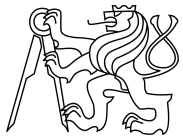
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Aplikace pro podporu tvorby odhadů pracnosti softwarových projektů
Student:	Juraj Polačok
Vedoucí:	Ing. Michal Petřík
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je návrh a implementace webové aplikace pro odhady pracnosti SW projektu na základě metod a postupů používaných ve firmě Profinit.

1. Analyzujte používané nástroje na tvorbu odhadů.
2. Seznamte se s metodikou odhadů zadavatele a její implementací.
3. Navrhněte architekturu a GUI požadované aplikace.
4. Zvolte technologie pro 3vrstvou web aplikaci a využijte je v implementaci.
5. Vytvořte dokumentaci a aplikaci otestujte.
6. Zhodnoťte řešení s ohledem na možný budoucí rozvoj.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 5. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalárska práca

Aplikácia pre podporu tvorby odhadov pracnosti softwarových projektov

Juraj Polačok

Vedúci práce: Ing. Michal Petřík

12. mája 2017

Poďakovanie

Chcel by som poďakovať Ing. Michalovi Petříkovi, vedúcemu bakalárskej práce, za odborné vedenie, pomoc a ochotu pri tvorbe tejto práce. Ďalej by som chcel poďakovať firme Profinit EU, s.r.o. za možnosť realizácie tejto práce. V neposlednom rade ďakujem svojim rodičom a priateľke za podporu počas celého štúdia.

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 užívaniu 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 užívať.

Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, a za akýmkoľvek účelom (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené. Každá osoba, ktorá využije vyššie uvedenú licenciu, sa však zaväzuje priradiť každému dielu, ktoré vznikne (čo i len čiastočne) na základe Diela, úpravou Diela, spojením Diela s iným dielom, zaradením Diela do diela súborného či zpracovaním Diela (vrátane prekladu), licenciu aspoň vo vyššie uvedenom rozsahu a zároveň sa zaväzuje sprístupniť zdrojový kód takého diela aspoň zrovnateľným spôsobom a v zrovnateľnom rozsahu ako je zprístupnený zdrojový kód Diela.

V Prahe 12. mája 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2017 Juraj Polačok. Všetky práva vyhradené.
Táto práca vznikla ako školské dielo na FIT ČVUT v Prahe. Práca je chránená medzinárodnými predpismi a zmluvami o autorskom práve a právach súvisiacich s autorským právom. Na jej využitie, s výnimkou bezplatných zákonných licencií, je nutný súhlas autora.

Odkaz na túto prácu

Polačok, Juraj. *Aplikácia pre podporu tvorby odhadov pracnosti softwarových projektov*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Táto bakalárska práca sa zaoberá tvorbou aplikácie pre podporu tvorby odhadov pracnosti softwarových projektov. Prácu možno rozdeliť na tri časti. Prvou častou je samotná analýza momentálne používaných nástrojov na tvorbu odhadov a oboznámenie sa s metodikou odhadov riešených vo firme Profinit. Druhou je návrh architektúry a užívateľského rozhrania. Pri návrhu architektúry je dôležité aby aplikácia bola webová a využívala 3vrstvú architektúru. Poslednou časťou je zhodnotenie riešenia s ohľadom na budúci vývoj tejto aplikácie. Aplikácia je vyvijána spolu s kolegom Milanom Vanclom, ktorý sa zaoberá dátový model.

Klíčová slova tvorba odhadov, webová aplikácia, Profinit, Java, Spring Boot, Vaadin

Abstract

This bachelor thesis deals with application Web Based Software Project Effort Estimation Tool. The thesis can be divided into three parts. The first part consists of analysis of currently used tools for managing estimations and aquire knowledge of methodics that are used in Profinit. The second part describes

creating design and user interface. It is important to use 3 tier web architecture as application architecture. The last part contains evaluation of created solution and discussion about further development of this application. This application is also developed by my colleague Milan Vancl, who deals with data model of this application.

Keywords estimation tool, web application, Profinit, Java, Spring Boot, Vaadin

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Hlavným cieľom tvorby odhadov pracnosti je určiť pracnosť potrebnú na dokončenie aktivity, napríklad projektu. Ak poznáme pracnosť, vieme k jednotlivým podaktivitám priradiť zdroje, z čoho je následne možné vypočítať čas potrebný na dokončenie celého projektu. Cieľom týchto odhadov je sa čo najviac priblížiť realite.

Jednou z metodík ako tvoriť odhad je porovnávaním s minulými projektami, ktoré majú podobné vlastnosti, preto je potrebné mať informácie o minulých odhadoch. V priebehu projektu sa odhady upresňujú, neustále sa menia. Tieto zmeny je nutné ukladať. Pre správne porovnanie potrebujeme stanoviť pravidlá, ktoré budú mať za dôsledok pevnú štruktúru, čo bude vo výsledku znamenať konzistentné odhady, ktoré budeme vedieť porovnávať.

Primárnou úlohou tejto bakalárskej práce je evidencia odhadov a ich správa. Samotná tvorba odhadov nie je cieľom tejto práce. Aplikácia sa vyvíja pod názvom Profinit Estimate a je výsledkom nielen tejto bakalárskej práce, ale aj práce kolegu Milana Vancla. Jeho časť sa zameriava na vytvorenie dátového modelu. Táto práca sa zaoberá využívaním tohto modelu a prácou s ním, ako aj návrhom architektúry a vytvorením užívateľského rozhrania pre tvorbu a správu odhadov.

Cieľom tejto aplikácie je vytvárať a spravovať odhady pracnosti podobným spôsobom, akým to bolo doposiaľ. Pri návrhu užívateľského rozhrania je dôležité vychádzať z tejto skutočnosti. Je potrebné, aby boli zachované isté princípy, ktoré fungovali v minulom riešení, napríklad možnosť relatívneho odkazovania sa na iné hodnoty, používanie jednoduchých matematických operácií, jednoduchá navigácia a intuitívne rozhranie. Výsledkom je aplikácia, v ktorej sa odhady spravujú jednoduchšie než doposiaľ.

Kapitola 1

Motivácia a cieľ práce

1.1 Motivácia

Táto práca má za úlohu vytvoriť aplikáciu, v ktorej je možné spravovať odhady pracnosti. Aplikácia má potenciál zjednodušiť prácu vytvárania odhadov pre firmu Profinit. Hlavným prínosom tejto aplikácie je jednotná správa všetkých odhadov pracnosti a ich vyhľadávanie. Aplikáciu bude používať viacero ľudí podieľajúcich sa na tvorbe odhadov pracnosti.

1.2 Cieľ práce

Cieľom tejto bakalárskej práce je:

- Analýza metodiky odhadov používaných vo firme Profinit
- Analýza používaných nástrojov pre tvorenie odhadov pracnosti
- Voľba vhodných technológií pre webovú aplikáciu
- Návrh grafického rozhrania (ďalej len GUI)
- Implementácia navrhnutých obrazoviek
- Otestovanie aplikácie a vytvorenie dokumentácie

Kapitola 2

Úvod k tvorbe odhadov pracnosti

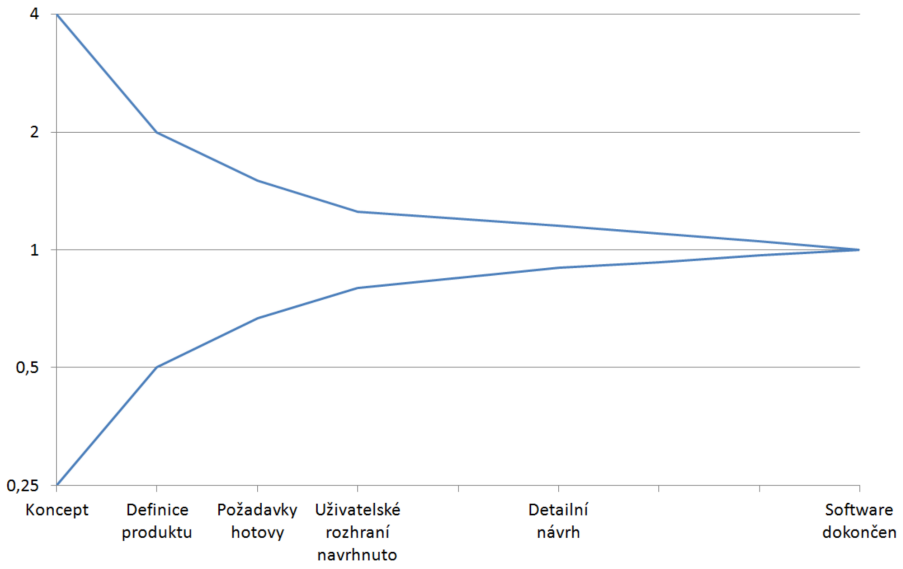
2.1 Prečo evidovať odhady

Jednou z možností ako sa dostávajú softwarové firmy k zákazkám je výberové konanie, v ktorom firma, prípadne štát, vypíše tender na vývoj aplikácie. Táto ponuka je rozposlaná viacerým softwarovým firmám, ktoré na základe špecifických požiadavok vypracujú ponuku, v ktorej určia, za koľko peňazí a za aké časové obdobie sú danú aplikáciu schopní dodať. V niektorých prípadoch môže byť požiadavkou termín dodania aplikácie, to znamená, že sa od dodávateľskej firmy požaduje len cenová ponuka. Následne si firma, prípadne štát, vyberie najlepšiu ponuku. Vzniká otázka, ako softwarové firmy vytvoria ponuku, resp. ako vypočítajú náklady na projekt a potrebný čas na jeho dokončenie. Prvou fázou je vytvorenie odhadu pracnosti, v ktorom sa odhadne pracnosť potrebná na vytvorenie projektu. Ak poznáme pracnosť a poznáme početnosť tímu, ktorý bude projekt vyvíjať, je možné zistiť požadované atribúty: cenu projektu a približný dátum dodania projektu.

Existuje množstvo metodík, ktoré sa snažia určiť spôsob, akým danú pracnosť vypočítať. Veľa metodík sa opiera o historické dáta minulých projektov. Princíp je jednoduchý. Porovná sa podobný minulý projekt s novým projektom a urobí sa zjednodušený predpoklad predpokladajúci rovnaký priebeh.[1, str. 93]. K tomu, aby sme takýto postup mohli aplikovať, je v prvom rade nutné disponovať týmito historickými dátami.

2.2 Kužeľ neistoty

Vývoj softwaru sa skladá z tisícok rozhodnutí ohľadom rôznych funkcií týkajúcich sa aplikácie. Nejasnosti samotného odhadu pramenia z nejasnosti, akým spôsobom budú dané funkcie riešené.[1, str. 35] Čím viac informácií je k dis-



Obr. 2.1: Kužel neistoty prevzaté z [2]

pozícií, tým presnejšie vieme odhadnúť pracnosť danej funkcie, čím upresníme odhad. Bol uskutočnený výskum, ktorý zistil, že projektové odhady sú predmetom predpovedateľného množstva neistoty v rôznych fázach projektu. Túto neistotu zachycuje kužeľ neistoty.[1, str. 35]

Na obrázku 2.1 je zachytená závislosť nepresnosti od fázy projektu. Vertikálna os obsahuje odchýlku odhadu od skutočnosti. Horizontálna os zachycuje rôzne fázy projektu. Spôsob čítania grafu je nasledovný. Ak reálna pracnosť projektu budú 4 človekomesiace, odhad pracnosti v konceptuálnej časti bude v intervale od 1 človekomesiaca po 16 človekomesiacov.

2.3 Štruktúrovaný expertný posudok

Pri odhadovaní danej aktivity je dôležité uvažovať v intervaloch, nie v konkrétnej hodnote. Je potrebné vytvoriť odhad pracnosti pre množinu problémov. Vytvorením tohto odhadu bude poverený vývojár, ktorý na tomto probléme bude pracovať. Vývojár vypracuje odhad pracnosti pre jednotlivé problémy spôsobom „aktivita — potrebný čas“. Tu je namieste otázka, ako tento vývojár postupoval. Je žiadúce spýtať sa, či uvažoval o bezproblémovom alebo o najhoršom možnom scenári. Znova bude poverený vytvorením tohto odhadu s tým, aby jednotlivým problémom nepridelil konkrétne číslo, ale interval — ak všetko pôjde bezproblémovo, ak pri riešení nastanú rôzne komplikácie. Následne sa porovná jeho prvý odhad s druhým. McConnell [1, str. 108] uvádza, že vo väčšine prípadov bude platiť, že jeho prvotné odhady počítali s bezproblémovým scenárom. V niektorých prípadoch dokonca nastane situácia,

v ktorej prvotný odhad problému X bol odhadnutý menším číslom než druhý odhad problému X. Je to tým, že uvažovanie o najhoršom scenári niekedy vedie k nutnosti prehodnotiť aj ideálnu situáciu.[1, str. 108]

Ak je k dispozícií interval pracnosti problému, vzniká otázka, ktoré číslo použiť. Použiť priemer nie je správne riešenie, pretože najhorší scenár je mnohonásobne horší ako očakávaný scenár. Takéto uvažovanie by viedlo vo výsledku k preceneniu odhadu pracnosti, teda reálna práca by trvala výrazne kratšie.[1, str. 108] K tomu slúži technika zvaná Program Evaluation and Review Technique (ďalej už len PERT). K využívaniu metódy PERT je potrebné pridať ďalší atribút k odhadu pracnosti. Tým je najpravdepodobnejší scenár, ktorý sa určí na základe odborného posudku (založenom na skúsenostiach). Očakávaný odhad pracnosti sa následne vypočíta vzťahom

$$\frac{NajhorsiScenar + BezproblemovyScenar + 4 * NajpravdepodnejsiScenar}{6}$$

Táto technika sa nazýva PERT. Zdroj [1, str. 109] uvádza, že niektorí odborníci na odhady doporučujú tento PERT vzťah upraviť na

$$\frac{2 * NajhorsiScenar + BezproblemovyScenar + 3 * NajpravdepodnejsiScenar}{6}$$

Argumentom, prečo je tento vzťah zmenený, je fakt, že ľudia majú sklony prikláňať odhady viac k optimistickému variantu.[3]

2.4 Metoda bottom-up

Metóda zdola-nahor alebo bottom-up funguje na princípe rozpadu jednotlivých aktivít na množinu podaktivít. Ako funguje rozpad je možné názorne ukázať na príklade. Úlohou je odhadnúť pracnosť projektu X. Projekt X sa určite bude skladať z častí ako je analýza, design, implementácia. Tieto časti sa budú skladať z menších častí. Čím detailnejší je rozpad, tým menšia chyba je zanesená do výsledku.

Takto funguje princíp dekompozície — rozpadu. Následne sa z týchto menších aktivít spraví súčet, vyplnia sa nadkategórie a takto sa postupuje rekurzívne, až kým sa nedôjde k samotnému projektu X. Takémuto rozpadu sa hovorí tiež Work breakdown structure (ďalej už len WBS).

Je zaujímavé, že takouto metodikou je možné dostať sa k relatívne presnému odhadu, i keď odhad samotných aktivít nemusí byť presný. Dôvod, prečo to funguje je objasnený v 2.5.

2.4.1 Kontrolný zoznam

Väčšina projektov realizovaných softwarovými firmami sa dá rozdeliť do kategórií, ktorými sú analýza, design, implementácia, testovanie, dodávka. Je vhodné vytvoriť kontrolný zoznam, ktorý špecifikuje zoznam aktivít, ktoré sa

často vyskytujú v danej kategórií. Pri tvorení nového odhadu je vhodné si tento zoznam prejsť a skontrolovať, či sa na nejakú podaktivitu nezabudlo. Štúdie predpovedania v rozličných oboroch zistili, že jednoduché kontrolné zoznamy zvýšia presnosť odhadov pripomínaním vecí, na ktoré by ľudia mohli zabudnúť.[1, str. 110]

2.5 Zákon veľkých čísel

Odhady spravené metodikou opísanou v 2.4 ťažia zo štatistickej vlasnosti nazývanej „Zákon veľkých čísel“. Ak by sa odhad nečlenil na menšie aktivity a rovno by sa odhadla jeho pracnosť, tento odhad by bol oproti realite buď podhodnotený alebo nadhodnotený. Pri odhade pracnosti menších aktivít je odhad opäť zaťažený chybou, ale platí, že niektoré aktivity boli podhodnotené a niektoré nadhodnotené. Vo výsledku sa tieto podhodnotenia a nadhotnotenia navzájom vyrušia.[1, str. 115]

Z teoretického hľadiska by mal tento predpoklad fungovať. V minulosti sa uskutočnil výskum, ktorý potvrdil túto teóriu aj v praxi.[1, str. 115]

2.6 V akých jednotkách odhadovať

Pri odhadovaní pracnosti sa stretávame s jednotkami „man-day“ (ďalej už len MD) čo v preklade znamená človekodeň, prípadne „man-hour“ (ďalej už len MH) čo v preklade znamená človekohodina. MH je jednotka, ktorá vyjadruje počet hodín potrebných na dokončenie aktivity. Medzi MH a MD platí vzťah

$$MD = 8 * MH$$

Ak odhadujeme aktivitu, ktorej vykonanie bude trvať približne 800MH, je názornejšie povedať 100MD. V niektorých projektoch sa možno zriedkavo stretnúť s pojmom „man-month“, čo v preklade znamená človekomesiac.

Kapitola 3

Analýza používaných nástrojov firmou Profinit

3.1 Metodika používana vo firme Profinit

Vo firme Profinit je používaná kombinácia rôznych metodík. Najčastejšie sa používa kombinácia metodiky bottom-up viz 2.4 a metodiky expertného posudku viz 2.3. Pri tvorení odhadov sa tiež používajú varianty, viz 3.1.1, predpoklady, viz 3.1.2 a záruka viz 3.1.3.

3.1.1 Variant

Pre najlepšie pochopenie variantu je použitý názorný príklad. Úlohou je implementovať kalkulátor, ktorého parametrom je reťazec. Odhad pracnosti tejto funkcie samozrejme závisí na tom, čo všetko tento kalkulátor dokáže. Ak by kalkulátor zvládal len jednoduché operácie typu $5 + 2 + 3$, implementácia je výrazne jednoduchšia v porovnaní s alternatívou, v ktorej je potrebné vyhodnocovanie priority operátorov, priority výrazov v zátvorke. Preto je vhodné pri tvorení odhadu pracnosti používanie variantov.

Takéto problémy sa pri odhadovaní vyskytujú veľmi často, preto je podstatné vytvárať pre jednu konkrétnu funkciu odhad pracnosti s viacerými variantmi. V tejto úlohe by sa odhad pracnosti rozdelil na dva varianty. V prvom variante by sa uvažovalo len o jednoduchej implementácií, v druhom o zložitejšej, pričom by tieto aktivity mali rozličné odhady pracnosti. Typický scenár je tiež požiadavka zákazníka na vytvorenie odhadu s viacerými variantmi. Zákazník potrebuje funkcionalitu a od softwarovej firmy žiada vypracovanie odhadu pracnosti na základe týchto variantov.

3.1.2 Predpoklad a obmedzujúce podmienky

Pri vytváraní nového odhadu je potrebné vedieť, čo sa odhaduje, a na základe čoho. Čím viac informácií o výslednom projekte je známych, tým presnejšie je možné vytvoriť odhad pracnosti.

Problémom je, že vo väčšine prípadov sú detailné informácie pre projekt neznáme. Ak by bolo úlohou implementovať webovú aplikáciu, ku ktorej zákazník poskytol minimum informácií, je vytvorenie odhadu pracnosti, ktorý by odpovedal reálnym hodnotám, takmer nemožné. Vytvárať odhad pracnosti v takomto prípade nemá zmysel. K tomu, aby bolo možné urobiť odhad presnejší, je nutné si dodefinovať nejaké obmedzenia a predpoklady, z ktorých sa bude vychádzať pri samotnej tvorbe. Ideálne je, ak tieto predpoklady a obmedzenia poskytne zákazník. V prípade nedostatku týchto informácií od samotného zákazníka je potrebné si dodefinovať vlastné predpoklady a vlastné obmedzenia s tým, že následne je možné tento odhad prezentovať zákazníkovi a argumentovať, prečo tento odhad vyzerá práve takto.

Príkladom obmedzujúcej podmienky je „Design obrazoviek nám bude dodaný zákazníkom“. Predpokladom môže byť „Obrazovka prihlásenia nebude obsahovať možnosť registrácie“. Po vytvorení odhadu s predpokladmi sa výsledný odhad pracnosti ukáže zákazníkovi, ktorý predpoklady potvrdí alebo zamietne. V prípade zamietnutia je potrebné nájsť so zákazníkom konsensus a predpoklad upraviť tak, aby spĺňal zákazníkove požiadavky, a na základe týchto nových predpokladov vytvoriť nový odhad pracnosti.

Niekedy sa stane, že tieto predpoklady nie je možné dodefinovať. Vtedy je potrebné použiť inú metodiku, napríklad T-shirt sizing (táto metodika nie je predmetom záujmu tejto bakalárskej práce).

3.1.3 Záruka

Po odovzdaní aplikácie zákazníkovi tento vývoj pre softwarovú firmu nekončí. Po nasadení do produkcie nastáva fáza, v ktorej je aplikácia v záruke, teda prípadné chyby v aplikácii musia byť opravené. Záruka sa počíta ako percentuálna časť jednotlivých kategórií. Pre správne nastavenie tejto hodnoty je potrebné vychádzať z minulých projektov, kde je ľahko dohľadateľné, aká pracnosť bola potrebná po nasadení do produkcie. Pre presnejší výpočet je ideálne, ak sa nastaví záruka pre každú kategóriu zvlášť. Vo firme Profinit existuje len jedno číslo vyjadrujúce záruku, ktoré je spoločné pre všetky kategórie.

Záruka nie je prítomná stále. Tento atribút sa počíta len v prípade, že dodávateľ záruku poskytuje.

3.2 Spôsob evidovania odhadov

Evidencia odhadov vo firme Profinit je riešená pomocou programu Microsoft Excel. Každý odhad je uložený v samostatnom súbore. Všetky odhady majú

Testování							
Varianta	Popis činnosti	Min (MD)	Max (MD)	Nej. prav. (MD)	Prům. (MD)	Oček. (MD)	
Biom podpis	[illegible]	6,00	8,00	7,00	7,00	7,00	
Biom podpis	[illegible]	0,50	1,00	0,75	0,75	0,75	
Biom podpis	[illegible]	3,00	5,00	4,00	4,00	4,00	
Biom podpis	[illegible]	0,75	1,25	1,00	1,00	1,00	
Biom podpis	[illegible]	2,50	3,50	3,00	3,00	3,00	
Biom podpis	[illegible]	0,50	0,75	0,63	0,63	0,63	
Biom podpis	[illegible]	0,25	0,50	0,38	0,38	0,38	
Biom podpis	[illegible]	1,00	2,00	1,50	1,50	1,50	
Biom podpis	[illegible]	1,00	2,00	1,50	1,50	1,50	
Biom podpis	[illegible]	1,50	2,50	2,00	2,00	2,00	
Biom podpis	[illegible]	1,75	2,25	2,00	2,00	2,00	
Biom podpis	[illegible]	1,50	2,50	2,00	2,00	2,00	
Archivace dok	[illegible]	2,50	3,50	3,00	3,00	3,00	
Mazání dok	[illegible]	0,50	1,00	0,75	0,75	0,75	
Načítání dok	[illegible]	2,00	4,00	3,00	3,00	3,00	
Distribuce	[illegible]	2,50	3,50	3,00	3,00	3,00	
Distribuce	[illegible]	0,50	1,00	0,75	0,75	0,75	

Obr. 3.1: Aktuálny spôsob vyplňovania položiek

Checklist testování
Tvorba testovacích scénářů pro zákazníka.
Otestování na vývojovém prostředí, otestování na QA prostředí. Zde musíme počítat i s časem, kdy je nutné seznámit se s problematikou, abychom vůbec mohli testovat - dalším aspektem může být:
Musíme pro testování speciálně nastavit prostředí (například instalace Selenium, SoapUI, jiný prohlížeč …)?
Musíme spouštět dávky nebo jinak modifikovat vývojové prostředí (například specifické hodnoty v databázi, emulace stavu aplikace, apod…)?
Potřebujeme rozhraní třetí strany (s tím mohou souviset certifikáty, přístupové údaje, …).
Budeme vytvářet nové regresní testy/upravovat stávající?
Testování finální dodávky před odesláním.
Kvalifikační testování.
Podpora akceptačního testování (testování na straně zákazníka).
Pokud výkonnostní požadavky uvádíme v analýze, musíme se jimi zabývat i v rámci testování.
Zde uvažujeme primárně testování nově přidané funkcionality, dále se testování může objevit i ve vlastní dodávce.

Obr. 3.2: Aktuálny spôsob vyplňovania kontrolného zoznamu

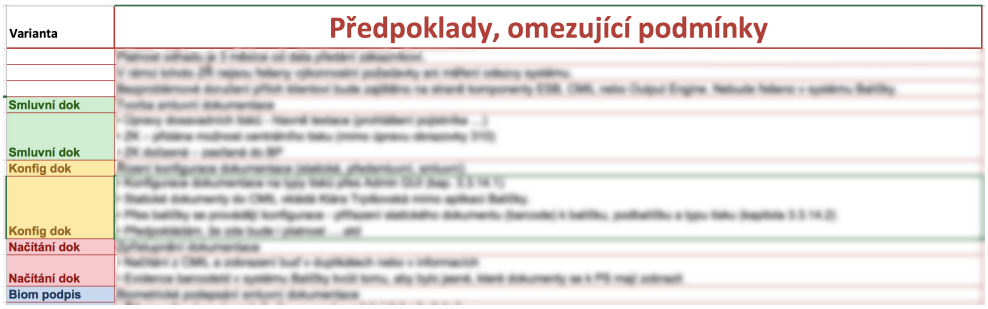
jednotnú štruktúru. Súbor obsahuje záložky, pričom každá záložka má iný význam.

3.2.1 Dekompozícia na kategórie

Spomínaný rozklad je riešený cez listy, pričom názov každého listu zodpovedá názvu kategórie, napríklad `Implementace`. Tieto listy obsahujú tabuľku, do ktorej je možné zadávať jednotlivé položky, pričom ku každej položke je možné vyplniť Min, Max, Nej. prav., čo sú hodnoty, ktoré boli opísané v časti, viz 2.3. Hodnota Min odpovedá bezproblémovému scenáru, hodnota max najhoršiemu možnému scenáru a hodnota Nej. prav. najpravdepodobnejšiemu scenáru. Na základe vyplnenia týchto dát sa vypočíta priemer z hodnoty min a max i očakávaná hodnota podľa metódy PERT. Jednotlivým položkám je tiež možné priradiť variant, viz 3.1. Záhlavie tabuľky obsahuje sumarizáciu jednotlivých scenárov s ohľadom na uvažované varianty. Uvažované varianty sa nachádzajú v liste Prehľad, viz 3.2.3. Na pravej strane od tabuľky sa nachádza kontrolný zoznam viz 3.2.

3.2.2 Zoznam predpokladov a obmedzujúcich podmienok

Na ďalšom liste sa nachádza zoznam predpokladov a obmedzujúcich podmienok, pričom je možné jednotlivým predpokladom prideliť variant, viz 3.3. Na



Obr. 3.3: Aktuálny spôsob vyplňovania predpokladov a obmedzujúcich podmienok

pravej strane sa rovnako ako pri záložkách s jednotlivými kategóriami nachádza tabuľka obsahujúca kontrolný zoznam.

3.2.3 Prehľad

Hlavným listom je prehľad a sumarizácia všetkých kategórií. Na tomto liste sa nachádza viac tabuliek. Hlavnou tabuľkou je súčet hodnôt jednotlivých kategórií. Tabuľka je rozšírená o ďalší riadok, ktorý obsahuje špeciálnu kategóriu `Záruka`, viz 3.1.3 a ďalší stĺpec, v ktorom je riziko pre každú kategóriu definované vzťahom, viz 3.1.

$$\frac{\begin{array}{c}najhorsiScenarDanejKategorie \\ - bezproblemovyScenarDanejKategorie\end{array}}{\begin{array}{c}celkovySucetNajhorsichScenarov \\ - celkovySucetBezproblemovychScenarov\end{array}} * 100 \tag{3.1}$$

Pod tabuľkou sa nachádza sumarizácia všetkých hodnôt. Prítomné sú 4 rôzne sumarizácie: realizácia, realizácia vrátane záruky, celkovo bez záruky, celkovo so zárukou. Do realizácie sa počíta súčet týchto kategórií: Design, Implementácia, Testovanie, PM (Projekt Manažment), Ostatné.

Napravo od tejto tabuľky sa nachádza ďalšia tabuľka obsahujúca podiel jednotlivých hodnôt k celku v danom scenári. Pre lepšie pochopenie je uvedený príklad. Výpočet pre kategóriu Implementace, pre hodnotu Max (najhorší scenár) je určený vzťahom, viz 3.2.

$$\frac{najhorsiScenar}{celkovySucetNajhorsichScenarov} * 100 \tag{3.2}$$

V tomto liste sa tiež vyskytuje prepínač, ktorý definuje, v čom sú vyjadrené jednotlivé hodnoty pre jednotlivé aktivity. Je možné vybrať medzi dvomi hodnotami — MD a MH. Tento prepínač slúži len na prepočítanie hlavnej tabuľky súčtu jednotlivých kategórií. Ak je nastavená hodnota MD, súčtom je skutočne súčet jednotlivých aktivít, ktoré boli vyplnené pre jednotlivé kategórie. V prípade nastavenia hodnoty MH, je postup rovnaký, ako keď bola

nastavená hodnota MD s tým rozdielom, že všetky hodnoty budú vydelené hodnotou 8.

Zoznam variantov, ktoré su dostupné a ktoré sa majú vo výslednom odhade pracnosti započítať, je možné vyplniť práve v tomto liste. Tento súbor dokáže spracovať nanajvýš 5 rôznych variantov. Samotný výpočet prebieha v listoch kategórií, kde sa celková hodnota vypočíta na základe zvolených variantov. Aktivity, ktoré nemajú zvolený žiaden variant, sú vo výsledku započítané stále.

Výsledný odhad, ktorý sa prezentuje zákazníkovi, musí byť vyplnený manuálne. Hodnoty sú len okopírované z vypočítaných hodnôt a následne manuálne zaokrúhlené. Štruktúru je možné vidieť na obrázku 3.4.

3.2.4 Hlavička

Excelovský súbor obsahuje tiež hlavičku. V nej sú informácie o názve projektu i informácie o rôznych verziách, viz 3.3.

3.3 Momentálne verzovanie

Verzovanie daného odhadu je riešené vo verzovacom systéme SVN, viz 5.5.1 alebo vo verzovacom systéme GIT, viz 5.5.2. Postup na vytvorenie novej verzie, prípadne revízie, je riešený následovným spôsobom

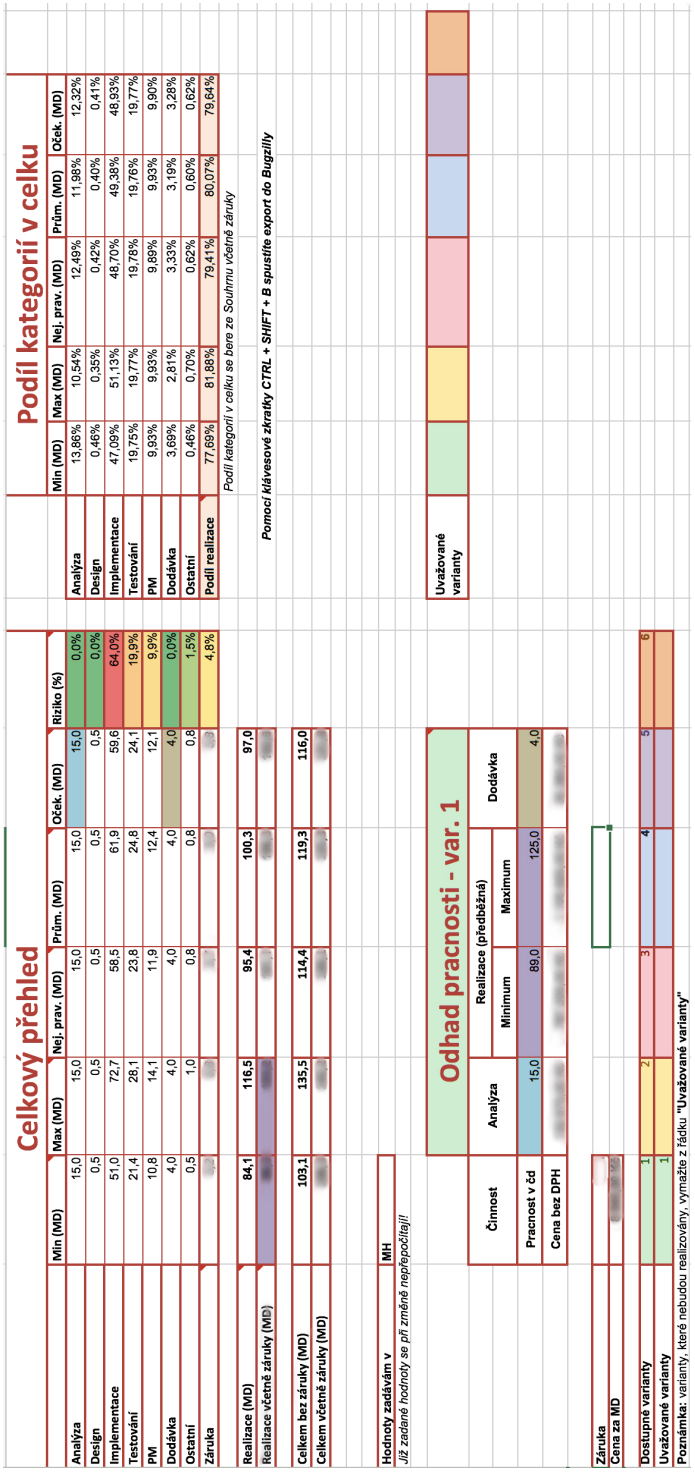
1. otvorenie a editácia aktuálnej verzie
2. pridanie informácie o novej verzii do listu Hlavička (Dátum, Autor verzie, Stručný popis)
3. commitnutie tohto súboru do SVN repozitára

3.4 Nedostatky súčasného riešenia

Súčasné riešenie nemá efektívne riešené verzovanie. Pre porovnanie dvoch rôznych verzií rovnakého odhadu je potrebné stiahnuť predošlú verziu cez verzovací systém SVN. Nie je jednoduché zistiť, v ktorých aktivitách došlo k zmene, prípadne zistiť, ktoré aktivity boli zmazané a ktoré pridané. Detekcia týchto zmien je dôležitá, pretože pri tvorení budúcich odhadov pracnosti bude možné predísť chybám, ktoré boli prítomné v minulosti.

Druhým problémom je vyhľadávanie v týchto odhadoch na základe autora, prípadne filtrovanie na základe veľkosti projektu. Z názvu súboru nie je jednoznačné o aký veľký projekt sa jedná. Táto informácia je dôležitá pri vytváraní nových odhadov z dôvodu potrebného porovnávania s podobnými projektmi.

Štruktúra jednotlivých odhadov je rovnaká pre všetky odhady. Nie je možné vytvoriť stromovú štruktúru pri tvorení dekompozície. To znamená, že nie je možné vytvoriť, napríklad v kategórií `Analýza`, rozpad na ďalšie



Obr. 3.4: Aktuálny spôsob zobrazenia prehľadu

podkategórie. WBS môže mať rozpad do ľubovolnej hĺbky. Túto skutočnosť nie je možné zachytiť.

Práva na editáciu a zápis nie sú riešené.

Kapitola 4

Požiadavky na aplikáciu

Cieľom aplikácie je nahradiť momentálne riešenie webovou podobou. Funkcionalitu, ktorá fungovala v Exceli, je potrebné preniesť do aplikácie.

4.1 Funkčné požiadavky

Funkčné požiadavky popisujú, akú funkcionalitu má aplikácia spĺňať. Zoznam funkčných požiadavok pre túto aplikáciu je nasledovný.

1. Aplikácia bude umožňovať ľubovoľnú dekompozíciu, nie len tú, ktorá sa momentálne používa v Exceli.
2. Dekompozícia bude definovaná šablónou.
3. Dekompozícia bude v GUI reprezentovaná vo forme tabuľky.
4. Základná jednotka odhadu (aktivity) bude obsahovať popis a ohodnotenie pracnosti: min, max, odhadováno.
5. Na základe ohodnotenia pracnosti bude dopočítaný priemer, ako aj očakávaná hodnota podľa metódy PERT.
6. Ohodnotenie pracnosti pre jednotlivé položky bude v jednotkách MD alebo MH. Túto hodnotu bude potrebné zvoliť pri vytváraní nového odhadu, pričom hodnotu nebude možné meniť v rámci odhadu.
7. Pri zobrazení daného odhadu bude možné spraviť prepočet medzi MD a MH. Samotné hodnoty to nezmení, prepočet bude informatívneho charakteru.
8. Jednotlivé odhady budú podporovať správu variantov, ako aj filtrovanie podľa týchto variantov.
9. Variant bude definovaný kľúčom a stručným popisom.

10. Aplikácia bude vedieť zobraziť súčet položiek jednotlivých kategórií v stromovej štruktúre.
11. Prepodklady a obmedzujúce podmienky budú viazané na varianty a na jednotlivé aktivity.
12. Prístup do systému bude podmienený prihlásením pomocou mena/hesla pomocou protokolu LDAP, viz 5.1.1.
13. Každý odhad bude patriť do kontextu, ku ktorému budú definované práva. Práva budú rozdelené na právo čítania a právo zápisu.
14. Aplikácia bude podporovať navigáciu pomocou klávesnice.
15. Aplikácia bude podporovať operáciu späť/undo.
16. Všetky zmeny medzi verziami budú logované a bude možné ich porovnávať jednotlivo.

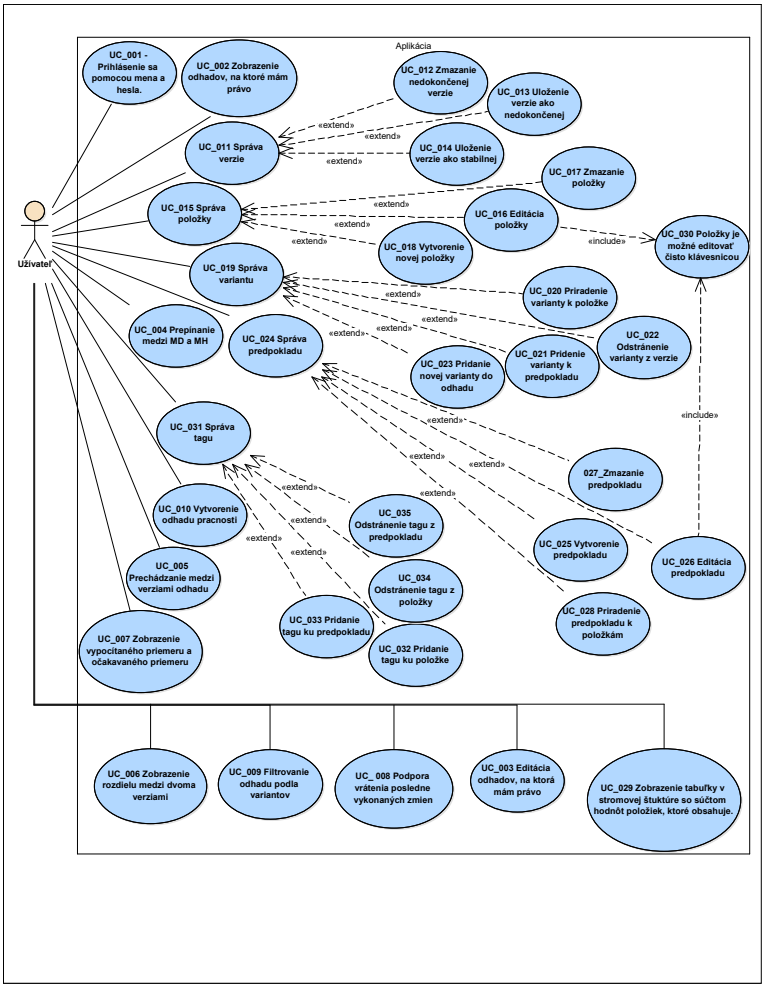
4.2 Prípady použitia

Prípad použitia (ang. usecase, ďalej už len UC) opisuje možnosti, ktoré systém ponúka jednotlivým aktérom. Aktérom sa rozumie účastník používajúci aplikáciu. Príkladom aktéra môže byť užívateľ, administátor. Každý aktér môže mať iné výsady a iné práva. V tejto aplikácií je aktérom len používateľ. Prípady užitia zachycuje obrázok 4.1. Význam UC je buď jasný z definície alebo je možné pochopiť jeho význam pri opise obrazovky, ktorého sa týka, viz 7. Pre lepšie pochopenie je význam nejasných UC vysvetlený nasledovne.

UC_001 Užívateľ sa môže do systému prihlásiť pomocou mena a hesla. LDAP server overí, či daný užívateľ má prístup do aplikácie.

UC_002 Po úspešnom prihlásení môže užívateľ zobraziť zoznam odhadov, ku ktorým má právo na čítanie.

UC_011, UC_015, UC_019, UC_024, UC_031 Uživateľ má prístup k editácií jednotlivých častí, v prípade, že sa nachádza v odhade, na ktorý ma právo na zápis. Druhou podmienkou na umožnenie editácie je, že sa užívateľ nachádza v rozpracovanej verzii odhadu. Výnimkou je vytvorenie novej verzie (UC_036). Vytvorenie novej verzie je možné v prípade, že sa užívateľ nachádza v stabilnej verzii. Vytvorenie novej verzie vytvorí kópiu tejto verzie, ktorej je pridelené nové číslo, názov a popis, ktorý vypĺnil užívateľ.



Obr. 4.1: Prípady použitia

Kapitola 5

Možnosti riešenia

5.1 Vymedzenie pojmov

5.1.1 LDAP

Zdroj [4] uvádza, že Lightweight Directory Access Protocol (ďalej už len LDAP) je adresárová informačná služba. V praxi to napríklad znamená zoznam zamestnancov firmy, ich prihlasovacie mená, domovské adresáre, osobné informácie, ich emailov alebo telefónnych čisel. Často používanou funkciou LDAP je overenie užívateľa (meno/heslo).

5.1.2 Java Enterprise Edition

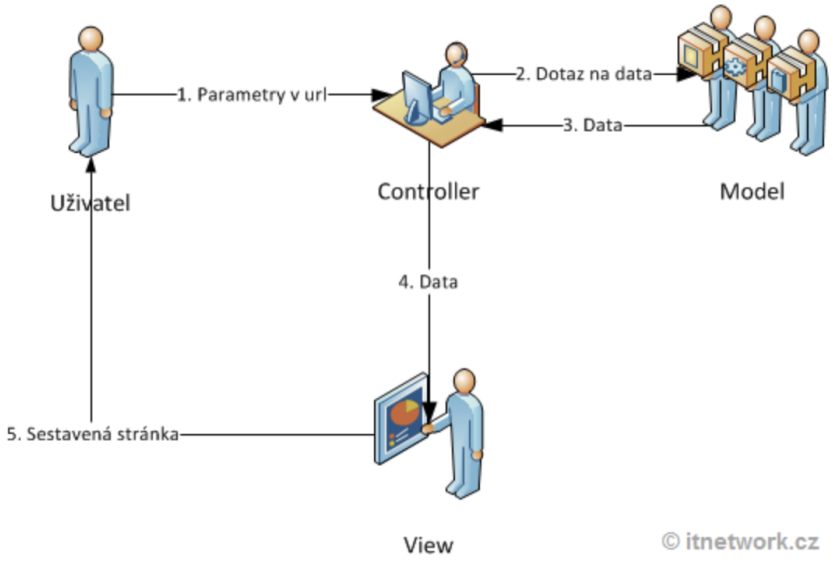
Java Enteprise Edition (ďalej už len Java EE) je technológia určená pre tvorbu podnikových aplikácií.[5] Aplikácie vytvorené pod touto platformou potrebujú aplikačný server, na ktorom aplikácia beží.

5.1.3 JavaServer Faces

„Java Server Faces je komponentne orientovaný framework, vyvinutý spoločnosťou Sun Microsystems, ktorý sa stal v prostredí Java EE od verzie 5 štandardom pri vývoji webových aplikácií. Jeho hlavnou úlohou je uľahčiť vývoj moderných aplikácii v Jave.“[6, str. 12] Framework využíva architektúru Model, View, Controller (ďalej už len MVC) viz 5.1.4.

5.1.4 Model, View, Controller

Základnou myšlienkou MVC architektúry je oddelenie logiky od výstupu. Rieši problém tzv. „špagetového kódu“, kedy sú v jednom súbore (triede) logické operácie a zároveň renderovanie výstupu. Súbor obsahuje databázové dotazy, logiku a napríklad rôzne html tagy. Všetko je do seba zamotané ako špagety.[7]



Obr. 5.1: Princíp MVC (prevzaté z [7])

Takýto kód je ťažko udržovateľný pre samotného autora, nie to ešte pre ostatných programatátorov. Cieľom MVC je striktne odlišovať logické celky.

M — Model udržuje dátovú štruktúru. V aplikácii je model realizovaný najčastejšie pomocou triedy, ktorá obsahuje dáta. Do modelovej vrstvy nepatria len samotné dáta, ale aj dómenová logika. Príkladom môže byť trieda typu „Človek“ obsahujúca atribút „Rok narodenia“. Výpočet veku človeka spadá do tejto vrstvy.

V — View je zobrazením dát z modelu. To, ako aplikácia vyzerá pre užívateľa, teda to, ako vyzerá GUI, patrí do view vrstvy.

C — Controller si možno predstaviť ako prostredníka medzi vrstvou modelu a vrstvou view. To, ako stránka vyzerá, rieši view. Samotné dáta a logiku obsahuje model. Úlohou controllera je získať dáta z modelu a odovzdať ich view vrstve, ktorá tieto dáta následne zobrazí. Úlohou je tiež opačný smer. Akciu vykonanú užívateľom vo view vrstve (napríklad registrácia) najprv spracuje Controller, ktorý môže vykonať validácie a následne zavolať logiku na modelovej vrstve (napríklad uloženie nového užívateľa). Určenie, ktorá stránka sa má zobraziť, rieši tiež Controller. Príkladom môže byť prístup na webovú stránku `localhost:8080/zobrazOdhady`. Controller spracuje parametre z url adresy a rozhodne čo sa má stať. V tomto prípade načíta dáta z modelu a odovzdá ich view vrstve, ktorá tieto dáta zobrazí.

Na diagrame 5.1 je názorne zobrazená situácia.

5.1.5 Maven

Maven je nástroj pre automatizáciu buildovania, správu aplikácie a jej závislosti. Definícia Maven projektu sa nachádza v súbore Project Object Model (POM). Jedná sa o súbor vo formáte XML, ktorý sa nachádza v koreňovom adresári projektu. Samotný súbor sa nazýva pom.xml. V tomto súbore sú údaje o projekte, napríklad do akej skupiny aplikácia patrí (groupId), jej jednoznačný názov v rámci skupiny (artifactId), číslo verzie daného projektu, definovanie rodičovského POM súboru a mnoho ďalšieho.[8].

5.1.6 Trojvrstvá architektúra

Trojvrstvá architektúra sa skladá z troch vrstiev. Definícia jednotlivých vrstiev je čerpaná zo zdroja [9].

- **Prezentačná vrstva** je časť, ktorá je viditeľná pre užívateľov, zaisťuje vstup požiadavok a prezentuje výsledky. Príkladom prezentačnej vrstvy môžu byť spomínané frontend frameworky, viz 5.3.
- **Aplikačná vrstva** sa nazýva tiež funkčnou vrstvou. Táto vrstva má na starosti biznis logiku. Zaisťuje výpočty a operácie medzi vstupnými a výstupnými požiadavkami a dátami.
- **Dátová vrstva** zaisťuje prácu s dátami ako vytváranie nových dát, editovanie dát a mazanie. Pri vývoji Java aplikácií sa na túto vrstvu zvyknú používať frameworky ako napríklad myBatis, Hibernate.

Hlavnou myšlienkou trojvrstvej aplikácie je striktná závislosť smerom dole a vždy len o jednu úroveň. Znamená to, že prezentačná vrstva komunikuje stále len s aplikačnou vrstvou, nikdy nie priamo s dátovou. Takéto rozdelenie má tú výhodu, že oddeľuje jednotlivé vrstvy tak, aby na sebe neboli závislé. Takéto rozdelenie prináša výhodu v momente, keď bude potrebné zmeniť jednu z týchto vrstiev. Nebude potrebné prepisovať celú aplikáciu, ale len túto jednu konkrétnu vrstvu. Ďalšou výhodou môže byť napríklad prítomnosť viacerých prezentačných vrstiev.

5.1.7 Vkladanie závislosti

Dependency injection v preklade znamená vkladanie závislosti (ďalej už len DI). DI je technika vkladania závislosti medzi jednotlivými komponentami programu.

5.2 Backend frameworky

Backend možno vnímať ako serverovú časť projektu, v ktorej sa odohráva celá biznis logika, rovnako aj prístup do databázy. Je to všetko to, čo nie je

frontend, viz 5.3. Jednou z požiadavok na výber technológii je, aby aplikácia bola postavená na technológiách bežne používaných vo firme Profinit, prípadne na riešení, ktoré je možné si rýchlo osvojiť.

5.2.1 Spring

Spring je open-source aplikačný framework vydaný pod licenciou Apache License 2.0., ktorý je určený na podporu riešení v mnohých oblastiach, napríklad: web, databázový prístup, cloud, bezpečnosť aplikácií či vývoj pre mobilné platformy.

Tento framework vznikol v roku 2002, stále sa vyvíja a má veľkú popularitu. Samotných modulov pre Spring je veľké množstvo. V tejto bakalárskej práci je opísaný len malý malé množstvo funkcionalít, ktoré boli použité priamo v aplikácií.

Spring je v súčasnej dobe možné spojazdniť pomocou technológie Maven, viz 5.1.5, prípadne Gradle, Ant alebo manuálne.

Tento framework tiež obsahuje Spring Inversion of Control (IoC), ktorý slúži k vytváraniu objektov, konfigurácii, prepojeniu a tiež k správe ich životného cyklu. Objekty, ktoré sú pod správou Spring IoC sa nazývajú beany.[10]

5.2.2 Spring boot

Spring Boot vznikol v druhej polovici roku 2013. Tento framework vznikol ako nadstavba na Spring Framework. Spring Framework je veľmi populárny, ale vytvorenie jednoduchej aplikácie zaberie veľa času. Spring Framework vychádza z myšlienky, že všetko, čo je potrebné, si nastaví autor sám. K tomu, aby to všetko nastavil, je potrebné detailnejšie zoznámenie sa s touto technológiou. Vo väčšine situácií sú tieto nastavenia nastavené na východzie hodnoty.

Tu vstupuje Spring Boot. Vychádza z myšlienky, že vo východzom stave je všetko nastavené tak, aby to fungovalo. Samozrejme, je tu možnosť všetko manuálne prepísať.

Pre Spring Boot existuje incializátor, v ktorom je možné si vybrať, ktoré závislosti budú v projekte použité a následne sa vygeneruje celý projekt, ktorý v sebe obsahuje tieto závislosti, a ktorý je pripravený na implementáciu požadovanej aplikácie. Inicializátor je možné nájsť na [11].

5.3 Frontend frameworky

Frontend možno vnímať ako klientskú časť aplikácie. Poskytuje prístup k službám serverovej časti pomocou grafického rozhrania. Užívateľ používa aplikáciu práve cez grafické rozhranie. Zjednodušene sa dá povedať, že frontend je to, čo užívateľ vidí.[12] Pri implementácií frontend časti je vhodné použiť frontend frameworky. Existujú časti, ktoré sa vyskytujú na viacerých stránkach

a možno ich vnímať ako bežné. Takouto časťou je napríklad tabuľka, v ktorej je možné pohybovanie sa pomocou klávesnice, pomocou stlačenia klávesy „ENTER“ je možné dostať sa do režimu editácie. Túto funkcionalitu je možné naimplementovať svojpomocne alebo využiť systém (framework), ktorý už má takúto funkcionalitu v sebe implementovanú. Oba prístupy majú svoje výhody a nevýhody.

5.3.1 Vlastné riešenie vs framework

Podľa [13] medzi hlavné výhody používania frontend frameworkov patrí úspora času pri vytváraní grafického rozhrania. Ďalšou výhodou takýchto frameworkov je častokrát „plug-and-play-functionality“. Tento pojem v preklade znamená „zapojiš a ideš funkčnosť“. V našom prípade to znamená, že pomocou zopár príkazov dokážeme vyriešiť pomerne zložitú funkcionalitu. V neposlednom rade výhoda používania populárnych frameworkov so sebou prináša aj veľkú komunitu ľudí používajúcich tieto frameworky. Táto komunita rieši problémy na diskusných fórach. Vďaka tomu, že je framework populárny, zvyknú byť tieto diskusné fóra aktívne, čo znamená, že väčšina problémov daného frameworku sa už pravdepodobne riešila v minulosti.

Hlavnou nevýhodou používania frameworku je spôsob, akým je písaný. Je písaný pre bežné použitie, no v praxi sa stáva, že od daného komponentu potrebujeme vlastnú funkcionalitu. Upravenie komponentu frameworku môže byt zložité a časovo veľmi náročné.

Implementácia vlastného riešenia je časovo náročnejšia, no aj napriek tomu dokáže toto riešenie v niektorých prípadoch ušetriť čas v budúcnosti. Autor tohoto riešenia má presný prehľad o tom, čo toto riešenie dokáže, a v prípade potreby zmien vie relatívne rýchlo tieto zmeny doimplementovať. Vlastné riešenie je rýchlejšie, pretože frameworky poskytujú veľkú množinu funkcií, ktoré častokrát pre konkrétne úlohy nie sú potrebné. Vo vlastnom riešení sa implementuje len to, čo je požadované.

Nevýhodou použitia vlastného riešenie je nutnosť neustáleho vývoja tohoto riešenia, napríklad kvôli vývoju prehliadačov.

Vzniká otázka, čo je výhodnejšie. Je dôležité spraviť prieskum frameworkov a potrieb aplikácie. Ak sa dôjde k záveru, že framework poskytuje takmer všetko, čo aplikácia vyžaduje, je správnym riešením použiť framework. Ak sa dôjde k záveru, že bude potrebné prepísať veľkú časť komponentov frameworku, pretože funkcionalita, ktorú poskytuje, nevyhovuje úplne potrebám aplikácie, je lepšie zvoliť vlastné riešenie.

5.3.2 Prehľad frontend frameworkov

Pri tvorení GUI pre aplikáciu je dôležité, aby vytváranie a editovanie nových aktivít bolo rýchle a intuitívne ako v aktuálnom riešení, ktoré k tomu využívalo Excel. Pri výbere možných frontendových frameworkov bolo prihlia-

dané na túto skutočnosť. Boli vybrané frameworky, ktoré dokázali zobrazovať a editovať aktivity v tabuľkách, prípadne v online tabuľkových procesoroch. Taktiež bolo prihliadané na to, aby aplikácia patrila pod Rich User Interface (RUI), čo znamená, že aplikácia je svojim správaním podobná desktopovej aplikácii. „RUI sa odkláňa od klasického prístupu tvorby webov, ktorý pozostával z množstva jednotlivých stránok, ktoré boli navzájom pospájané pomocou URL linkov. V tomto prípade sa totiž generuje iba jedna stránka (tzv. „Single page aplikácia"), ktorá zobrazuje zmeny kódu v závislosti na aktivite používateľa pomocou AJAX-u".[14] Všetky vymenované frontend frameworky je možné prepojiť s backendovým frameworkom Spring.

5.3.2.1 Vaadin Framework

Vaadin je open-source framework šírený pod licencou Apache License 2.0. Framework je určený pre vytváranie moderných webových aplikácií, ktoré vyzerajú výborne, fungujú výborne a robia vývojára a samotných užívateľov šťastnými.[15] Hlavnou myšlienkou Vaadinu je rozdelenie jednotlivých komponentov na dva odlišné modely, a to na stranu serverovú a stranu klientskú, kde práve strana serverová ponúka silu vývoja. Vďaka serverovej strane je pri vývoji možné zabudnúť na internetové stránky a programovať užívateľské rozhranie ako klasický počítačový Java program. Pomocou Vaadinu je dokonca možné vytvoriť aplikáciu bez znalosti jazykov ako sú HTML a JavaScript.[16] Klientská strana spúšta v aplikácií JavaScript, preto nie je potrebná inštalácia dodatočných pluginov pre webový prehliadač. Vaadin využíva pre vykreslenie užívateľského rozhrania framework Google Web Toolkit.

Tento framework je zvažovaný hlavne kvôli podpore pre zobrazovanie a editáciu tabuľky. Od Vaadin verzie 7.5 vydanej v druhej polovici 2015 je prístupný komponent s názvom „Grid". Tento komponent poskytuje skvelú navigáciu po tabuľke pomocou šípok na klávesnici, ako aj intuitívne editovanie jednotlivých položiek. Tento komponent je výborne zdokumentovaný. Samotý grid má možnosti radenia, filtrovania. Zobrazovanie iných hodnôt v editovacom móde a mimo editovacieho módu je možné riešiť cez konvertory, ktoré Vaadin poskytuje.

Pre zobrazenie stromovej štruktúry samotný Vaadin neposkytuje komponent, ktorý by splnil túto funkčnosť. Vo Vaadin 8, vydaný 21.2.2017, sa momentálne implementuje natívna podpora komponentu „TreeGrid". Zatiaľ je tento komponent len v alpha verzii Vaadinu, jeho dokončenie sa odhaduje na začiatok leta 2017. Vaadin je populárny pre jednoduché vytváranie vlastných komponentov, ktoré je možné publikovať priamo na stránkach Vaadinu. Tieto vlastné komponenty sa nazývaju add-ons (doplnky). Pre účel stromovej štruktúry je možné použiť doplnok Vaadin TreeGrid prístupný na stránke, viz [17]. Tento doplnok je vydaný pod licenciou Apache Licence 2.0. Jedná sa o rozšírenie implementácie samotného Gridu.

Vzniká otázka, či je lepšie si napísať vlastný komponent alebo použiť Vaadin 7. Prihliadajúc na to, že Vaadin 8 existuje len zopár dní, je vhodnejšie ísť cestou Vaadinu 7. Dôvodom nie je len vyššie spomínaná funkcionalita, ale tiež rozšírenosť verzie 7. Vaadin 7 existoval od roku 2013, preto je väčšia šanca, že problémy spojené s Vaadinom 7 budú na diskusných fórach riešené detailnejšie v porovnaní s Vaadin verziou 8. Vzhľadom na to, že je táto verzia na trhu krátko, je veľká pravdepodobnosť, že väčšina problémov ešte nebola riešená vôbec.

5.3.2.2 PrimeFaces

PrimeFaces je open-source framework určený pre JavaServer Faces(ďalej už len JSF), viz 5.1.3. Framework je šírený pod licenciou Apache License 2.0. PrimeFaces je odľahčená knižnica postavená na myšienke zachovať túto knižnicu čo najmenšiu. Pridávanie riešení tretích strán môže do projektu priniesť zbytočnú zložitosť, ktorá nie je potrebná, čo by malo za následok spomalenie aplikácie. Autori PrimeFaces vyvíjajú tento framework bez týchto tretích strán, čím zaisťujú maximálnu efektivitu. PrimeFaces pozostáva len z jedného súboru jar, bez žiadnych závislostí a žiadneho konfigurovania.[18]

PrimeFaces sa riadi motom: „Dobrý UI komponent by mal skryť zložitosť ale zachovať flexibilnosť.“[18] Tento framework má jednu z najväčších komunít, ktorá sa podieľa na jeho vývoji.

Tento framework je zvažovaný hlavne kvôli podpore pre zobrazovanie stromovej štruktúry. Jedná sa konkrétne o komponent na zobrazovanie stromovej štuktúry rozpadu kategórií TreeTable viz [19]. Na stránkach sa nachádza tiež ukážka použitia tohoto komponentu v programe. Ako je možné vidieť, jeho používanie je triviálne. Na stránke je v záhlaví možné vyberať medzi možnosťami, ktoré daný komponent podporuje. Pre potreby našej aplikácie sú určite potrebné funkcie ako radenie, voliteľná veľkosť stĺpcov a odchytávanie udalostí.

Druhým dôvodom pre výber tohoto riešenia je zobrazovanie a editácia tabuľky. Zobrazovanie jednotlivých aktivít v danej kategórii vie riešiť komponent DataTable, viz [20]. Na stránke je tiež ukážka použitia. V záhlaví sa nachádzajú jednotlivé funkcionality, ktoré daný komponent poskytuje. Potrebná funkcionalita pre našu aplikáciu je pohodlná editácia jednotlivých aktivít, čím sa rozumie pohybovanie sa po tabuľke pomocou šípok na klávesnici. Komponent DataTable podporuje editáciu. Editovanie funguje v poriadku, no nespĺňa požiadavku na pohybovanie sa po tabuľke. Z ukážok poskytnutých v deme tiež nie je jasné, ako zobrazovať v editovacom móde hodnoty iné, ako tie, ktoré sú zobrazované mimo edit módu. Táto funkcionalita je žiadúca pre hodnoty určené matematickým vzťahom. V Exceli, pri použití funkcie `=5+5`, je v momente opustenia editovaceho módu zobrazená hodnota 10. V roku 2011 bol vytvorený komponent s názvom „Sheet“, ktorý napodobňoval funkciolatiu z Excelu.[21] Tento komponent skončil v beta verzii a nebol v nových

Tabuľka 5.1: ZK licencia (prevzaté z [23])

Produkt	**Licencia**
ZK CE	LGPL (zdarma pre open source projekty a komerčné použite)
ZK PE	ZOL alebo Komerčná licencia
ZK EE	ZOL alebo Komerčná licencia

verziách ďalej podporovaný. Pri použití tohoto frameworku by bolo potrebné doimplementovať vyššie spomínanú funkcionalitu.

5.3.2.3 ZK Framework

ZK framework je najpoužívanejším ajaxovým frameworkom, ktorý poskytuje najjednoduchší spôsob vytvorenia modernej webovej aplikácie v jazyku Java. Podobne ako vo framerowku Vaadin, je tu možnosť implementovať UI čisto len za pomoci Java kódu. Tento prístup sa podobá vývoju desktopových aplikácií, v ktorých nebolo potrebné riešiť JavaScriptové programovanie, problémy medzi rôznymi prehliadačmi, zložitú ajaxovú komunikáciu a ďalšie veci, ktoré sťažujú vývoj samotnej aplikácie. ZK využíva pre vykreslenie užívateľského rozhrania jQuery a JSON.[22]

Framework ponúka možnosť výberu troch verzií, viz 5.1.

Produkty ZK PE, ZK EE je možné používať zdarma, v prípade, že výsledná aplikácia bude spadať pod licenciu ZK Open source. Ak aplikácia nebude vyvíjaná ako open source, tieto produkty spadajú do komerčnej licencie.

Niektoré komponenty, prípadne rozšírenia pre daný komponent, je potrebné si zakúpiť.

Tento framework je zvažovaný hlavne kvôli komponentu ZK Spreadsheet, ktorý umožňuje pracovať s Excelovskými súbormi priamo v okne prehliadača. Dá sa to prirovnať ku Google Sheets, s tým rozdielom, že všetky zmeny je možné odchytávať a priradiť im vlastný význam. Idea tohoto riešenia by spočívala vo vykreslení prázdneho excel súboru, do ktorého by sa v controlleri vyplnili hodnoty, ktoré by boli z databázy. Pri samotnom ukladaní by sa tento excel súbor opäť spracoval a zmeny by sa perzistovali do databázy. Problémom je, že niektoré žiadúce funkcie sú prístupné len v platených variantoch tohoto komponentu. Jedná sa konkrétne o funkcie na validáciu, filtrovanie a hlavne prepojenie podpory práce s JSP, prípadne JSF súbormi.

5.4 Databáza

Databáza je množina štruktúrovaných dát alebo informácií uložených v počítačovom systéme. Všetky aplikácie, ktoré potrebujú uchovávať dáta musia tieto dáta niekam ukladať. Návrh modelu a výber databázovej vrstvy riešil ko-

lega Milan Vancl v [24]. Tejto oblasti sa detailnejšie venuje v jeho bakalárskej práci.

5.5 Verzovací systém

Verzovací systém hraje veľkú rolu vo vývoji aplikácií. Primárnym účelom verzovacích systémov je zaznamenávanie zmien a propagácia týchto zmien medzi vývojármi, ktorí na tejto aplikácii pracujú. Medzi najpoužívanejšie verzovacie systémy v dnešnej dobe patria nástroje git a SVN. V rámci spolupráce s Milanom Vanclom bolo pre tento projekt potrebné použiť verzovací systém.

5.5.1 SVN

SVN je sytém pre správu a verzovanie zdrojových kódov, jeho plný názov je Apache Subversion. Projekt je verzovaný v centrálnom úložisku na serveri. Pri akejkoľvek zmene je potrebné mať prístup k tomuto serveru.

5.5.2 Git

Git predstavuje distribuovaný systém. Hlavným rozdielom medzi git a SVN je to, že každý vývojár pracuje v lokálnom repozitári, do ktorého môže commitovať zmeny. K tomu nie je potrebné, aby mal pri každej zmene prítomné spojenie so serverom.

Kapitola 6

Zvolené riešenie

6.1 Backend framework

Ako backend framework bol zvolený framework Spring Boot. Hlavným dôvodom tohoto výberu bolo to, že samotný framework obsahuje plne funkčné komponenty bez časovo náročnej manuálnej konfigurácie. Táto výhoda umožnila takmer okamžitý štart vývoja aplikácie. Ďalšou výhodou je zabudovaný Tomcat server, ktorý sa spustí automaticky pri štarte aplikácie. Ak v budúcnosti bude na tejto aplikácií pracovať viacero ľudí, bude spojazdnenie vývojového prostredia značne urýchlené. V neposlednom rade bolo pri výbere tohoto frameworku prihliadané na to, aby vývojári vo firme Profinit poznali túto technológiu a mohli jednoducho pokračovať vo vývoji. V porovnaní s frameworkom Spring tento variant nie je o nič ochudobnený. Všetky funkcie, ktoré poskytuje Spring Framework je možné nastaviť aj v Spring Boot frameworku.

6.2 Frontend framework

Pri výbere frontend frameworku sa kládol najväčší dôraz na podporu komponentov, vďaka ktorým by bolo jednoduché editovať tabuľku a prechádzať kategórie v stromovej štruktúre. Najvhodnejším kandidátom bol framework Vaadin. Prvotne bol zvažovaný framework ZK, no pri prvom nasadení vznikli otázky, ako riešiť veci, ktoré sú podporované len v platenej verzii. Tieto veci boli vymenované v 5.3.2.3. Využitie tohoto frameworku by najviac kopírovalo súčasné riešenie, ale implementovať komponenty, ktoré boli platené, by bolo časovo náročné.

Pri návrhu GUI bolá zvažovaná idea, v ktorej by aplikácia nekopírovala aktuálne riešenie úplne, ale vychádzala by z inej myšlienky. Pri využití ZK frameworku, by bol postup riešený nasledovným spôsobom. Užívateľ môže v danom liste robiť čokoľvek, a až po uložení sa vykonajú validácie, ktoré tieto zmeny zamietnu, prípadne povolia. Alternatívny prístup by bol nasledovný. Užívateľ nebude môcť robiť čokoľvek tak, ako tomu bolo v Exceli, ale jeho možnosti

budú obmedzené len na striktné úkony. Postup vývoja by bol riešený spôsobom pridávania funckionalít, ktoré sú požadované. K takému riešeniu bol najvhodnejší framework Vaadin. V porovnaní s frameworkom PrimeFaces podporoval pohodlné pohybovanie sa po tabuľke a následnú jednoduchú editáciu.

6.3 Verzovací systém

Tento projekt bol vyvíjaný pod verzovacím systémom Git. Pred konkurečným riešením SVN sa uprednostnil predovšetkým kvôli možnosti publikovania menších zmien lokálne, bez potreby pripojenia na internet.

Kapitola 7

Návrh obrazoviek pre aplikáciu

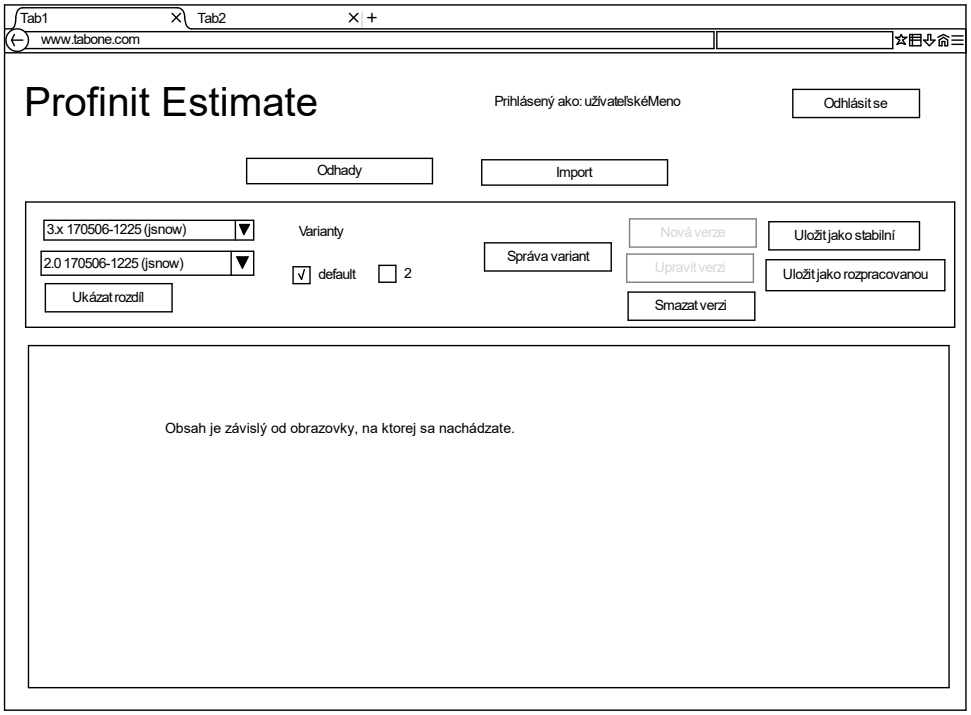
Táto kapitola sa venuje návrhu obrazoviek, ktoré boli vytvorené v programe Enterprise Architect a opísaniu funkčnosti jednotlivých komponentov. Pri návrhu obrazoviek sa kladol dôraz na jednoduchosť a funkčnosť. Na všetkých obrazovkách je len fragment celku, ktorý sa mení v závislosti od obrazovky. Rozloženie prvkov je viditeľné na obrázku 7.1. V popise obrazovky, ktorá obsahuje iné rozloženie, to bude explicitne spomenuté.

7.1 Zoznam odhadov

Obrazovka pre zoznam odhadov, viz 7.2, obsahuje zoznam všetkých odhadov, na ktoré má užívateľ právo na čítanie. Táto obrazovka nevychádza zo šablóny 7.1. Pre každý riadok budú prítomné štyri stĺpce, ako je viditeľné na obrázku. Pri niektorých odhadoch nie je vyplnený autor. Je to z toho dôvodu, že jednotlivé verzie môžu byť v dvoch rôznych stavoch — stabilné a rozpracované. Vzhľadom na to, že na editácii rozpracovanej verzie odhadu môže pracovať viacero užívateľov, je autor editácie neznámy. V momente, keď sa táto verzia uloží ako stabilná, bude autor jednoznačný. Stĺpec `Posledná zmena` obsahuje informáciu o čase a dátume uloženia stabilnej alebo rozpracovanej verzie.

7.2 Správa kategórií

Na tejto obrazovke, viz 7.3, sa nachádza stromová štruktúra dekompozície daného odhadu. To, ako je samotný odhad rozpadnutý, je definované šablónou, z ktorej vznikol. Hodnoty min, max, nej. prav. odpovedajú hodnotám z Excelu. Hodnota min pre kategóriu Analýza je vypočítaná ako súčet hodnôt min všetkých jej podkategórií, prípadne súčet hodnôt min aktivít v prípade, že kategória už neobsahuje ďalšie podkategórie. Hodnota priemer je vypočítaná ako priemer hodnôt min, max. Očakávaná hodnota je vypočítaná podľa metody PERT, viz 2.3.



Obr. 7.1: Wireframe — kostra obrazoviek

Pri kliknutí na akýkoľvek riadok tejto tabuľky nastane presmerovanie na obrazovku Správa položiek, viz 7.3 s položkami danej kategórie. V prípade kliku na kategóriu, ktorá obsahuje ďalšie vnorené kategórie, nastane tiež presmerovanie na obrazovku Správa položiek s tým, že sa načítajú všetky aktivity prislúchajúce všetkým podkategóriám. Ak podkategórie obsahujú ďalšie podkategórie, postupuje sa obdobným spôsobom, až kým sa nedôjde k aktivitám.

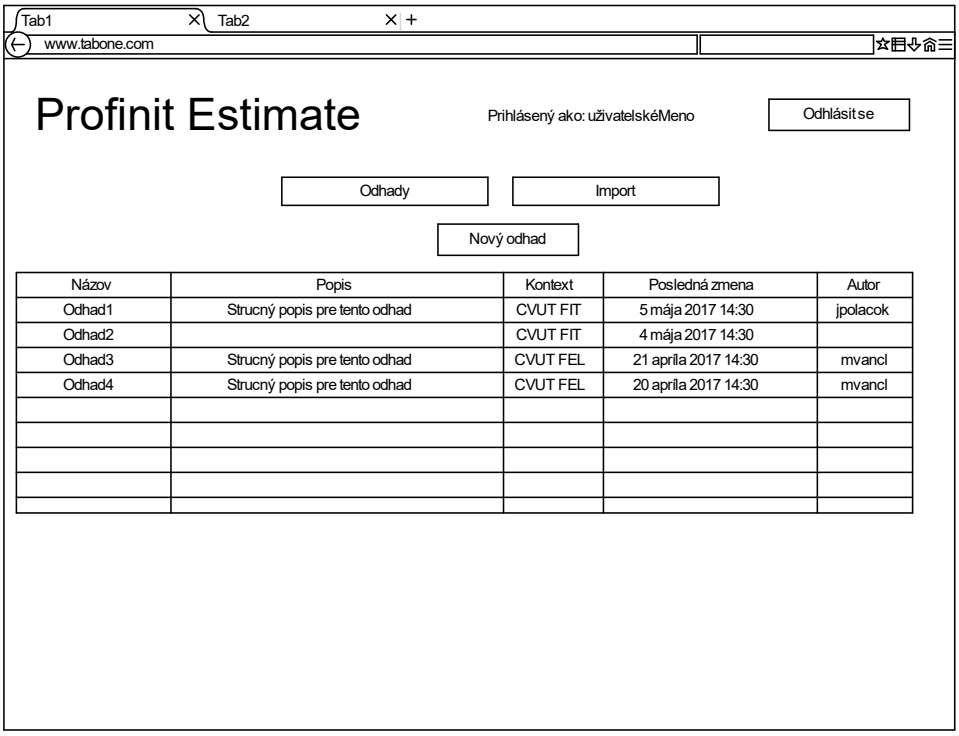
Pod prvou tabuľkou sa nachádza tabuľka obsahujúca jeden riadok. Ide o tabuľku, ktorá obsahuje informácie o počte predpokladov odhadu v danej verzii. V prípade kliknutia na tabuľku nastane presmerovanie na obrazovku Správa predpokladov, viz 7.4.

Poslednou tabuľkou na tejto obrazovke je podieľ jednotlivých kategórií v celku. Výpočet je spravený rovnako ako bolo opísané v 3.2.3.

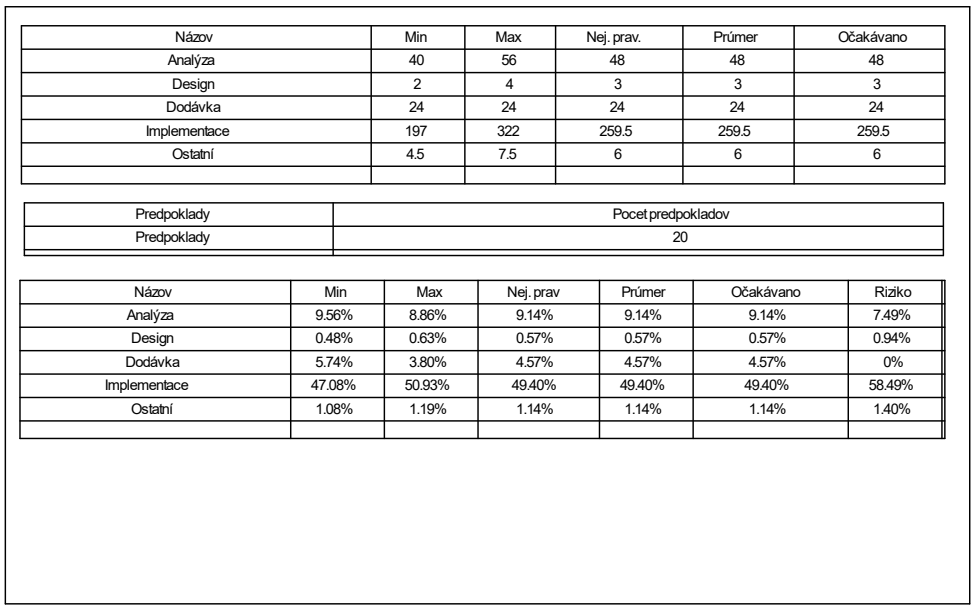
7.3 Správa položiek

Obrazovku, viz 7.4, pre správu položiek možno rozdeliť na dve oblasti, ktoré sú v tomto kontexte nazvané ľavá a pravá strana.

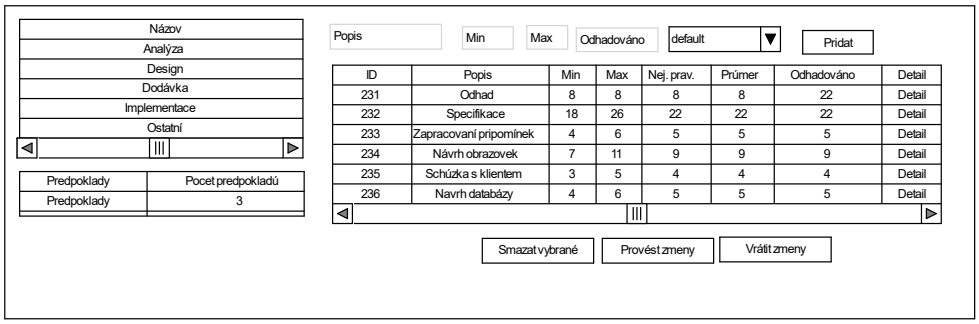
Naľavo je možné vidieť tabuľku obsahujúcu kategórie a pod ňou tabuľku s predpokladmi. Ide o komponent, ktorý bol opísaný v 7.2.



Obr. 7.2: Wireframe — zoznam odhadov



Obr. 7.3: Wireframe — prehľad odhadu

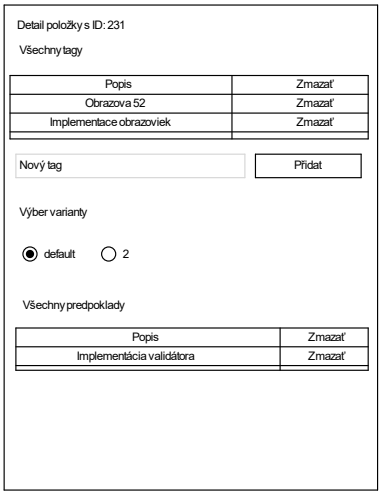


Obr. 7.4: Wireframe — prehľad položiek

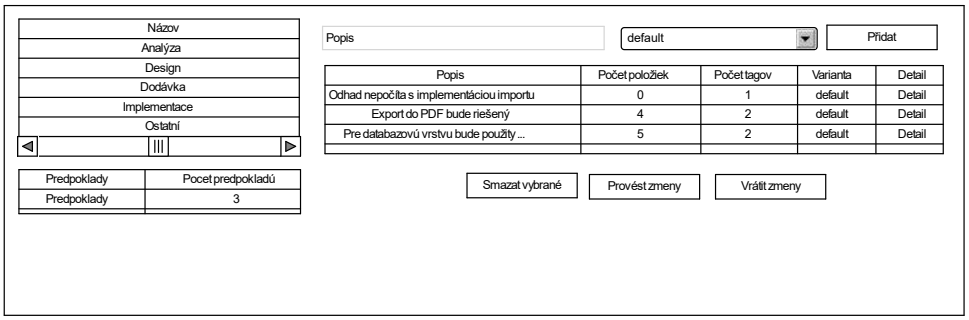
Na pravej strane je možné vidieť tabuľku obsahujúcu jednotlivé aktivity, ktoré sú v danej kategórií. To, pre akú kategóriu sa položky momentálne zobrazujú, je možné vidieť v tabuľke naľavo (kategória, pre ktorú sú položky zobrazené, je zvýraznená). Tabuľka jednotlivých aktivít pozostáva z ID, ktoré je automaticky generované, a atribútov, ktoré boli definované v 3.1. Posledný stĺpec obsahuje tlačítko `Detail`. Kliknutie na toto tlačítko otvorí okno opísané v 7.3.1. Nad tabuľkou sa nachádzajú štyri textové polia, jeden selectbox a tlačítko na pridanie. Cez tento formulár je možné pridať novú aktivitu. Formulár je viditeľný len v momente, keď sa užívateľ nachádza vo verzii, ktorá je rozpracovaná a má zámok (v kontexte tejto kapitoly ďalej už len *má právo*). Viac o logike zámku je popísané v 7.5. Druhou podmienkou pre zobrazenie tohoto formuláru je, aby vybraná kategória neobsahovala žiadne podkategórie. Model je navrhnutý tak, že aktivity je možné pridávať len do kategórií, ktoré nemajú podkategórie. Pod tabuľkou sa nachádzajú tlačítka. Tieto tlačítka sú zobrazené, ak užívateľ *má právo*. Význam jednotlivých tlačítok je následovný. `Smazat vybrané` zmaže položky, ktoré boli vybrané. `Provést změny` uloží zmeny vykonané v tabuľke do modelu (zmena hodnoty popisu, prípadne zmena hodnoty). `Vrátit změny` vráti posledné zmeny. Na túto funkcionalitu sa vzťahujú nasledovné akcie: Pridanie novej položky, Zmena hodnoty pre položky, Zmazanie položky.

7.3.1 Detail položky

Táto obrazovka, viz 7.5, obsahuje detaily pre konkrétnu aktivitu. Obsahuje zoznam tagov, ktoré sú viazané na aktivitu. Pod zoznamom tagov sa náchadza výber variantu. Týmto spôsobom je možné meniť variant, do ktorého daná položka patrí. Tieto tagy je možné mazať i pridávať nové. Úpravy tagov a zmena variantov je možná len v prípade, že užívateľ *má právo*. Táto obrazovka je zobrazená len ako vyskakovacie okno.



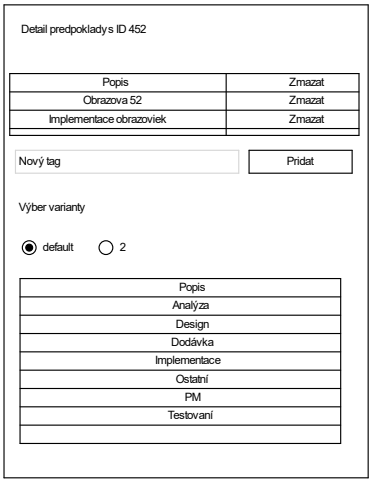
Obr. 7.5: Wireframe — detail položky



Obr. 7.6: Wireframe — prehľad predpokladov

7.4 Správa predpokladov

Táto obrazovka, viz 7.6, je podobná obrazovke pre správu položiek viz 7.3. Líši sa len tabuľkou na pravej strane. Tabuľka pozostáva z popisu predpokladu, počtu položiek, ktoré sú viazané na tento predpoklad, počtu tagov a variantu, do ktorého patrí. Posledným stĺpcom je tlačítko `Detail`. Kliknutie na toto tlačítko otvorí okno opísané v 7.4.1. Pri vytváraní nového predpokladu je potrebné vyplniť textové pole a vybrať variant, ku ktorému sa viaže. V prípade nevyplnenia variantu je použitý východzí variant (default). Tlačítka majú rovnaký význam ako tlačítka opísané v 7.3 s tým rozdielom, že tieto tlačítka neupravujú aktivity ale predpoklady. Práva na editáciu sú riešené rovnako ako v 7.3.



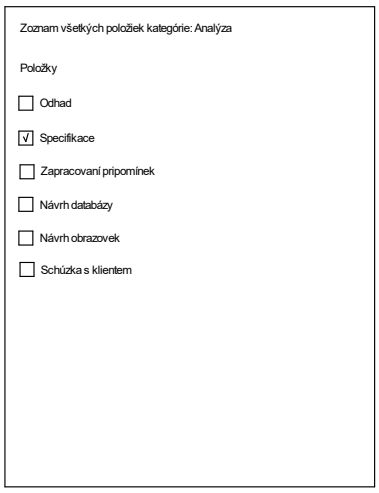
Obr. 7.7: Wireframe — detail predpokladu

7.4.1 Detail predpokladu

Táto obrazovka, viz 7.7, obsahuje detaily pre konkrétny predpoklad. Obsahuje zoznam tagov, ktoré su viazané na predpoklad. Pod zoznamom tagov sa nachádza výber variantu. Pod týmto výberom sa nachádza stromová štruktúra kategórií. V prípade kliknutia na konkrétnu kategóriu sa objaví nové okno obsahujúce všetky položky danej kategórie. Pri každej položke je checkbox, viz 7.8 . V prípade, že je checkbox zaškrnutý, znamená to, že daná položka je viazaná na tento predpoklad. Táto stromová štruktúra je použitá pre rýchlu navigáciu a rýchle označovanie aktivít. Táto obrazovka je zobrazená len ako vyskakovacie okno.

7.5 Navigačný panel

Navigačný panel je prítomný na väčšine obrazoviek, viz 7.1. Je rozdelený na dve úrovne. Prvá úroveň obsahuje informácie o odhade (názov odhadu, stav verzií, časovú jednotku odhadu). Pod stavom verzií sa rozumie informácia ohľadom toho, či existuje rozpracovaná verzia. V prípade, že existuje, je vypísaný vlastník zámku. V prvej úrovni je prepínač medzi MD a MH. Tento prepínač má informatívny charakter. Hodnoty v jednotlivých položkách sa v skutočnosti neprepočítavajú. Druhá úroveň je rozdelená na 3 sekcie. Sekcie budú opisované zľava doprava. V prvej sekcii sa nachádzajú dva selectboxy a jedno tlačítko. Prvý selectbox `Verze` špecifikuje, ktorá verzia je momentálne načítaná. V prípade, že sa vyberie iná hodnota, dôjde k refreshu stránky a načíta sa zvolená verzia. Formát, ktorý je zobrazený, je nasledovný — Číslo verzie, dátum uloženia verzie - čas uloženia - (autor). Druhý selectbox je referenčná verzia. Po vybratí referenčnej verzie je možné kliknuť na tlačítko *Ukázat*



Obr. 7.8: Wireframe — výber položiek

rozdíl. V prípade, že neboli zvolené rovnaké hodnoty, nastane presmerovanie na obrazovku 7.7.1.

Druhou sekciou je zoznam variantov. Pri každom variante je checkbox. V prípade, že je variant zaškrnutý, zobrazujú sa položky, ktoré patria do daného variantu. Hodnoty v tabuľke, ktorá zobrazuje kategórie, sa tiež okamžite prepočítajú. Táto sekcia obsahuje tlačítko `Správa variant`. Po kliknutí sa otvorí okno opísane v 7.6.

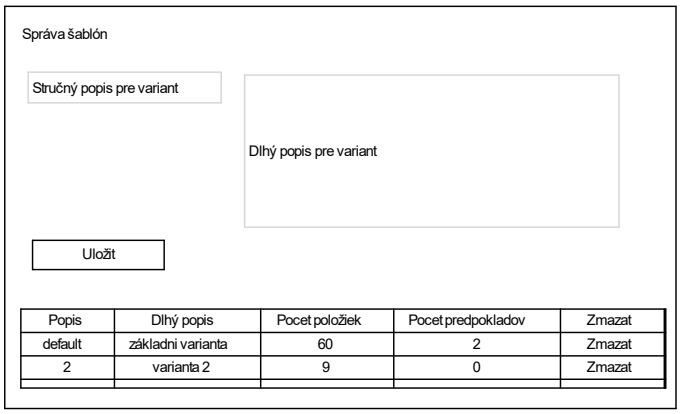
Poslednou sekciou je správa verzií. Nachádza sa tu 5 tlačítok, ktoré sú povolené, prípadne zakázané, v závislosti od práv užívateľa, od stavu samotného odhadu a od stavu danej verzie.

- *Nová verze* — tlačítko je povolené len v prípade, že užívateľ *má právo* na zápis do odhadu a momentálne neexistuje žiadna rozpracovaná verzia v tomto odhade. Kliknutím na tlačítko sa objaví okno, v ktorom je potrebné vyplniť názov novej verzie a stručný popis. Následne sa vytvorí nová verzia, ktorá je kópiou verzie, z ktorej bola vytvorená.

- *Upravit verzi* — tlačítko je povolené len v prípade, že zvolená verzia je v stave nedokončená (unfinished) a užívateľ *má právo* na zápis. V momente kliknutia na toto tlačítko sa užívateľ stáva majiteľom zámku. Tento zámok má informatívny charakter.

- *Smazat verzi* — tlačítko je povolené len v prípade, že užívateľ vlastní zámok. Kliknutím na tlačítko sa rozpracovaná verzia zmaže.

- *Uložit jako rozpracovanou* — tlačítko je povolené len v prípade, že užívateľ vlastní zámok. Kliknutím na tlačítko sa zmeny z modelu zapíšu do



Obr. 7.9: Wireframe — správa variant

databázy a užívateľ stratí zámok. Verzia je stále označovaná ako nedokončená (unfinished). Je možné uložiť aj verziu, ktorá obsahuje chyby. Chybou môže byť napríklad nesprávne vyplnená hodnota pre položku. Správnosť jednotlivých položiek kontroluje kalkulátor, viz 8.9.

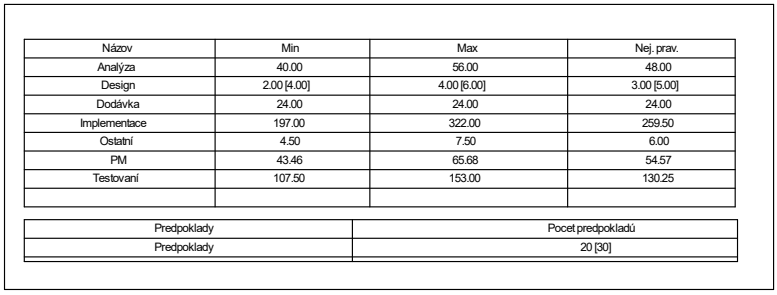
- *Uložit jako stabilní* — tlačítko je povolené len v prípade, že užívateľ vlastní zámok. Kliknutím tlačítka sa verzia uloží ako stabilná len v prípade, že odhad neobsahuje žiadnu chybu.

7.6 Správa variantov

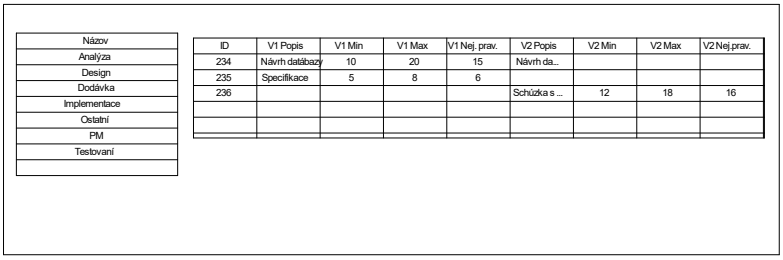
Obrazovka pre správu variantov, viz 7.9, obsahuje zoznam všetkých variantov, ako ich krátky, tak aj dlhý popis. V tabuľke je tiež zobrazená informácia o tom, koľko položiek a koľko predpokladov sa viaže na tento variant. V prípade, že užívateľ *má právo* tak je možné jednotlivé varianty zmazať (s výnimkou východzieho variantu). V prípade, že užívateľ chce zmazať variant, ktorý obsahuje aspoň jednu položku alebo predpoklad, objaví sa nové okno, v ktorom môže užívateľ presunúť všetky predpoklady a položky do inej, existujúcej kategórie alebo zmazať všetky položky a predpoklady, ktoré patrili do daného variantu. Obrazovka obsahuje textové polia (Krátky popis, Dlhý popis) pre vytvorenie nového variantu. Táto obrazovka je zobrazená len ako vyskakovacie okno.

7.7 Porovnávanie dvoch verzií

Porovnávanie dvoch verzií možno rozdeliť na dve obrazovky.



Obr. 7.10: Wireframe — porovnávanie kategórií



Obr. 7.11: Wireframe — porovnávanie položiek

7.7.1 Porovnávanie kategórií

Na obrazovke, viz 7.10, je zobrazená len jedna tabuľka v stromovej štruktúre. V prípade, že sa hodnota prvej bunky rovná hodnote druhej bunky, je vypísaná hodnota iba raz. Ak sa hodnoty nerovnajú, formát výpisu je nasledovný `x [y]`, kde `x` je hodnota bunky v pôvodnej verzii, hodnota `y` je hodnota bunky v porovnávanej verzii. Kliknutie na konkrétnu kategóriu zobrazí všetky položky danej kategórie tiež v porovnávacom režime.

7.7.2 Porovnávanie položiek

Na obrázku, viz 7.11, je možné vidieť štruktúru porovnávania. Čo na obrázku nie je vidieť, sú farebné odlíšenia. V prípade, že v pôvodnej verzii existuje položka, ktorá v porovnávanej nie je, písmo v celom riadku je zelenou farbou. Ak je to opačne, je červenou. V situácii, v ktorej došlo k zmene akejkoľvek hodnoty položky, je písmo oranžové. Detailnejšie informácie sa nachádzajú v tooltipe. Tooltip sa zjaví v momente, keď sa prejde na danú položku myšou. V tooltipe sú napríklad informácie o zmene variantu, zmene predpokladov a tagov.

Kapitola 8

Realizácia

Táto kapitola sa zaoberá priebehom tvorenia webovej aplikácie. Zaoberá sa problémami, ktoré nastali pri samotnom vývoji a tiež ich riešením.

8.1 Vytvorenie projektu v Spring Boot

Ako už bolo spomínané v 5.2.2, projekt je možné vytvoriť pomocou inicializátora na stránke [11]. Na tejto stránke je potrebné vyplniť Skupinu (Group), ktorá bola vyplnená hodnotou profinit.eu a artefakt (Artifact), ktorého hodnota bola Profinit Estimate. Pri výbere závislostí je možné zobraziť stránku so všetkými závislosťami. Pre túto aplikáciu boli vybrané nasledovné závislosti: Security (za účelom riešenia logovacieho systému), Web (závislosť potrebná pre vývoj webovej aplikácie), Vaadin (integrácia s frontendovým frameworkom), Neo4j (prepojenie s databázou Neo4j. Viac do hĺbky to rozoberá Milan Vancl v [24]), LDAP (závislosť potrebná na prepojenie Spring Security s LDAP protokolom).

Po vybratí týchto závislostí bolo možné vygenerovať projekt. Projekt vzniknutý týmto generátorom obsahoval vyplnený POM súbor, viz 5.1.5. Generátor vygeneroval triedu `ProfinitEstimateApplication`, ktorá obsahovala anotáciu `@SpringBootApplication`. Zdroj [25] vysvetľuje, čo daná anotácia znamená. Táto anotácia v sebe skrýva anotácie `@Configuration`, `@EnableAutoConfiguration` a `@ComponentScan`, ktoré boli veľmi často používané Spring Boot vývojármi, preto pre zjednodušenie vznikla anotácia, ktorá ich zaobaľuje. `@Configuration` dáva Springu informáciu o tom, že daná trieda môže obsahovať konfiguračné nastavenia, ako napríklad správu bean. Tieto konfigurácie je možné evidovať v XML súbore, ale priamo vývojári Springu Boot odporúčajú používať anotáciu `@Configuration` . `@EnableAutoConfiguration` slúži k nastaveniu Springu na základe závislostí, ktoré aplikácia obsahuje. V prípade tejto aplikácie sleduje závislosti, ktoré boli pridané cez Maven. `@ComponentScan` slúži k výhľadávaniu komponentov. Tieto komponenty je ná-

```
@Theme("valo")
@SpringUI
public class MyUI extends UI {
  @Autowired
  private Greeter greeter;

  @Override
  protected void init(VaadinRequest request) {
      setContent(new Label(greeter.sayHello()));
  }
}
```

Obr. 8.1: Prepojenie Vaadinu so Springom (prevzaté z [26])

sledne možné vložiť pomocou anotácie `@Autowired` do ostatných tried, ktoré túto závislosť potrebujú. DI je opísaný v 5.1.7.

Takto vytvorený projekt beží pod embedovaným Tomcat serverom. Táto vlastnosť je výhodná pri vývoji aplikácie, nakoľko nie je potrebné inštalovať a konfigurovať aplikačný server. K spusteniu aplikácie je potrebné mať nainštalovaný program Maven a Javu vo verzii 1.8. Aplikácia sa spúšťa príkazom `mvn spring-boot:run`.

8.2 Integrácia Spring Boot s Vaadin Frameworkom

Framework Vaadin má dobrú integráciu s frameworkom Spring Boot. Zdroj [26] uvádza postup ako jednoducho prepojiť Spring Boot aplikáciu s Vaadin Frameworkom. Pokiaľ bol projekt vytvorený so závislosťou na Vaadin, Maven si stiahol knižnice Vaadinu pri builde. Vďaka anotácii `@EnableAutoConfiguration` túto závislosť Spring Boot detekoval a automaticky nakonfiguroval integráciu s Vaadinom. Pre prepojenie Vaadin Frameworku s DI Springu je potrebné vytvoriť triedu s anotáciou `@SpringUI`, viz 8.1. Touto anotáciou je zabezpečené to, aby daná trieda patrila pod systém bean v Springu. Táto skutočnosť je potrebná hlavne kvôli tomu, aby sme do danej stránky mohli injektovať ďalšie beany pod správou Springu. Ako uvádza ukážka, je tam znázornená trieda, do ktorej je možné injektovať beany spravované Springom. Táto trieda ma tiež anotáciu `@Theme("valo")`. Téma slúži k vlastnému definovaniu vzhľadu jednotlivých komponentov.

Pohybovanie sa medzi stránkami možno riešiť dvoma spôsobmi. Prvým spôsobom je definovať pre každú stránku iné rozloženie, viz 8.2. Anotácia `@SpringView(name = UIScopedView.VIEW_NAME)` odovzdá Springu informáciu o tom, že ak užívateľ príde na webovú stránku #UIScopedView.VIEW_NAME

```
@UIScope
@SpringView(name = UIScopedView.VIEW_NAME)
public class UIScopedView extends VerticalLayout
implements View {
...
 @Override
  public void enter(ViewChangeEvent event) {
      // This view is constructed in the init() method()
  }
....
}
```

Obr. 8.2: Zobrazovanie rôznych View v závislosti od URL adresy (prevzaté z [26])

```
@SpringUI
public class InitScreen extends UI{

  @Autowired
  private SpringViewProvider viewProvider;
  protected void addMainPanel(){
    VerticalLayout mainPanel = new VerticalLayout();
    mainPanel.setHeightUndefined();
    layout.addComponent(mainPanel);
    Navigator navigator = new Navigator(this, mainPanel);
    navigator.addProvider(viewProvider);
 }
}
```

Obr. 8.3: Zaregistrovanie navigátora pre komponentu

tak sa zavolá metóda `enter` a nastane vykresľovanie definované v triede `UI-ScopedView`.

V tejto aplikácii je pohyb medzi stránkami riešený spôsobom, v ktorom sa neprekresľuje celá stránka, ale len jeden konkrétny komponent, viz 8.3. Takéto správanie bolo žiadúce hlavne kvôli tomu, že hlavička stránky a prepínanie sa medzi odhadmi a importom je prítomné na každej stránke. V použitom riešení nebolo potrebné do každej stránky pridávať tieto komponenty nanovo.

Ako je možné vidieť v ukážke, trieda `InitScreen` má injektovanú triedu `SpringViewProvider`, ktorá je zodpovedná za správu URL odkazov. Na základe anotácií `@SpringView(name = Názov)`, ktoré sú prítomné pred každou triedou, ktorá rieši obrazovku, Spring mapuje jednotlivé URL fragmenty na

danú triedu. Logika, ktorá obsahuje v sebe tieto informácie je prítomná práve v inštancii `viewProvider`. Inštancia s touto logikou bola namapovaná do inštancie `navigator`, ktorý je spojený s konkrétnym komponentom `mainPanel`. Tento kód má za následok, že pri prechádzaní medzi rôznymi stránkami definovanými URL adresou, sa bude meniť obsah len vo vnútri komponentu `mainPanel`.

8.3 Štruktúra projektu

Aplikáciu možno rozdeliť na dve časti. Prvou časťou je samotná aplikácia, viz 8.4 a druhou testy, viz 8.5. Ako je možné vidieť v tabuľke 8.4, jednotlivé vrstvy nie sú riešené ako nezávislé komponenty, ale sú v zdieľanom repozitári. Milan Vancl [24] sa zaoberal databázovou vrstvou, ako aj implementáciou správy importu a exportu. Táto práca sa zaoberala vytvorením prezetančnej vrstvy a vytvorením kalkulátora, viz 8.9. Aplikačnú vrstvu možno rozdeliť do dvoch oblastí. Oblasť aplikačnej vrstvy, ktorá súvisela s načítavaním dát do modelu z dabatabázy ako aj ukladanie modelu do databázy, riešil Milan Vancl v [24]. Táto práca v aplikačnej vrstve implementovala funkcie, ktoré boli potrebné pre správne fungovanie prezentačnej vrstvy. Jednalo sa konkrétne o pridávanie, mazanie položiek a predpokladov, prepájanie predpokladov s položkami, správe variantov (vytváranie, priradzovanie položiek a predpokladov k variantu, mazanie).

8.4 Prihlasovanie do aplikácie

Systém prihlasovania je riešený pomocou zabudovaného Spring Security LDAP servera. Tento server obsahuje prihlasovacie údaje troch rôznych užívateľov a to `jsnow`, `mrayder` a `sbaratheon`. Títo traja užívatelia boli vytvorení pre demonštrovanie systému oprávnenia. Samotný LDAP server spolu s integráciou s aplikáciou vychádzal z manuálu, viz [27]. Samotné práva k jednotlivým odhadom sú uložené v databáze tejto aplikácie. LDAP server slúži na overenie, či má daný užívateľ prístup do aplikácie. Jednotlivé práva pre užívateľa sa po úspešnom prihlásení načítavajú z databázy prostredníctvom API z databázovej vrstvy.

8.5 Práca s dátami

Pri zobrazovaní jednotlivých obrazoviek bolo potrebné mať prístup k dátam. Práca Milana Vancla [24] sa zaoberala práve touto problematikou. V aplikácii implementoval databázovú vrstvu, ktorej API bolo možné využívať pri implementácii frontendovej časti. Sprostredkovanie k reálnym dátam, ako aj ukladanie a editácia bolo riešené týmto prístupom. Rozdelenie aplikácie na trojvrstvú architektúru umožnilo vyvíjať aplikáciu takýmto spôsobom.

```
main.java.eu.profinit.estimate
├── configuration ..... Konfigurácia pre LDAP
├── model ..... Databázová vrstva
├── repo ..... Repozitár k správe databázy
├── service ..... Aplikačná vrstva
├── session ..... Triedy na správu relácie
├── utility
│   ├── calculator ..... Výpočet priemeru a očakávanej hodnoty
│   ├── category ..... Výpočet hodnôt pre kategórie
│   ├── deleted ..... Evidencia zmazaných entít
│   ├── file ..... Načítanie súborov
│   ├── porter ..... Správa importu a exportu odhadu
│   ├── reference ..... Kalkulátor pre výpočet hodnôt položiek
│   └── uploader ..... Podpora nahrávania súboru
└── view ..... Prezentačná vrstva
    ├── compare ..... Obrazovky pre porovnávanie dvoch verzií
    ├── converter ..... Konvertory pre komponent Grid
    ├── model ..... Dátový model pre prezentačnú vrstvu
    └── utility
        └── undo ..... Podpora pre vrátenie posledných zmien
            ├── item ..... Vrátenie zmien na obrazovke položiek
            └── precondition ... Vrátenie zmien na obrazovke predpokladov
```

Obr. 8.4: Štruktúra aplikácie

```
test.java.eu.profinit.estimate
├── repo ..... Testovanie repozitára
└── utility ..... Testovanie utilít ako export, import a výpočet hodnôt položiek
```

Obr. 8.5: Štruktúra aplikácie — testy

8.6 Načítavanie odhadu

Pri zobrazovaní odhadu bolo potrebné načítať informácie o odhade. Prvá stránka, ktorá sa zobrazila, obsahovala informácie o všetkých kategóriách ako aj vypočítané hodnoty min, max, odhadováno. K tomu, aby tieto údaje mohli byť zobrazené, bolo potrebné načítať celý odhad v danej verzii. Pod celým odhadom sa rozumejú všetky atribúty, ktoré definujú odhad. Vzťahy medzi jednotlivými atribútmi definuje dátový model, viz 8.6. Túto štruktúru kopírujú triedy v balíčku `model`. Mapovanie z grafovej databázy na objekty v Jave zastrešuje Neo4j — OGM Object Graph Mapper. Samotnú prácu s databázou zastrešujú triedy v balíčku `repo`, ktoré komunikujú s databázou prostredníc-

tvom Spring Data Neo4j. Viac sa týmto technológiam venuje práca [24].

Tento odhad je potrebné obohatiť. Jednotlivé hodnoty v položkách odhadu nemusia byť striktne číslom. Hodnota môže obsahovať vzorec, prípadne byť závislá na inej hodnote. Preto je potrebné vypočítať skutočné hodnoty pre všetky položky. Samotný výpočet je opísaný v 8.9. Tento výpočet edituje inštancie, ktoré sa nachádzajú v balíčku `model`. K tomu, aby tento výpočet neprebiehal pri každom prekreslení stránky, je použitá trieda `SessionManager`, ktorá je Springovskou beanou s rozsahom (ang. scope) typu `SessionScope`. Typ `SessionScope` znamená, že v momente, keď sa injektuje táto beana do iných tried, je vytvorená jedinečná inštancia pre každú reláciu (ang. session). Názorným príkladom môže byť prihlásenie sa. Užívateľ vyplní meno a heslo a úspešne sa prihlási. Pri zobrazení ďalšej stránky sa už od užívateľa nevyžaduje heslo, pretože aplikácia má informáciu o tom, že tento užívateľ bol úspešne prihlasený. Táto informácia je uložená v relácii.

Pri načítavaní stránky sa zakaždým skontroluje, či je v inštancii `SessionScope` obohatený odhad. V prípade, že tento odhad obohatený nie je (užívateľ pristupuje k odhadu prvýkrát alebo mení verziu daného odhadu), celý odhad sa načíta z databázy a obohatí sa pomocou kalkulátora.

Obdobným spôsobom je riešené porovnávanie dvoch verzií. Do `SessionScope` je tiež vložený porovnávaný odhad.

8.7 Komponent grid

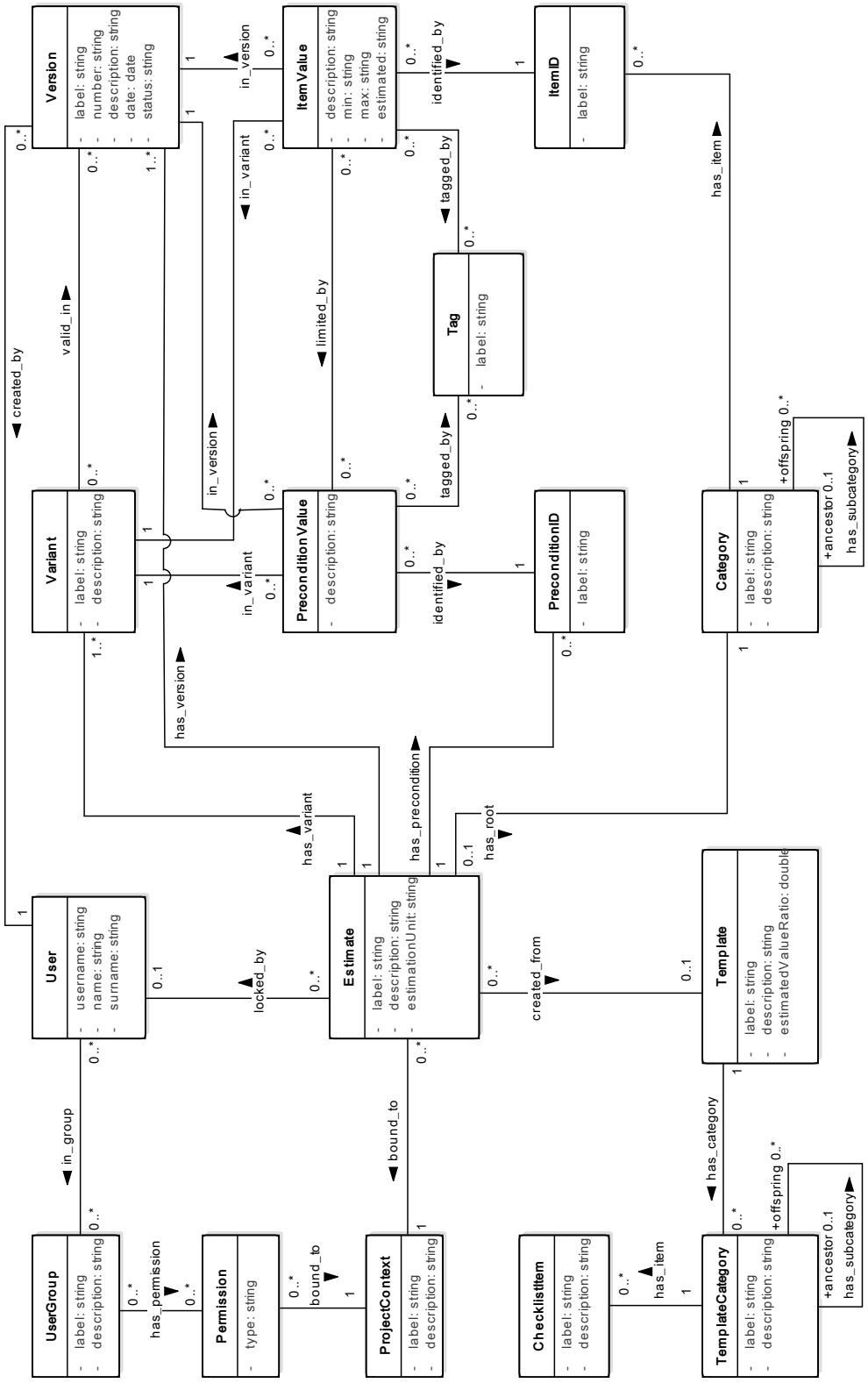
Vo väčšine prípadov bol na zobrazovanie dát použitý komponent Grid. Tento komponent má kvalitne vypracovanú dokumentáciu. Stručný zoznam funkcií je čerpaný z [28].

Grid je komponent na zobrazovanie dát v tabuľke. Dáta v tabuľke sú viazané priamo v gride alebo pomocou kontajnerov.

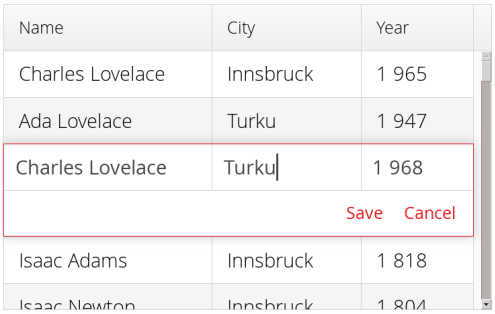
8.7.1 Ako dostať dáta do Gridu

Kontajner je štruktúra, ktorá obsahuje samotné dáta. V tejto aplikácii bol najčastejšie používaný kontajner `BeanItemContainer`. Parametrom kontajnera je kolekcia vlastnej triedy. Parametre jednotlivých atribútov sú zistené pomocou getterov a setterov. Tento kontajner je následne potrebné vložiť do gridu. Grid zistí prítomnosť kontajnera a automaticky pridá potrebné stĺpce na ich zobrazenie. Všetko prebieha v pozadí.

Druhou možnosťou je nepoužiť kontajner, ale manuálne pridávať stĺpce. Jednotlivý stĺpec je definovaný názvom a typom. Toto riešenie je výhodnejšie v prípade, že je potrebné zobraziť iba zopár hodnôt v tabuľke. Riešenie pomocou `BeanItemContainer`, vytvorí pre každý getter jeden stĺpec. Ak máme entitu s množstvom atribútov a je potrebné zobraziť iba zopár, je potrebné väčšinu stĺpcov schovať, prípadne odstrániť.



Obr. 8.6: Dátovy model (prevzaté z [24])



Obr. 8.7: Grid v buffered móde (prevzaté z [28])

8.7.2 Editovanie položiek

Editovať hodnoty v gride je možné v dvoch módoch. Nazývajú sa buffered a unbuffered.

Buffered, viz 8.7, v editovacom móde obsahuje popisky ako `Save` a `Cancel`. Tieto texty je možné zmeniť. Pohybovanie sa a editácia nie je taká pohodlná ako v unbuffered verzii, nakoľko očákavaná reakcia pri stlačení klávesy `enter` je uloženie zmien a posunutie sa o riadok nižšie. Unbuffered je mód, ktorý sa viac podobá editácií v Exceli. Problémom pri unbuffered móde je nemožnosť odchytiť akciu, v ktorej užívateľ zmení hodnotu. Tento problém bol vyriešený spôsobom, v ktorom užívateľ po skončení editácií potrebuje tieto zmeny uložiť do modelu stlačením tlačítka `Provést zmeny`, ktoré je viditeľný na obrázku 7.4. Po stlačení spomínaného tlačítka sa detekciou zistí, ktoré hodnoty boli zmenené. Urobí sa prepočet vysvetlený v 8.9, prekreslí sa tabuľka obsahujúca kategórie (zmena hodnoty vynúti celkové súčty v kategóriách) a tiež tabuľka obsahujúca položky. Pri editácii hodnoty min na hodnotu `=5+5` je táto hodnota zobrazená až do momentu, kým sa neklikne na tlačítko `Provést zmeny`.

8.7.3 Konvertory

Grid dokáže v tabuľkách zobrazovať aj hodnoty z vlastných objektov. Pre správne zobrazovanie je potrebné implementovať vlastný konvertor, ktorý určí logiku toho, čo sa má zobraziť. Táto funkcionalita bola potrebná pre vypĺňanie jednotlivých hodnôt pre položky. Správanie je také isté ako v Exceli. Do danej bunky je možné napísať vzorec, napríklad `=5+5`, no po opustení tejto hodnoty sa nezobrazuje táto hodnota, ale hodnota `10`. Táto funkcionalita bola dosiahnutá pomocou použitia vlastnej triedy `PairValueAndCalculatedValue`, ktorá obsahovala vypočítanú hodnotu aj skutočnú hodnotu. Následne boli vytvorené dva konvertory. Jeden pre zobrazovanie vypočítanej hodnoty, ktorý bol použitý pre zobrazovanie mimo editovacieho režimu. Tento konvertor bol nastavený priamo na daný stĺpec, ktorý obsahoval dáta typu `PairValueAndCalculatedValue`. Druhý konvertor bol použitý pre zobrazovanie hodnoty

v editovacom móde. Nastaviť, aby bol použitý tento konvertor v editovacom režime nebolo také jednoduché ako v minulom prípade. Grid umožňuje pre každý stĺpec nastaviť vlastný editovací komponent (napr. TextField). V tomto prípade boli na atribúty min, max, odhadováno vytvorené tri rôzne inštancie typu TextField, ktorými sa nastavil druhý typ konvertora. Tieto TextFieldy bolo potrebné prepojiť s jednotlivými stĺpcami.

8.7.4 Radenie

Komponent grid umožňuje tiež radiť tabuľku podľa jednotlivých stĺpcov. Hodnoty typu min, max, odhadováno sú v tabuľke ukladané ako Stringy a nie double. Dôvod je ten, že jednotlivé hodnoty nemusia byť striktne len číselného formátu. V prípade nevalidného vstupu je zobrazená chyba. Viac o kontrole, viz 8.9.1. Hodnoty sú uložené ako `String`, ktorý reprezentuje čísla. Pri radení grid používa funkciu `compareTo`, ktorá je preťažená z interfacu `Comparable`. Ak sú číselné hodnoty chápané ako Stringy, pri usporiadaní prvkov 10,9,125 od najväčšieho po najmenšie to dopadne nasledovne: 9,125,1, čo neodráža skutočnosť. Pre tieto účely bola vytvorená trieda `WrapperSortedString`, ktorá mala upravenú funkciu porovnávania aby reflektovala skutočnosť.

8.8 Komponent TreeGrid

Pre zobrazenie stromovej štuktúry bolo potrebné do projektu pridať doplnok Vaadin TreeGrid [17]. Doplnky sa do projektu pridávajú pomerne jednoducho. Je potrebné do POM súboru pridať jednu závislosť a definovať repozitár, z ktorého sa má doplnok stiahnuť, viz 8.8. V prípade, že doplnok obsahuje aj klientskú časť kódu, je potrebné skompilovať widget obsahujúci tento doplnok pre aplikáciu. Ku kompilácií widgetu je potrebné v adresári **src/main/resources** vytvoriť súbor **Widgets.gwt.xml**, v ktorom sa definuje, ktoré doplnky sú používané. V tomto prípade bolo potrebné vytvoriť súbor s obsahom, viz 8.9. Predposledným krokom bolo pridanie anotácie do triedy, ktorá využívala funkcionalitu daného widgetsetu. V tomto prípade stačilo tento widget zaregistrovať do hlavnej triedy Vaadinu a to `InitScreen` pomocou anotácie `@Widgetset("Widgets")` .

Tento komponent je rozšírením klasického gridu s podporou stromového zobrazenia. Samotné dáta sú uložené v triede, ktorá dedí od `HierarchicalContainer`. Zoznam metód a spôsob, akým majú byť implementované je možné vidieť na demo ukážke prístupnej na [30]. Podobným spôsobom bol implementovaný Container pre zobrazenie stromového zobrazenia kategórií.

```
<dependency>
    <groupId>com.vaadin</groupId>
    <artifactId>vaadin-treegrid</artifactId>
    <version>0.7.4</version>
</dependency>

<repository>
    <id>vaadin-addons</id>
    <url>http://maven.vaadin.com/vaadin-addons</url>
</repository>
```

Obr. 8.8: Pridanie Vaadin doplnku do POM definície (prevzaté z [17])

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC
"-//Google Inc.//DTD Google Web Toolkit 2.5.1//EN"
 "http://google-web-toolkit.googlecode.com/
svn/tags/2.5.1/distro-source/core/
src/gwt-module.dtd">
<module>
    <inherits name="org.vaadin.treegrid.WidgetSet" />
    <inherits name="com.vaadin.DefaultWidgetSet" />
</module>
```

Obr. 8.9: Vytvorenie Widgetsetu (prevzaté z [29])

8.9 Implementácia kalkulátora

Jednou z požiadavok je, aby aplikácia vedela spracovávať referencie na iné položky ako aj jednoduché matematické operácie, ako tomu bolo v programe Excel. Formát vypĺňania jednotlivých hodnôt je nasledovný. Hodnota môže byť číslo, matematický vzťah podporujúci jednoduché operácie typu súčet, rozdiel, násobenie, delenie. Podpora priority vyhodnocovania výrazov je taktiež podporovaná. Podobne ako v Exceli je potrebné hodnotu obsahujúcu výraz začať znakom `=`. Príkladom validného vstupu je `=9(5+5)*1.25-14)*0.3`. Druhou podmienkou je, aby bolo možné odkazovať na iné hodnoty. Každá položka má jednoznačne určené ID a má hodnoty min, max, odhadováno. Hodnotu každej položky je možné vyjadriť formátom `#45.min#, #45.max# , #45.est#`, čo vyjadruje odkaz na minimálnu, maximálnu a odhadovanú hodnotu položky s ID 45.

Pri vyhodnocovaní týchto referencií je dôležité poradie, v akom budú jednotlivé hodnoty vyhodnocované. Predstavme si situáciu, v ktorej by sme mali

3 položky, pričom 1. by odkazovala na 2. a 2. na 3. V takomto prípade by sme najprv museli vyhodnotiť hodnotu 3. položky, po jej vyhodnotení by bolo možné vyhodnotiť 2. položku a nakoniec vyhodnotiť 1. položku. Riešením je nájdenie topologického usporiadania. Topologické usporiadanie možno v našom prípade definovať ako poradie, v akom je potrebné vyhodnocovať jednotlivé hodnoty. Pre riešenie tohoto problému bol použitý upravený Kahnov algoritmus. Ide o grafový algoritmus, preto je potrebné k pochopeniu Kahnovho algoritmu vysvetliť pojmy:

- **Uzol** v našom prípade možno chápať ako konkrétnu hodnotu konkrétnej položky. Nakoľko má každá položka hodnoty min, max, odhadováno, jedna položka sa skladá z troch uzlov.
- **Hrana** je spojenie medzi dvomi uzlami. Hrany môžu byť orientované alebo neorientované. V našom prípade bude hrana reprezentovať závislosť jednej položky na druhej. Jedná sa o orientovanú hranu, nakoľko je potrebné rozlišovať, ktorý uzol je závislý od ktorého uzla. Ak je uzol A závislý na uzle B, nie je to to isté, ako keď je uzol B závislý na uzle A.
- Pojmy **vstupná hrana** a **výstupná hrana** možno najlepšie definovať na názornej ukážke. Predstavme si situáciu, v ktorej máme dve položky. Prvá má ID 1, druhá má ID 2. Hodnota min položky 1 sa rovná =#2.min#. Znamená to, že medzi položkou 1 a položkou 2 je hrana. Položka 1 je závislá na položke 2. Na túto hranu možno pozerať z dvoch pohľadov. Pre položku 2 sa táto hrana javí ako vstupná. Zoznam vstupných hrán, teda možno chápať ako zoznam položiek, ktoré sú závislé na tejto položke. Pre položku 1 sa táto hrana javí ako výstupná. Zoznam výstupných hrán možno chápať ako zoznam položiek, na ktorých je daná položka závislá.
- **Cyklická závislosť** znamená, že hodnotu nie je možné vyčísliť. Príkladom je situácia, v ktorej uzol A ukazuje na uzol B a uzol B ukazuje na uzol A.

Upravený Kahnov algoritmus vychádza z originálneho Kahnovo algoritmu [31]. Tento algoritmus možno opísať v týchto bodoch

1. Prejdi všetky uzly a urč im vstupné a vystupné hrany.
2. Urob kópiu výstupných hrán.
3. Nájdi všetky uzly, ktorých počet výstupných hran je rovný hodnote 0 a vlož ich do fronty.
4. Vyber a následne odstráň uzol z fronty. Tento uzol označme A. Uzlu je možné vyčísliť jeho hodnotu, viz 8.9.2. Na základe vstupných hrán zisti,

ktoré uzly su závislé na uzle A. Týmto uzlom odstráň z výstúpných hrán hranu, ktorá ukazovala na uzol A. V prípade, že počet výstupných hrán (uzol už nebude závislý na žiadnej nevypočítanej hodnote) bude rovný hodnote 0, pridaj uzol do fronty.

5. Posledný krok opakuj, kým nebude fronta prázdna.

6. Prejdi všetky uzly a skontroluj počet ich výstupných hrán. V prípade, že výstupná hrana nie je rovná hodnote 0, znamená to, že v závislostiach existuje cyklická závislosť.

Tento algoritmus nevysvetľuje ako dôjde k samotnému vyčísleniu hodnoty. V prvom kroku algorimu je spomínané, že je potrebné prejsť všetky uzly a určiť im vstupné a výstupné hrany. Postup určovania týchto hrán a samotnej hodnoty je nasledovný.

1. Skontroluj, či hodnota nie je číslo. Ak je, nastav túto hodnotu ako vypočítanú a skonči.

2. Skontroluj, či hodnota nie je matematický výraz bez externých závislostí. Ak je, urč jej hodnotu, viz 8.9.2 a skonči.

3. Skontroluj, či výraz obsahuje referencie na iné položky. Ak áno, danému uzlu pridaj do výstupných uzlov všetky tieto závislosti. Všetkým uzlom, na ktoré odkazuje daný výraz, pridaj vstupnú hranu na tento uzol.

Pri kontrole prebieha tiež kontrola, či výraz odkazuje na hodnotu, ktorá je prítomná, ako aj na validitu výrazu. Implementácia používa `HashMap`, v ktorej kľúč je jednoznačné id a hodnota je samotný uzol. Uzol je reprezentovaný vypočítanou hodnotou, skutočnou hodnotou. Vstupné a výstupne hrany sú riešené ako `HashSet`, preto samotné pridávanie a odstraňovanie je riešené v konštantnom čase.

8.9.1 Chyby pri vyhodnocovaní

Pri vyhodnocovaní jednotlivých položiek môžu nastať tieto chyby

- **CYCLE** — hodnota nemohla byť vyhodnotená, pretože je v cyklickej závislosti.
- **INVALID** — hodnota nemohla byť vyhodnotená, pretože odkazuje na hodnotu, ktorá obsahuje chybu.
- **UNKNOWN_REFERENCE** — hodnota nemohla byť vyhodnotená, pretože odkazuje na položku s ID, ktoré neexistuje.
- **BAD_FORMAT** — hodnota nemohla byť vyhodnotená, pretože je v nevalidnom formáte. Príklad `5+5`, prípadne `=5+#2.sss#`.

```
public static double evaluateFromString(String input)
throws ParseException,
EvaluationException, NumberFormatException {
    ExpressionParser parser =
      new SpelExpressionParser();
    String resultInDouble =
      convertNumbersToDouble(input);
    Expression exp =
      parser.parseExpression(resultInDouble);
    return exp.getValue(Double.class);
}
```

Obr. 8.10: Vyhodnocovanie výrazu pomocou SpEL)

- **EMPTY** — výraz je prázdny.

Tieto chyby sa taktiež zobrazujú v GUI v samotnom gride.

8.9.2 Spring Expression Language

Vyhodnocovanie matematického výrazu bolo riešené použitím technológie Spring Expression Language (ďalej už len SpEL), viz 8.10. String je vo formáte `(5+5)/2*3`, teda výraz nezačína znakom `=`. Do tejto funkcie vstupuje reťazec, ktorý obsahuje len čísla. Hodnota formátu `=5+#3.min#` je pred volaním tejto funkcie upravená spôsobom, v ktorom je podreťazec `#3.min#` nahradený vypočítanou hodnotou.

Problémom pri použití tejto technológie bola celočíselná aritmetika. Výraz `1.0 + 5/3` bol vyhodnotený ako `2`. V 8.10 je možné vidieť volanie metódy `convertNumbersToDouble(input)`, ktorá upravuje reťazec na formát, v ktorom sú všetky čísla vyjadrené ako čísla formátu double. Príkladom môže byť hodnota `1+4+2`. Táto hodnota je upravená na hodnotu `1.0+4.0+2.0`.

8.9.3 Zmena hodnôt

Časti kalkulátora sú používané tiež v časti, v ktorej užívateľ zmení, pridá alebo odstráni hodnotu. Každej oblasti sa venuje samostatná sekcia.

8.9.3.1 Pridanie novej hodnoty

Pridanie hodnoty je najjednoduchšia úloha. Nová hodnota pozostáva z troch uzlov. Každému uzlu sa vypočíta hodnota. Pri výpočte hodnoty môžu nastať tri situácie.

- Hodnota je číslo.

- Hodnota je vzorec, ktorý sa vypočíta pomocou SpEL.
- Hodnota obsahuje referencie. Všetky uzly majú už vyplnené hodnoty, prípadne vyplnenú chybu. V prípade, že hodnota odkazuje na uzol s chybou, pre tento uzol sa nastaví chybový príznak a skončí sa. V prípade, že sú všetky položky validné, vypočíta sa hodnota pomocou SpEL.

Cyklus v tejto fáze nemôže vzniknúť, pretože žiadna položka nemohla odkazovať na položku s týmto ID, nakoľko ID pre túto položku ešte neexistovalo.

8.9.3.2 Odstránenie hodnoty

Pri odstraňovaní položky je potrebné kontrolovať vstupné hrany všetkých troch uzlov. Môže sa stať, že užívateľ zmazal položku, na ktorú ukazovali iné položky. V prípade, že takáto situácia nastane, je potrebné rekurzívne nastaviť všetkým uzlom chybový príznak na `UNKNOWN_REFERENCE`. Všetkým uzlom, ktoré odkazovali na túto hodnotu, je potrebné nastaviť príznak `INVALID` a takto rekurzívne prejsť všetký vnútorné hrany a všetkým opäť nastaviť príznak `INVALID`.

8.9.3.3 Zmena hodnoty

Pri zmene hodnoty môže nastať viacero situácií. Zo začiatku sa kalkulátor snaží určiť hodnotu nového výrazu. Najprv sa skontroluje, či nová hodnota nie je číslo. Ak hodnota nie je číslo, zistí sa, či výraz obsahuje referencie na iné položky. V prípade, že neobsahuje, je použitý SpEL na výpočet hodnoty.

Predpokladajme, že výraz neobsahoval žiadnu referenciu. To znamená, že výraz je vyčíslený alebo obsahoval chybu. V prípade, že je výraz vyčíslený, je potreba rekurzívne aktualizovať všetky položky, ktoré odkazovali na túto položku. Pri aktualizácii môžu nastať dve situácie. Prvou je, že uzol, ktorý odkazoval na menený uzol bol v cykle, ktorý spôsoboval uzol, ktorý ma momentálne novú hodnotu. V takom prípade je možné hodnotu pre tento uzol vyhodnotiť a zrušiť chybový príznak, ktorý bol `CYCLE` a v prípade možnosti úspešného výpočtu za použitia SpEL, je možné zrušiť chybový príznak, prípadne ho nastaviť na `INVALID`. Zrušenie tohoto cyklu je nutné propagovať rekurzívne.

V predošlom odseku bola rozobratá situácia, v ktorej zmenená hodnota neobsahovala referenciu. V tomto odseku je rozobratá situácia, v ktorej zmenená hodnota referenciu obsahuje. Prvým krokom je zistiť, či táto nová hodnota nemá za následok vznik cyklu. Pomocou algoritmu BFS, viz [32], sa skontrolujú všetky výstupné hrany, pričom sa kontroluje, či sa medzi týmito uzlami nenachádza uzol, ktorý je práve menený. Ak sa tento uzol nájde, znamená to, že zmenená hodnota má za následok vznik cyklu. Preto je potrebné propagovať chybový príznak `CYCLE` všetkým uzlom, ktoré odkazovali na tento menený uzol. Samozrejme, nielen priamym uzlom, ale aj všetkým uzlom rekurzívne.

V prípade, že zmenená hodnota nevytvorí nový cyklus, je možné vyhodnotiť hodnotu tejto položky pomocou SpEL, ako bolo spomínané v poslednej odrážke v 8.9.3.1. V prípade, že novú hodnotu bolo možné vypočítať, postupuje sa rovnako ako v prvom odseku tejto sekcie, v ktorom bolo opísané, akým spôsobom sa aktualizujú položky, ktoré odkazovali na tento menený uzol.

8.9.3.4 Vrátenie celého riadku

Užívateľ môže zmazať položku, čo môže viesť k tomu, že položkám, ktoré záviseli na tejto hodnote bol nastavený chybový príznak `UNKNOWN_REFERENCE`. Táto funkcia bola popísaná v 8.9.3.2. Aplikácia umožňuje vracať posledné vykonané zmeny, viz 8.10. Pri vrátení tejto hodnoty nemožno postupovať spôsobom, akým bolo postupované pri pridávaní novej položky, pretože pri pridávaní novej položky nebola žiadna hodnota položky závislá na tej novej. Pri vrátení ale tomu tak nemusí byť. Pri odstraňovaní položky boli informácie o všetkých troch uzloch skopírované a uložené. Pri vrátení sa tieto skopírované údaje použijú a zistia sa vstupné hrany. Prejdú sa všetky uzly, ktoré boli v `HashSet` vstupných hrán (všetky tieto položky mali nastavené chybový príznak `UNKNOWN_REFERENCE`, kvôli tomu, že ukazovali na hodnotu, ktorá po vymazaní neexistovala). Tieto hodnoty je možné skúsiť vyhodnotiť nanovo. Nie je zaručené, že tieto uzly už bude možné v poriadku vyhodnotiť. Uzol, môže byť tiež v cykle, prípadne môže odkazovať na ďalšiu hodnotu, ktorá neexistuje. V prípade, že uzol bolo možné vyhodnotiť, je potrebné rekurzívne prejsť opäť všetky vstupné hrany. Ak mal uzol hodnotu `UNKNOWN_REFERENCE`, všetky uzly, ktoré odkazovali na túto hodnotu boli tiež nevalidné s chybovým príznakom `INVALID`.

8.10 Vrátenie zmien

Jednou z požiadavok na vývoj aplikácie bola funkcia Undo. Jedná sa o funkciu, pomocou ktorej je možné vrátiť posledné vykonané zmeny. Táto funkcionalita bola implementovaná do editácie samotných položiek ako aj do editácie predpokladov. Na dosiahnutie tejto funkcionality bol použitý návrhový vzor Command Pattern.

Myšlienka vychádza z toho, že v programe sa bude evidovať zoznam všetkých úkonov, ktoré užívateľ vykonal. Tieto úkony sú ukladané do fronty. Ak je k dispozícií presný zoznam úkonov, ktoré užívateľ vykonal, je možné ku každej akcii vykonať akciu opačnú. Názorným príkladom je situácia, v ktorej užívateľ pridal novú položku. Užívateľ chce túto aktivitu vrátiť, teda je potrebné položku zmazať. Ku každej akcii je jednoznačne určená protiakcia. Pre pridanie položky je protiakciou odstránenie tejto položky. Pri zmene hodnoty položky je protiakciou opätovná zmena na starú hodnotu.

V aplikáci bolo vytvorené rozhranie s názvom `UndoWrapper`, ktoré obsahovalo metódu `undo`. Rozhranie implementovali rôzne triedy. Pre správu položky

to boli triedy `UndoAddItemValue`, `UndoChangedItemValue` a `UndoDeletedItemValue`. Aplikácia používala dve fronty typu `UndoWrapper`. Prvá fronta slúžila na evidenciu úkonov pre položky a druhá na evidenciu úkonov pre predpoklady. Pri vykonaní zmeny (napr. pridanie novej položky) bola do prvej fronty zaradená inštancia `UndoAddItemValue`. Pri stlačení tlačítka `Vrátit zmeny` sa z fronty jednoducho vybral a odstránil posledný pridaný prvok a zavolala sa metóda `undo`. Samotná fronta je uložená triede `SessionManager`, teda v beane spravovanej Springom kvôli tomu, aby tieto zmeny bolo možné vykonať aj po prechode medzi stránkami, napríklad pri prechode do inej kategórie.

Kapitola 9

Testovanie a dokumentácia

9.1 Unit testy pre kalkulátor pre vyhodnotenie výrazov

Unit testovanie je automatické testovanie, ktoré overuje korektnosť implementácie. V rámci aplikácie boli spravené unit testy pre komponent, ktorý vyhodnocuje hodnoty buniek, viz 8.9.

Pre otestovanie funkčnosti bolo spravených 10 testov, ktorých úlohou bolo pokryť všetky situácie, ktoré mohli pri vyhodnocovaní nastať. Kontrolovala sa detekcia cyklov, správnosť výpočtov aj správne detekovanie problémov.

- základny test — úlohou je zistiť, či kalkulátor dokáže spracovať číselné hodnoty, ktoré sú uložené v reťazci
- test jednoduchých výrazov — úlohou je zistiť, či kalkulátor dokáže validne vyhodnotiť výrazy typu `=(0.5+0.4)*2`
- test jednoduchého odkazovania — úlohou je zistiť, či kalkulátor dokáže správne spracovať odkazovanie na inú hodnotu
- zložitejší test odkazovania — úlohou je zistiť, či algoritmus na určenie správneho topologického usporiadania funguje správne a či vyhodnotenie jednotlivých hodnôt bolo validné
- detekcia cyklov test — úlohou je zistiť, či kalkulátor dokáže detekovať cyklus a či správne nastavil chybový príznak hodnotám, ktoré boli v cykle
- test detekcie chybových vstupov — úlohou je zistiť, či kalkulátor dokáže detekovať nevalidné vstupy všetkých typov, viz 8.9.1

Druhou časťou boli testy, ktoré sa zaoberali zmenami, ktoré boli opísané v 8.9.3.

- test zmeny hodnoty bez referencie — úlohou je zistiť, či sa validne prepočítala zmenená hodnota a tiež všetky hodnoty, ktoré odkazovali na uzol, ktorý bol menený
- test zavedenia cyklu — úlohou je zistiť, či kalkulátor dokáže detekovať zavedenie cyklu, t.j. či kalkulátor úspešne nastaví chybový príznak všetkým prvkom, ktoré sa kvôli zmenenej hodnote nachádzajú v cykle
- test zavedenia chyby — úlohou je zistiť, či kalkulátor dokáže v prípade chyby nastaviť chybový príznak všetkým uzlom, ktoré odkazujú na túto hodnotu
- test zmeny hodnoty s referenciou — úlohou je zistiť, či sa validne prepočíta zmenená hodnota a tiež všetky hodnoty, ktoré odkazovali na uzol, ktorý bol menený
- test odstránenia a pridania položky — úlohou je zistiť, či sa po odstránení položky správne nastaví chybový príznak `UNKNOWN_REFERENCE` všetkým prvkom, ktoré odkazovali na už neexistujúcu položku. Druhou úlohou je pridanie tejto položky späť a kontrola, či položky, koré predtým odkazovali na túto hodnotu, už neobsahujú chybový príznak.

9.2 Manuálne testovanie scenárov

Testovanie frontendu je riešené pomocou testovacích scenárov, ktoré pokrývajú všetky prípady užitia, viz 4.2. Aplikácia je testovaná na testovacích dátach, ktoré je možné nahrať pomocou skriptu, ktorý sa nachádza na priloženom CD. Na priloženom CD je tiež návod na konfiguráciu skriptu a samotné spojazdnenie aplikácie.

Skript vytvorí testovacie dáta, v ktorých sú traja užívatelia: `jsnow`, `sbaratheon`, `mrayder`. Heslo k jednotlivým užívateľom je zhodné s menom. Skript tiež vytvorí jeden odhad pracnosti, ktorý je v kontexte `CASTLE_BLACK`. K tomuto kontextu má užívateľ `jsnow` práva na čítanie aj na zápis. Užívateľ `sbaratheon` má k tomuto kontextu len právo na čítanie. Užívateľ `mrayder` k tomuto kontextu nemá ani právo na čítanie.

9.2.1 Kontrola prístupu

Tento testovací scenár pokrýva `UC_001, UC_002, UC_003`. Pri prihlasovaní sa použije heslo zhodné s užívateľským menom, pokiaľ nie je uvedené inak.

Testovací scenár 1.1 Prihlás sa ako užívateľ `jsnow`, použi heslo a skontroluj zoznam odhadov, ktoré vidíš.

Očakávaný výstup: Užívateľ uvidí odhad vytvorený v kontexte `CASTLE_BLACK`.

Testovací scenár 1.2 Prihlás sa ako užívateľ `sbaratheon` a skontroluj zoznam odhadov, ktoré vidíš.

Očakávaný výstup: Užívateľ uvidí odhad, vytvorený v kontexte `CASTLE_BLACK`.

Testovací scenár 1.3 Prihlás sa ako užívateľ `mrayder` a skontroluj zoznam odhadov, ktoré vidíš.

Očakávaný výstup: Užívateľ neuvidí žiaden odhad.

Testovací scenár 1.4 Prihlás sa ako užívateľ `mrayder`, vstup do odhadu EST_FULL a skontroluj, či je možné založiť, prípadne editovať verziu. Skontroluj, či je možné akýmkoľvek spôsobom editovať varianty (vytvoriť/zmazať variant), predpoklady (editovať/vytvárať/mazať, pridávať tag, meniť variant, prepájať položky) alebo položky (editovať/vytvárať/mazať, pridávať tag, meniť variant, prepájať položky).

Očakávaný výstup: Založenie novej verzie, prípadne editácia nie je možná. Editovanie akejkoľvek časti nie je možné v čítacom režime.

Testovací scenár 1.5 Prihlás sa ako užívateľ `jsnow`, vstup do odhadu EST_FULL a skontroluj, či je možné založiť, prípadne editovať verziu.

Očakávaný výstup: Založenie novej verzie je možné.

Testovací scenár 1.6 Prihlás sa ako užívateľ `jsnow`. Heslo použi iné ako `snow`.

Očakávaný výstup: Systém detekuje nevalidnú kombináciu mena a hesla a neumožňuje vstup do systému.

9.2.2 Kontrola výpočtov

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL. Testovací scenár pokrýva `UC_029, UC_009, UC_007`.

Testovací scenár 2.1 Zobraz si akúkoľvek kategóriu a ručne vypočítaj súčet položiek min, max, nej. prav., priemer a očakávanú hodnotu. Tento súčet následne porovnaj s hodnotami, ktoré udáva samotná kategória.

Očakávaný výstup: Hodnoty budú rovnaké.

Testovací scenár 2.2 Zobraz si kategóriu `PM`, vyber len variantu 2 a skontroluj, či sa prekeslila stránka. Spočítaj počet položiek a následne ručne vypočítaj súčet položiek min, max, nej. prav, priemer a očakávanú hodnotu. Tento súčet následne porovnaj s hodnotami, ktoré udáva samotná kategória.

Očakávaný výstup: Počet položiek bude 2. Súčet hodnôt bude rovnaký.

Testovací scenár 2.3 V navigačnom paneli odznač defaultný variant a variant 2. Skontroluj, či tabuľka s kategóriami obsahuje vo všetkých bunkách hodnotu nula. Prejdi všetky kategórie a skontroluj, či sú v niektorej kategórii viditeľné položky. Tiež skontroluj, či sa na stránke s predpokladmi nachádza nejaký predpoklad.

Očakávaný výstup: Tabuľka obsahuje vo všetkých bunkách hodnotu nula. Pri prechádzaní jednotlivými kategóriami sa nikde nevyskytuje žiadna položka. Pri zobrazení predpokladov sa na stránke nevyskytoval ani jeden predpoklad.

9.2.3 Kontrola prepočtu medzi MD a MH

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL. Testovací scenár pokrýva `UC_004`.

Testovací scenár 3.1 Skontroluj, či v záhlaví tabuľky je uvedené, v akých jednotkách sú jednotlivé hodnoty.

Očakávaný výstup: Záhlavie tabuľky vyzerá Min (MH), Max (MH), Nej. prav (MH), Priemer (MH), Očekáváno (MH).

Testovací scenár 3.2 Vyber si akúkoľvek kategóriu a skontroluj ako vyzerá záhlavie tabuľky jednotlivých položiek.

Očakávaný výstup: Záhlavie tabuľky vyzerá Min (MH), Max (MH), Nej. prav (MH), Priemer (MH), Očekáváno (MH).

Testovací scenár 3.3 V navigačnom paneli prepni zobrazenie z MH na MD a skontroluj, či sa hodnoty správne prepočítali. Prepočet je správny ak platí, že nová hodnota je osemnásobne menšia ako predošlá. Taktiež skontroluj, či sa zmenilo záhlavie tabuliek, ktoré boli kontrolované v Testovacom scenári 3.1 a 3.2.

Očakavaný výstup: Prepočítanie hodnôt funguje v poriadku, záhlavie sa zmenilo, formát je rovnaký ako v testovacom scenári 3.1, 3.2, ale namiesto MH je tam hodnota MD.

Testovací scenár 3.4 Vytvor novú verziu. Skontroluj, či je možné v navigačnom paneli nastaviť hodnotu MH a či je možné túto hodnotu zmeniť na MD.

Očakavaný výstup: Hodnota je nastavená na MH a hodnotu nie je možné zmeniť na MD.

9.2.4 Kontrola editovanie položiek a predpokladov

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL. Pre tento odhad vytvoril novú verziu. Túto

verziu môže editovať, nakoľko ma zámok. Testovací scenár pokrýva UC_011, UC_013, UC_015, UC_016, UC_017, UC_018, UC_024, UC_027, UC_026, UC_025, UC_003, UC_006, UC_005, UC_030

Testovací scenár 4.1 Vyber si kategóriu `Analýza`, klikni na tabuľku zoznam položiek a vyskúšaj nasledovné klávesové skratky. Pohyb po tabuľke je možný pomocou šípok (dole, hore, vpravo, vľavo). Pri stlačení klávesy `enter` je možné jednotlivé hodnoty editovať. V editovacom režime je možné prejsť na bunku vpravo stlačením klávesy `tab`, bunku vľavo `shift + tab`, stlačením klávesy `enter` je možné prejsť na bunku dole. Klávesa `esc` ukončí editovanie. Skontroluj, či systém reaguje tak, ako bolo popísané v testovacom scenári. Ten istý postup aplikuj na editovanie predpokladov.

Očakávaný výstup: Systém reaguje presne tak, ako bolo opísané v testovacom scenári.

Testovací scenár 4.2 Vyber si kategóriu `Analýza` a začni editovať jednotlivé položky. Zvoľ si akúkoľvek položku a zmeň jej popis, hodnotu min, max a odhadováno. Následne stlač klávesovú skratku `ctrl + u`. Prepni sa na inú kategóriu a následne sa vráť späť na kategóriu `Analýza`. Skontroluj, či hodnoty, ktoré si zmenil, sú skutočne zmenené. Túto verziu ulož ako rozpracovanú (stlač `Uložit jako rozpracovanou`). Odhlás sa a znova sa prihlás. Skontroluj, či je hodnota stále zmenená. Postup znova zopakuj s tým rozdielom, že namiesto použitia klávesovej skratky `ctrl + u` použi tlačítko `Provést změny`.

Očakávaný výstup: Systém reaguje presne tak, ako bolo opísané v testovacom scenári. V prvom kroku sú hodnoty uložené v modeli, v druhom kroku sú všetky hodnoty uložené v databáze.

Testovací scenár 4.3 Tomuto scenáru predchádza testovací scenár 4.2. Vyber si kategóriu `Analýza` a zmaž položku, ktorú si v minulom kroku needitoval. Zmazať položku je možné označením pomocou klávesnice stlačením `space` a následne stlačením tlačítka `Smazat vybrané`. Po odstránení zvolenej položky skontroluj, či sa daná položka z tabuľky odstránila, taktiež skontroluj, či sa správne prepočítali hodnoty kategórie `Analýza`. Následne v tejto kategórii vytvor novú položku. Skontroluj, či sa položka pridala do tabuľky a či sa správne prepočítali hodnoty kategórie `Analýza`. Túto verziu ulož ako rozpracovanú (stlač `Uložit jako rozpracovanou`). Odhlás sa a znova prihlás a skontroluj, či položka, ktorú si zmazal, je skutočne zmazaná. Tiež skontroluj, či položka, ktorú si pridal, je pridaná.

Očakávaný výstup: Mazanie položky prebieha v poriadku. Stránka sa v poriadku prekreslila a kategória sa správne prepočítala. Pridávanie položky tiež funguje v poriadku.

Testovací scenár 4.4 Tomuto scenáru predchádza testovací scenár 4.3. Skontroluj, či odhad obsahuje dve stabilné verzie a jednu rozpracovanú, skús sa medzi nimi prepínať a skontroluj, či sa hodnoty menia v závislosti od zvolenej verzie. V druhom kroku porovnaj rozpracovanú verziu s poslednou stabilnou. V navigačnom paneli vyber `Verze` ako rozpracovanú verziu a `Referenční verze` ako stabilnú. Následne stlač tlačítko `Ukázat rozdíl`. Skontroluj, či tabuľka obsahuje vypočítané hodnoty oboch verzií a či ich správne porovnala. Zvoľ kategóriu `Analýza` a skontroluj, či tabuľka obsahuje informáciu o zmazanej položke (písmo je červené), pridanej položke (písmo je zelené) a editovanej položke (písmo je oranžové). Položky, ktoré neboli naprieč verziami menené sú čiernou farbou.

Očakávaný výstup: Systém reaguje presne tak, ako bolo opísané v testovacom scenári.

Testovací scenár 4.5 Vykonaj obdobné úkony ako v testovacom scenári 4.2, 4.3 s rozdielom, že budeš testovať `Predpoklady` a nie kategóriu `Analýza`.

Očakávany výstup: Systém reaguje presne tak, ako bolo opísané v testovacom scenári.

9.2.5 Správa variant

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL. Pre tento odhad vytvoril novú verziu. Túto verziu môže editovať, nakoľko ma zámok. Testovací scenár pokrýva `UC_019, UC_020, UC_021, UC_022, UC_023`.

Testovací scenár 5.1 Klikni na tlačitko `Správa variant`. Zobrazí sa vyskacovacie okno, v ktorom je zoznam všetkých variantov. V tomto okne je možné pridať nový variant, ale aj zmazať už existujúce varianty. Vytvor nový variant a zmaž variant s popisom 2. Zjaví sa otázka, či sa všetky položky a predpoklady majú presunúť do iného variantu, prípadne, či sa majú zmazať. Zvoľ presun do variantu s popisom default. Skontroluj, či sa v navigačnom paneli objavil nový variant a či sa zmazaný variant odstránil. Pri zobrazení akejkoľvek kategórie klikni na detail akejkoľvek položky a skontroluj, či je možné položke nastaviť novovytvorený variant a či nie je možné vybrať zmazaný variant. Postup poslednej vety zopakuj pre akýkoľvek predpoklad. Následne verziu ulož ako rozpracovanú. Odhlás sa a prihlás a skontroluj či je nový variant prítomný.

Očakávaný výstup: Nový variant je možné vytvoriť. Existujúci variant je možné zmazať. Novovytvorený variant je možné priraďovať položkám a predpokladom. Zmazaný variant už nie je možné nastaviť položke. Po uložení sa vytvorený variant uloží do databázy a zmazaný variant sa odstráni.

9.2.6 Undo command

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL. Pre tento odhad vytvoril novú verziu. Túto verziu môže editovať, nakoľko má zámok. Testovací scenár pokrýva `UC_008`.

Testovací scenár 6.1 Vyber si kategóriu `Analýza`. Vykonaj tieto zmeny v striktnom poradí. Zmaž akúkoľvek položku, vytvor novú položku, zmeň hodnotu akejkoľvek položky, klikni na tlačítko `Provést zmeny`. Stlač tlačítko `Vrátit zmeny`, prípadne stlač kombináciu kláves `ctrl + z`. Po každom stlačení skontroluj, či sa prekreslila tabuľka obsahujúca položky, ako aj výsledná tabuľka s kategóriami. Po prvom vykonaní akcie skontroluj, či sa vrátili zmeny, ktoré si vykonal. Po druhom vykonaní skontroluj, či sa odstránila položka, ktorú si pridal. Po treťom vykonaní skontroluj, či sa pridala položka, ktorú si odstránil. Po štvrtom vykonaní už nie je čo vrátiť. Skontroluj, či sa objavila notifikácia informujúca o tejto skutočnosti. Tento postup zopakuj pri predpokladoch.

Očakávaný výstup: Vrátenie zmien funguje tak, ako bolo opísané v testovacom scenári.

9.2.7 Správa tagov

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL. Pre tento odhad vytvoril novú verziu. Túto verziu môže editovať, nakoľko má zámok. Testovací scenár pokrýva `UC_031`, `UC_032`, `UC_032`, `UC_033` a `UC_034`, `UC_035`.

Testovací scenár 7.1 Vyber si akúkoľvek kategóriu, zobraz si detail akejkoľvek položky. V detaile vytvor tejto položke dva nové tagy. Verziu ulož ako rozpracovanú. Odhlás sa a prihlás. Skontroluj, či daná položka obsahuje tagy, ktoré boli pridané. Tento postup zopakuj pre akýkoľvek predpoklad.

Očakávany výstup: Tagy je možné pridávať jednotlivým položkám aj predpokladom.

Testovací scenár 7.2 Tomuto scenáru predchádza testovací scénár 4.3. Zobraz si detail položky, ku ktorej si priradil tagy. Tieto tagy teraz vymaž. Odhlás sa a prihlás. Skontroluj, či tagy sú z položky zmazané. Tento postup zopakuj pre predpoklad, ktorému si nastavil tagy.

Očakávany výstup: Tagy je možné zmazať z predpokladu aj položky.

9.2.8 Prepojenie predpokladov na položky

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL. Pre tento odhad vytvoril novú verziu. Túto verziu môže editovať, nakoľko má zámok. Testovací scenár pokrýva `UC_028`.

Testovací scenár 8.1 Klikni na zoznam predpokladov. Choď do detailu akéhokoľvek predpokladu. V zozname kategórií si vyber akúkoľvek kategóriu. Zobrazí sa nové okno, v ktorom je možné zaškrtnúť položky, ktoré sú viazané k tomuto predpokladu. Označ niektoré položky. Verziu ulož ako rozpracovanú. Odhlás sa a prihlás. Opäť choď do detailu daného predpokladu a skontroluj, či položky, ktoré si označil, sú vybrané. Tieto položky následne odznač. Verziu ulož ako rozpracovanú. Odhlás sa a prihlás. Choď do detailu daného predpokladu a skontroluj, či sú tieto položky odznačené.

9.2.9 Vytvorenie nového odhadu

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow`. Testovací scenár pokrýva `UC_010`.

Testovací scenár 9.1 Klikni na tlačítko `Vytvorit nový odhad`. Zobrazí sa nové okno, v ktorom sú údaje, ktoré je potrebné vyplniť pre založenie nového odhadu. Ako šablónu zvoľ možnosť `MULTI_LEVEL_TEMPLATE_MOCK`. Projektový kontext zvoľ `CASTLE_BLACK`. Označenie a popis zvoľ ľubovoľné. Stlač tlačítko `Vytvořit`. Skontroluj, či si presmerovaný na stránku s prehľadom položiek, či máš zámok k danému odhadu a či je možné hneď začať vyplňovať predpoklady. Odhlás sa a prihlás. Skontroluj, či v zozname odhadov je odhad, ktorý si vytvoril.

Očakávaný výstup: Vytvorenie odhadu prebehlo v poriadku. Po vytvorení nasledovalo presmerovanie na vypĺňanie jednotlivých predpokladov.

Testovací scenár 9.2 Tomuto testovaciemu scenáru predchádza testovací scenár 9.1. Skontroluj, či je možné kategórie prechádzať v stromovej štruktúre. Choď do kategórie TMPL_CAT_2 -> TMPL_CAT_2.2 a pridaj novú položku. Novú položku pridaj do kategórie TMPL_CAT_2 -> TMPL_CAT_2.1. V tabuľke s kategóriami klikni na kategóriu TMPL_CAT_2 a skontroluj, či sú viditeľné položky zo všetkých podkategórií.

Očakávaný výstup: Zobrazenie položiek pre kategóriu, ktorá obsahuje vnorené kategórie, zobrazí všetky položky všetkých vnorených kategórií.

9.2.10 Správa verzie

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL, ktorý obsahuje len stabilné verzie. Testovací scenár pokrýva `UC_012`, `UC_014` a `UC_036`

Testovací scenár 10.1 Vyber si verziu a vytvor z nej novú verziu. Túto verziu následne zmaž. Odhlás sa a prihlás. Skontroluj, či novovytvorená verzia je skutočne zmazaná.

Očakávaný výstup: Verzia je po zmazaní skutočne zmazaná z databázy.

Testovací scenár 10.2 Vyber si verziu a vytvor z nej novú verziu. Urob zmenu, ktorá obsahuje nevalidnú hodnotu v položke (napríklad nastav minimálnu hodnotu na `5+2`), zmeny ulož do modelu stlačením `Provést změny` a ulož verziu ako stabilnú. V prípade, že sa túto verziu nepodarilo uložiť, zmaž ju.

Očakávaný výstup: Verziu nie je možné uložiť, nakoľko existuje položka, ktorá obsahuje chybu.

Testovací scenár 10.3 Vyber si verziu a vytvor z nej novú verziu. Verziu ulož ako stabilnú. Odhlás sa a prihlás. Skontroluj, či je verzia skutočne uložená ako stabilná.

Očakávaný výstup: Verzia je uložená do databázy a je označená ako stabilná.

9.2.11 Validácia vstupov

Tento testovací scenár predpokladá, že je užívateľ prihlásený ako `jsnow` a nachádza sa v odhade EST_FULL, vo verzii ktorá je rozpracovaná. Testovací scenár pokrýva `UC_012` a `UC_014`

Testovací scenár 11.1 Choď do akejkoľvek kategórie. Pridaj tam novú položku, ktorá má jednu z hodnôt prázdnu. Postup zopakuj pre pridanie nového predpokladu.

Očakávaný výstup: Nie je možné vytvoriť položku ani predpoklad, ktorý má aspoň jeden atribút prázdny.

Testovací scenár 11.2 Choď do akejkoľvek kategórie. Pridaj tam novú položku, pričom popis daj ľubovoľný, min nastavíš na 5+2, max na 10, a nej. prav. na 11. Skontroluj, či sa po pridaní novej položky zobrazuje hodnota min ako `INVALID`. Tiež skontroluj, či sa v danej kategórii zobrazuje hodnota min ako `ERROR`.

Očakávaný výstup: Tabuľka položiek obsahuje pri novovytvornej položke text `INVALID`. Hodnota min v tabuľke s kategóriami pri danej kategórii obsahuje text `ERROR`.

9.3 Dokumentácia

Dokumentácia k aplikácii sa nachádza na priloženom CD. Dokumentácia obsahuje:

- popis jednotlivých obrazoviek a funkčnosti systému
- manuál k používaniu kalkulátora opísaneho v 8.9

- dokumentovaný zdrojový kód — Javadoc

Kapitola 10

Súčasný stav a vyhliadky do budúcna

Aplikácia umožnujúca vytvárať odhady pracnosti a ich nasledovnú správu je hotová. Aplikácia umožňuje vytvoriť nový odhad. Jednotlivým odhadom je možné pridávať nové položky, predpoklady, varianty. Položky a predpoklady je možné medzi sebou ľubovoľne prepájať. Každá položka a predpoklad je viazaný na variant, pričom túto hodnotu je možné zmeniť. Aplikácia tiež podporuje intuitívne editovanie jednotlivých hodnôt pre položky. Pri editovaní položiek je možné použiť jednoduché matematické operácie, ako aj odkazovanie sa na iné položky. Prechod medzi kategóriami je rýchly a jednoduchý. Odhady pozostávajú z verzií, ktoré je možné vytvárať a editovať, prípadne mazať. Tieto verzie je možné medzi sebou porovnávať. Pri porovnávaní verzií je jednoduché detekovať, ktoré položky pribudli, ktoré boli odstránené a ktoré boli nejakým spôsobom zmenené. Na verifikáciu užívateľov sa momentálne používa vstavaný LDAP server. Aplikácia podporuje systém oprávnení (právo na zápis / čítanie). Jednotlivé práva pre užívateľov sú uložené v databáze tejto aplikácie.

Frontend tejto aplikácie momentálne dokáže vytvárať nové odhady len z preddefinových šablón, ktoré sú manuálne vytvárané. Priaznivejšia by bola možnosť, v ktorej by sa šablóna dala vytvoriť cez intuitívne grafické rozhranie. Teraz je šablónu možné vytvárať len ľuďmi, ktorí poznajú dátový model a majú skúsenosti s jazykom Cypher. Pre nasadenie do produckie je potrebné vymeniť vstavaný LDAP server za reálny Profinití LDAP. S toutou zmenou súvisí tiež implementácia správy užívateľov. Je potrebné vytvoriť administračnú konzolu, v ktorej bude môcť administrátor priradzovať jednotlivým užívateľom oprávnenia k jednotlivým kontextom. V budúcnosti je tiež potrebné implementovať schvaľovací proces, v ktorom bude možné odhad exportovať až po schválení. Testovanie frontendu je momentálne riešené manuálnym prechádzaním testovacích scenárov. Tento prístup by bolo vhodné nahradiť automatickým testovaním, napríklad za pomoci Selenium testov.

Záver

Cieľom práce bolo vytvoriť pre firmu Profinit webovú aplikáciu pre podporu tvorby odhadov pracnosti softwarových projektov na základe metód a postupov používaných vo firme Profinit. Tento cieľ bol dosiahnutý.

Na začiatku bola vykonaná analýza riešenia, ktoré sa aktuálne používa. Po vykonanej analýze a zoznámení sa s metodikou vo firme Profinit bolo možné začať uvažovať o výbere vhodných technológií.

Prieskum výberu backend frameworku, v ktorom sa bude vyvíjať aplikácia, spočíval vo výbere medzi frameworkami Spring a Spring Boot. Kvôli jednoduchšej konfigurácii a rýchlejšiemu vývoju bol vybratý framework Spring Boot. Pre frontendovú časť pripadali do úvahy framework PrimeFaces, ZK a Vaadin. Po hlbšej analýze sa ukázalo, že najvhodnejší je framework Vaadin.

Ďalšou časťou bol návrh obrazoviek, ktorý kladol dôraz na jednoduchosť a funkčnosť.

Predposlednou časťou bola realizácia, v ktorej bolo opísané prepojenie frameworku Spring Boot s frontendovým frameworkom Vaadin. Stručne boli rozobraté komponenty, ktoré sa používali v aplikácii najčastejšie. Pri samotnej implementácii bolo potrebné naimplementovať systém, ktorý vyhodnocoval bunky, ktorých hodnota bola závislá na iných hodnotách. Bolo potrebné detekovať cykly a určiť poradie vyhodnocovania buniek.

Posledná časť obsahovala popis unit testov pre systém vyhodnocovania buniek ako aj popis testovacích scenárov, ktoré testovali funkčnosť aplikácie z pohľadu prezentačnej vrstvy.

Ciele tejto práce boli splnené.

Literatúra

[1] McConnell, S.: *Odhadování softwarových projektů*. Brno: Computer Press, vyd. 1. vydání, 2006, ISBN 8025112403.

[2] Macinka, V.: *Techniky stanovení nákladů softwarových projektů*. Diplomová práce, Masarykova Univerzita, 2009.

[3] Buckler, C.: 10 Reasons Why Software Project Estimates Fail. *SitePoint Pty Ltd* [online], 29. 6. 2010, [cit. 2017-05-08]. Dostupné z: `https://www.sitepoint.com/10-reasons-why-software-project-estimates-fail/`

[4] Zapletal, L.: Lehký úvod do LDAP. *Root.cz* [online], 24.7.200, [cit. 2017-05-08]. Dostupné z: `https://www.root.cz/clanky/lehky-uvod-do-ldap/`

[5] Integrated Cloud Application & Platform Services: *Java EE at a Glance* [online]. 2017, [cit. 2017-05-05]. Dostupné z: `http://www.oracle.com/technetwork/java/javaee/overview/index.html`

[6] Gracík, P.: *Návrh a implementácia systému pre správu obchodných prípadov*. Diplomová práce, Masarykova Univerzita, 2013.

[7] Čápka, D.: MVC architektura. *ITnetwork* [online], 2016, [cit. 2017-05-08]. Dostupné z: `https://www.itnetwork.cz/navrhove-vzory/mvc-architektura-navrhovy-vzor/`

[8] The Apache Software Foundation: *POM Reference* [online]. 6.5.2017, [cit. 2017-05-05]. Dostupné z: `https://maven.apache.org/pom.html#What_is_the_POM`

[9] Three-Tier Architecture. *Technopedia* [online], c2017, [cit. 2017-05-08]. Dostupné z: `https://www.techopedia.com/definition/24649/three-tier-architecture`

[10] Pivotal Software, Inc: *The IoC container* [online]. 2016, [cit. 2017-05-05]. Dostupné z: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html

[11] Pivotal Software, Inc: *Spring Initializr* [online]. 2017, [cit. 2017-05-05]. Dostupné z: https://start.spring.io

[12] Bradford, L.: What is a Front-end Framework and Why Use One? *The Balance* [online], 15.2.2017, [cit. 2017-05-08]. Dostupné z: https://www.thebalance.com/what-is-a-front-end-framework-and-why-use-one-2071948

[13] Gerchev, I.: Front-end Frameworks: Custom vs. Ready-to-use Solutions. *SitePoint* [online], 4.11.2014, [cit. 2017-05-08]. Dostupné z: https://www.sitepoint.com/front-end-frameworks-custom-vs-ready-use-solutions/

[14] Zubčák, T.: *Webový framework s modulem pro správu uživatelů*. bachelor thesis, Masarykova Univerzita, 2009.

[15] Vaadin Ltd: *Introduction* [online]. 2017, [cit. 2017-05-05]. Dostupné z: https://vaadin.com/introduction#what

[16] Kubový, T.: *Vaadin - přechod na novou technologii existujícího portfolia produktů*. Diplomová práce, Západočeská univerzita v Plzni, 2014.

[17] Vaadin Ltd: *Vaadin TreeGrid* [online]. 2017, [cit. 2017-05-05]. Dostupné z: https://vaadin.com/directory#!addon/vaadin-treegrid

[18] Primetek: *Why Primefaces* [online]. 2017, [cit. 2017-05-05]. Dostupné z: https://www.primefaces.org/whyprimefaces/

[19] Primetek: *TreeTable* [online]. 2015, [cit. 2017-05-05]. Dostupné z: https://www.primefaces.org/showcase/ui/data/treetable/basic.xhtml

[20] Primetek: *DataTable* [online]. 2015, [cit. 2017-05-05]. Dostupné z: https://www.primefaces.org/showcase/ui/data/datatable/basic.xhtml

[21] Primetek: *PrimeFaces Brings Excel To JSF* [online]. 20.7.2011, [cit. 2017-05-05]. Dostupné z: https://www.primefaces.org/primefaces-brings-excel-to-jsf/

[22] Potix Corporation: *Top Reasons* [online]. 2017, [cit. 2017-05-05]. Dostupné z: https://www.zkoss.org/whyzk/TopReasons

[23] Potix Corporation: *ZK Licensing* [online]. 2017, [cit. 2017-05-05]. Dostupné z: https://www.zkoss.org/license

[24] Vancl, M.: *Návrh datového modelu aplikace pro podporu tvorby odhadů pracnosti softwarových projektů*. bachelor thesis, České vysoké učení technické v Praze, 2017.

[25] Pivotal Software, Inc: *Using the SpringBootApplication annotation* [online]. 2017, [cit. 2017-05-05]. Dostupné z: `https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#using-boot-using-springbootapplication-annotation`

[26] Vaadin Ltd: *Vaadin Spring* [online]. 27.2.2017, [cit. 2017-05-05]. Dostupné z: `http://vaadin.github.io/spring-tutorial/`

[27] Pivotal Software, Inc: *Authenticating a User with LDAP* [online]. 2017, [cit. 2017-05-05]. Dostupné z: `https://spring.io/guides/gs/authenticating-ldap/`

[28] Vaadin Ltd: *Grid* [online]. 2017, [cit. 2017-05-05]. Dostupné z: `https://vaadin.com/docs/-/part/framework/components/components-grid.html`

[29] Vaadin Ltd: *Using Add-ons With Maven* [online]. 2017, [cit. 2017-05-05]. Dostupné z: `https://vaadin.com/directory/help/using-vaadin-add-ons/maven`

[30] Wagner, A.: Vaadin TreeGrid add-on. `https://github.com/vaadin/tree-grid/blob/master/treegrid-demo/src/main/java/org/vaadin/treegrid/demo/DemoContainer.java`, 30.12.2016, [cit. 2017-05-08].

[31] Agarwal, C.: Kahn's algorithm for Topological Sorting. *Geeks for geeks* [online], 2016, [cit. 2017-05-08]. Dostupné z: `http://www.geeksforgeeks.org/topological-sorting-indegree-based-solution/`

[32] Černý, J.: BFS, hledání nejkratší cesty. *Algoritmy eu* [online], 2016, [cit. 2017-05-08]. Dostupné z: `http://algoritmy.eu/zga/pruchod-grafu/bfs-hledani-nejkratsi-cesty/`

Dodatok **A**

Zoznam použitých skratiek

LDAP Lightweight Directory Access Protocol

GUI Grafické rozhranie

PERT Program Evaluation and Review Technique

WBS Work breakdown structure

MD Man-day

MH Man-hour

MVC Model, view, controller

POM Project Object Model

JSF Java Server Faces

JSP Java Server Pages

DI Dependency injection

IoC Inversion of Control

API Application programming interface

ZK CE ZK Community Edition

ZK PE ZK Professional Edition

ZK EE ZK Enteprise Edition

GUI Graphic user interface

UI User interface

JAR Java Archive

HTML HyperText Markup Language

XML Extensible markup language

OGM Object Graph Mapper

SpEL Spring Expression Language

BFS Breadth-first search

Dodatok **B**

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `dokumentace` ..... dokumentácia aplikácie
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF
  - `thesis.ps` ..... text práce vo formáte PS