu původního odhadu pracnosti by

si vedoucí projektu ani nemusel všimnout, že pro dokončení issue bylo potřeba mnohem víc času než původně odhadoval.
Kvůli tomuto problému byl interně přidán atribut Original estimation, který vznikne při vytvoření issue a v čase by se již neměl měnit. Při vzniku issue je hodnota atributů Original estimation a estimation stejná. Po dokončení práce na issue pak lze porovnat přesnost původního odhadu, porovnáním atributů Original estimation a estimation, vedoucího projektu s realitou, což může vést v budoucnu k přesnějším odhadům. Ve zbytku práce budu, pro lepší čitelnost, atributy Spent time, Estimation a Original estimation nazývat hromadně atributy time trackingu.

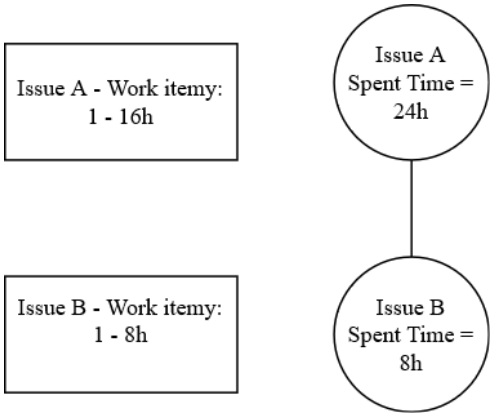
3.3.8 Aktualizace issue

Při práci na projektu často dochází ke změně jednotlivých issue. Ať už je to přidáním komentáře, work itemu nebo změnou hodnot některých z atributů.

3.3.9 Změna atributu spent time

Jelikož issue mohou být provázány, pak přidáním work itemu na jedno issue ovlivní atribut spent time všem jeho předkům. Všem předkům změněného issue se do hodnoty atributu spent time přičte hodnota přidaného work itemu.
[12] Uvedeme si příklad na následující struktuře projektu.

Obrázek 3.2 Výpočet atributu spent time

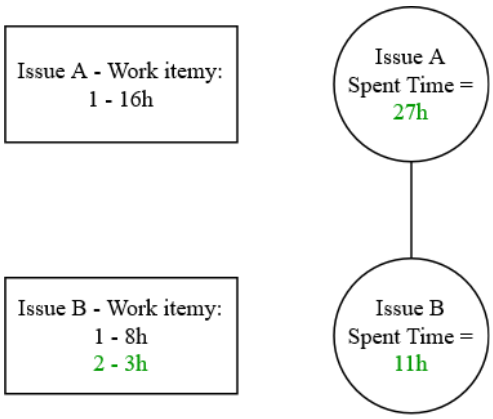


Zdroj: vlastní tvorba.

Z příkladu lze vidět, že hodnota atributu spent time issue A je součtem vlastních work itemů a work itemů jeho potomků (v tomto případě pouze issue B)

V následujícím obrázku si ukážeme, jak se hodnoty změní po přidání work itemu s hodnotou tří hodin do issue B.

Obrázek 3.3 Přidání workitemu k issue



Zdroj: vlastní tvorba.

Po přidání work itemu do issue B se zároveň změnila i hodnota atributu spent time pro issue A. Work itemy issue A zůstávají beze změny.

3.3.10 Změna atributů estimation

Při změně atributu estimation může docházet k problému se synchronizací potomka s předkem. Pokud změníme hodnotu atributu estimation pro rodičovské issue, pak dojde k chybě synchronizace rodiče se synem. [12]

V tomto případě Youtrack označí pouze issue přímo změněné jako aktualizované a jeho předky

neaktualizuje – nemění hodnoty atributu estimation. Toto chování umožňuje, aby hodnota atributu estimation předka byla menší než hodnota jeho potomka, což je nevalidní stav projektu. Na příkladu s projektem zabývající se stavbou domu by pak mohla konstrukce střechy mít větší odhad pracnosti než stavba celého domu, jehož je konstrukce střechy součástí. Z důvodu tohoto chování youtracku aplikace dopočítává hodnoty lokálně, aby předešla nevalidnímu stavu projektu. Podrobnější popis jak aplikace tuto situaci řeší, bude pospán v části analýzy.

3.4 Softwarová cache

Softwarová cache (dále jen cache) je dočasné paměťové úložiště pro data, ke kterým se přistupuje často případně pro data, jejichž způsob získání trvá velmi dlouho.
Například lze cache použít při komunikaci s jinými systémy nebo databázemi, kde si výsledky komunikace uložíme do již zmíněné dočasné paměti a při příštím získáváním těchto dat nemusíme zatěžovat ostatní systémy, ale danou informaci získáme z cache mnohem rychleji.
Zavedení cache nemusí sloužit pouze pro snížení zátěže ostatních systému, ale může pomáhat i lokálně. Představme si, že v aplikaci probíhá velmi složitý výpočet, na jehož konci získáme požadovaná data. Tyto data můžeme následně uložit do paměti cache a můžeme se tak v budoucnu vyhnout opakovanému složitému výpočtu a nahradit jej jednoduchým přístupem do paměti cache.[17]
Jako příklad si můžeme uvést jednoduchou funkci, která z roku narození a současného roku dopočítává věk dané osoby. Po každém volání této funkce by se prováděla operace odečtení roku narození od současného roku. Pomocí zavedení cache bychom si mohl již jednou spočítaný věk uložit do paměti, a tedy při opakovaném dotazu na věk konkrétní osoby bychom se vyhnuli dopočítávání věku.
Důležité je podotknout, že zavedení cache má i svoje záporné stránky, které podrobněji popíšu v jedné z následujících kapitol. Na našem jednoduchém příkladu s věkem osoby je tento nedostatek zřetelný. Obsah paměti cache se nijak neaktualizuje a tedy kdybychom funkci volali po deseti letech, pořád by vracela stejný věk jako při prvním volání a tedy by osoba byla o deset let mladší než ve skutečnosti je.
Příklad funkce si ukážeme níže v pseudokódu:

```
function ZiskejVěkOsoby(DatumNarozeníOsoby, SoučasnýRok)
    if Obsah paměťi cache je prázdný then
        Ulož výsledek rozdílu (SoučasnýRok – DatumNarozeníOsoby) do paměti cache
        return (SoučasnýRok – DatumNarozeníOsoby)
    else
        return Obsah paměti cache
    end if
end function
function Main(...)
    ZískejVěkOsoby(1997,2021)        ▷ Výsledek = 24 - 1x operace odčítání
    ZískejVěkOsoby(1997,2021)        ▷ Výsledek = 24 - 0x operace odčítání
    ZískejVěkOsoby(1997,2022)        ▷ Výsledek = 24 - 0x operace odčítání
end function
```

Při prvním volání funkce proběhne operace odčítání a uložení výsledku do paměti cache. Druhé volání pouze získává obsah paměti cache a neproběhne žádná operace odčítání, a tedy jsme ušetřili čas této operace. V posledním volání funkce (v roce 2022) se opět vrátí obsah cache, která není aktualizovaná a vrací tak starý výsledek. Očekávaný výsledek je 25. Z příkladu je patrné, že je potřeba řešit problém s neaktuálními daty v paměti cache. Zrychlení funkce zavedením cache by nemělo smysl, kdyby získaná data nebyla validní.

3.4.1 Invalidace cache

Se zavedením cache je potřeba zavést i metody k její údržbě. Je nutné rozpoznat, kdy data v cachi již nejsou aktuální a cache invalidovat (vyčistit). Invalidace cache je pouhé odebrání obsahu její paměti a při následném volání funkce je potřeba provést znovu výpočet a nový (aktualizovaný) výsledek opět uložit do cache.[17] V našem příkladu dopočítávání věku osoby bychom si uchovávali v proměnné rok uložení vypočítaných dat (věku) do cache a v případě že by tento rok neodpovídal roku současnému, pak bychom cache invalidovali a uložili do ní nový výsledek. K invalidaci cache nemusí docházet pouze při aktualizování dat, ale i v případě, že dochází paměti aplikace.
Funkce by se tedy pozměnila takto:

---

function ZiskejVěkOsoby(*DatumNarozeníOsoby*, *SoučasnýRok*)

if *RokUloženíDatDoCache* != *SoučasnýRok* **then**
Invaliduj cache

if Obsah paměťi cache je prázdný **then**
Ulož výsledek rozdílu (*SoučasnýRok* – *DatumNarozeníOsoby*) do paměti cache
RokUloženíDatDoCache = *SoučasnýRok*
return (*SoučasnýRok* – *DatumNarozeníOsoby*)
else
return Obsah paměti cache
end if

RokUloženíDatDoCache = 2021

function Main(...)
ZískejVěkOsoby(1997,2021) ▷ Výsledek = 24 - 1x operace odčítání
ZískejVěkOsoby(1997,2021) ▷ Výsledek = 24 - 0x operace odčítání
ZískejVěkOsoby(1997,2022) ▷ Výsledek = 25 - 1x operace odčítání
ZískejVěkOsoby(1997,2022) ▷ Výsledek = 25 - 0x operace odčítání
end function

3.4.2 Nevýhody cache

Jak jsme již zmínili, cachování (tj. použití cache) má i své slabé stránky a v některých případech by bylo její zavedení dokonce na škodu. Například u naší funkce dopočítávající věk se neděje žádná složitá operace, a tedy zavedením cache bychom rozdíl v rychlosti pravděpodobně nepoznali. Co bychom poznali je ovšem zvýšení nároku na paměť aplikace. Představme si seznam i jen tisíců osob, u kterých bychom si museli jednotlivě uchovávat informaci o jejich věku v paměti cache. Při větším počtu osob by aplikaci brzo došla paměť, což paradoxně aplikaci zpomalí.

3.5 Testování softwaru

Testování je proces při kterém ověřujeme správnou funkčnost subjektu jako je například aplikace. Pomocí testování předcházíme výskytu chyb, která mohou mít v budoucnosti závážné dopady. Díky testování můžeme zvýšit výkon aplikace a snížit cenu jejího vývoje, jelikož potenciální chyby mohou být odhaleny při vývoji a tedy náklady na jejich opravu budou značně nižší.[13]

3.5.1 Unit testy

Unit testy (také jednotkové testy) slouží k otestování individuálních jednotek systému. Například může jít o otestování, zda metoda či funkce pracují, jak se očekává. Tento typ testování probíhá ve fázi vývoje a zpravidla jej vytváří samotní vývojáři. Výhoda unit testování je možnost reagovat na vzniklé chyby okamžitě, což významně urychlí jejich opravu.[14]

```
@Test
public void cisloJeSude() {
  Assert.assertEquals(True, isEven(8));
  Assert.assertEquals(False, isEven(7));
}
```

3.5.2 Regresní testy

Tento typ testů slouží k ověření, že nové změny nijak nenarušily původní chování aplikace. Regresní testy pomáhají například při migraci aplikace nebo jejímu složitějšímu refaktoringu, jelikož na konci úprav můžeme ověřit, že aplikace stále funguje, jak se očekává.[13]

3.6 REST API

3.6.1 API

API (Application Programming Interface) je rozhraní které zajišťuje komunikaci a výměnu dat mezi dvěma platformami. Mezi tyto platformy například patří webové a mobilní aplikace.[19]

3.6.2 REST

REST (Representational State Transfer) je architektura pro webové API která pracuje přímo se zdroji platformy. Tyto zdroje musí mít unikátní identifikátor nazývaný URI a můžeme s nimi pracovat pomocí následujících operací[20]:

- Create – Vytvoření zdrojů.
- Retrieve – Získání zdrojů.
- Update – Změnu zdrojů.
- Delete – Smazání zdrojů.

Kapitola 4

Analýza současné verze aplikace

Aplikace vznikla za účelem zobrazení WBS ze systému Youtrack. Jedná se o praktický nástroj projektového vedoucího, který během chvíle může zjistit stavy jednotlivých úkolů nebo celkového projektu. Analýza bude rozdělena do dvou jednotlivých částí. V první řadě proběhne analýza původní aplikace z pohledu samotného fungování aplikace. Následně bude pozornost zaměřená na refaktoring původní aplikace. Původní aplikace nebyla verzovaná a její funkcionalitu nepokrývaly žádné testy. Z tohoto důvodu se těmto tématům v analýze podrobněji věnovat nebudeme.

4.1 Využití konfigurace Youtracku v aplikaci

V kapitole Typ issue jsme uvedli seznam typů, které může issue nabývat. V aplikaci pracujeme s kategoriemi typů. Využívané kategorie jsou

- Task
- Envelope
- Collector

4.1.1 Task (list grafu)

Do první kategorie se řadí typ Task. Jedná se většinou o atomický prvek v roli listu ve stromové struktuře projektu. Pod tímto typem si můžeme představit jednotlivé úkoly pro členy týmu například opravy konkrétních chyb, přidání konkrétní funkcionality do projektu zákazníka apod. Do této kategorie patří zejména typy Task, Bug Feature.

4.1.2 Envelope (zastřešující uzel)

Typ envelope představuje jednoduchý obal pro seskupení více issue spadajících do podobné kategorie. Jedná se převážně o podstrom v daném projektu, kde tento issue kategorie envelope reprezentuje kořen podstromu.

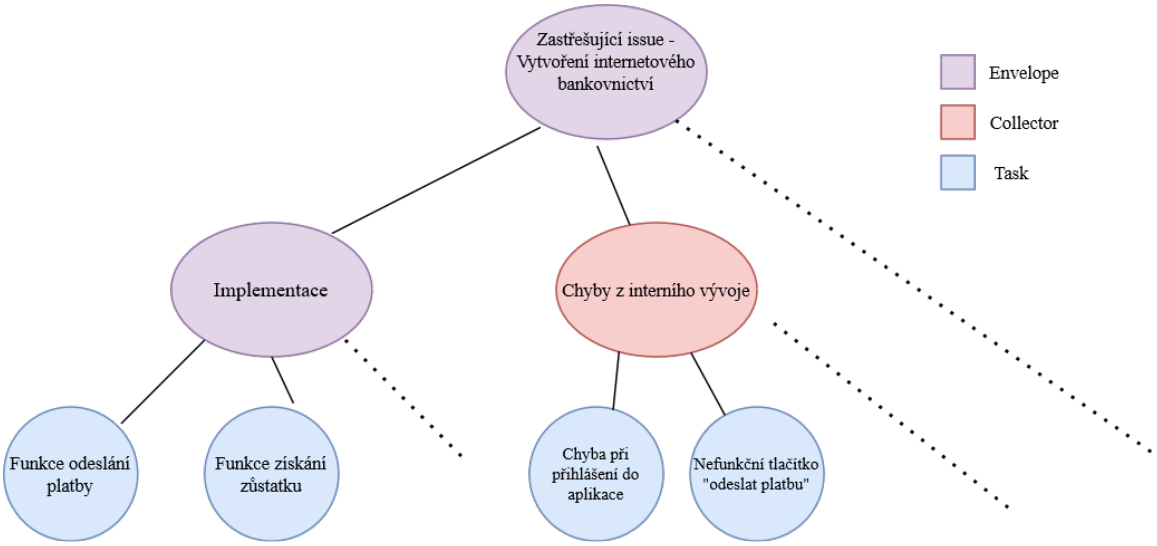
Envelope si můžeme představit jako issue zastřešující celý projekt. Potomci tohoto issue pak můžou být jednotlivé tasky, ale i další issue kategorie envelope. Rozdíl mezi kategorií envelope a taskem je mimo jiné fakt, že na envelope žádný člen týmu nevykazuje žádnou pracnost. Celková pracnost issue této kategorie se dopočítává z jeho potomků. Stejně tomu je i pro atributy estimation a original estimation. Díky zavedení tohoto typu pak vedoucí práce může jednodušeji kontrolovat celkovou situaci jednotlivých komponent projektu (např.: vývoj, testování, opravy

chyb) a nemusí procházet jednotlivé issue daných komponent. Do této kategorie patří pouze typ epic, ale je možné v konfiguraci Youtracku vytvořit vlastní typ issue spadající do této kategorie.

4.1.3 Collector

Do poslední kategorie patří typ collector. Má stejnou strukturu jako Envelope a tedy se jedná o pouhou obálku pro jednotlivé issue. Rozdíl mezi Envelope a collectorem spočívá v tom, že Envelope při jeho vytvoření už zná své potomky, zatímco u collectoru jeho potomky při vzniku neznáme, pouze víme, že v průběhu práce na projektu mu vzniknou. Typický příklad collectoru je Oprava chyb. Při odhadování pracnosti projektu se podle interních pravidel vždy počítá s tím, že při vývoji vzniknou chyby a bude je potřeba opravit, ale konkrétně je neznáme. Podle náročnosti projektu odhadneme potřebný čas pro daný collector. U collectoru stejně jako u Envelope se nevykazuje pracnost, ale oproti Envelope se mu vyplňují atributy estimation a orignial estimation. U jednotlivých potomků collectoru pak následně nezáleží na atributech estimation a original oestimation, protože jsou předem určeny pro collector a tedy se nijak nedopočítávají, tak jak je tomu u Envelope. Dopočítává se pouze atribut spent time potomků, jelikož na collector nelze vykazovat pracnost. Do této kategorie spadá typ collector, ale opět lze tuto kategorii rozšířit.

Obrázek 4.1 Reprezentace množiny issue jako stromu



4.2 Algoritmus aplikace

Fungování aplikace lze rozdělit na dvě hlavní činnosti.

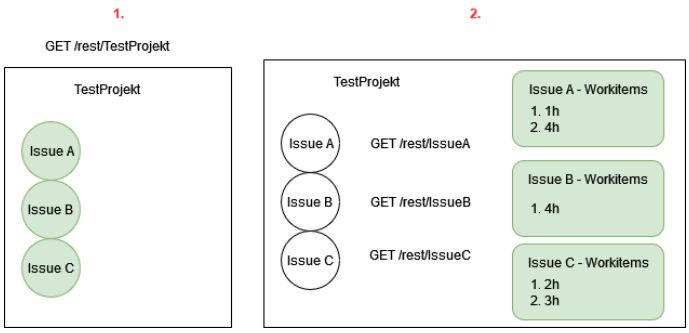
1. Získávání dat ze systému Youtrack
2. Lokální dopočítávání hodnot atributů time trackingu a vizualizace

4.2.1 Získávání dat ze systému Youtrack

Aplikace komunikuje se systémem youtrack pomocí REST API rozhraní. Pro každý projekt (který definujeme v konfiguraci aplikace) se první řadě aplikace dotáže systému Youtrack na

všechna issue, které do projektu patří. Následně se aplikace pro každý získaný issue dotáže na jeho work itemy. V poslední řadě je potřeba získaná issue provázat. Jedá se o jednouchý proces, kde podle vazeb získaných ze systému Youtrack, přiřadíme jednotlivým issue reference na jejich rodiče, případně i potomky. S takto sestavenou kolekcí issue se přesuneme k druhé části aplikace a to k lokálnímu dopočítání atributů time trackingu.

Obrázek 4.2 Získávání dat ze systému Youtrack



Zdroj: vlastní tvorba.

4.2.2 Lokální dopočítávání hodnot atributů timetrackingu

V kapitole Změna atributů estimation jsme popsali, jak youtrack aktualizuje issue a k jakým problémům to vede. Z tohoto důvodu se hodnoty atributů time trackingu dopočítávají lokálně v aplikaci, aby nemohl nastat nevalidní stav projektu. Tedy, i když na straně youtracku dojde k chybě synchronizace mezi předkem a potomkem, tak v aplikaci se správně spočítá hodnota atributů obou issue.

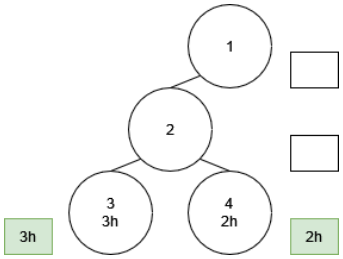
Počítání jednotlivých atributů pro jednotlivé kategorie je odlšiný.

---

function SPOČÍTEJORIGINALESTIMATION
$SumaHodnotAtributůOriginalEstimationVšechPotomků = 0$
for all $Potomek$ **do**SumaHodnotAtributůOriginalEstimationVšechPotomků+ = $spočítejOriginalEstimation(Potomek)$ **end for**

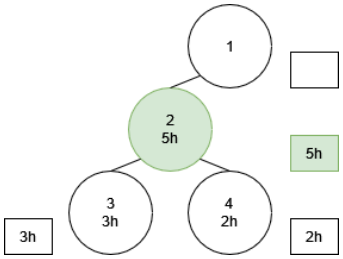
if Kategorie == Envelope **then**
return $SumaHodnotAtributůOriginalEstimationVšechPotomků$
end if
if Kategorie == Collector **then**
return $VlastníHodnotaAtributuOriginalEstimation$
end if
return $SumaHodnotAtributůOriginalEstimationVšechPotomků + VlastníHodnotaAtributuOriginalEstimation$
end function
=0

Obrázek 4.3 Uložení atributů do cache



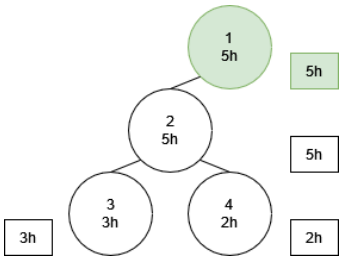
Zdroj: vlastní tvorba.

Obrázek 4.4 Sečtení hodnot potomků - vrchol 2



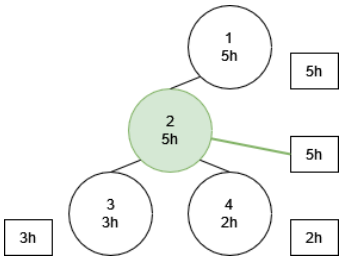
Zdroj: vlastní tvorba.

Obrázek 4.5 Sečtení hodnot potomků - vrchol 1



Zdroj: vlastní tvorba.

Obrázek 4.6 Sečtení vrcholu 2 - uložení hodnoty z cache



Zdroj: vlastní tvorba.

Než začneme dopočítát atributy, tak je potřeba si najít v kolekci issue, získané z předchozí komunikace s youtrackem, všechna kořenová issue. Nad těmito issue pak zvlášť začneme dopočítávat atributy. Dopočítání probíhá rekurzivně, tedy pro dopočítání hodnot jednoho kořenového issue musím prvně dopočítat hodnoty jeho potomků. Operaci dopočítání atributů time trackingu jednoho issue budeme nazývat sečtení issue.

Uvedeme si příklad samotného průchodu stromem projektu se strukturou podle obrázku. Tedy nebereme ohled ani na kategorii issue ani na konkrétní atribut time trackingu issue.

Pro kořen stromu (vrchol 1) je potřeba, aby se dopočítal vrchol 2, pro který je potřeba aby se sečetli vrcholy 3 a 4. Vrcholy 3 a 4 jsou listy v tomto stromu a již nemají potomky, a tak pouze vrátí své hodnoty. Vrchol 2 nyní zná hodnotu svých potomků, které přičte ke svým hodnotám a výsledek pouze předá vrcholu 1, který nyní už zná celkovou hodnotu jeho potomků a tento výsledek přičte ke svým hodnotám a součet uloží. Jak jsem již zmínil, vrchol 2 na rozdíl od vrcholu 1 výsledek pouze předá vrcholu 1, a tedy jej nemá uložený. Je nutné provést dopočítání vrcholu 2 stejně jako u vrcholu 1 a tím se mu hodnoty potomků přičtou k jeho dosavadním hodnotám. Toto dopočítání je pak potřeba provést pro všechna zbylé issue. Lze tady vidět redundantní dopočítávání hodnot pro všechna issue (kromě kořenového issue), protože při dopočítávání hodnot kořene se automaticky dopočítají hodnoty jeho potomků, akorát se neuloží.

Tento problém s nadbytečným dopočítáváním hodnot byl v původní aplikaci vyřešen zavedením globální cache, kde se při dopočítávání jednotlivých issue výsledek uložil do cache (na obrázcích značená čtvercem). Dopočítání hodnot kořene stromu pak probíhalo stejně, až na to, že každé issue si do cache uložilo součet jeho hodnot a hodnot jeho potomků. Funkce dopočítávající hodnoty se sice volá opět na každý issue, ale už se nezanořuje do jeho potomků, ale rovnou vrací hodnoty uložené v cache.

Po zavedení globální cache se pro vrchol 1 průchod nezmění, ale pro dopočítání vrcholu 2 se prvně ptáme, jestli vrchol 2 má v cache uložený součet svých hodnot a hodnot jeho potomků. V případě, že má (z průchodu pro Vrchol 1), pak obsah cache bereme jako výsledek a dále se nezanořujeme a přesouváme se na Vrcholy 3 a 4 kde dopočítání proběhne stejně. V případě, že

by v cachce hodnoty uložené nebyly, tak bychom hodnoty vrcholu 2 museli dopočítat stejným způsobem jako při průchodu pro Vrchol 1, tedy pomocí zanoření se do jeho potomků a teprve až pak bychom se přesunuli na vrcholy 3 a 4.

Dále se globální cache využívá nad získanými issue z youtracku, kde plní jednoduchou funkci kde uloží již získané data do cache a ty se pak zobrazují v aplikaci, aby se znovu nemuseli dotazovat youtracku při obyčejném obnovení stránky, ale pouze na žádost uživatele, když chce zobrazit změny youtracku v aplikaci. To vede k tomu, že hodnoty z youtracku nemusí být v aplikaci vždy aktuální. Nevýhodou zavedení pouze globální cache je, že při jakékoliv změně issue v systému youtrack , kterou chceme zobrazit v aplikaci, musíme celou cache invalidovat a tedy se dotazovat rozhraní youtracku na všechny issue a nejen na ty změněné a opět všem issue dopočítávat hodnoty time trackingu, což při větším množství projektů v systému youtrack může vést i k několika minutovému načítání dat, při změně pouhého jednoho issue. Po načtení všech issue ze systému youtrack a jejich následném dopočítání, se issue v aplikaci stromově vykreslí

4.2.3 Stanovení hypotézy

Na základě porovnávání složitostí původní verze aplikace a zavedením víceúrovňové cache stanovíme hypotézu při průměrné změně 5% issue. Tedy v projektu se 100 issue se obvykle v konkrétním čase mění pouze 5 issue a zbytek není potřeba aktualizovat.
Jelikož oproti původní aplikaci pracujeme pouze nad množinou změněných issue, pak očekáváme, že se aplikace při průměrné změně zrychlí v průměru alespoň 10x, bereme-li v potaz režii cache. Pro potvrzení nebo vyvrácení hypotézy provedeme měření rychlosti původní aplikace nad testovacím projektem, který obsahuje 216 issue a maximální hloubka zanoření je 5. Dále stanovíme, že při přidání dalších 2 projektů s dohromady 400 issue se aplikace při změně stejných issue jako v předchozím testování zrychlí alespoň 20-krát.

4.2.4 První měření:

První měření proběhne pouze nad jedním projektem (označme jej projekt A) s 216 issue.

Počet změněných issue	Původní doba trvání (s)	Očekávaná doba trvání po optimalizaci (s)
1	18	1,5
5	18	1,8
10	18	2,6
20	18	4,1

Tabulka 4.1 Vyhodnocení výsledků optimalizace z prvního měření - Zdroj: Vlastní tvorba.

4.2.5 Druhé měření:

Při druhém měření přidáme do konfigurace další 2 projekty s dohromady 400 issue. Měření proběhne nad stejnými issue jako z prvního měření (Tedy pouze nad projektem A)

Počet změněných issue	Původní doba trvání (s)	Očekávaná doba trvání po optimalizaci (s)
1	55	1,5
5	55	1,8
10	55	2,6
20	55	4,1

Tabulka 4.2 Vyhodnocení výsledků optimalizace z druhého měření - Zdroj: Vlastní tvorba.

4.3 Refaktoring

4.3.1 Opakování proměnných

Pro komunikaci se systémem Youtrack pomocí REST rozhraní je potřeba definovat mj. formát, v jakém systém Youtrack vrací požadované zdroje a případně způsob autentizace. Tyto informace se přes restové rozhraní posílají v hlavičce a jsou potřeba posílat při každém požadavku na zdroje. V původní aplikaci každá metoda komunikující se systémem Youtrack si tuto hlavičku inicializuje zvlášť ve svém těle. V případě, že by byla potřeba hlavička upravit (například změna způsobu autentizace), tak se tato úprava musí provést v každé metodě. V původní aplikace byla hlavička inicializována sedmkrát. Při úpravě hlavičky by se snadno mohlo stát, že zapomeneme upravit jednu metodu ze sedmi a tedy vznikne chyba které by se dalo po jednoduché úpravě kódu v budoucnosti vyhnout.

4.3.2 Extrakce proměnných do konfigurace

Pokud chceme získat seznam všech issue daného projektu ze systému Youtrack pomocí rozhraní REST, pak musíme Youtracku předat parametr název projektu ze kterého chceme daná issue získat. Tento parametr se opět inicializuje v těle metody a jedná se tedy o konstantu. V případě, že budeme chtít změnit v aplikaci projekt z Youtracku, ze kterého budeme získávat data, poté musíme upravit zdrojový kód. V případě časté změny projektu je to velmi nepraktický přístup a v některých případech (například pokud není dostupný zdrojový kód, ale pouze spustitelný soubor aplikace) změnu ani provést nelze.

Obrázek 4.7 Redundantní inicializace proměnných - č.1

```
public Map<String, String> getIssueNames(Set<String> issueIds) {
    String requestIds = String.join(",", issueIds);
    MultiValueMap<String, String> headers = new LinkedMultiValueMap<>();
    headers.add("Accept", "application/json");
    headers.add("Authorization", "Bearer " + youtrackToken);

    try {
```

Zdroj: vlastní tvorba.

Obrázek 4.8 Redundantní inicializace proměnných - č.2

```
public Map<String, YoutrackIssue> readIssueByFilter(String filter) {
    MultiValueMap<String, String> headers = new LinkedMultiValueMap<>();
    headers.add("Accept", "application/json");
    headers.add("Authorization", "Bearer " + youtrackToken);

    Map<String, YoutrackIssue> result = new HashMap<>();

    int max = 500;
```

Zdroj: vlastní tvorba.

4.3.3 Komentáře

Přidáním komentáře ke složitějším částem kódu pomáhá v budoucnu rychlejšímu porozumění daného problému. Vhodné je i komentovat určité celky pro lepší čitelnost programu, například popis třídy a jednotlivých metod.

4.3.4 Ošetření nedefinovaných hodnot

Přidáním komentáře ke složitějším částem kódu pomáhá v budoucnu rychlejšímu porozumění daného problému. Vhodné je i komentovat určité celky pro lepší čitelnost programu, například popis třídy a jednotlivých metod. V aplikaci především chybí komentáře pro jednotlivé metody čímž proces porozumění fungování aplikace je značně náročnější.

Kapitola 5
Praktická část

V praktické části bude předvedeno řešení problémů z části analýzy. V první řadě se tato část bude věnovat verifikování hypotézy ohledně optimalizace aplikace. Následně zde dopodrobna bude popsáno fungování aplikace po optimalizaci. Na konci této kapitoly zde bude předvedeno řešení dílčích cílů práce jako je refaktoring , testování a REST API.

5.1 Optimalizace

Jak už jsem v předchozí kapitole zmínil, při libovolné změně issue ze systému Youtrack se musí načíst všechny data ze systému youtrack pro všechny projekty a následně se všem issuem v projektech musí dopočítat hodnoty atributů timetrackingu. Jako řešení tohoto problému jsem zvolil zavedení více úrovňového cachování. Při změně jednotlivých issue nebude potřeba aktualizovat všechna issue (spolu s tím i invalidovat jejich cache), ale jenom těm dotčených změnou.

5.1.1 Získávání dat ze systému Youtrack

V první řadě popíšu, jak se změní způsob získávání dat ze systému Youtrack. První spuštění aplikace proběhne stejně jako v původní verzi a navíc se do proměnné uloží datum a čas tohoto spuštění. Při změně libovolného issue se aplikace nebude dotazovat Youtracku na všechna issue pro každý projekt, ale dotáže se pouze na seznam změněných issue od data spuštění. Následně pak datum spuštění aktualizuje na datum zaslání dotazu na změněné issue. Takto při další změně v budoucnu získáme issue pouze změněná v té konkrétní změně a ne všechna změněná issue od prvního spuštění aplikace. Způsob načítání work itemů pro seznam změněných issue se nemění a tedy pro každý změněný issue se aplikace dotáže na jeho work itemy a uloží je. Už zde je vidět částečná optimalizace. Při změně jednoho issue není potřeba manipulovat s daty projektů, do kterých změněná issue nepatří. Dále by se zrychlení projevilo u načítání work itemů jelikož množina změněných issue je podmnožinou všech issue, pak je počet dotazů na načtení work itemů ve většině případů menší.

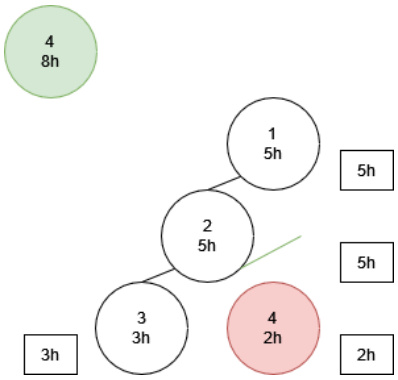
5.1.2 Lokální dopočítávání hodnot atributů time trackingu

V této části je potřeba přidat mezikrok a to proces výměny neaktuálních issue za ty již aktualizované. Výměna se odehraje ve dvou krocích. V prvním kroku je potřeba aktualizovat vazby rodiči a všem přímým potomkům neaktuálního issue na aktualizovaný issue. V tomto momentě původní issue zaniká a nahrazuje ho již aktualizovaný. V druhém kroku pak musí dojít k invali-

daci cache hodnot atributů time trackingu všem předkům aktualizovaného issue a tím zabránit nevalidnímu stavu projektu (Změna atributů estimation).

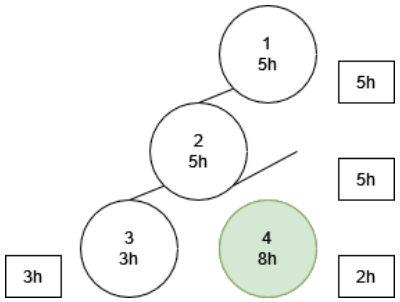
Po invalidaci a výměně všech aktualizovaných issue již dojde k dopočítání hodnot atributů time trackingu stejným způsobem jako v původní aplikaci. V případě, že jsme změnili issue pouze v jednom projektu, pak kořenová issue ostatních projektů nebudou invalidována a jejich dopočtení proběhne v konstantím čase. Ve změněném projektu proběhne dopočítání původním způsobem, akorát při dopočítávání kořene se budou dopočítávat pouze issue které byly změněny anebo předci změněných issue. Ostatním issue se cache neinvalidovala a tedy se dopočítávat nemusí.

Obrázek 5.1 Aktualizace is. č.4 - rozbití vazeb



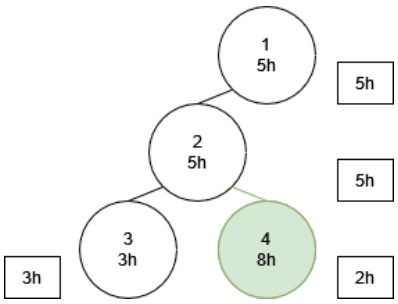
Zdroj: vlastní tvorba.

Obrázek 5.2 Aktualizace is. č.4 - nahrazení issue



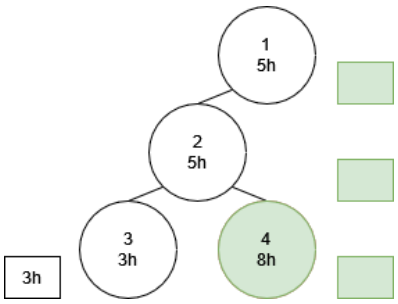
Zdroj: vlastní tvorba.

Obrázek 5.3 Aktualizace is. č.4 - spojení vazeb



Zdroj: vlastní tvorba.

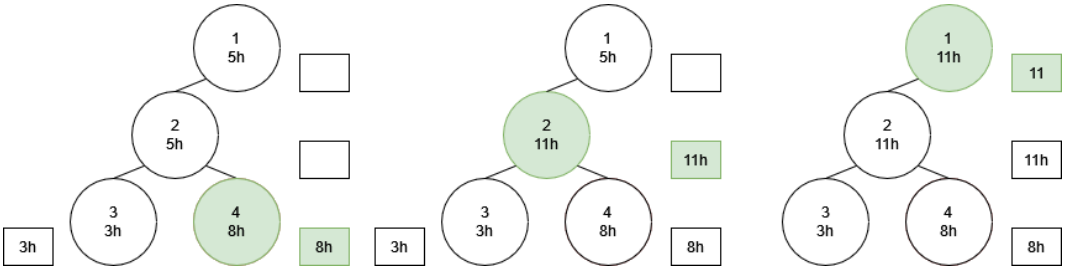
Obrázek 5.4 Aktualizace is. č.4 - invalidace cache



Zdroj: vlastní tvorba.

.

Obrázek 5.5 Aktualizace is. č.4 - dopočítání paměti cache



Zdroj: vlastní tvorba.

5.1.3 Vyhodnocení výsledků optimalizace

5.1.4 První měření:

První měření proběhne pouze nad jedním projektem (označme jej projekt A) s 216 issue.

Počet změněných issue	Očekávaná doba trvání po optimalizaci (s)	Skutečná doba trvání po optimalizaci (s)
1	1,5	1,3
5	1,8	1,6
10	2,6	2,4
20	4,1	3,9

Tabulka 5.1 Výsledky optimalizace 1- první měření - Zdroj: Vlastní tvorba.

5.1.5 Druhé měření:

Při druhém měření přidáme do konfigurace další 2 projekty s dohromady 400 issue. Měření proběhne nad stejnými issue jako z prvního měření (Tedy pouze nad projektem A)

Počet změněných issue	Očekávaná doba trvání po optimalizaci (s)	Skutečná doba trvání po optimalizaci (s)
1	1,5	1,3
5	1,8	1,6
10	2,6	2,4
20	4,1	3,9

Tabulka 5.2 Výsledky optimalizace - druhé měření - Zdroj: Vlastní tvorba.

Z výsledků měření lze prohlásit stanovenou hypotézu za potvrzenou. Nová verze aplikace je tedy mnohonásobně rychlejší. Přidáváním více projektů do aplikace se rychlostní rozdíly verzí aplikací ještě více rozšiřují. U druhého měření můžeme vypozorovat, že přidáním druhého projektu do aplikace se nezmění její rychlost. Muselo by dojít v nově přidaném projektu ke změněn libovolného issue. Tedy nová verze aplikace není závislá na počtu projektů (a jejich issue), ale pouze na jejich změnách.

5.2 REST API

Pro možnost komunikace cizích aplikací s aplikací naší, byly v práci vytvořeny základní metody umožňující komunikaci přes REST rozhraní. Výhodou těchto API oproti API systému Youtrack je kompletnost a korektnost vrácených issue. Narozdíl od Youtracku, se posílají workitemy spolu s daným issue a není potřeba po obdržení issue posílat následující požadavek na jeho workitemy. Nové API zaručují validitu hodnot atributů timetrackingu, jelikož jejich dopočítání probíhá bezpečně na straně aplikace.

Seznam nově vytvořených metod:

- ById - metoda vrací issue na základě jeho unikátního identifikátoru
- ByProject - metoda vrací všechna issue spadající pod daný projekt
- All - metoda vrací seznam všech issue ze všech projektů

5.3 Refaktoring

V kapitole Refaktoring byly uvedeny drobné nedostatky aplikace z pohledu refaktoringu, které v této části práce budou vyřešeny.

5.3.1 Opakování proměnných

Problém opakování proměnné hlavičky v každé metodě komunikující se systémem Youtrack byla vyřešena vytvořením samostatné metody. Ostatní metody využívající této proměnné volají tuto nově vytvořenou funkci a získávají hlavičky v podobě návratové hodnoty.

Obrázek 5.6 Extrakce hlaviček do metody

```
private MultiValueMap<String, String> getHeaders() {
    return CollectionUtils.toMultiValueMap(
            Map.of(
                    "accept", List.of(MediaType.APPLICATION_JSON.toString()),
                    "Authorization", List.of("Bearer " + youtrackToken)
            )
    );
}
```

Zdroj: vlastní tvorba.

5.3.2 Extrakce proměnných do konfigurace

Jelikož je v budoucnosti možné, že bude potřeba měnit projekt nad kterým aplikace pracuje, tak pro zjednodušení zakomponování této změny byla proměnná projektu přesunuta do konfigurace. Nyní uživatel mění pouze konfigurační soubor a nemusí zasahovat do zdrojového kódu aplikace.

5.4 Pokrytí testy

Původní aplikaci nepokrývali žádné testy a tedy ověřit, že aplikace funguje správně bylo obtížné. V rámci této práce přibyly testy, které ověřují funkčnost dopočítávání lokálních atirbutů time-trackingu a práci s projekty v podobě grafů.

Kapitola 6
Závěr

V rámci bakalářské práce byly stanoveny cíle týkající se úpravy a rozšíření aplikace spojené s optimalizací, refaktoringem, testováním a verzováním. Na základě provedené analýzy byly zjištěny nedostatky v aplikaci v podobě neefektitivy při manipulaci s daty ze systému Youtrack. Při sebemenší změně issue nastává problém příliš zdlouhavé aktualizace, při které se zbytečně aktualizují všechna issue, včetně těch nezměněných. Toto zbytečné čekání odrazovalo uživatele od pravidelného aktualizování dat, což vedlo k nebezpečí práce s neaktuálními daty.

Teoretická část vymezuje základní koncepty týkající se sledovaného tématu k pochopení následně podnikutých kroků při analýze dat a příslušných řešení. V této části bakalářské práce došlo k definování pojmů, například Youtrack, WBS, issue, cache atp.

Analytická část práce se zabývala zkoumáním fungování původní aplikace. V rámci této části došlo k zjištění základních nedostatků, které aplikace obsahuje. Díky analýze byla zjištěna významná časová náročnost při práci s původní aplikací. Dochází zde například k měření doby trvání aktualizace projektu při daném počtu změněných issue. Tato doba je pozorována na původní aplikaci a následně porovnávána s očekávanou dobou trvání aktualizace po optimalizaci. Toto měření bylo provedeno ve dvou variantách. Součástí analytické části bylo také zkoumání aplikace a zjišťování nedokonalostí z pohledu refaktoringu.

Praktická část se odvíjí z obou výše zmíněných částí, jak teoretické, tak analytické. Na základě těchto poznatků dochází v aplikaci k optimalizaci zavedením víceúrovňové cache. Současně zde dochází k přesnému vyhodnocení výsledků optimalizace. Naměřené hodnoty jsou porovnávány s očekávanou dobou trvání, která byla stanovena v analytické části. Výsledky vykazují pozitivnější hodnoty v rámci každé testované skupiny.

Díky provedení optimalizace jsou již data využívána v aplikaci vždy aktuální, jelikož optimalizace několiknásobně zrychlila a zefektivnila proces aktualizování. Nyní tedy uživatelům nehrozí nebezpečí práce s neaktuálními daty.

V praktické části se nadále podařilo splnit zbylé dílčí cíle. Bylo zavedeno verzování kódu do systému Git, byly vytvořeny testy ověřující správnou funčknost počítání a jiné manipulace dat. V neposlední řade došlo ke vzniku základních metod pro komunikaci s aplikací přes REST rozhraní a refaktoringu drobných nedostatků.

V rámci potenciálního rozšíření aplikace by bylo vhodné rozšířit RESTové rozhraní o více metod. Následně by se pak mohlo uvažovat i o podrobnějším testování, například zavedením integračních testů. Z pohledu refaktoringu by se v aplikaci mohlo více využít nástroje pro průběžnou

kontrolu kvality kódu, jako je třeba nástroj SonarQube.

Literatura

[1] BURDE, Durgesh, Project Management Process [online]. [cit. 5.1.2022]. Dostupné z: https://ssrn.com/abstract=1284391

[2] ALUTBI, Mohaymen. WORK BREAKDOWN STRUCTURE [online]. [cit. 5.1.2022]. Dostupné z: https://www.researchgate.net/publication/342163727_WORK_BREAKDOWN_STRUCTURE_WBS

[3] JETBRAINS. Introduction to Youtrack [online].[cit.5.1.2022]. Dostupné z: https://www.jetbrains.com/help/youtrack/standalone/YouTrack-Documentation.html

[4] JETBRAINS. YouTrack Classic [online]. [cit. 6.1.2022]. Dostupné z: https://www.jetbrains.com/help/youtrack/standalone/Issues.html

[5] JETBRAINS. State [online]. [cit. 6.1.2022].Dostupný z: https://www.jetbrains.com/help/youtrack/standalone/Search-and-Command-Attributes.html#state

[6] JETBRAINS. Issue [online]. [cit. 4.1.2022]. Dostupné z: https://www.jetbrains.com/help/youtrack/devportal/v1-Issue.html

[7] JETBRAINS. Create and Edit Issues [online]. [cit. 4.1.2022]. Dostupné z: https://www.jetbrains.com/help/youtrack/standalone/Create-and-Edit-Issues.html#create-issue

[8] JETBRAINS. Link Issues [online]. [cit. 5.1.2022]. Dostupné z: https://www.jetbrains.com/help/youtrack/standalone/Link-Issues.html

[9] JETBRAINS. Projects [online]. [cit. 5.1.2022]. Dostupný z: https://www.jetbrains.com/help/youtrack/standalone/Managing-Projects.html

[10] JETBRAINS. Enable Time Tracking [online]. [cit. 5.1.2022]. Dostupný z: https://www.jetbrains.com/help/youtrack/standalone/Time-Management-Tutorial.html#enable-time-tracking

[11] JETBRAINS. Add Work Items To Issues [online]. [cit. 5.1.2022]. Dostupný z: https://www.jetbrains.com/help/youtrack/standalone/Add-Work-Items-to-Issue.html#add-work-items-time-tracking-tab

[12] JETBRAINS. Track Spent Time [online]. [cit. 5.1.2022]. Dostupný z: https://www.jetbrains.com/help/youtrack/standalone/Add-Work-Items-to-Issue.html

[13] IBM. What is software testing? [online]. [cit. 4.1.2022]. Dostupné z: https://www.ibm.com/se-en/topics/software-testing

[14] GURU 99. What is software testing? Definition, Basics Types in Software Engineering [online]. [cit. 4.1.2022]. Dostupné z: https://www.guru99.com/software-testing-introduction-importance.html

[15] KNOP, Dušan a kol. Algoritmy a grafy 1 (BI-AG1), Přednáška č. 1 Základy grafů [online]. [cit. 5.1.2022]. Dostupné z: https://courses.fit.cvut.cz/BI-AG1/media/lectures/bi-ag1-p1-handout.pdf [Soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém CD].

[16] KNOP, Dušan a kol. Algoritmy a grafy 1 (BI-AG1), Přednáška č. 2 Souvislost, DFS, stromy, orientované grafy [online]. [cit. 5.1.2022]. Dostupné z: https://courses.fit.cvut.cz/BI-AG1/media/lectures/bi-ag1-p2-handout.pdf [Soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém CD].

[17] JAVA POINT a kol. Spring Boot Caching [online]. [cit. 4.1.2022]. Dostupné z: https://www.javatpoint.com/spring-boot-caching

[18] MASARYKOVA UNIVERZITA a kol. Cache paměti [online]. [cit. 4.1.2022]. Dostupné z: https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/CACHE.HTML

[19] SANTORO a kol. Web Application Programming Interfaces (APIs): general-purpose standards, terms and European Commission initiatives. Luxembourg: Publications Office of the European Union, 2019, ISBN 978-92-76-13183-0.

[20] REST API. What is REST? [online]. [cit. 6.1.2022]. Dostupný z: https://restfulapi.net/

Příloha A

Seznam použitých zkratek

API Application programming interface
REST Representational state transfer
FTFP Fix time fix price
WBS Work breakdown structure

Příloha B

Obsah přiloženého média

- `readme.txt` ..... Stručné informace ke konfiguraci pro spuštění aplikace
- `jar` ..... adresář se spustitelným jar souborem
- `src`
  - `main` ..... zdrojové kódy implementace
  - `test` ..... zdrojové kódy testů
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF