FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání bakalářské práce

| | |
|---|---|
| **Název:** | Rozšíření aplikace pro podporu tvorby odhadů SW o integrační moduly |
| **Student:** | Ivan Havasi |
| **Vedoucí:** | Ing. Michal Petřík |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Webové a softwarové inženýrství, zaměření Softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | do konce letního semestru 2022/2023 |

## Pokyny pro vypracování

Pokyny pro vypracování:
1. Proveďte analýzu používaných nástrojů na tvorbu odhadů a sledování projektů (issue tracking nástroje) v prostředí zadavatele.
2. Seznamte se s metodikou odhadů využívanou zadavatelem a současným řešením v aplikaci Estimate (budou poskytnuty odpovídající podklady).
3. Navrhněte architekturu a zvolte vhodné technologie pro integraci nástroje na v kontextu zadavatele nejpoužívanější issue tracking nástroje. Klíčovou oblastí zde bude vhodný návrh univerzálního import/export rozhraní a taktéž možnost úpravy odhadů před vlastním exportem pro potřeby konkrétního issue tracking systému.
4. Implementujte a zdokumentujte navržené změny/moduly.
5. Zhodnoťte výhody a nevýhody vašeho řešení vzhledem k možnosti budoucího rozvoje a rozšiřitelnosti.

---

*Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 5. listopadu 2021 v Praze.*

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

# Rozšíření aplikace pro podporu tvorby odhadů SW o integrační moduly

***Ivan Havasi***

Katedra softwarového inženýrství
Vedúci práce: Ing. Michal Petřík

10. mája 2022

# Poďakovanie

Rád by som poďakoval Ing. Michalovi Petříkovi za jeho pomoc, ktorú mi poskytol počas vedenia tejto práce. Tiež by som rád poďakoval firme Profinit EU, s.r.o. za možnosť realizácie tejto bakalárskej práce. Nakoniec by som sa chcel poďakovať rodine a priateľom za ich podporu.

# Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb, autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k používaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo používať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, ale len pre nezárobkové účely. Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 10. mája 2022 . . . . . .

### Odkaz na túto prácu

Havasi, Ivan. *Rozšíření aplikace pro podporu tvorby odhadů SW o integrační moduly*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

# Abstrakt

Táto bakalárska práca sa zaoberá problematikou tvorby odhadov práce a zaznamenávaním týchto odhadov do systémov určených na sledovanie projektov, tzv. issue tracking systems. Konkrétne sa pozerá na systémy používané vo firme Profinit EU, s.r.o. Zameriava sa na nedostatok, ktorý tvorí to, že z používaného systému na tvorbu odhadov neexistuje jednoduchý spôsob exportovania odhadov do programov na sledovanie projektov, ktoré sa vo firme používajú.

Práca prináša analýzu týchto systémov. Potom analýzu a implementáciu riešenia. Toto riešenie predstavuje integráciu z firemného systému na tvorbu odhadov, do dvoch najpoužívanejších externých systémov na sledovanie projektov. Táto integrácia prináša exportovanie v niekoľkých krokoch, kde si užívateľ môže nastaviť všetky potrebné hodnoty tak, aby exportované dáta už nemusel nijako upravovať. V práci sú ešte zhodnotené výhody a nevýhody tohto riešenia, a taktiež sa rozoberá možnosť budúcich úprav.

**Klíčová slova** tvorba odhadov pracnosti, riadenie projektu, softvérový projekt, Gitlab, Youtrack, API

# Abstract

This bachelor thesis deals with the problem of creating work effort estimates and the subsequent tracking of these estimates in issue tracking systems. Specifically, it looks at the systems used at Profinit EU, s.r.o. It focuses on the shortcoming of current solution, which is used to create work effort estimates. Currently, there is no easy way to export estimates to issue tracking systems, that are used at the company.

The thesis presents an analysis of these systems. Then the analysis and implementation of the solution. This solution is an integration from the company's system to create estimates, into the two most used external project tracking systems at the company. This solution brings an export window with several steps, where the user can set all the necessary values so that the exported data no longer needs to be modified by hand. The advantages and disadvantages of this solution are further evaluated in the thesis, and the possibility of future modifications is also discussed.

**Keywords** creation of work effort estimates, project managment, software project, Gitlab, Youtrack, API

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Úvod

Vo svete softvérových produktov sa neustále začínajú nové a končia staré, či už úspešné alebo neúspešné, softvérové projekty. Samozrejme, každý zákazník chce, aby sa práve jeho projekt zaradil do kategórie tých úspešných. Avšak, nie vždy sa to musí podariť. Dôvodov, prečo môže byť softvérový projekt neúspešný je nespočetne veľa.

Často sa stáva, že sa projekt predraží, jeho vývoj trvá príliš dlho, alebo sa jednoducho nezvládne konkrétna požiadavka zákazníka. Aj preto takmer vždy na projekt dohliada projektový manažér, ktorý sa usiluje o to, aby žiadna z týchto situácií nenastala. Toto určite nie je jednoduchá úloha a vyžaduje si veľa práce, času a pozornosti.

V dnešnej dobe si tento manažér dokáže pomôcť, napríklad, aj systémami na sledovanie projektov. V týchto systémoch môže sledovať, ako prebieha plnenie jednotlivých menších úloh, ale aj projektu ako celku. Na to, aby mohol vedieť, či projekt zaostáva, musí mať vypracované časové odhady, s ktorými potom môže porovnať momentálnu situáciu. Avšak, väčšinou sa na tvorbu odhadov používajú iné systémy ako na sledovanie projektov. Preto musí projektový manažér často prenášať dáta alebo údaje z jedného systému do druhého.

A práve toto je časť, ktorej sa v rámci tejto bakalárskej práce budem venovať. Návrh, implementácia a otestovanie integrácie medzi systémom na tvorbu odhadov vo firme Profinit a najpoužívanejšími systémami na správu projektov (tzv. issue tracking systems). Vo firme sa na správu projektov najčastejšie používa: YouTrack a Gitlab issue tracker. Predtým, ako začneme s implementáciou, je dôležité si ujasniť metodiku odhadov firmy, odhady samé o sebe a ako fungujú systémy na sledovanie projektov. Verím, že aj vďaka tomuto riešeniu sa zjednoduchší a zefektívni práca projektových manažérov, ktorý sa budú môcť viac venovať projektom, vďaka čomu sa ušetrí čas a systém Estimate bude správne integrovaný na ďalšie nástroje, ktoré sa používajú vo firme.

Kapitola 1

# Ciele práce

Hlavným cieľom tejto práce je implementovať integráciu medzi systémom na tvorbu odhadov a systémami, ktoré sú určené na sledovanie projektov. Vo firme Profinit je vyvíjaný systém Estimate, ktorý budeme rozširovať o danú integráciu. Toto rozhranie bude exportovať dané odhady do najviac používaných systémov na správu projektov vo firme. Samozrejme, najprv musíme takéto rozhranie správne navrhnúť a po implementácii ho dostatočne otestovať. Pre správnu implementáciu je dôležitá analýza nástrojov, ktoré sa vo firme používajú. V systéme Estimate je dôležité sa pozrieť na všetky technológie, štruktúru projektu a vhodné umiestnenie tohto rozhrania. Pre externé aplikácie je dôležité si naštudovať ich API, cez ktoré budeme exportovať odhady.

Taktiež je dôležité to, aby sme v práci vysvetlili základné pojmy, s ktorými budeme pracovať. Ako sa dajú sledovať projekty a konkrétne úlohy na týchto projektoch. Prečo je dôležité si vytvárať odhady a ako vytvoriť správny odhad. A hlavne ako tieto dve časti spolu súvisia. Vytvoriť analýzu obidvoch systémov, ktorú potom budeme môcť využiť pri konkrétnej implementácii.

Kapitola 2

# Riadenie softvérového projektu

Každý z nás určite používa veľké množstvo softvérových produktov. Tieto produkty nám často uľahčujú život, pomáhajú nám pri každodenných úlohách alebo nás jednoducho dokážu zabaviť. Je ale dôležité si uvedomiť, že za každým takýmto úspešným produktom je úspešný projekt, ktorý tento produkt vytvoril. Väčšinou na takýchto projektoch pracujú tímy ľudí vedené projektovým manažérom, ktorý dohliada na správne dokončenie projektu. V prípade agilných tímov je táto pozícia rozdelená medzi všetkých členov tímu.

## 2.1 Čo je to softvérový projekt

Podľa metodiky projektoveho riadenia Prince2 [1] je projekt dočasná organizácia vytvorená s cieľom dodávky jedného alebo viacerých produktov. Takýto projekt je vždy unikátny. Nikdy sa nestretneme s dvoma projektami, ktoré budú úplne totožné. Je tiež dočasný. Trvá len pokiaľ nie je dodaný produkt, a potom zaniká. Dôležité je vedieť, že takýto projekt má veľkú mieru neistoty. To znamená, že existuje možnosť, že tento projekt nedosiahne svoj cieľ.

### 2.1.1 Úspešný softvérový projekt

Vieme, že projekt má veľkú mieru neistoty, ale aj ak sa nám projekt podarí dokončiť, nemôžeme ho vždy nazvať úspešným. Úspešný projekt je podľa Steve McConnela [2, str. 4] taký, ktorý dodrží svoje náklady, časový plán a aj ciele s určitou kvalitou. Dôležité je, aby úspešný projekt nemusel posúvať dátum dokončenia ani navyšovať svoju cenu. Po vytvorení detailného plánu by mal projekt dodržať predpoklady s odchylkou približne 10 %. Tento cieľ je v dnešnej dobe dosiahnuteľný bežným projektovým manažérom na normálnom projekte. [2, str. 4]

## 2.2 Životný cyklus softverového projektu

Existuje mnoho spôsobov ako definovať životný cyklus softvérového projektu. V tejto kapitole si predstavíme najbežnejšie životné cykly softverových projektov, ktoré sa používajú. [3]

### 2.2.1 Vodopád

Toto rozdelenie sa nazýva Vodopád, práve preto, že celý cyklus prebieha v smere od začiatku do konca. Výhody takéhoto prístupu sú, že už na začiatku dobre vieme ako bude celý projekt vyzerať. Vieme, čo budeme implementovať a môžeme sa na to pripraviť. Dokážeme vytvoriť presnejší plán s predikovateľnejším rozsahom, cenou a časom. [3] Životný cyklus vyzerá nasledovne:

- plánovanie
- analýza
- dizajn architektúry
- implementácia
- testovanie
- nasadenie

Medzi najväčšie nevýhody patrí to, ako reaguje na zmeny. [3] Ak sa nenachádzame v plánovacej časti je veľmi zložité a často až nemožné doplniť do projektu ďalšie časti, lebo to by narušilo celý plán. Väčšinou sa preto žiadne veľké zmeny v tomto životnom cykle po plánovaní už nevykonávajú.

### 2.2.2 Iteratívny vývoj

Tento cyklus rieši niektoré problémy Vodopádu. Najmä to, ako môžeme reagovať na zmeny. Skladá sa z viacerých iterácií, kde každá iterácia vyzerá ako jeden malý Vodopád. Vďaka tomuto prístupu dokážeme reagovať na zmeny vždy v ďalšej iterácii. Avšak, takýto projekt už nebude mať tak presný plán ako Vodopád, keďže v každej ďalšej iterácii budeme robiť nové zmeny a reagovať na požiadavky zákazníka. [3]

### 2.2.3 Agilný vývoj

V agilnom vývoji sa sústredime na rýchle vývojové cykly. Vytvárame nové verzie často, aj keď obsahujú iba malé zmeny. V každej verzii otestujeme produkt a následne túto verziu vydáme. Dôležitou súčasťou je zapojiť zainteresované strany z podnikovej sféry (*business stakeholders*), aby poskytovali spätnú väzbu počas celého procesu vývoja. [3] Spolu s Agile sa zvykne používať

aj rámec *Scrum*, ktorý pomáha štruktúrovať zložitejšie vývojové projekty. [3] V Scrume tími pracujú v krátkych šprintoch (dva až štyri týždne), v ktorých spĺňajú priradené úlohy. Pomocou denných stretnutí potom tím monitoruje postup projektu. [3]

## 2.3 Projektový trojuholník

Projektový trojuholník (tiež železný trojuholník alebo projektové obmedzenia) [4] je modelom základných parametrov projektu. Trojuholník na každej svojej strane obsahuje:

- Scope – rozsah projektu.
- Cost – náklady na projekt.
- Time – čas potrebný na projekt. [4]

V tele trojuholníka sa potom uvádza aj parameter Quality, teda kvalita daného projektu. [4]

V novšej literatúre sa môžu objaviť aj iné parametre. Napríklad PMBOK Guide má pridané parametre Resources a Risk. [5, str. 6] Avšak my si vystačíme so základnými parametrami.

Hlavnou úlohou tohto trojuholníka je zobrazovať vzťahy medzi všetkými parametrami. A to, že pridáme alebo naopak znížime jeden konkrétny parameter, nám môže ovplyvniť všetky ostatné parametre a teda aj celý projekt. Napríklad, ak chceme ukončiť projekt rýchlejšie (znížiť čas potrebný na dokončenie projektu), tak môžeme zmenšiť rozsah alebo zväčšiť náklady projektu. Pre zníženie celkových nákladov musíme skrátiť čas alebo rozsah projektu. A nakoniec pre zníženie rozsahu projektu je potrebné pridať čas alebo náklady.

Zatiaľ som nijako nespomenul kvalitu. Tá sa bude meniť, ak nebudeme proporcionálne upravovať ostatné parametre. Čiže, ak znížeme náklady na projekt, ale nezmenšíme rozsah celého projektu, posunie sa kvalita projektu nadol. Často môžeme meniť iba rozsah projektu a čas, ktorý na projekte strávime, pretože cena zvykne byť fixne daná zákazníkom. To obmedzuje naše reálne možnosti posunu parametrov v trojuholníku.

Avšak každý softvérový projekt je jedinečný, a preto nám niekedy už ani tento trojuholník nepomôže. V prípade, že na projekte už máme veľké množstvo ľudí (veľké náklady), tak pridanie ešte ďalších kolegov, aby sme projekt urýchlili, môže mať opačný dôsledok, čiže projekt sa môže ešte spomaliť a predĺžiť. [6] Rovnako pridanie ľudí v konečnej fázi projektu nám často nepomôže projekt urýchliť, pretože zaučenie a pomoc novým členom tímu zaberá čas pôvodných členov. Rovnako v projektoch, ktoré nie sú dôsledné riadené nám úpravy parametrov trojuholníka kvalitu zvýšiť vôbec nemusia. [6]

## 2.4 Viditeľnosť priebehu projektu

Dôležitou časťou riadenia projektu je mať dobrú „viditeľnosť“. To znamená vedieť presne povedať aktuálny a pravdivý stav projektu. [2, str. 42] Odpovedá vykonaná časť projektu jeho časovému plánu a rozpočtu? Ak na túto otázku na projekte nevieme odpovedať znamená to, že dostatočne na projekt „nevidíme“ a nemusíme ho dokázať dostatočne riadiť. Príklady činností, ktoré zvyšujú viditeľnosť projektu:

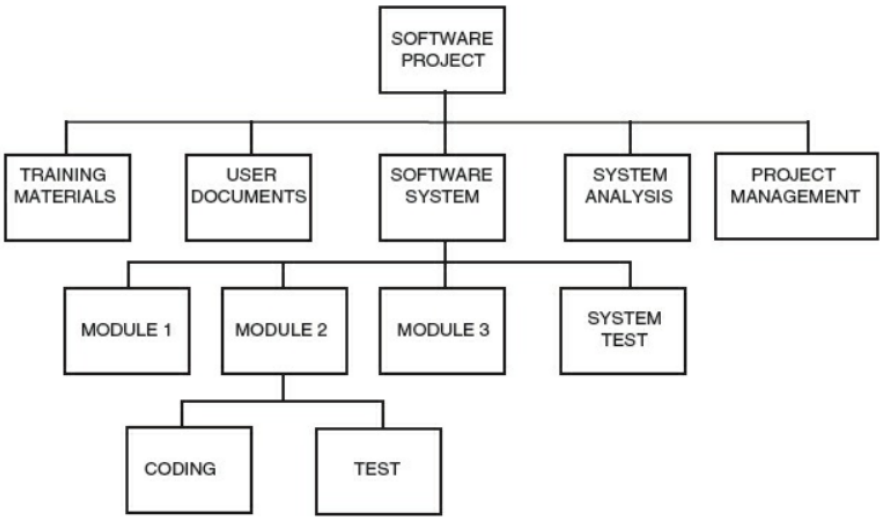
- Preskúmanie projektu po dokončení prvých približne 10 %, aby sme sa vedeli rozhodnúť, či má zmysel v projekte vôbec pokračovať.
- Pravidelne porovnávať aktuálny stav projektu s jeho plánom, aby sme dokázali včas zistiť, či náš plán funguje alebo tento plán musíme nejako upraviť.
- Pri konkrétnych úlohach používať vždy iba označenia, ktoré hovoria o tom, či je úloha hotová alebo nie. To znamená nepoužívať percentuálne odhady, pretože tie nemusia byť vôbec presné.
- Často uvádzať produkt do stavu, v ktorom ho budeme schopný nasadiť, aby sme si mohli overiť stav a kvalitu produktu.
- Prehodnotenie odhadov na konci každej časti projektu s cieľom zlepšenia budúcich odhadov a plánov. [2, str. 42 – 43]

Mať úspešný projekt znamená, mať dobrú viditeľnosť. Dobrú viditeľnosť nedosiahne tím, ktorý sa bude iba spoliehať na to, že ju získa automaticky, ale ten, ktorý ju zapracuje do plánu projektu od jeho úplného počiatku. [2, str. 43]

## 2.5 WBS

Hierarchická štruktúra činností (z anglického Work breakdown structure – WBS) je zoskupenie aktivít projektu, ktoré organizuje a definuje celkový rozsah práce na projekte. Skladá sa z viacerých úrovní, ktoré spolu tvoria stromovú štruktúru, a každá nižšia úroveň detailnejšie definuje prácu na projekte. [7, str. 16] Príklad môžeme vidieť na obrázku 2.1.

WBS sa používa ako jeden z hlavných nástrojov na pomoc pri definovaní práce, ktorá musí byť na projekte ukončená preto, aby boli splnené ciele projektu. Poskytuje rámec organizovania a dekompozície práce na vhodnú úroveň detailov pre plánovanie a riadenie projektu.




Obr. 2.1: Príklad WBS rozdelenia. [7, str. 20]

### 2.5.1 Projektová aktivita

Prvok práce, ktorá je vykonaná na projekte, ktorá môže byť jednoducho definovaná slovesom. Aktivita zvyčajne obsahuje očakávanú dobu trvania, očakávanú cenu a očakávané požiadavky na zdroje. [7, str. 15] Aktivity sú často rozdelené do konkrétnych úloh.

## 2.6 Sledovanie softvérového projektu

Projektový manažér, ktorý sleduje softvérový projekt určite nemá jednoduchú úlohu. Zo strany zákazníka je požadované, aby sa projekt dokončil načas, ale často sa stane, že zo strany vyvojárov je počuť o tom, ako sa termíny nestíhajú. Samozrejme, nie vždy je dokončenie projektu načas tou najdôležitejšou vecou. Niekedy sa oplatí termín predĺžiť a dostať tak kvalitnejší výstup.

Jednou z úloh manažéra je vyhnúť sa problémom a dohliadnuť na dodržanie termínov, k tomu sa často využívajú nástroje na sledovanie projektov. Tieto nástroje uľahčujú sledovanie stavu konkrétnych menších častí projektu ako aj celého projektu. Manažér sa tak môže pozrieť na to, či dátum splnenia úloh zodpovedá stanovenému termínu. Ak niekde uvidí problém, môže ho začať riešiť čo najskôr, a tým zabrániť nahromadeniu väčšieho množstva problémov.

### 2.6.1 Vlastnosti programov

V súčasnosti existuje veľké množstvo programov, ktoré nám pomáhajú so sledovaním projektov. Nie je jednoduché nájsť program vhodný na špecifický

projekt, a preto sa pozrieme na najdôležitejšie vlastnosti, aké by mal náš program obsahovať:

- Prispôsobivosť – niekedy potrebujeme program, v ktorom si budeme zaznamenávať iba chyby, inokedy potrebuje aj ďalšie činnosti, napríklad sledovanie rôznych typov úloch, tvorba testovacích prípadov, možnosť vlastných polí...
- Historia zmien jednotlivých úloh – často si v projekte potrebujeme automaticky zaznamenávať konkrétne zmeny, ktoré sa udiali pri danej úlohe, avšak nie všetky programy na sledovanie projektov takúto možnosť poskytujú.
- Jednoduchosť – je dôležité vybrať si taký program, ktorý bude pre nás jednoduchý na používanie a rýchlo sa s ním naučíme pracovať.
- Oprávnenia – pomáhajú nám nastaviť všetky akcie, ktoré môže konkrétny užívateľ vykonať. Úprava úloh, vytváranie úloh, zmena odhadu času úlohy...
- Prehľad a metriky – program na sledovanie úloh by nám mal umožniť rýchlo sa dostať k potrebným informáciam, či už priorita úlohy, alebo napríklad miera výskytu chýb v čase projektu.
- Pracovný postup – pracovný postup, ktorý je na projekte, by mal byť zlepšený pomocou programu na sledovanie projektu. Mal by pomôcť pri manažovaní a riešení úloh a chýb.
- Integrácia na VCS – tento program by mal obsahovať jednoduchú integráciu na systémy verzovania. [8]

Jednou nezmienenou vlastnoťou je možnosť použitia API programu. To môže slúžit na získavanie, úpravu alebo nastavenie dát. Tejto téme sa ale budeme viac venovať na konkrétnych príkladoch v kapitole 4.

### 2.6.2 Programy

Medzi známe programy určené na sledovanie softvérových projektov patria:

- Bugzilla
- YouTrack
- JIRA
- Trac
- Gitlab issues

- Github issues
- Azure DevOps

V dnešnej dobe všetky tieto programy poskytujú dostatočnú základnú funkčnosť – sledovanie projektu. Rozdiely sú často v detailoch. Rozdiel zvykne byť aj v možnosti hosťovania tohto programu. Niektoré poskytujú *cloudové* riešenie, kedy nie je potrebné nič nastavovať a rovno sa môžeme pustiť do práce. Na iné musíme použiť vlastné servery. Najdôležitejším rozdielom potom môže byť cena.

Kapitola 3

# Odhad

V oblasti softvérového inžinierstva sa slovo odhad používa celkom často. Je ale dôležité ho vedieť použiť správne. Význam tohto slova podľa Synonimického slovníka slovenčiny [9] je: „*približné určenie veľkosti množstva, hodnoty a pod. niečoho*“. Takže význam slova odhad už poznáme, ale my potrebujeme určiť odhad pracnosti.

## 3.1 Prečo odhadovať

Na projekt si biznis vždy vyčlení určité financie a za ne požaduje určitý produkt. K cieľu projektu vzniknú rôzne plány, avšak často sa stane, že naše plány na konci projektu nezodpovedajú realite. Buď sme nezvládli naimplementovať všetko, čo zákazník požadoval, alebo sme boli nútení minúť viac ako bol náš rozpočet. Niekedy tieto rozdiely nie sú veľké a dajú sa upraviť aj počas projektu. Problém, avšak nastáva ak sú plán a realita od seba vzdialené na približne viac ako 20 %. [10, str. 17] V tomto prípade nedokáže projekt zachrániť ani skúsený manažér. Práve pred týmto nás dokážu zachrániť odhady. Hlavným cieľom odhadu je určiť, či je plán dostatočne realistický, aby mohol byť dodržaný. Ak sa nám podarí vytvoriť dostatočne presné odhady, ktoré skombinujeme s dobrým plánovaním a dodržiavaním týchto plánov je väčšia pravdepodobnosť, že dokážeme ukončiť projekt tak, ako sme si ho naplánovali. [10, str. 17]

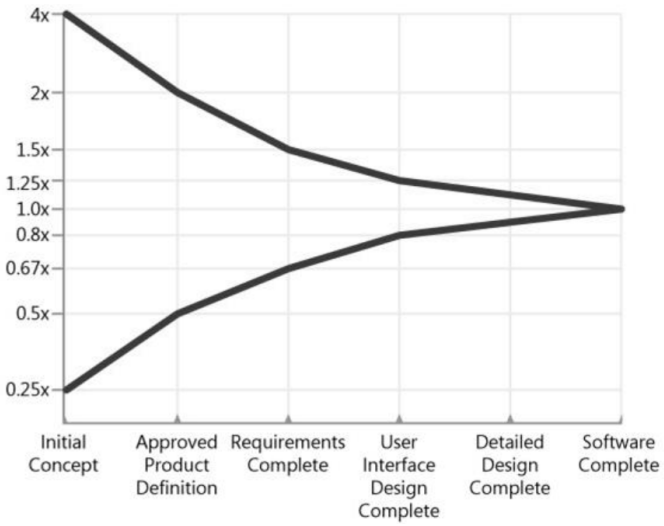
## 3.2 Čo je to odhad pracnosti

Mnohokrát, v oblasti softvérových projektov, počujeme, že čas potrebný na pridanie nejakej novej vlastnosti do nášho projektu sa odhaduje na konkrétny počet hodín, dní, možno mesiacov. Dá sa ale takáto obrovská činnosť odhadnúť takto presne? Pokúsiť sa o to určite môžete, ale ako hovorí Steve McConnel [10, str. 24] ¾ softvérových projektov nedodrží svoje pôvodné odhady alebo

rovno zlyhá. Vytvoriť presný odhad je takmer nemožné. Pri odhadoch je dôležité nevytvárať presné odhady, ale intervaly, pri ktorých sme si dostatočne istí, že sa do nich daná činnosť zmestí.

## 3.3 Kužeľ neistoty

To, aké presné sú naše odhady počas projektu, nám zobrazuje kužel neistoty. Na osi X má životný cyklus projektu od počiatočného konceptu až po ukončenie celého projektu. Na osi Y je pravdepodobnosť chyby, ktorá môže nastať pri tvorbe odhadov. Teda ako veľmi sa náš odhad môže netrafiť do reality. Kužel neistoty tu krásne ukazuje to, čím dlhšie na projekte pracujeme, tým lepšie a presnejšie odhady môžeme vytvárať. [10, str. 32]




Obr. 3.1: Kužeľ neistoty. [10, str. 32]

### 3.3.1 Oblak neistoty

Horším prípadom kužeľa neistoty je *Oblak neistoty.* [10, str. 34] Nastáva vtedy, ak projekt nie je dobre riadený a ľudia, ktorí tvoria odhady nemajú dostatok znalostí alebo skúseností, a preto sa ich odhady nemusia ani časom zlepšovať. Je dôležité mať presne zadefinovaný cieľ projektu a všetky požiadavky na softvér – to dokáže značne znížiť kolísavosť odhadov, a tým pádom sa viac priblížiť ku kužeľu. [10, str. 34]

## 3.4 Časové jednotky odhadov

Pri časových odhadoch sa používajú jednotky ako *man-hour* a *man-day* (skrátene MH a MD). MH vyjadruje množstvo práce, ktorú dokáže jeden človek odpracovať za jednu hodinu. [11] MD nevyjadruje počet dní, ale 8 násobok MH. To v praxi namená, že ak má nejaká činnosť odhad 2 MD, tak v MH to je $2MD * 8 = 16MH$, čiže 16 man-hours.

## 3.5 Metódy tvorby odhadov

Ešte predtým ako si vyberieme konkrétnu techniku, ktorú použijeme na tvorbu odhadov na našom projekte je dôležité zohľadniť hlavné body:

- Čo odhadujeme – nie vždy chceme odhadnúť to isté. Na niektorých projekt môžeme odhadovať koľko úsilia potrebujeme, ako dlho bude trvať projekt alebo koľko vlastností musíme pridať.
- Veľkosť projektu – pri menších projektoch väčšinou nemôžeme použiť štatisticky orientované techniky, ktoré sú naopak vhodné pre veľké projekty. Výhodu majú stredne veľké projekty, ktoré môžu použiť väčšinu techník malých a veľkých projektov.
- Štýl vývoja softvéru – sekvenčný alebo iteratívny štýl vývoja má pri vytváraní odhadov hlavný rozdiel v tom, aký pomer požiadavkov sa definuje v počiatočných fázach projektu oproti fázam, ktoré nasledujú po začatí vývoja.
- Vývojové štádium – niektoré techniky fungujú lepšie v širších častiach kuželu neistoty, iné lepšie v častiach, kedy už máme nejaké dáta, ktoré môžeme využiť na spresnenie ďalších odhadov.
- Možná presnosť – väčšinou chceme, čo najpresnejšie odhady, avšak často tieto presné odhady majú vysokú cenu. Preto sa niekedy môžeme rozhodnúť pre menej presný odhad, ktorý ale získame rýchlejšie a teda s nižšou cenou. [10, str. 46]

Teraz sa môžeme stručne pozrieť na vybrané metódy.

### 3.5.1 Metóda počítania

V odhadovaní by sme sa nemali obmedzovať na to, čo si tradične predstavíme pod pojmom odhad. Ak je to možné, ako prvé by sme mali výsledok spočítať. Tento postup nám dokáže dať najpresnejšie odhady. [10, str. 65]

Niekedy, ak si odpoveď nemôžeme spočítať priamo je dobré si spočítať niečo iné a potom vypočítať výsledok pomocou nejakých kalibračných dát.

Napríklad ak odhadujeme počet ľudí, ktorí sedia v miestnosti pri stoloch a vieme, že je pri jednom stole by malo sedieť 5 ľudí. Stačí nám vynásobiť počet stolov číslom 5.

V softvérových projektoch je mnoho vecí, ktoré sa môžu spočítať. V skorej časti vývojového štádia sa dajú počítať nové vlastnosti, prípady použitia alebo marketingové požiadavky. V strednej časti vývojového štádia dokážeme spočítať dáta trochu detailnejšie a konkrétne môžeme počítať technické požiadavky, požiadavky na zmeny, obrazovky, databázové tabuľky alebo aj dialógové okná. Ku koncu projektu (alebo vývojového cyklu) počítame ešte detailnejšie, teda napríklad kód, ktorý sme už napísali, počet odhalených chýb, triedy alebo aj veci spomenuté v skorších fázach. [10, str. 65 – 66]

### 3.5.2 Metóda historických dát

Kalibrácia dát dokáže prekonvertovať počty na odhady. Každý odhad používa nejakú kalibráciu, či už výslovnú, alebo nevýslovnú. Typy dát, ktoré môžeme použiť na kalibráciu:

- Dáta z projektu – dáta, ktoré boli získané v skorších fázach projektu.
- Dáta z organizácie – historické dáta z organizácie, ktorá na projekte pracuje.
- Dáta z odvetvia – dáta z iných organizácii, ktoré vyvíjajú rovnaký typ softvérovéru ako je softvér, ktorý vyvíjame v projekte. [10, str. 75 – 76]

Dáta z organizácie a projektu sú veľmi nápomocné a dokážu pomôcť k tvorbe veľmi presných odhadov. Dáta z odvetvia slúžia najmä ako záloha, pokiaľ nezískame vlastné dáta. Odhady, ktoré sa vytvárajú z historických dát z projektu alebo organizácie sú presnejšie. [10, str. 75]

### 3.5.3 Metóda expertného odhadu

Táto metóda je jednou z najpožívanejších metód v praxi. Až 72 % odhadov na projektoch je založených na Metóde expertných odhadov. [12] Rozdiel tiež nastáva pri intuitívnom expertnom odhade a odhade, ktorý je štruktúrovaný, takýto odhad dokáže byť presnejší.

Ľudia, ktorí budu pracovať na úlohách, ku ktorým chceme vytvoriť odhady, dokážu vytvoriť presnejšie odhady týchto úloh ako ľudia, ktorí na nich pracovať nebudú. Toto, ale platí hlavne na odhady, ktoré sa používajú na konkrétne úlohy a sú aspoň v strednej fáze vývoja.

K presnejším odhadom nám dokáže pomôcť aj rozdelenie väčších odhadov na viacero malých, kde si môžu malé úlohy odhadnúť špecializovaní ľudia.

Ďalšou dobrou pomôckou je tvorba intervalových odhadov, ktoré obsahuje najlepší, najhorší a najpravdepodobnejší možný scenár. Pri týchto intervaloch

môžeme použiť aj techniku PERT (Program Evaluation and Review Technique), ktorá nám vypočíta očakávaný prípad. Vzorec PERT je: [10, str. 81]

$$ExpectedCase = \frac{BestCase + (4 * MostLikelyCase) + WorstCase}{6}$$

Najpravdepodobnejší scenár je tvorený expertnými odhadmi, a preto sa môže stať, že bude optimistickejší ako realita. V tomto prípade nám môže pomôcť PERT a jeho vzorec, ktorý je upravený tak, že ráta s optimistickým najpravdepodobnejším odhadom: [10, str. 81]

$$ExpectedCase = \frac{BestCase + (3 * MostLikelyCase) + (2 * WorstCase)}{6}$$

Poslednou dôležitou časťou je porovnávanie odhadov s reálnymi výsledkami, aby sme si mohli zlepšovať naše expertné odhady. Tým jednoduchším spôsobom je iba určiť, či bola reálna hodnota v intervale najlepšieho a najhoršieho scenára. Zložitejší spôsob počíta chybu aproximácie (z anglického *Magnitude of Relative Error*) pomocou vzorca: [10, str. 82]

$$MRE = |\frac{(ActualResult - EstimatedResult)}{ActualResult}|$$

Následne porovnávame hodnotu MRE. Čím menšia je táto hodnota, tým presnejšie boli naše odhady.

### 3.5.4 Metóda dekompozície

Tato známa metóda tvorby odhadov pomocou dekompozície funguje na princípe rozdelenia väčších celkov na menšie (tie môžme rozdeliť na ešte menšie...), ktoré potom odhadujeme po jednom. Nakoniec sčítame všetky jednotlivé odhady, čiže máme odhad pre celý veľký celok. Jednotlivé menšie úlohy môžeme odhadovať pomocou metód spomenutých vyššie.

Takýto spôsob benefituje zo *Zákona veľkých čísel*, ktorý v našom prípade hovorí, že ak vytvoríme jeden veľký odhad, tak môžeme odhad oveľa nadhodnotiť alebo podhodnotiť. Avšak, ak vytvoríme viacej menších odhadov, tak jednotlivé odchylky sa navzájom zrušia a náš odhad bude nakoniec presnejší. [10, str. 86]

Pomôcť s dekompozíciou nám môže aj WBS, vďaka ktorému nezabudneme na žiadne malé časti. [10, str. 87] Tiež nám pomôže s porovnaním iných projektov, na ktorých sme WBS použili. Vo vytvorenom WBS vidíme všetky časti projektu, na ktoré by sme nemali zabudnúť – to sú naše jednotlivé malé časti, ktoré môžeme odhadovať.

Tento typ dekompozície sa nazýva *Dekompozícia zhora nadol*. Smer dekompozície môžeme otočiť. V tom prípade nám vznikne *Dekompozícia zdola*

*nahor*. Tá by sa mala používať najmä v tímoch, ktoré sa dobre vyznajú vo svojich projektoch. V tomto prípade príde zákazník s nápadom na nejakú menšiu zmenu. Keďže tím pracuje na projekte už dlhšiu dobu a dobre sa v ňom vyzná, tak dokáže zmenu rýchlo vykonať bez potreby dekompozície zadania na viacero menších úloh. Tento spôsob môže byť rýchlejší, ale je vhodný iba do tímov, ktoré na daných projektoch pracujú dlhšie a dobre ich poznajú.

### 3.5.5 Metóda story points

Táto metóda sa používa pri Agilnom vývoji (konkrétne Extreme programming). Úlohy, v tomto kontexte *stories*, sa hodnotia spoločne členmi tímu, ktorí sa dohodnú na konkrétnom bodovom ohodnotení danej story. Akú škálu použiť pri hodnotení stories? Story points, ktoré sa udeľujú, by nemali závisieť na žiadnych konkrétnych číslach – žiadne dni práce, počty riadkov kódu alebo kalendárny čas. . .Najdôležitejšie je, že členovia tímu používajú rovnakú stupnicu, na ktorej sa včas zhodli, a taktiež, že to hodnotia všetci naraz. [10, str. 105]

Následne sa vytvorí prvá iterácia, do ktorej sa priradia úlohy za predom dohodnutý súčet story points. Po dokončení iterácie je dobré sa spätne pozrieť na vykonanú prácu a vyhodnotiť koľko story points bolo doručených (sčítať story points za všetky zhotovené úlohy). Až teraz sa môže tím pokúsiť odhadnúť jeden story point na úsilie, ktoré je potrebné na doručenie úlohy, čas alebo počet kalendárnych dní, ktoré sú nutné na dokončenie úloh. Znova platí, čím dlhšie na projekte pracujeme, tým presnejšie naše odhady budú. Tento proces porozumenia tomu, čo jeden story point vlastne znamená je špecifický pre každý tím, pretože každý tím si zvolí inú škálu udeľovania story points. Takže úplne rovnaká úloha môže byť v dvoch tímoch hodnotená úplne rozdielne a bude to správne ohodnotená úloha. [10, str. 106]

Dôležité je tiež to, že jeden story point nám hodnotí celý životný cyklus úlohy od jej analýzy cez implementáciu až po testovanie.

Kapitola 4

# Analýza

Ešte predtým ako sa pozrieme na konkrétnu implementáciu riešenia, je potrebné sa pozrieť na doterajšiu aplikáciu Estimate. Aké technológie používa a ako ich môžeme využiť v náš prospech. Potom sa musíme ešte zaoberať API týchto programov, ktoré chceme zintegrovať, aby sme do nich vedeli exportovať odhady.

## 4.1 Proces vo firme

Vo firme Profinit sa na tvorbu odhadov používa softvér Estimate, ktorý je vhodný na odhady tvorené pomocou dekompozície. Na sledovanie projektu sa najčastejšie používajú práve dva programy, a to YouTrack a Gitlab Issues. Tieto programy sú integrované s interným systémom Profis, do ktorého sa vykazuje reálny čas strávený na danej úlohe. Časové hodnoty z Profisu sa potom raz denne importujú do YouTracku alebo Gitlabu.

V praxi, často vidíme, že je vytvorený odhad v aplikácii Estimate, a potom sú všetky hodnoty – popis konkrétnych častí odhadov alebo časové hodnoty – ručne prepisované do spomenutých programov na sledovanie projektu. Takéto spôsoby ručného prepisovania a kopírovania hodnôt z jedného políčka do druhého sú nevhodné pre prácu človeka. Prácu človeka zbytočne predlžujú, často sa pri takýchto všedných úlohách môže človek pomýliť, a potom si túto chybu ani nevšimne. Následne musí vynaložiť ďalší čas na hľadanie chyby a jej opravu. To všetko za predpokladu, že chyba nespôsobí ďalšie a väčšie problémy.

Avšak tento typ úloh je ako stvorený pre počítače. Pri presúvaní hodnôt z jedného programu do druhého sa počítač nepomýli a zvládne to v zlomku sekundy. Práve preto som sa rozhodol vytvoriť možnosť vyexportovania odhadov, vytvorených v softvéry Estimate, do dvoch najpoužívanejších programov na tvorbu odhadov vo firme Profinit.

Metodika tvorby odhadov člení každý odhad na tieto časti:

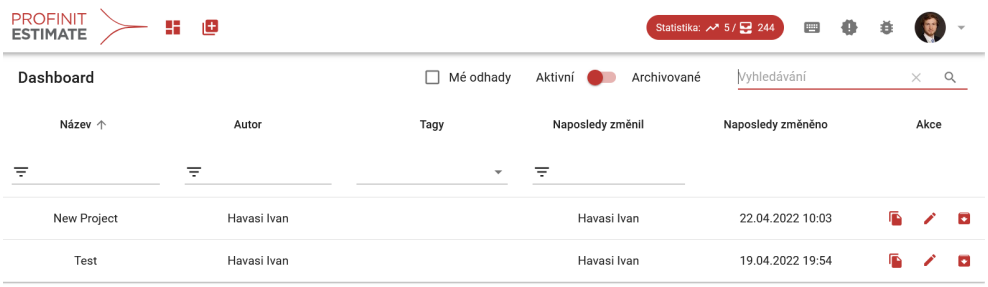
- Analýza
- Design
- Implementácia
- Testovanie
- Project Management
- Tvorba dodávky
- Ostatné

Samozrejme, toto neznamená, že každý odhad musí byť rozdelený do všetkých častí, ale znamená to, že kazdý odhad by mal mať rovnakú alebo aspoň podobnú štruktúru.

Taktiež sa v metodike počíta záruka (zdroj do metodiky), ktorá nám odhaduje predpokladaný čas, ktorý bude použitý na rôzne opravy. Takýto čas je najlepšie si odvodiť z historických dát. Odporúčanou veľkosťou záruky je hodnota 5 % zo súčtu všetkých častí, na ktoré je odhad členený.

## 4.2 Aplikácia Estimate

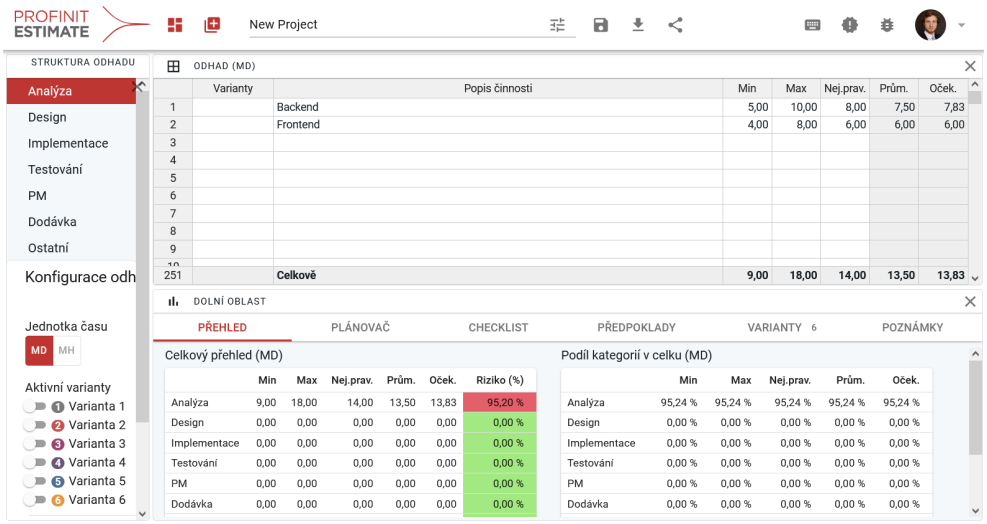
Estimate je aplikácia určená na tvorbu odhadov. Vyvinutá bola vo firme Profinit a jej cieľom bolo zjednodušiť tvorbu odhadov tak, aby sa vo firme prešlo od používania šablón v tabuľkových procesorov k jednotnému systému.




Obr. 4.1: Obrazovka s výberom projektov v aplikácii Estimate.

Aplikácia ponúka tvorbu odhadov podľa internej metodiky. Po prihlásení sa dostaneme na obrazovku s výberom všetkých projektov 4.1, ku ktorým máme prístup. Tu si môžeme buď vytvoriť nový projekt, alebo začať upravovať jeden zo starších projektov. Potom sa dostávame na obrazovku konkrétneho projektu (obrázok 4.2). Cely projekt má nejakú štruktúru. To znamená, že obsahuje kategórie, do ktorých sú odhady delené. Do každej kategórie si môžeme

zaznamenať ľubovoľné odhady. Odhady sa zaznamenávajú do tabuľky uprostred obrazovky. Tá obsahuje stĺpce: varianta odhadu, popis činnosti a potom tri stĺpce pre odhadované časy (minimum, maximum, najpravdepodobnejší), ktoré si môžeme vyplniť. Potom sa v tabuľke ešte vyskytujú dva stĺpce (priemer a očakávaný odhad času), ktoré sa vypočítajú automaticky podľa metodiky.




Obr. 4.2: Obrazovka s konkrétnym projektom v aplikácii Estimate.

Varianty sú dôležitou súčasťou. Užívateľ si môže každý odhad označiť inou variantou a varianty si môže následne v projekte zapnúť alebo vypnúť.

Aplikácia poskytuje dve časové jednotky pre odhad času. Tie sú *man-days* a *man-hours*.

V momente, keď sme spokojní s odhadmi, ktoré sme si vytvorili, si ich uložíme a ak odhady potrebujeme dostať von z aplikácie, tak máme iba dve podobné možnosti. Pomocou tlačítka v hlavičke si vieme celý projekt vyexportovať do HTML alebo CSV formátu.

## 4.3 Stretnutie s reálnymi používateľmi

Pred začatím práce prebehlo online stretnutie s používateľmi aplikácie Estimate. Na stretnutí bol *Delivery Manager* z veľkého projektu vo firme, k jeho pracovným činnostiam patrí aj vytváranie odhadov, a preto to bola vhodna osoba na stretnutie.

Cieľom bolo získať, čo najlepší prehľad o tom, aká možnosť exportu odhadov do programov na sledovanie projektu by bola pre nich najlepšia. Rovnako bolo cieľom zistiť, akým spôsob exportujú odhady teraz a či by sme tento sspôsob vedeli napodobniť aj v našej implementácii.

Výsledkom tohto hovoru bolo zistenie, že aplikáciu nepoužívajú tak často, ako by chceli. Jedným z hlavných problémov bolo práve exportovanie odhadov do externých aplikácií. Prvotné odhady sú vytvorené v tabuľkovom procesore s vždy rovnakou štruktúrou – rozpad pracnosti na WBS. Ku konkrétnym odhadom zvyknú pridávať ďalšie informácie – typ, ktorý bude mať nové issue, poznámky a odhadovaná pracnosť. Potom sa rozhodnú, ktoré odhady vôbec budú exportovať, a ktoré zatiaľ nechcú ďalej používať. Odhady označené na export pomocou vlastného konvertoru prekonvertujú do tvaru, ktorý externá aplikácia rozpozná a nakoniec ich naimportujú do externej aplikácie.

Takže nový exportér by mal mať možnosť exportovať odhady, ktoré majú:

- Rôzne kategórie (analýza, implementácia...).
- Rôzne typy issues, ktoré majú vzniknúť.
- Možnosť pridania poznámky alebo nejakej ďalšej informácie.
- Odhadovanú pracnosť.

Takýto export by mal byť intuitívny a jednoduchý tak, aby sa nový používateľ nemusel učiť nič nové. Rovnako by mal byť dobre integrovaný do aplikácie Estimate či už vizuálne, alebo technologicky.

## 4.4 Návrh

Podľa môjho názoru je najlepším spôsobom exportu odhadu vytvorenie dialógového okna, ktoré dokáže previesť užívateľom celým procesom exportovania odhadov. Celý tento proces má viacero krokov, a preto by aj takéto dialógové okno malo mať kroky zložené z obrazoviek, na ktorých sa budú dať nastavovať konkrétne detaily podľa požiadavky užívateľa. Nastavovať tieto detaily pre každý samostatný odhad v dialógovom okne nie je veľmi praktické a už vôbec nie prehľadné. Práve preto sa budú upravovať nastavenia pre jednotlivé katégorie. Týmto štýlom docielime kompromis medzi nutnými zmenami pri každom odhade a jednoduchom všeobecnom nastavení, ktoré by platilo pre celý projekt.

Iným možným návrhom by bolo zapracovanie všetkých nastavení priamo do hlavnej tabuľky s odhadmi. Výhodou tejto možnosti je schopnosť upravovať a nastavovať rôzne hodnoty pre každý odhad samostatne. Nevýhodou je spojenie dvoch rozličných činností (tvorba odhadu a export odhadu) do jednej. Nie každý užívateľ, ktorý si vytvára odhady ich potrebuje exportovať do externej aplikácie. Tiež by to zhoršilo prehľadnosť hlavnej tabuľky s odhadmi.

## 4.5 Použité technológie

Aplikácia Estimate je rozdelená na tri časti, z ktorých každá používa moderné technológie – programovací jazyk a frameworky:

**Prezentačná vrstva** TypeScript, React, Material-UI.

**Aplikačná vrstva** Kotlin, SpringBoot.

**Dátová vrstva** JSON, MongoDB.

Tieto technológie využijeme na to, aby sme mohli vytvoriť rozhranie na exportovanie odhadov do dvoch programov na sledovanie projektu – Youtrack a Gitlab issues.

### 4.5.1 Prezentačná vrstva

Bežný jazyk, ktorý sa používa na vývoj na frontende je JavaScript. V tejto aplikácii ale použitý nie je. Miesto neho sa používa TypeScript, ktorý sa niekedy môže nazývať aj *typovaným JavaScriptom*. [13] Veľkou výhodou je, že každý funkčný JavaScript kód je aj TS kódom. Najväčším rozdielom je používanie typov. Aj keď nejakej hodnote (napr. premennej) explicitne nepriradíme typ, tak TS si ho odvodí z hodnoty, ktorú premenná nadobudne a dokáže nás upozorniť na nesprávne narábanie s danou hodnotou. Toto sa nám veľmi hodí, ak pracujeme na väčšom množstve kódu a chceme zabrániť zbytočným chybám.

V tejto aplikácii bola na frontende použitá knižnica *React*. Je to JavaScript knižnica určená na vytvárane používateľského rozhrania. [14] Vytváranie rozhrania robí jednoduchým aj vďaka použitiu zapuzdrených komponentov. React dokaže efektívne aktualizovať a zobraziť vždy iba tie potrebné komponenty, keď sa menia dáta.

Konkrétny dizajn nám pomáha vytvárať knižnica *Material-UI*, ktorá slúži na jednoduché pridávanie už nadizajnovaných komponent. [15] Sú to napríklad tlačítka, inputy, texty...Každý komponent môžeme jednoducho a rýchlo použiť v našej aplikácii bez dlhého nastavovania alebo dizajnovania.

Aplikácia je rozdelená do mnohých komponentov, čo ju robí prehľadnou. My toto rozdelenie v našej práci využijeme a budeme pridávať vlastné komponenty na export.

### 4.5.2 Aplikačná vrstva

Staticky typovaný moderný programovací jazyk. Takto by sme mohli predstaviť *Kotlin* v jednej vete. Jazyk, ktorý beží nad JVM. Práve toto je jednou z veľkých výhod tohto jazyka. Pri nových a moderných jazykoch často môže nastať problém s nedostatkom kvalitných knižníc, ale keďže je Kotlin interoperabilný s Javou, [16] môžete využívať všetky tie knižnice, ktoré ponúka Java.

Ďalšou výhodou interoperability s Javou je jednoduchý štart pre ľudí, ktorí vyvíjajú v Jave. Medzi ďalšie rozdiely oproti Jave patrí napríklad: null-safety, Kotlin nemá checked exceptions, extension functions, companion objects. . .[17] Okrem iného Kotlin ponúka možnosť preloženia kódu do JavaScriptu a od roku 2019 je Kotlin preferovaným jazykom na vývoj na platforme Android. [18]

Dobrú podporu v Kotline má aj *Spring Framework*. Poskytu komplexný programovací a konfiguračný model pre moderné podnikové aplikácie alebo hocijaké Java aplikácie. Prvá produkčná verzia bola vydaná v roku 2004. [19] Spring Framework sa snaží zjednodušiť vývoj aplikácii tak, aby sme sa my mohli sústrediť na biznis logiku a nemuseli sa stále dookola zaoberať podobnou aplikačnou logikou. Medzi najväčšie výhody patrí *dependency injection*. [20] To znamená, že Spring dokaže vložiť rôzne objekty (závislosti) do iného objektu. Ďalšími výhodami sú: validácia, data binding, vytvaranie webových aplikácii pomocou *Spring MVC* alebo *WebFlux*, jednoduché testovanie web serverov pomocou *WebTestClient* a mokovanie, scheduling. . .

Rozšírením Spring Frameworku je *Spring Boot*, ktorý okrem toho všetkého, čo dokáže Spring Framework, vytvorí aplikáciu, ktorú môžeme *rovno spustiť*. [21] Je prednastavený s najlepšími knižnicami, ktoré buď poskytuje Spring Framework, alebo sú z tretej strany. Tieto knižnice vyberá tím zo Springu. [21] Okrem pripravenej Spring aplikácie tiež poskytuje: vstavaný Tomcat HTTP web server, vybrané závislosti, automatické nastavenie Springu, produkčné vlastnosti (metriky, zistenie stavu aplikácie).

### 4.5.3 Dátová vrstva

Asi najbežnejším spôsobom uchovávania dát sú SQL databázy. Používajú sa relačné databázové systémy ako PostgreSQL, MySQL, či Oracle Database. Avšak v tejto aplikácii bol zvolený iný prístup. NoSQL sú netabuľkové databázy, ktoré uchovávajú dáta iným spôsobom ako klasické relačné databázy. Poskytujú flexibilné schémy a jednoducho sa škálujú aj na veľké množstvo dát. Hlavnými typmi NoSQL databáz sú:

- Dokumentová
- Kľúč-hodnota
- Wide-column
- Grafová

Aplikácia Estimate používa multiplatformnú open-source *dokumentovú databázu MongoDB*. Táto možnosť bola zvolená, pretože odhady sú ukladané a držané v JSON šablónach (objektoch). A práve toto je jeden z najlepších dôvodov na použitie dokumentových NoSQL databáz, akou je MongoDB. S JSON objektami dokáže jednoducho, dobre a rýchlo pracovať. Tiež nám uľahčuje iteratívny vývoj, keď sa často mení databázová schéma. V klasických

relačných databázach by sa neustále musel prepisovať alebo upravovať databázový model, čo by spomaľovalo celkový vývoj. Toto boli hlavné dôvody, prečo bola táto možnosť zvolená. [22, str. 20, 60 – 61]

## 4.6 Youtrack

Youtrack je nástroj na riadenie projektu, ktorý si môžeme prispôsobiť k našim potrebám. Ponúka možnosti sledovania projektov a jednotlivých úloh, možnosti používania agilných tabulí (*agile boards*), plánovania šprintov a vydaní nových verzií, vytváranie vlastných pracovných postupov a procesov a ďalšie. [23] Je to proste nástroj, ktorý nám pomáha sledovať úplne celý projekt. A ešte si ho môžeme prispôsobiť podľa naších predstáv.

Pre nás najdôležitejšou súčasťou Youtracku je jeho API, pomocou ktorého dokážeme upraviť všetko to, čo v tejto práci budeme potrebovať. Na konkrétne časti z API, ktoré sa nám zídu sa pozrieme v nasledujúcej podkapitole.

### 4.6.1 API

Tabuľka 4.1: Tabuľka so zákadnými údajmi pre Youtrack API

| | Hodnoty |
|---|---|
| Názov | Youtrack REST API |
| Základná časť url | `https://example.youtrack.cloud/api` |
| Formát tela | JSON |
| Autorizácia | Permanent token |

Všetky údaje ohľadom Youtrack API môžeme nájsť v oficiálnej dokumentácii. [24] Toto REST API zvláda nasledujúce činnosti:

1. Importovať úlohy z predošlého programu na sledovanie projektov.
2. Vytvárať, upravovať a rôzne ďalšie operácie s jednotlivými úlohami.
3. Upravovať projekty, agilné tabule, vlastné polia alebo spájanie viacerých úloh.

V tejto práci sa budeme venovať iba tomu, čo bude neskôr potrebné, takže prvému bodu sa vyhneme úplne a zameriame sa na vytváranie a úpravu nových úloh.

Aby sme vôbec vedeli komunikovať s týmto API, musíme sa dokázať autorizovať. Youtrack ponúka dve možnosti:

**Permanent Token Authorization** autorizácia s použitím permanentného tokenu.

**OAuth 2.0 Authorization** autorizácia s OAuth 2.0 protokolom, kde nám Youtrack pridelí access token pre každé prihlásenie zo strany klienta. [25]

Pre naše účely je lepšie ísť cestou *Permanent Token Authorization*, pretože nám stačí vygenerovať jeden token s administrátorskými právomocami, ktorý bude potom v aplikácii *Estimate* uložený a ona ho potom bude môcť používať pre všetkých svojich užívateľov. Stačí token pripojiť do hlavičky každého požiadavku, konkrétne k parametru *Authorization*. Naproti tomu, ak by sme si vybrali druhú možnosť, používateľ by sa musel pri každom novom prihlásení do aplikácie a pokuse o export prihlásiť aj do svojho Youtrack účtu.

Teraz, keď sa už dokážeme autorizovať, môžeme začať s vytváraním *HTTP požiadavkov*. Tie majú mať svoje telo vo formáte *JSON*. [24]

Pre naše potreby budeme potrebovať *HTTP požiadavky* na tieto činnosti:

1. Získanie všetkých prístupných projektov.
2. Získanie špeciálnych polí (časový odhad, štítky).
3. Vytvorenie novej úlohy.
4. Nastavenie špeciálnych polí pre dané úlohy.
5. Nastavenie rodiča pre konkrétne úlohy.

Pre získanie zoznamu všetkých projektov môžeme použiť *GET požiadavok* poslaný na url `/admin/projects?fields=id,name,shortName`. Ak by sme si chceli skusiť takýto požiadavok prevolať, môžeme použiť nástroj príkazového riadku curl, ktorý slúži aj na prevolávanie HTTP požiadavkov. [26] Nesmieme zabudnúť na nastavenie *permanentného tokenu* v hlavičke. 4.3

```
curl -X GET \
'https://example.youtrack.cloud/api/admin/projects' \
-H 'Accept: application/json' \
-H 'Authorization: Bearer perm:PERM-TOKEN' \
-H 'Content-Type: application/json'
```

Obr. 4.3: Príklad curl príkazu na vytvorenie HTTP požiadavku na získanie zoznamu projektov v Youtrack API

Návratovou hodnotou je pole projektov so všetkými hodnotami, my sme si avšak v url požiadavku zadefinovali práe tri hodnoty, a to: *id*, *name* a *shortName*. Okrem toho sa nám vždy vráti aj špeciálna hodnota *$type*, ktorá nám určuje JSON objekt, 4.4 ktorý je v našom prípade projekt.

Keďže máme zoznam projektov, môžeme si vybrať jeden konkrétny projekt, s ktorým budeme ďalej pracovať. Budeme preňho potrebovať zistiť štítky,

```
[
    {
        "shortName": "BP",
        "name": "Bakalárska práca",
        "id": "0-1",
        "$type": "Project"
    }
]
```

Obr. 4.4: Príklad návratovej hodnoty z požiadavku na získanie zoznamu projektov

ktoré sa používajú na označenie každej úlohy. Na tento úkon tiež zvolíme *GET požiadavok* s url adresou `/admin/projects/{projectId}/customFields?fields=id,field(id,name),bundle(name,values(id,name))`. V url adrese použijeme id projektu z minulého požiadavku. V odpovedi 4.5 sa nám vráti meno aj id konkrétnych štítkov, ktoré potom môžeme nastaviť novej úlohe.

Od GET požiadavkov sa teraz presunieme k POST požiadavkom. Prvým je vytvorenie novej úlohy. V url adrese `/api/issues?fields=id,idReadable` sme si nastavili tvar návratovej hodnoty, ktorá bude obsahovať iba *id* a *idReadable* danej úlohy. Často sa v *POST požiadavkoch* vypĺňa aj telo a tento prípad nie je výnimkou.4.6 Vyplníme základné hodnoty ako *project*, kde vyplníme id projektu, ktoré sme získali v predchádzjúcom požiadavku, 4.4 *summary* – názov, *description* – popis. Časový odhad a typ úlohy sú *špeciálne hodnoty*, ktoré treba vyplniť trochu zložitejšie. Pri type použijeme jeden z názvov, ktoré sme získali v požiadavku 4.5. Hodnota časové odhadu sa zapisuje v *ľudsky čitateľnom formáte*. To znamená, že ak chceme zapísať hodnotu 10 hodín a 5 minút, použijeme: „10h 5m“. Navratovu hodnotu sme si znova nastavili pomocou url parametru *fields*, takže sa nám vráti *id* a *idReadable*, ktoré použijeme pri spájaní dvoch rôznych úloh.

A práve toto je posledným požiadavkom, ktorý vytvoríme. Posielať budeme POST požiadavok `/admin/projects/{projectId}/customFields?fields=id,field(id,name),bundle(name,values(id,name))`. Táto url adresa slúži na používanie rôznych príkazov, ktoré môžeme poslať na server, ktorý ich potom vyhodnotí. My to využijeme na nastavenie rodičovskej úlohy. Do query parametru to zapíšeme ako „subtask of {idReadable}“ a určíme id našej „detskej“ úlohy.4.7

## 4.7 Gitlab

Na rozdiel od Youtracku, Gitlab nie je iba nástrojom na sledovanie projektov. Samozrejme, zvláda aj túto činnosť, ale okrem toho sa používa hlavne ako

```
[
    {
        "bundle": {
            "values": [
                {
                    "name": "Task",
                    "id": "90-0",
                    "$type": "EnumBundleElement"
                },
            ],
            "name": "Priorities",
            "$type": "EnumBundle"
        },
        "field": {
            "name": "Priority",
            "id": "87-0",
            "$type": "CustomField"
        },
        "id": "108-7",
        "$type": "EnumProjectCustomField"
    },
]
```


Obr. 4.5: Príklad návratovej hodnoty z požiadavku na získanie štítkov na projekte

webový *Git* repozitár. Gitlab taktiež zvláda všetko od plánovania projektu cez vývoj, testovanie buildov, jednoduchého nasadenia až po monitorovanie a manažovanie daného projektu. [27]

Čo má spoločné s Youtrackom je, že tiež ponúka kvalitné API, ktoré nám pomôže so všetkým, čo k tejto práci potrebujeme. Znova sa na všetko dôležité z API pozrieme v nasledujúcej podkapitole.

### 4.7.1 API

Tabuľka 4.2: Tabuľka so zákadnými údajmi pre Gitlab Issues API

| | Hodnoty |
|---|---|
| Názov | Gitlab Issues API |
| Základná časť url | `https://git.my-domain.eu/api/v4` |
| Formát tela | JSON |
| Autorizácia | Impersonation Token |

```
{
    "project": {
        "id": "0-1"
    },
    "summary": "With estimation",
    "description": "If you are reading this, it worked",
        "customFields": [
    {
        "name":"Estimation",
        "$type":"PeriodIssueCustomField",
        "value":
            {
                "presentation":"20h"
            }
    },
    {
        "value": {
            "name": "Task",
            "$type": "EnumBundleElement"
        },
        "name": "Type",
        "$type": "SingleEnumIssueCustomField"
    }
    ]
}
```

Obr. 4.6: Ukážka tela požiadavku na vytvorenie novej úlohy

```
{
"query": "subtask of BP-26",
"issues": [
        {
        "id": "2-85"
        }
    ]
}
```

Obr. 4.7: Príklad tela pri požiadavku na vytvorenie spojenia medzi dvoma úlohami

Táto REST API funguje veľmi podobne ako tá predošlá. 4.6.1 Znova budeme používať údaje z oficiálnej dokumentácie. [28] Autorizácia taktiež

funguje podobne, avšak Gitlab ponúka viacej možností. Tu je výber tých najdôležitejších:

**OAuth 2.0 Authorization** autorizácia s OAuth 2.0 protokolom.

**Personal access token** autorizácia pomocou tokenu, ktorý si môže každý užívateľ vygenerovať.

**Impersonation Token** podtyp *Personal access tokenu*, ktorý nám dovolí predstierať, že sme nejaký užívateľ.

Pre naše potreby, vytváranie a správa úloh, je najlepšou možnosťou použiť *Impersonation Token*, ktorý nám vygeneruje administrátor s nutnými právami. Potom tento token jednoducho pripojíme do hlavičky každého požiadavku do parametru *Authorization*.

Na začiatok potrebujeme zistiť zoznam projektov. Stačí vytvoriť GET požiadavok na url `/projects`. Týmto získame zoznam všetkých projektov, 4.8 z ktorých si môžeme vybrať jeden, s ktorým chceme pracovať ďalej.

```
[
    {
        "id": 620,
        "description": "",
        "name": "bakalarka-havasi",
        "name_with_namespace": "Ivan Havasi / bakalarka-havasi",
        "path": "bakalarka-havasi"
    }
]
```

Obr. 4.8: Príklad zoznamu projektov s vybranými hodnotami

Id projektu, ktoré sme získali z minulého požiadavku na získanie celého zoznamu projektov, teraz využijeme na získanie štítkov, ktoré sa dajú úloham na tomto projekte priradiť. Štítky získame zaslaním GET požiadavku na url `/projects/{projectId}/labels`. Z návratovej hodnoty 4.10 si uchováme id pre nasledujúce požiadavky.

```
{
    "title" : "Issues with auth2",
    "state" : "opened",
    "description" : "rest api created issue"
}
```

Obr. 4.9: Príklad tela požiadavku na vytvorenie novej úlohy

```
[
    {
        "id" : 1,
        "name" : "bug",
        "color" : "#d9534f",
        "text_color" : "#FFFFFF",
        "description": "Bug reported by user",
        "description_html": "Bug reported by user",
        "open_issues_count": 1,
        "closed_issues_count": 0,
        "open_merge_requests_count": 1,
        "subscribed": false,
        "priority": 10,
        "is_project_label": true
    }
]
```


Obr. 4.10: Príklad zoznamu štítkov

Teraz sa pozrieme na POST požiadavky. Prvým je vytvorenie novej úlohy. Pošleme požiadavok na url `/projects/projects/{projectId}/issues`, kde vyplníme id projektu. Ešte pred odoslaním musíme vytvoriť telo požiadavku, 4.9 do ktorého vložíme nadpis, popis a stav úlohy. Z toho, čo sa nám vrátilo si uložíme dve hodnoty, a to *id* a *iid*. Obidve sú interné hodnoty Gitlabu pre identifikáciu konkrétej úlohy.

```
{
    "target_project_id": 620,
    "target_issue_iid": 5
}
```

Obr. 4.11: Príklad tela požiadavku na prepojenie dvoch úloh

```
{
    "duration": "4h15m"
}
```

Obr. 4.12: Príklad tela požiadavku na nastavenie časového odhadu

Keďže sme si už vytvorili úlohu, môžeme jej nastaviť špeciálne hodnoty. Začneme nastavením rodičovskej úlohy. Stačí poslať POST požiadavok na url

`/projects/{projectId}/issues/{childIssueId}/links`. V url adrese uvedieme id projektu a id našej úlohy, teda úlohy, ktorej chceme priradiť rodiča. Do tela požiadavku 4.11 znova uvedieme id projektu a *iid* cieľovej úlohy, takže nášho rodiča.

Nakoniec vytvoríme POST požiadavok, ktorý nastaví časový odhad. Posielame ho na url adresu `/projects/620/issues/{issueId}/time_estimate`. Znova je potrebné uviesť id projektu a id našej úlohy, ktorej časový odhad chceme nastaviť. Do tela požiadavku musíme uviesť čas. Ten sa zapisuje v *ľudsky čitateľnom formáte*, ktorý sme si už ukázali pri predchádzajúcom API. 4.6.1 Takže k hodnote *duration* iba pripíšeme časovú hodnotu. 4.12

Kapitola 5

# Implementácia

V tejto kapitole sa pozrieme na to, ako sme naimplementovali rozšírenie pre aplikáciu Estimate. Toto rozšírenie dokáže vyexportovať odhady do Youtracku a Gitlab Issues.

Implementacia prebiehala *iteratívnym spôsobom* v týždňových intervaloch. To znamená, že jeden deň v týždni prebehlo stretnutie, kde sa predstavili najnovšie zmeny, teda všetko, čo bolo naprogramované. Po kontrole práce sa dohodol postup na ďalší týždeň. Takto sa postupovalo počas celého vývoja. Tento spôsob bol zvolený preto, lebo takýto štýl je rýchly, dokáže rýchlo reagovať na nové nápady a zmeny, taktiež *proof of concept* (teda predvedenie prvej funkčnej implementácie) vznikol hneď na začiatku vývoja. Vďaka tomu bolo hneď vidieť, že na práci má zmysel pokračovať a to, čo sme si zadali na začiatku je dosiahnuteľné.

## 5.1 Frontend

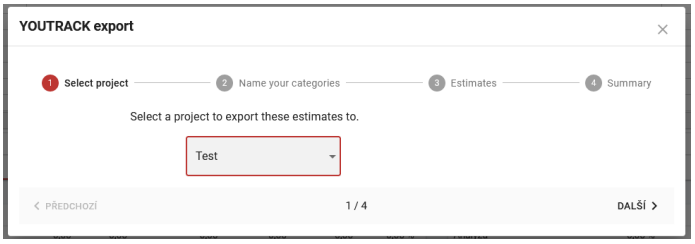
### 5.1.1 Návrh

Vďaka iteratívnemu spôsobu vývoja sa dizajn dialógového okna, ktoré slúži na export, menil z týždňa na týždeň v náväznosti na konzultácie, ktoré prebiehali každý týždeň. V tejto práci nebudeme prechádzať úplne všetky zmeny, ktoré vznikali každý týždeň. Zameriame sa iba na tie najdôležitejšie alebo najväčšie zmeny.

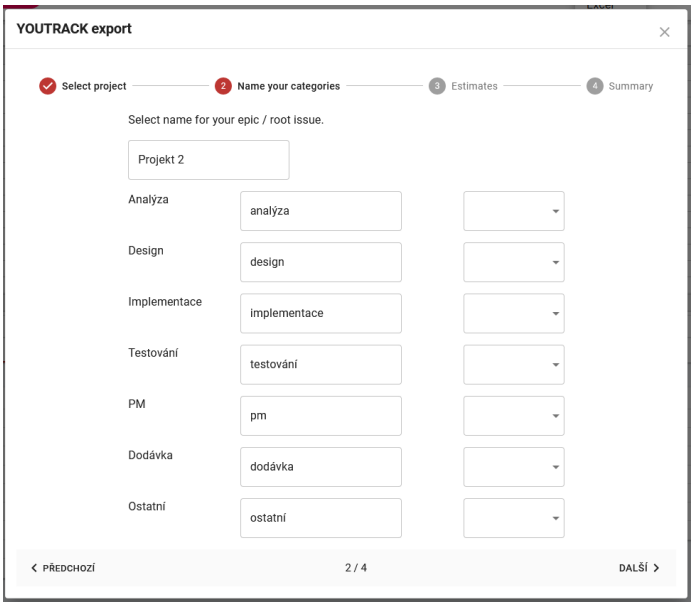
Úplne prvá iterácia dizajnu dialógového okna bola veľmi jednoduchá. Dal sa vybrať projekt, do ktorého sa bude odhad exportovať. Potom sa dalo iba stlačiť tlačítko exportu a odhad by sa vyexportoval. Okno neobsahovalo žiaden sprevádzajúci text. Jediný text, ktorý mohol byť nápomocný bola stránka, na ktorú sa má exportovať odhad (v našom prípade Youtrack).

Druhá iterácia toho už veľa vylepšila. Jednoduché dialógové okno sa zmenilo na okno so štyrmi krokmi. Každý krok tohto okna nastavoval rôzne údaje a dáta k exportu. V prvom kroku si vyberieme projekt, do ktorého budeme


Obr. 5.1: Prvá iterácia dizajnu dialógového okna.

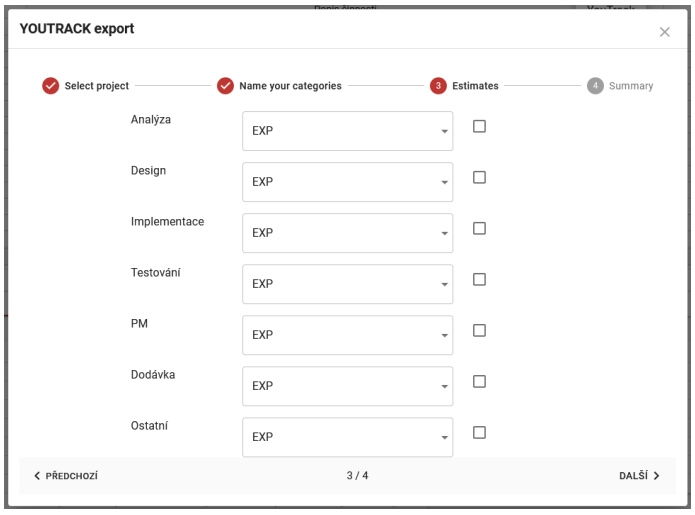



Obr. 5.2: Druhá iterácia dizajnu dialógového okna. Prvý krok.

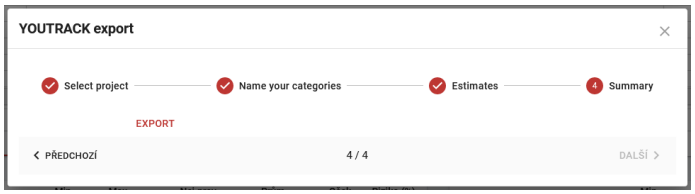



Obr. 5.3: Druhá iterácia dizajnu dialógového okna. Druhý krok.

exportovať. Toto bolo v podstate všetko, čo zostalo rovnaké z prvej iterácie. V ďalšom okne sa dal nastaviť názov hlavnej úlohy (epic). Pre každú kategóriu, ktorá sa používa v aplikácii sa dalo nastaviť spoločné krátke zhrnutie a hlavne typ, ktorý dostane úloha v Youtracku. V treťom okne sa dá nastaviť, ktorý typ času chceme použiť vo vyexportovanom odhade. Aplikácia Estimate totižto ponúka viacero možností. Minumum, maximum, priemer, najpravdepodobnejší a očakávaný čas. Rovnako sa pre každú kategóriu dala nastaviť záruka, teda koľko percent z celkového času danej kategórie ma byť určených na neočakávané momenty, úpravu bugov, hľadanie bugov. . .Posledný krok zobrazuje zhrnutie všetkých nastavení, ktoré si užívateľ nastavil.



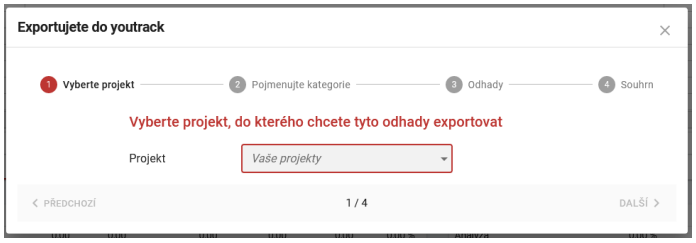

Obr. 5.4: Druhá iterácia dizajnu dialógového okna. Tretí krok.



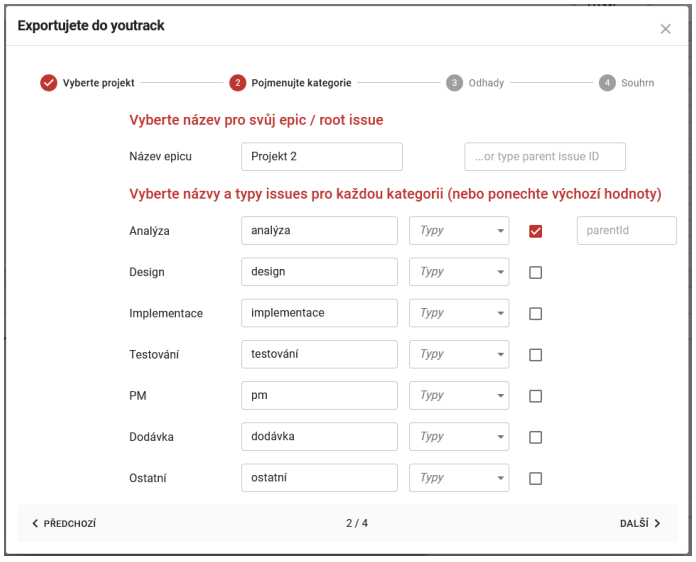

Obr. 5.5: Druhá iterácia dizajnu dialógového okna. Štvrtý krok.

Tretia iterácia prináša hlavne dizajnové zmeny. V každej časti pribudli nadpisy, ktoré v skratke vysvetľujú, čo sa v konkrétnej časti dá nastaviť. Celá aplikácia Estimate používa dva jazyky – čestinu a angličtinu, preto bolo v tejto iterácii všetko preložené do obidvoch jazykov a pri prepnutí jazyka v nastaveniach sa všetok text zobrazí v danom jazyku. Okrem toho pribudla možnosť v druhom kroku nastaviť si vlastnú rodičovskú úlohu pre celý odhad alebo pre jednotlivé kategórie. To znamená, že ak už niekto mal hlavnú úlohu (epic) vy-

tvorený pred exportovaním odhadu, stačí mu napísať id danej úlohy do nového políčka a ostatné úlohy sa prepoja s týmto rodičom.

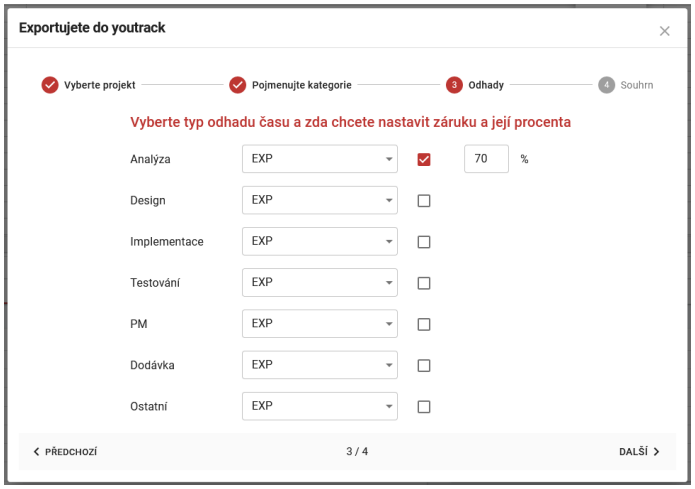



Obr. 5.6: Tretia iterácia dizajnu dialógového okna. Prvý krok.

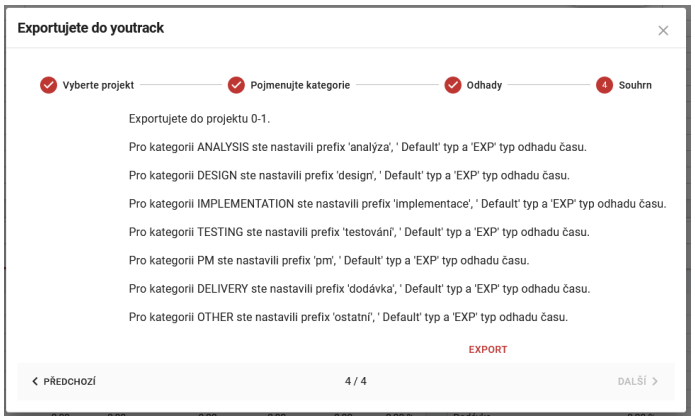



Obr. 5.7: Tretia iterácia dizajnu dialógového okna. Druhý krok.

V poslednej iterácii boli brané do uváhy aj podnety, ktoré vznikli v záverečnom online stretnutí s používateľmi Estimate. Na základe tohto stretnutia bolo pridané políčko 5.10 pre výber *vlastného poľa*, ktoré má informácie o odhade. Ďalším podnetom bolo zobrazenie stavu exportu. Čiže zobraziť užívateľovi informáciu o tom, že aplikácia pracuje. To sme vyriešili pridaním piateho kroku, v ktorom užívateľ môže exportovať odhady 5.12. Po kliknutí na tlačítko *Export* sa odhady začnú exprotovať a užívateľovi sa zobrazí načítacia obrazovka 5.14. Ak sa odhady vyexportujú bez problémov, dialógové okno sa zavrie. Ak nastane nejaký problém, tak sa užívateľovi zobrazí v okne 5.13 informácia o chybe. Takto užívateľ bude vždy presne vedieť, či sa export podaril. Okrem toho bola trochu upravená aj obrazovka so zhrnutím 5.11 tak, aby bola lepšie čitateľná. Hlavné bolo zvýrazniť najdôležitejšie údaje.




Obr. 5.8: Tretia iterácia dizajnu dialógového okna. Tretí krok.

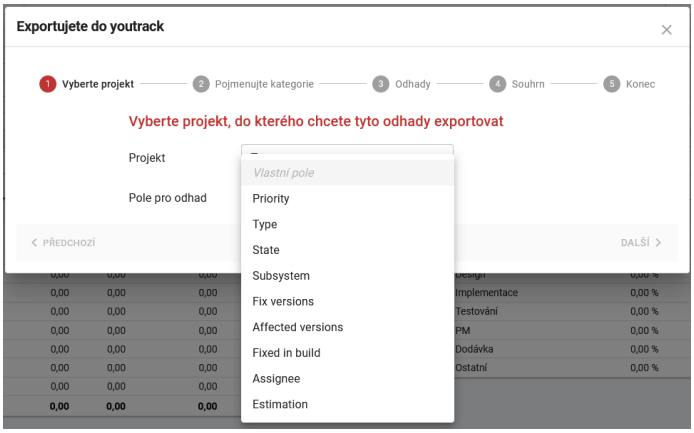



Obr. 5.9: Tretia iterácia dizajnu dialógového okna. Štvrtý krok.

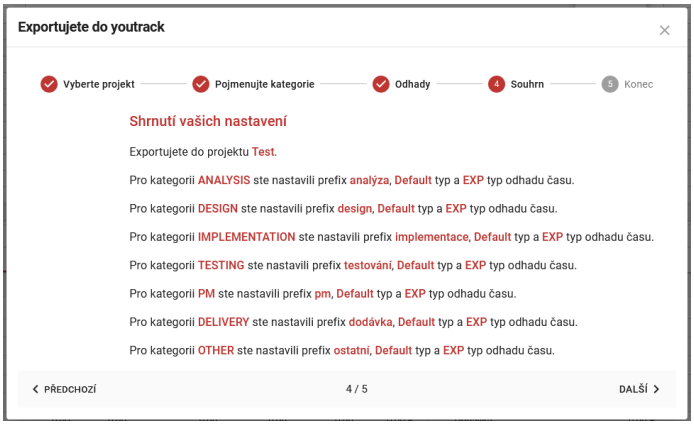
### 5.1.2 Kód a technológie

Ako už bolo spomenuté v analýze prezentačnej vrstvy 4.5.1, na frontende je použitá knižnica *React*. Celá aplikácia je rozdelená do menších *komponent*, ktoré sa dajú viackrát použiť. V tomto štýle pokračovala aj moja práca. Ako vidieť na obrázku 5.15, každá stránka tvorí samostatný komponent a v prípade, že v danej stránke používame podobné alebo rovnaké riadky, tak sú aj pre ne vytvorené samostatné komponenty. Tieto komponenty sa potom často generujú v nejakom *loope*, čo môžeme vidieť na obrázku 5.16. V tomto prípade chceme vygenerovať komponent *ExportSummaryRow* (riadok na obrazovke zhrnutie) pre každú kategóriu, ktorú môžeme v aplikácii odhadovať.

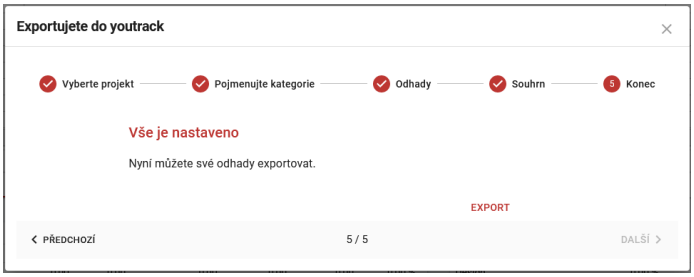
Doteraz sa dalo exportovať odhady iba do HTML alebo CSV formátu. To sa robilo pomocou malého menu, ktoré sa zobrazilo po kliknutí na tlačítko



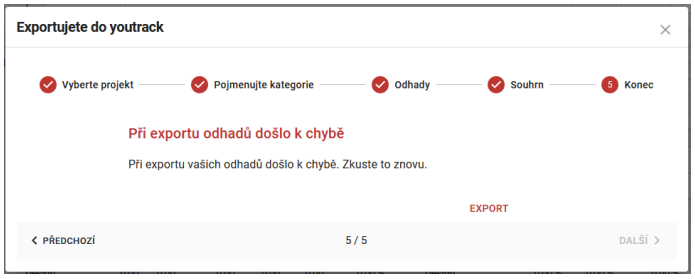

Obr. 5.10: Štvrtá iterácia dizajnu dialógového okna. Pridané pole pre výber vlastného poľa pre odhady.




Obr. 5.11: Štvrtá iterácia dizajnu dialógového okna. Upravená obrazovka so zhrnutím.




Obr. 5.12: Štvrtá iterácia dizajnu dialógového okna. Normálny stav obrazovku v poslednom kroku.




Obr. 5.13: Štvrtá iterácia dizajnu dialógového okna. Obrazovka v chybovom stave.


Obr. 5.14: Štvrtá iterácia dizajnu dialógového okna. Načítanie alebo export odhadov.

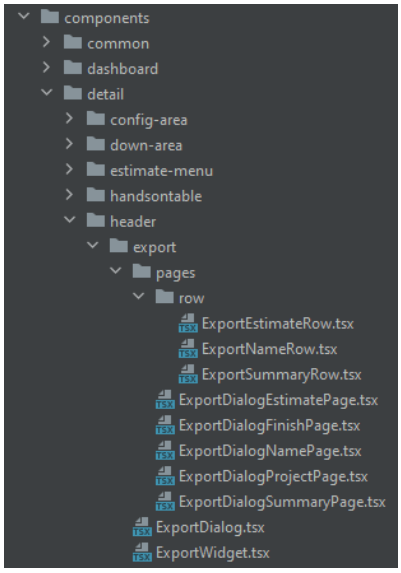
v hlavičke (*header*). Preto som si vytvoril priečinok *export*, v ktorom sú uložené všetky komponenty, ktoré sa využívajú v exportovacom dialógovom okne. Hlavnou „hlavou“ je súbor `ExportWidget.tsx`. Tento súbor renderuje všetkých päť obrazoviek (jedna obrazovka pre každý krok). Nakoniec aj posiela *požiadavok* na exportovanie všetkých odhadov.

Pre jednotný štýl aplikácie sa používa knižnica *Material-UI*, ktorá ponúka veľké množstvo už nadefinovaných komponent. Na obrázku 5.17 vidíme kód na vytvorenie HTML elementu *input*, jeho následné zobrazenie v aplikácii môžeme vidieť na obrázku 5.7 ako pole „Název epicu“. Vytvorenie tohto alebo aj podobného elementu je jednoduché a každý element potom bude používať jednotný štýl. Taktiež, celé dialógové okno je rozložené do mriežky (používa sa na to Material-UI komponent *Grid*). Toto rozloženie je pre nás výhodné najmä preto, lebo všetky objekty v dialógovom okne sa dajú dobre rozdeliť na jednotlivé riadky. Či už je to nadpis, text, nejaké pole alebo kombinácia všetkého.

Aplikácia Estimate je navrhnutá tak, že dokáže zobraziť všetok text v češtine alebo angličtine. K tomuto účelu sa používa knižnica *React-intl*. Každý text alebo hláška, ktorá je použitá je uložená v súboroch `cs.json` a `en.json`. Podľa aktualné zvoleného jazyka sa nám potom zobrazi aplikácia v správnom jazyku.

Na spravovanie stavu aplikácie sa používa knižnica *Redux*. S tým je pevne

spätá knižnica *Redux-Saga*, ktorá funguje ako *middleware* pre Redux. Výhodou je jednoduché odosielanie požiadavkov na naše backend API a následné spracovanie odpovede, ktorú sme dostali.




Obr. 5.15: Štruktúra pridaných súborov na frontende.

```
categories.map(category => {
    return (
    <ExportSummaryRow
        key={category.name}
        category={category}
        possibleType={
            types?.find(type => category.type == type.id)
            }
    />
    );
})
```

Obr. 5.16: Kód, ktorý generuje riadok pre každú kategóriu.

## 5.2 Backend

Backend časť programu sa zaoberá spracovávaním požiadavkov, ktoré prídu z frontendu. Všetky ich vyhodnotiť a následne poslať vlastné požiadavky

```
<OutlinedInput
value={name}
onChange={epicNameOnChange}
margin='dense'
/>
```

Obr. 5.17: Ukážka Material-UI komponentu.

nástrojom určeným na sledovanie projektov. Spracovať odpoveď, ktorú dostane a odoslať informáciu o odpovedi alebo údaje z odpovedi späť na frontend. S nástrojmi budeme komunikovať pomocou ich API, ktoré sme predstavili v 4.6.1 a 4.7.1.

Požiadavky z frontendu sa spracuvávajú v `EstimateController.kt`. Ten má niekoľko metód na pokrytie všetkých požiadavkov, ktoré potrebujeme na export odhadu. Obrázok 5.18 ukazuje, ako vyzerá jeden konkrétny controller, ten sa stará o získavanie projektov (v externej aplikácii), na ktoré má daný užívateľ prístup. Na frontend potom vracia zoznam všetkých projektov.

```
@GetMapping(EXPORT_ESTIMATE_PROJECTS)
fun getExportProjects(
    @RequestParam("system") system: ExportSystem
    ): List<Project> {
    logger.info { "Get user projects" }
    return exportService.getUsersProjects(system)
}
```

Obr. 5.18: Kód controlleru, ktorý získava projekty z externého systému.

Môžeme si všimnúť, kód `logger.info`. Tento kód slúži na logovanie informácii do konzole. Na to je použitá knižnica `MicroUtils/kotlin-logging`.

Ak budeme pokračovať ďalej, dostaneme sa do `ExportService`. Táto service zaisťuje úlohy na servisnej vrstve pre exportovanie odhadov. V našom príklade budeme pokračovať metódou `getExportTypesAndFields`. Táto metóda 5.19 na základe vstupného argumentu *projectId* zavolá metódu správneho klienta (v našom prípade máme iba klientov pre *Gitlab* a *Youtrack*) na získanie typov a vlastných polí, ktoré sa používajú na danom projekte na externom systéme.

*Youtrack* a *gitlab* sú *Factory*. Čiže objekty, ktoré nám vytvoria iné objekty. V našom prípade z nich dostaneme klientov, ktorí dokážu komunikovať s API konkrétneho systému. Do tejto service nám ich vkladá *SpringFramework* pomocou DI. Prečo vôbec používame *Factory*? Na vytvorenie nového klienta je potreba veľa rôznych dát, preto ho vytvárame pomocou *Builder dizajnového*

```
override fun getExportTypesAndFields(
    projectId: String, system: ExportSystem
    ): ExportTypesFields {
    val data = when (system) {
        ExportSystem.YOUTRACK -> {
            val types = youtrack.newClient()
                .getProjectTypes(projectId)
            val fields = youtrack.newClient()
                .getProjectFields(projectId)
            ExportTypesFields(types, fields)
        }
        ExportSystem.GITLAB -> {
            val types = gitlab.newClient()
                .getProjectTypes(projectId)
            ExportTypesFields(types, emptyList())
        }
        else -> {
            throw IllegalArgumentException(
                "Unknown export system $system")
        }
    }

return data
}
```

Obr. 5.19: Kód z ExportService, ktorý je určený na získavanie typov a vlastných polí používaných na projekte, ktorý je na externom systéme.

*vzoru*. Takto dokážeme jednoducho a prehľadne vytvoriť správneho klienta. Príklad vytvorenia webového klienta môžeme vidieť na obrázku 5.20. V tomto konkrétnom prípade potrebujeme nastaviť základnú url adresu, hlavičku na autorizáciu a hlavičky na určenie formátu dát, ktoré buď posielame, alebo prijímame.

Aj tieto Factory aj webový klienti sa nachádzajú v samostatnom module *estimate-integrations*. Tento nový modul slúži pre všetky integrácie, ktoré sú alebo budú v aplikácii Estimate. V našom prípade to sú integrácie na systémy na sledovanie projektov, do ktorých exportujeme naše odhady. Z tohto modulu sa potom posielajú *požiadavky* na externé systémy. Spracuvávajú sa návratové hodnoty a tie sa potom vrátia na servisnú vrstvu, kde sa s nimi môže pracovať ďalej. Aplikácia bola už predtým rozdelená do viacerých modulov. Napríklad *estimate-frontend*, *estimate-domain*...Každý tento modul má na starosti inú činnosť. Rovnako aj estimate-integrations sa stará o zasielanie a spracovávanie

```
private fun buildWebClient(): WebClient {
    return WebClient.builder()
        .defaultHeader(HttpHeaders.AUTHORIZATION,
            "Bearer $permToken")
        .baseUrl(baseUrl)
        .defaultHeader(HttpHeaders.CONTENT_TYPE,
            MediaType.APPLICATION_JSON_VALUE)
        .defaultHeader(HttpHeaders.ACCEPT,
            MediaType.APPLICATION_JSON_VALUE)
        .build()
}
```

Obr. 5.20: Kód určený na vytvorenie webového klienta.

požiadavkov. K tomu potrebuje iné závislosti, ktoré nie sú potrebné nikde inde.

V našom prípade je tou najdôležitejšou závislosťou *Spring Webflux*. Ten dokáže posielať požiadavky na externé systémy. Používame to na posielanie *GET* a *POST* požiadavkov 5.21. Tento univerzálny kód voláme vždy, keď sa snažíme získať zoznam objektov. Dôležité je, že vždy môžeme použiť iný objekt v zozname. Vďaka tomu nám stačilo vytvoriť iba jednu metódu, ktorú môžeme použiť viackrát. Voláme ho ale vždy z webového klienta, ktorý potom ešte môže odpoveď trochu upraviť alebo spracovať a pošle ju späť na servisnú vrstvu. Príklad takéhoto volania je na obrázku 5.22.

```
fun<T> getListRequest(
    url: String,
    client: WebClient,
    responseType: Class<T>): List<T> {
    val result = client
        .get()
        .uri(url)
        .retrieve()
        .bodyToFlux(responseType)
    return result
        .collect(Collectors.toList())
        .share().block()
        ?: throw Exception(
        "Error while getting list from url $url.")
}
```

Obr. 5.21: Kód, ktorý posiela GET požiadavok a získava zoznam objektov.

```
override fun getProjectTypes(projectId: String):
    List<ExportNameAndId> {
    return getListRequest(
        "/projects/$projectId/labels",
        defaultClient,
        ExportNameAndId::class.java)
}
```

Obr. 5.22: Kód vo webovom klientovy, ktorý nám získava typy úloh, ktoré sú na danom projekte v externom systéme.

## 5.3 Testovanie

Písanie kódu je síce dôležité, ale rovnako dôležité je aj jeho testovanie. Jedným zo základných typov testovania sú *unit testy*. Tie kontrolujú činnosť častí kódu, napríklad metód a funkcií. V práci na backende používame mokovací framework *Mockito*. Tento framework nám dovolí nastaviť návratové hodnoty pre konkrétne volania metód alebo funkcií z mokovaných objektov. [29] Na obrázku 5.23 môžeme vidieť príklad unit testu metódy `getUsersProjects`. V tomto konkrétnom príklade sme nastavili tri návratové hodnoty pre tri volania. Takže, ak naša metóda použije jedno z týchto volaní, tak dostane naspäť nastavenú návratovú hodnotu a my sa môžeme sústrediť iba na testovanie funkcionality našej metódy. V našom prípade nám to urýchľuje a zjednodušuje testovanie, pretože nemusíme zapínať a volať externý systém (Youtrack), aby sme z neho získali nejaké hodnoty. Na konci testu si tiež overíme, že metódy, ktoré by mali byť volané, tak volané naozaj aj boli a tie, ktoré volané byť nemajú, tak ani neboli. Rovnako ešte porovnávame výslednú hodnotu s tým, čo očakávame. Ak by metódy neboli volané alebo by sme dostali inú výslednú hodnotu, tak by náš test neprešiel. Takto si vieme jednoducho otestovať všetko správanie naších metód.

K dobrému kódu patrí aj *code review*, v ktorom sa kontroluje kvalita kódu. Toto prebiehalo s vedúcim práce pri každej iterácii. Dokumentovanie kódu je tiež dôležitou činnosťou, a preto sme aj v tejto práci pokračovali s dokumentáciou metód a funkcií tak, ako sa to v minulosti na tomto projekte zaviedlo.

Okrem toho sme si vytvorili lokálny Youtrack server, ktorý sme využívali na vývoj a následné testovanie integrácie, pretože sme sa počas vývoja museli pripojiť na nejaký Youtrack server, ale pripojovať sa na produkčný server by nebolo dobré riešenie. Takto sme si mohli vyskúšať reálnu funkčnosť našej integrácie. Rôzne verzie Youtracku priamo implikujú nutnosť rôznych nastavení aplikácie na základe toho, na akom prostredí je spustená (lokálne, testovacie, produkčné prostredie). Dôležité hodnoty pre každé prostredie vieme nastaviť v konfiguračných súboroch, kde môžeme napríklad nastaviť rozdielne url

adresy pre Youtrack API podľa toho, na akom prostredí sa teraz aplikácia nachádza.

Keďže bola práca vyvíjaná iteratívne, tak bola nová funkčnosť otestovaná každý týždeň vedúcim práce a rovnako počas záverečného stretnutia bola predvedená, a teda aj otestovaná celková implementácia.

```
@Test
fun getUserProjectsYoutrackTest() {
    val user = LoggedUser("testUsername",
                          "Test User",
                          emptySet())
    val projectList = listOf(Project("P-1", "Project1"))

    `when`(userService.getLoggedUser()).thenReturn(user)
    `when`(youtrack.newClient()).thenReturn(youtrackWebClient)
    `when`(youtrackWebClient.getUserProjects(user.username))
        .thenReturn(projectList)

    val result = exportService
        .getUsersProjects(ExportSystem.YOUTRACK)

    assertEquals(result, projectList)
    verify(userService, times(1)).getLoggedUser()
    verify(youtrackWebClient, times(1))
        .getUserProjects(user.username)
    verify(gitlabWebClient, never())
        .getUserProjects(user.username)
}
```

Obr. 5.23: Príklad unit testu metódy getUserProjects.

## 5.4 Vysledný stav

V minulých častiach sme rozoberali to, ako boli naprogramované jednotlivé časti integrácie. Teraz sa pozrieme na to, ako nakoniec vyzerá prechod takýmto exportovacím dialógovým oknom.

Ak je užívateľ spokojný s vytvorenými odhadmi, tak ich musí najprv uložiť. Po uložení môže otvoriť menu na export, kde má viacero možností: HTML, CSV, Youtrack, Gitlab. Užívateľ teda môže odhady exportovať do jedného z dvoch externých systémov. Po vybratí a kliknutí na danú možnosť sa otvorí dialógové okno s prvou stránkou. Ak si užívateľ vybral systém Youtrack, tak si bude musieť okrem projektu, do ktorého chce odhady exportovať, vybrať

aj pole (*custom field*), do ktoré sa vkladá časový odhad. Na obrázku 5.10 vidíme pole pre systém Youtrack (*Pole pro odhad*). Až po vybraní obidvoch možností sa užívateľovi umožní pokračovať na ďalšiu stránku. Na všetkých ostatných stránkach sú už prednastavené základné hodnoty, takže sa užívateľ môže iba prekliknúť na poslednú stránku a vyexportovať odhad.

Ak sa rozhodne zostať a všetko si nastaviť, tak sa mu naskytne možnosť zadania vlastného ID pre hlavnú úlohu (*epic*) alebo, ak tak neurobí, aspoň nastavenie mena tohto epicu. Potom si môže nastaviť prefix pre názvy úloh z každej kategórie alebo ID vlastnej úlohy, ktorá sa nastaví ako rodič všetkých nových úloh. Tiež si môže nastaviť konkrétny typ, ktorý budú mať úlohy v danej kategórii. Vždy podľa toho, čo je podporované na konkrétnom projekte.

Na tretej stránke si užívateľ môže nastaviť typ časového odhadu teda, či sa bude vo výsledku zobrazovať priemerný, minimálny, maximálny, najpravdepodobnejší alebo očakávaný odhadovaný čas (všetky tieto možnosti ponúka aplikácia Estimate).

Štvrtá stránka potom užívateľovi zobrazí všetky jeho nastavenia. Užívateľ si tak môže skontrolovať, že všetko nastavil správne, keď tak sa vrátiť a niektoré veci upraviť.

Posledná stránka už umožňuje užívateľovi všetko exportovať. Podľa toho, či sa export podaril sa potom užívateľovi zobrazí iná informácia. V prípade, že sa odhad podaril sa dialógové okno zavrie a vpravom dolnom rohu sa zobrazí hláška v zelenej farbe, ktorá informuje o úspechu.

Nespomenul som ešte varianty. Tie boli tiež implementované do práce a to tak, že ak užívateľ varianty používa a prejde na export, tak sa mu budú exportovať iba tie, ktoré sú aktívne. Ak by chcel importovať všetky odhady bez ohľadu na varianty, tak musí buď všetky varianty nastaviť ako aktívne alebo neaktívne.

Ako samozrejmosť tiež beriem implementáciu časovej jednotky odhadov (man-hours alebo man-days). To znamená, že ak užívateľ v projekte používa man-days, tak aj hodnoty v exporte budú v man-days.

## 5.5 Stretnutie po implementácii

V náväznosti na stretnutie, ktoré sme mali ešte pred implementáciou riešenia 4.3, sme po naimplementovaní riešenia znova pripravili stretnutie s rovnakou osobou. Cieľom bolo získať spätnú väzbu a možno aj nejaké rady.

Stretnutie začalo dobre. Predstavili sme celé riešenie vytvárania exportov do programu Youtrack. Aj sme si vyskúšali jeden projekt z aplikácie Estimate vyexportovať.

Odpoveď bola veľmi kladná, všetky dôležité body, ktoré boli spomenuté v úvodnom stretnutí boli naimplementované. Keďže členovia mali skúsenosti s Youtrack API, ponúkli nám dobré rady. Tou prvou bolo, že pole pre časový odhad v Youtracku si môže každý projekt premenovať alebo aj zmazať, a preto

by sa nemalo v exporte používať pevne pole *Estimate*. Ďalšou radou bolo pridanie piatej obrazovky, ktorá by zobrazovala, že sa odhad načítava, a teda sa aplikácia nezasekla. Obidve tieto rady sme využili a zapracovali ich v štvrtej iterácii návrhu 5.1.1 a následne naimplementovali.

Okrem toho ponúkol aj svoju víziu do budúcnosti, teda veci, ktoré by sa ešte mohli v budúcnosti naimplementovať. Napríklad vytvorenie šablóny, ktorú by si užívateľ vedel raz na projekte nastaviť, a potom by ju mohol stále používať na nastavenie hodnôt v exportnom dialógovom okne. Pridať práva v aplikácii Estimate, ktorými by sme mohli pridať nového človeka do projektu, a ten by potom mohol projekt exportovať. Zobrazenie projektov v prvej obrazovke dialógového okna 5.6 podľa toho, či má užívateľ právo na pridávanie odhadov v danej externej aplikácii (napr. Youtrack).

# Záver

Na začiatku práce sme si určili ciele. Návrh a implementácia integrácie, určenej na export odhadov, analýza programov, ktoré bude integrácia využívať a predstavenie problematiky tvorby odhadov a riadenia projektu. Tieto ciele sme postupne napĺňali.

Najprv sme si predstavili problematiku odhadov a riadenia projektov. Zistili sme, že obidve tieto témy sú úzko spojené. Následne sme si zanalyzovali terajšiu aplikáciu *Estimate* a ďalšie aplikácie (Youtrack a Gitlab), ktoré sú používané vo firme, a teda podporujú proces vedenia softvérového projektu.

Tieto znalosti sme potom využili pri návrhu a implementácii vlastného riešenia, ktoré rieši nedostatok aplikácie Estimate. Ten bol, že aplikácia nepodporovala exportovanie odhadov priamo do externých programov, ktoré slúžia na riadenie projektov. Toto riešenie tiež ponúka rôzne možnosti úprav exportovaných odhadov (nastavenie záruky, názvu a ďalšie možnosti). Pred a po implementácii sme sa stretli s reálnymi používateľmi aplikácie Estimate, od ktorých sme získali cennú spätnú väzbu, ktorú sme ešte dokázali využiť na zlepšenie nášho riešenia.

K tomuto rozšíreniu sa dá do budúcnosti pridať ešte niekoľko vecí. Najzaujímavejšou je vytvorenie šablón, ktoré by si užívateľ mohol vyplniť iba raz, a potom by ich mohol použiť pri viacerých exportoch. Takto by sa zamedzilo opakovaniu zadávania rovnakých hodnôt, ak exportujeme z jedného projektu viackrát. Ďalšou možnosťou je pridanie iných externých programov, do ktorých by sme mohli exportovať odhady.

# Bibliografia


1. BUEHRING, Simon. What is a project? In: *What is a PRINCE2?* [Online]. 2022 [cit. 2022-04-04]. Dostupné z : `https://www.whatisprince2.net/prince2-project-management#project`.
2. MCCONNELL, Steve. *Software Project Survival Guide*. Redmond: Microsoft press, 1998. ISBN 1-57231-621-7.
3. BUEHRING, Simon. 6 Basic SDLC Methodologies: Which One is Best? In: *Robert Half* [online]. 2021 [cit. 2022-04-04]. Dostupné z : `https://www.roberthalf.com/blog/salaries-and-skills/6-basic-sdlc-methodologies-which-one-is-best`.
4. The project triangle. In: *Microsoft Support* [online]. 2022 [cit. 2022-04-04]. Dostupné z : `https://support.microsoft.com/en-us/office/the-project-triangle-8c892e06-d761-4d40-8e1f-17b33fdcf810`.
5. INSTITUTE, Project Management. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) Fifth Edition*. Newton Square: Project Management Institute, 2013. ISBN 978-1-935589-67-9.
6. BROOKS Frederick P., Jr. *The Mythical man-month: essays on software engineering*. 2. vyd. New York: Addison-Wesley, 1975. ISBN 0-201-00650-2.
7. HAUGAN, Gregory T. *Effective Work Breakdown Structures*. Vienna: Berrett-Koehler Publishers, 2001. ISBN 1-56726-135-3.
8. BLAIR, Stephen. *A Guide To Evaluating a Bug Tracking System* [online]. MetaQuest Software Inc, 2004 [cit. 2022-04-04]. Dostupné z : `https://web.archive.org/web/20060419180529/http://www.metaquest.com/Downloads/Docs/EvaluatingABugTrackingSystem.pdf`.
9. *Slovníkový portál Jazykovedného ústavu Ľ. Štúra SAV* [online]. 2004 [cit. 2022-04-04]. Dostupné z : `https://slovnik.juls.savba.sk/?w=odhad`.
10. MCCONNELL, Steve. *Software estimation: demystifying the black art*. Redmond: Microsoft press, 2006. ISBN 0-73560-535-1.

11. Man-hour. In: *Merriam-Webster.com Dictionary, Merriam-Webster* [online]. [B.r.] [cit. 2022-04-04]. Dostupné z : https://www.merriam-webster.com/dictionary/man-hour.

12. KITCHENHAM, Barbara; PFLEEGER, Shari; MCCOLL, Beth; EAGAN, Suzanne. A case study of maintenance and development estimation accuracy. *Journal of Systems and Software*. 2002.

13. TypeScript is JavaScript with syntax for types. In: *TypeScript* [online]. 2022 [cit. 2022-04-04]. Dostupné z : https://www.typescriptlang.org/.

14. React. In: *React* [online]. 2022 [cit. 2022-04-04]. Dostupné z : https://reactjs.org/.

15. Move faster with intuitive React UI tools. In: *Material UI* [online]. 2022 [cit. 2022-04-04]. Dostupné z : https://mui.com/.

16. A modern programming language that makes developers happier. In: *Kotlin* [online]. [B.r.] [cit. 2022-04-04]. Dostupné z : https://kotlinlang.org.

17. Comparison to Java. In: *Kotlin* [online]. 2022 [cit. 2022-04-04]. Dostupné z : https://kotlinlang.org/docs/comparison-to-java.html.

18. Kotlin is now Google's preferred language for Android app development. In: *TechCrunch* [online]. 2019 [cit. 2022-04-04]. Dostupné z : https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development.

19. Spring Framework 1.0 Final Released. In: *Spring* [online]. 2004 [cit. 2022-04-04]. Dostupné z : https://spring.io/blog/2004/03/24/spring-framework-1-0-final-released.

20. Spring Framework. In: *Spring* [online]. 2022 [cit. 2022-04-04]. Dostupné z : https://spring.io/projects/spring-framework.

21. Spring Boot. In: *Spring* [online]. 2022 [cit. 2022-04-04]. Dostupné z : https://spring.io/projects/spring-boot.

22. PANSKÝ, Petr. *Analýza a optimalizace UX nástroje pro podporu tvorby odhadů pracnosti*. Praha, 2021. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.

23. Powerful project management for all your teams. In: *YouTrack* [online]. [B.r.] [cit. 2022-04-04]. Dostupné z : https://www.jetbrains.com/youtrack/.

24. YouTrack REST API. In: *YouTrack* [online]. 2022 [cit. 2022-04-04]. Dostupné z : https://www.jetbrains.com/help/youtrack/devportal/youtrack-rest-api.html.

25. Log in to YouTrack. In: *YouTrack* [online]. 2022 [cit. 2022-04-04]. Dostupné z : `https://www.jetbrains.com/help/youtrack/devportal/api-log-in-to-youtrack.html`.

26. Documentation Overview. In: *Curl* [online]. [B.r.] [cit. 2022-04-04]. Dostupné z : `https://curl.se/docs`.

27. The One DevOps Platform. In: *GitLab* [online]. 2022 [cit. 2022-04-04]. Dostupné z : `https://about.gitlab.com/`.

28. Issues API. In: *Gitlab* [online]. [B.r.] [cit. 2022-04-04]. Dostupné z : `https://docs.gitlab.com/ee/api/issues.html`.

29. More info. In: *Mockito* [online]. [B.r.] [cit. 2022-04-04]. Dostupné z : `https://site.mockito.org/#more`.

Dodatok A

# Zoznam použitých skratiek

**API** Application programming interface

**DI** Dependency injection

**HTTP** Hypertext Transfer Protocol

**JSON** JavaScript Object Notation

**JVM** Java Virtual Machine

**MD** Man-day

**MH** Man-hour

**REST** Representational state transfer

**SQL** Structured Query Language

**TS** TypeScript

**WBS** Work breakdown Structure

Dodatok B

# Obsah priloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF