UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠVO IN INFORMATIKO

Andraž Požar

Aplikacija za pregled časovnih in finančnih metrik podjetja ter generiranje poročil

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE

RAČUNALNIŠTVO IN INFORMATIKA

Mentorica: doc. dr. Polona Oblak

Ljubljana, 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00056/2012
Datum: 11.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ANDRAŽ POŽAR**

Naslov: **APLIKACIJA ZA PREGLED ČASOVNIH IN FINANČNIH METRIK PODJETJA TER GENERIRANJE POROČIL**

DESKTOP APPLICATION FOR GATHERING TIME AND FINANCIAL METRICS AND REPORT GENERATION

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V delu opišite in razložite razvoj aplikacije za pregled časovnih in finančnih metrik podjetja ter generiranje poročil, ki na njih temeljijo. Utemeljite izbiro tehnologij ter predstavite možnosti za bodočo nadgradnjo.

Mentor:

doc. dr. Polona Oblak

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Andraž Požar,

z vpisno številko 63080200,

sem avtor/-ica diplomskega dela z naslovom:

Aplikacija za pregled časovnih in finančnih metrik podjetja ter generiranje poročil

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr. Polone Oblak

  in somentorstvom (naziv, ime in priimek)

  _____

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 21. 9. 2012 Podpis avtorja/-ice: _____

KAZALO VSEBINE

KAZALO SLIK

KAZALO DIAGRAMOV

SEZNAM UPORABLJENIH KRATIC IN SIMBOLOV

OLAP	(OnLine Analytical Processing) Pristop k hitrem reševanju večdimenzionalnih analitičnih poizvedb.
OLTP	(OnLine Transaction Processing) Sistem za prenosno usmerjene realizacije aplikacije. Predvsem gre tu za poizvedovanje in shranjevanje podatkov v podatkovno bazo.
SQL	(Structured Query Language) Poseben programski jezik ustvarjen za urejanje in poizvedbo po podatkih v podatkovnih bazah.
JAVA	Programska platforma uporabljena za realizacijo velikega števila spletnih in namiznih aplikacij.

POVZETEK

Zagotavljanje kvalitete storitev je primarna naloga vsakega uspešnega podjetja. Toda le kvalitetne storitve niso dovolj za dober razvoj, potreben je tudi pregled nad učinkovitostjo zaposlenih in upravljanjem s financami. Podjetja imajo torej navadno zaposleno vsaj eno osebo, ki nadzira vse omenjeno. Način pridobivanja potrebnih podatkov je odvisen od podjetja. V visokotehnološkem podjetju, v katerem opravljam študentsko delo, je bilo pridobivanje podatkov zamudno, saj je zahtevalo uporabo več spletnih orodij ter dodatno manipulacijo s podatki v preglednicah. Razvita aplikacija predstavlja poenotenje vseh orodij za spremljanje časovnih in finančnih metrik kot tudi ustvarjanje poročil. Poročila so ključen vir informacije za vodje velikih projektov, saj jim pokažejo tako učinkovitost razvijalcev kot tudi razmerje med finančnimi prilivi in odlivi, ki so se zgodili na tem projektu. Aplikacija tako razbremenjuje osebo, ki nadzira kakovost storitev, in zmanjšuje možnosti za napake pri ustvarjanju poročil.

KLJUČNE BESEDE

Java, OLAP, PDF, grafični vmesni, namizna aplikacija, poročilo

ABSTRACT

Part of every successful company is quality assurance, yet quality assurance alone is not enough to achieve good company growth. For that also overview of employee efficiency and finance metrics is needed. To achieve that companies usually have at least one person who is responsible for supervising aforementioned indicators. A way of gathering needed data varies between companies. In the case of high-technology company where I was working, the acquiring of data and generation of reports was a time-consuming process since it required the use of many web tools and additional manipulation with acquired data. The developed application represents the unification of all tools needed to supervise financial and time metrics as well as functionality required for report generation. Reports are the key source of information for project managers and present employee efficiency as well as the ratio between the money income and outcome on the project. The application therefore unburdens the quality assurance supervisor and reduces the probability of errors.

KEYWORDS

Java, OLAP, PDF, graphical user interface, desktop application, report

1. UVOD

Naloga vsakega uspešnega podjetja je skrben pregled nad učinkovitostjo in uspešnostjo svojih zaposlenih kot tudi podroben nadzor nad financami podjetja. Da se to doseže, je potrebno zagotoviti sproten pregled nad časom, ki so ga zaposleni opravili na določenem projektu, ter nadzorom nad finančnimi transakcijami celotnega podjetja. Pregled omenjenih metrik je opravilo, ki za to odgovorni osebi pobere veliko časa. Dodaten problem predstavlja narava dela, saj je potrebno podatke najprej pridobiti z ustreznimi poizvedbami iz podatkovne baze ter nato s podatki manipulirati v preglednicah.

Omenjeni problemi so povod k potrebi za razvoj aplikacije, ki poenostavi delo osebe, ki je odgovorna za nadzor nad kvaliteto storitev podjetja ter uspešnim vodenjem financ. Aplikacija mora torej omogočati pregled nad časom, ki je bil porabljen na določenem projektu, ter finančnimi transakcijami celotnega podjetja. Za dosega tega cilja je potrebno razumevanje sistema vodenja časovnih in finančnih evidenc v podjetju ter seznanitev s podatkovnimi bazami, ki te podatke vsebujejo. Končni produkt aplikacije je poročilo, ki zajame vse podatke v izbranem časovnem intervalu in jih predstavi odgovorni osebi kot tudi ostalim zaposlenim v podjetju.

Najprej je bilo potrebno zbrati podatke o času, ki so ga zaposleni porabili na projektih, na katerih so delali. Ti podatki se za časovni interval, ki si ga izbere uporabnik, preberejo iz podatkovne baze in predstavijo uporabniku aplikacije. Ravno tako si lahko uporabnik ogleda finančne transakcije v podjetju, tako da izbere časovni interval, za katerega ga podatki zanimajo in opcijsko tudi projekt, na katerega se ti nanašajo. Finančni podatki so nato uporabniku predstavljeni v obliki preglednice, ki jo lahko skopira v orodja za upravljanje s preglednicami (Microsoft Excell). Ustvariti je potrebno tudi pogled, ki omogoča potrjevanje finančnih podatkov in upravljanje s tipi stroškov pri finančnih transakcijah, saj se nato te potrjene podatke uporabi za generiranje poročil. Končni in najpomembnejši del aplikacije je generiranje poročila, ki se ga uporabi za ovrednotenje uspešnosti podjetja in zaposlenih ter zajema časovne in finančne podatke. Poročilo se generira za izbrani časovni interval in ga je mogoče, če je uporabnik z njim zadovoljen, shraniti v datoteko PDF. Dodan je tudi pogled, v katerem lahko uporabnik še vedno izvaja lastne poizvedbe SQL, ter preglednica za prikaz rezultatov. S tem je bila dosežena polna funkcionalnost aplikacije v primerjavi z delom, ki ga je morala odgovorna oseba prej opraviti samostojno.

2. OZADJE

Za uspešno realizacijo aplikacije za pregled časovnih in finančnih metrik podjetja je potrebno podrobno razumevanje poslovne organizacije podjetja ter obstoječih informacijskih sistemov znotraj podjetja, kot so orodja za beleženje časa, ki ga je razvijalec porabil na projektih, orodje za pregled finančnih transakcij itd. Potrebno pa se je tudi seznaniti s podatkovno bazo, v katero se te informacije shranjujejo, ter z njeno arhitekturo.

2.1. ORGANIZACIJA PODJETJA

Vsako podjetje je organizirano na način, ki je navadno specifičen za dotično podjetje, saj je cilj te organizacije čim boljša prilagodljivost naravi dela, ki ga opravlja podjetje. V primeru visoko tehnološkega podjetja X, kjer sem opravljal študentsko delo, se ta organizacija v grobem deli na dva dela: organizacija poslov in organizacija kadra znotraj podjetja. V naslednjih poglavjih bo nekaj organizacij na kratko predstavljenih.

2.1.1. ORGANIZACIJA POSLOV

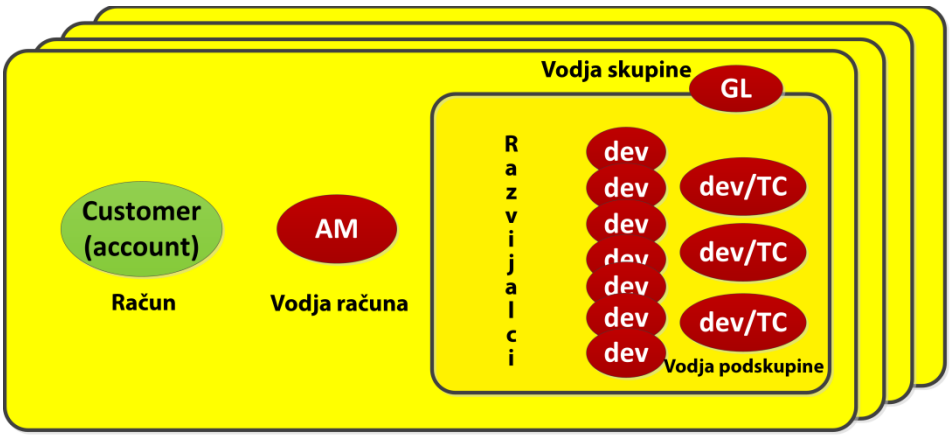
Vsako naročilo, ki pride s strani stranke, dobi svojega upravitelja Upravitelj je za določeno naročilo odgovoren in sprejema odločitve o tem, kako bo potekalo delo. Naročilo je nato običajno razdeljeno na več projektov. Vsak projekt predstavlja nek zaključen del naročila, vsi projekti skupaj pa predstavljajo celoto. Projekt se na dalje deli na pogodbene naloge, ki znova predstavljajo nek zaključen del tega projekta. Vsaka pogodbena naloga ima lahko več podnalog, ki lahko vsebujejo tudi svoje podnaloge. Globina tega drevesa ni v nobeni dimenziji omejena in je za vsak projekt različna, obstajajo pa priporočila, ki naj bi se jih odgovorne osebe držale. Podnaloga zgleda kot zahtevek za opravljanje nekega dela. Ko je podnaloga ustvarjena, se v njej opiše, kaj je pričakovan končen rezultat in se jo izstavi v last razvijalcu. Od tod dalje je ta razvijalec odgovoren za uspešno opravljanje naloge in lahko to podnalogo razdeli na manjše podnaloge, ki jih opravi sam ali jih da v last komu drugemu.

Kot omenjeno, je namen podnaloge, osebi, kateri je bila izročena, opisati problem, katerega mora rešiti in zahtevan rezultat. Obenem pa služi kot sredstvo, na katerem se beleži čas, ki ga je ta oseba vložila v delo za to podnalogo. Vsak razvijalec, ki dela na določeni podnalogi, ima torej dolžnost, da porabljen čas ob kratki obrazložitvi dela piše na to podnalogo. Na takšen način se pridobi dober pregled nad porabljenim časom razvijalcev in se lahko razbere, kje je prišlo do zamud; v primeru, če projekt ni dokončan v predvidenem času.

Glede na naročnika projekta obstaja znotraj podjetja neka logična razdelitev. Vsak naročnik, ki izda naročilo v vrednosti višji od predpisane, dobi znotraj podjetja svoj račun. Na ta račun je nato vezano vse delo in vse finančne transakcije, ki so se v povezavi s tem naročilom zgodile. Svoj račun ima tudi vsak vodja skupine razvijalcev. Na ta račun se vežejo naročila, ki ne presegajo prej omenjene vrednosti in so bila zaupana določenemu vodji skupine. Dodatni računi so še za naročila znotraj podjetja, torej interna naročila ter za dislocirane enote podjetja.

2.1.2. ORGANIZACIJA KADRA

Znotraj podjetja so razvijalci organizirani v skupine. Vsako skupino vodi vodja skupine, ki ima, kot že prej omenjeno, tudi svoj račun, na katerega se veže vse delo te skupine. Ta vodja je navadno tudi upravljalec računa. V vsaki skupini je več podskupin, ki imajo svoje vodje. Vsaka podskupina ima torej svojega vodjo in določeno število razvijalcev, ki jih vodja nadzira in jim, če je to potrebno, pomaga (glej sliko 1).

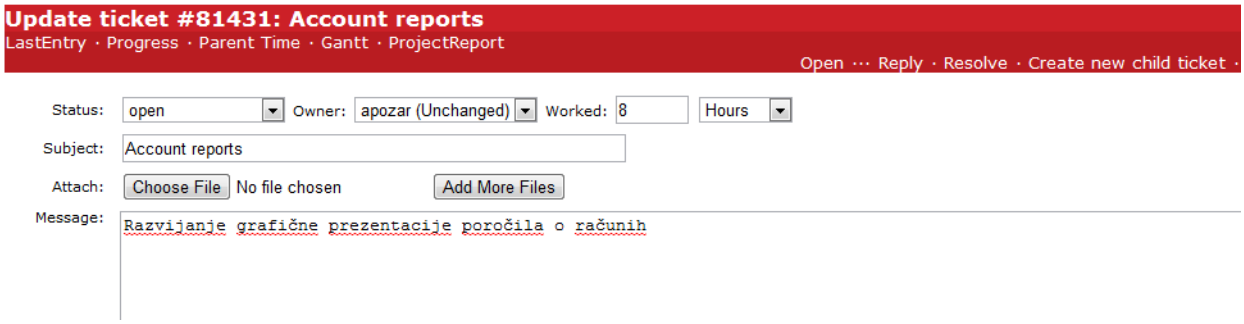


Slika 1: Organizacija kadra

2.2. PROCES DELA

2.2.1. VNOS ČASA, KI GA JE ZA DELO PORABIL RAZVIJALEC

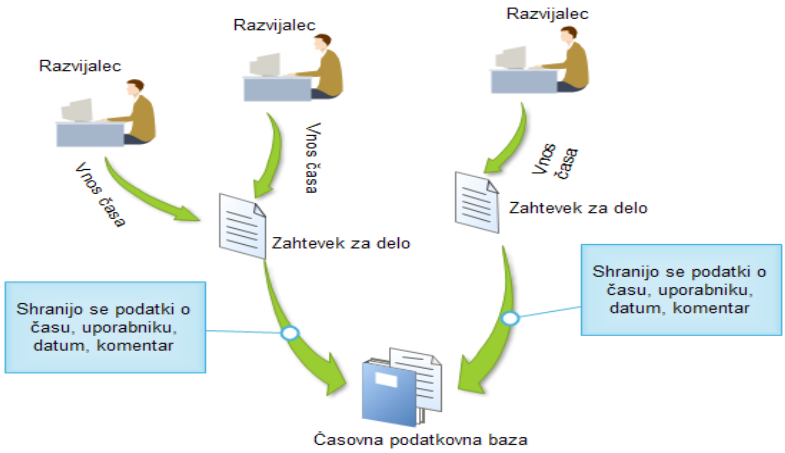
Ko razvijalec dobi zahtevo za opravljanje določene naloge, mora poskrbeti, da čas, ki ga je na nalogi porabil, vpiše. Vpis poteka preko spletne aplikacije, ki je uporabna za izstavljanje zahtev za opravljanje naloge ter beleženje časa. Ob vsakem vnosu časa mora razvijalec tudi na kratko opisati, kaj je v tem času naredil (slika 2).



Slika 2: Aplikacija za vnos časa porabljenega za delo ter opis dela

Tako pripravljeni podatki se nato shranijo v časovno podatkovno bazo. Vsak vnos je vezan na nalogo, ki je v tem primeru »81431: Account reports«, ta naloga pa na projekt. Vsak projekt pa naprej na račun, znotraj katerega je odprt. Tako urejeni podatki so primerni za nadaljnjo obdelavo, saj je mogoče vedno ugotoviti, koliko časa je bilo porabljenega na računu, projektu, nalogi.

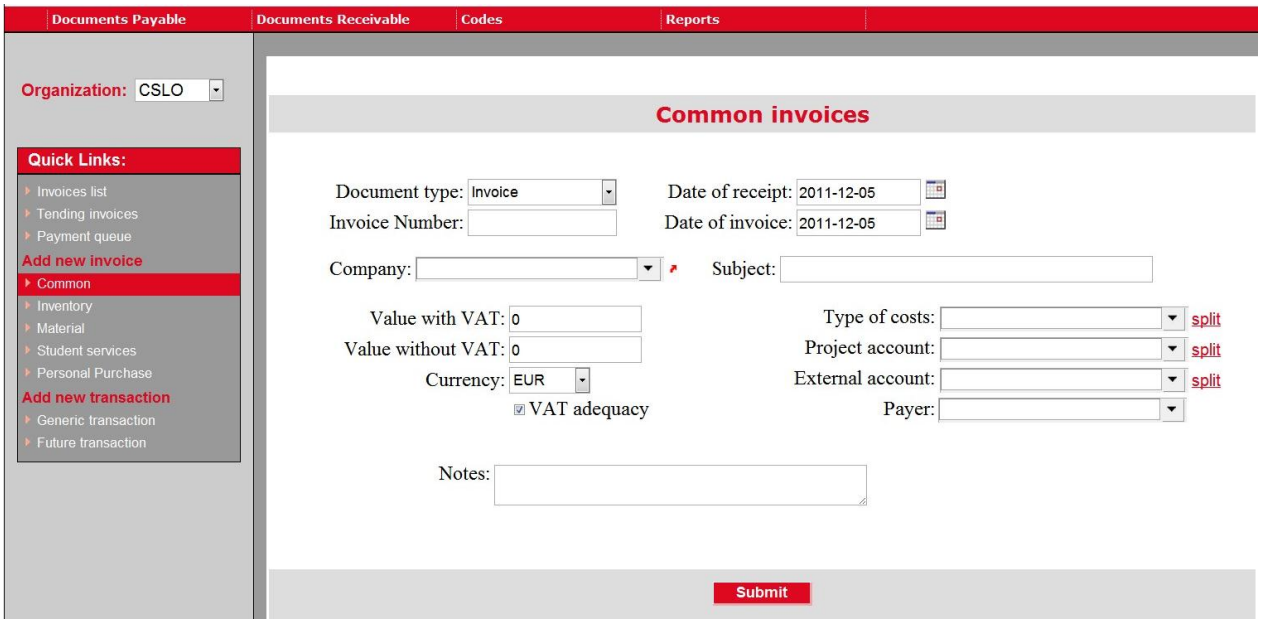
Na sliki 3 je prikazan potek dela in podatkov pri shranjevanju časovnih podatkov v podatkovno bazo.



Slika 3: Potek dela in tok podatkov pri vnosu časa v podatkovno bazo

2.2.2. VNOS FINANČNIH PODATKOV

Vodenje finančnih podatkov poteka s strani računovodstva. Oseba zadolžena za nadzor nad financami za shranjevanje transakcij uporablja spletno aplikacijo, ki je temu namenjena. Tu je mogoč vnos vseh transakcij, na sliki 4 pa je predstavljeno okno za vnos stroškov.



Slika 4: Orodje za vnos stroškov

2.3. ZGRADBA PODATKOVNE BAZE

Poznavanje podatkovne baze, iz katere se črpajo informacije, ki se nato uporabijo v aplikaciji, je osnovna in zelo pomembna naloga. Iz baze je potrebno prebrati določene informacije ter jih pravilno obdelati, da so lahko v določeni obliki predstavljene uporabniku. Glede na vrsto informacije se podatki shranjujejo v dve bazi: finančno in časovno.

2.3.1. PODATKOVNA BAZA S ČASOVNIMI PODATKI

Za prikaz časovnih metrik je potrebno imeti več kot le podatek o tem, koliko časa je nekdo delal na določenem projektu. Za ugotavljanje učinkovitosti posameznika ali skupine razvijalcev na čelu z vodjo skupine, je potrebno za temeljito analizo imeti naslednje podatke:

- račun (račun, na katerega so želeni podatki vezani)
- ime projekta
- tip projekta:
  - notranji
  - zunanji
  - trajni projekti
- razvijalec
- vodja podskupine (kateri pripada razvijalec)
- skupina razvijalcev (kateri pripada razvijalec)

Natančnejša obrazložitev potrebe po teh podatkih ter njihova medsebojna povezanost in uporabnost bo obrazložena v prihodnjih poglavjih.

Opravljanje poizvedbe po teh podatkih zahteva dostop do naslednjih tabel znotraj časovne podatkovne baze (diagram 1):

- CustomFields
- ObjectCustomFieldValues
- Queues
- Tickets
- Users
- Transactions

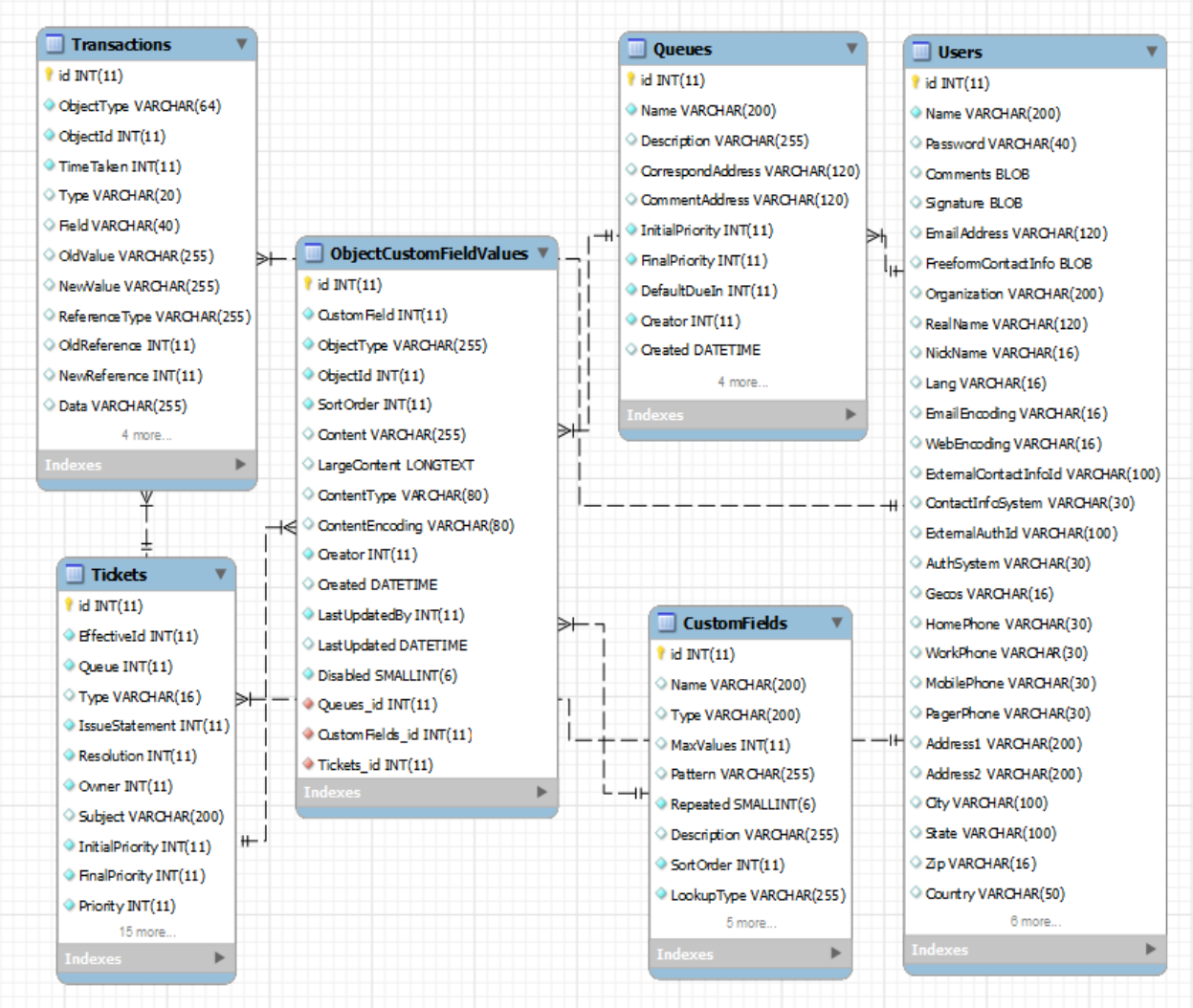


Diagram 1: Diagram zgradbe tabel potrebnih za shranjevanje časovnih podatkov

Razlaga vsebine potrebnih tabel znotraj podatkovne baze:

CustomFields:

Ta tabela služi kot šifrant celotne podatkovne baze. Vsaka informacija, ki se lahko nanaša na uporabnika, projekt, nalogo …, ima svojo identifikacijsko številko, ki se uporabi za dostop do želene informacije v ObjectCustomFieldValues tabeli.

ObjectCustomFieldValues:

Znotraj te tabele so shranjene vse dodatne informacije o uporabnikih, projektih, nalogah … S pridobitvijo identifikacijske številke iz tabele CustomFields je mogoče dostopati do informacij o tipu projekta, računu projekta itd.

Queues:

Tabela hrani vse projekte, ki so bili kadarkoli odprti. Tu so shranjeni le osnovni podatki, kot so ime projekta, opis projekta, datum odprtja.

Tickets:

Tabela hrani vse naloge, ki so bile kadarkoli odprte. Tu so shranjeni le osnovni podatki o nalogah, kot so ime naloge, lastnik naloge, datum odprtja naloge.

Users:

Tabela predstavlja seznam vseh ljudi, ki predstavljajo kader podjetja, kot tudi tistih, ki jih ni več v podjetju (to je potrebno zato, da se lahko preveri čas tudi za preteklost). Podatki shranjeni v tej tabeli so identifikacijska številka osebe, uporabniško ime, polno ime, elektronski naslov itd.

Transactions:

Znotraj te tabele je večina podatkov, ki so vezani na nalogo. Tu se beležijo spremembe imena naloge, datuma naloge, ocenjenega časa do dokončanja ter tudi čas, ki ga je razvijalec na tej nalogi vpisal in njegov komentar. Ta čas predstavlja končni rezultat pregleda časovnih podatkov.

2.3.2. PODATKOVNA BAZA S FINANČNIMI PODATKI

Za pridobivanje finančnih podatkov se stvari nekoliko bolj zapletejo. Za pridobitev vseh potrebnih podatkov za eno finančno transakcijo je namreč potrebno združiti podatke iz šestih tabel znotraj finančne podatkovne baze (diagram 2):

- Transaction
- TransactionSplit
- AccountInfo
- CompanyAccount
- DocumentPayableAdditionalData
- DocumentReceivableAdditionalData

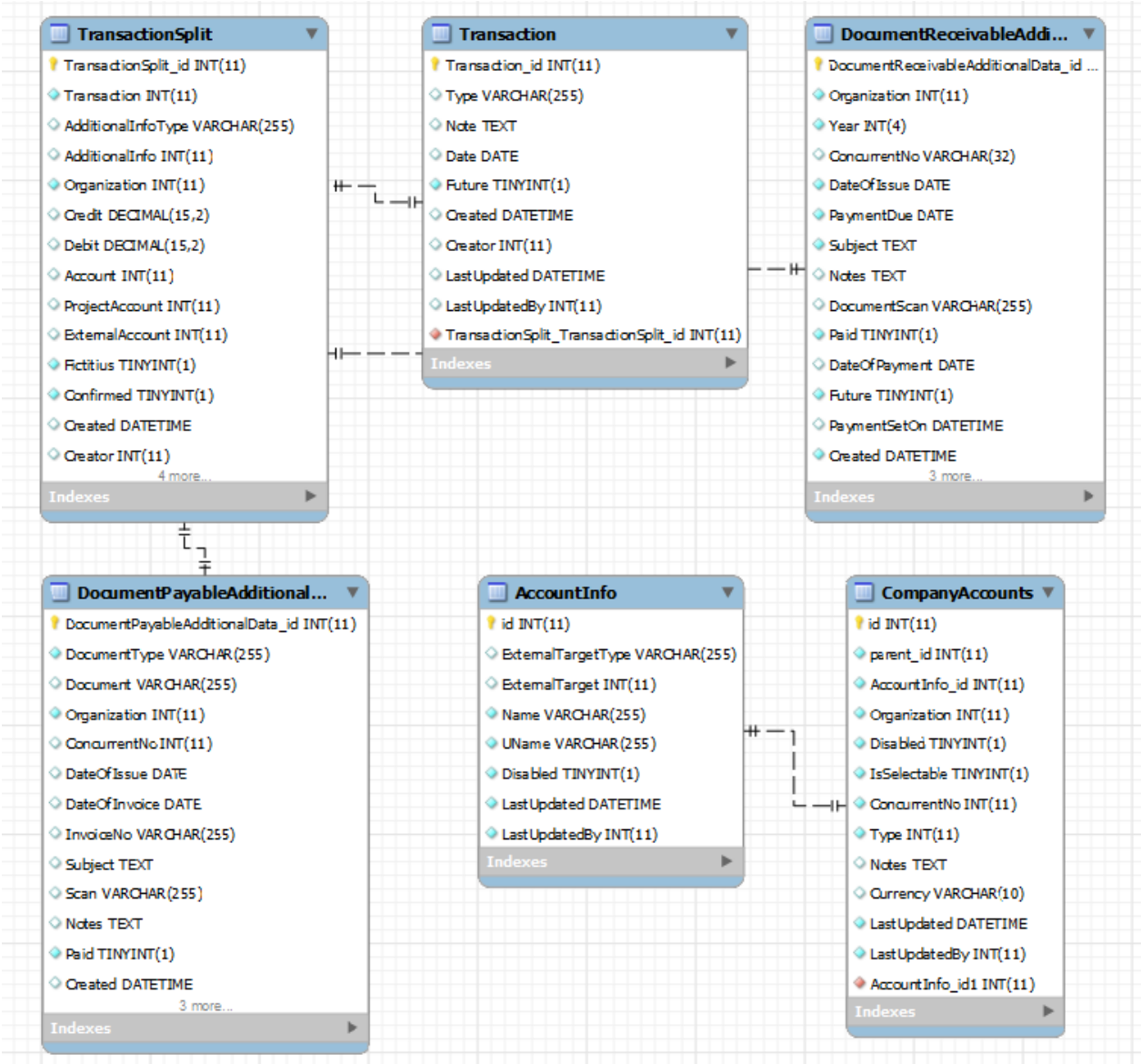


Diagram 2: Diagram zgradbe tabel potrebnih za shranjevanje finančnih podatkov

Razlaga vsebine potrebnih tabel znotraj podatkovne baze:

Transaction:

V tej tabeli so le osnovni podatki o transakciji, kot so tip transakcije, datum transakcije in identifikacijska številka.

TransactionSplit:

Tabela je povezna s tabelo »Transaction« preko identifikacijske številke transakcije. Tu je zapisana količina denarja, ki se je v transakciji prenesla. Ločimo priliv ali »Credit« in odliv ali »Debit«.

AccountInfo:

Vsak projekt ima svojo identifikacijsko številko, ki služi za ugotavljanje, na katerem projektu se je določena transakcija izvedla.

CompanyAccount:

Vsebuje informacijo o valuti ter skupaj z vnosi iz »AccountInfo« tabele tvori vse potrebne informacije o finančnem računu določenega projekta.

DocumentPayableAdditionalData:

Vsebuje dodatne informacije o odlivu financ na projektih. Iz te tabele je mogoče ugotoviti, kakšen je tip finančnega dokumenta, ki je bil izdan, opis transakcije, ali je transakcija že bila izvršena itd.

DocumentReceivableAdditionalData:

Vsebuje enake informacije kot »DocumentPayableAdditionalData«, le da gre v tem primeru za prilive in ne odlive.

3. RAZVOJ APLIKACIJE

Razvoj aplikacije je potekal v več delih. Najprej je bilo potrebno zbrati vse zahteve in želje naročnika, ki je v tem primeru podjetje samo. Ko so bile zahteve zbrane, se je delo nadaljevalo z izgradnjo ideje in predstavitvijo ideje naročniku. Ko je bil naročnik z idejo zadovoljen, se je začela izgradnja arhitekture in njeno preverjanje ter na koncu dejansko pisanje aplikacije.

3.1. RAZVOJNE ZAHTEVE IN PREDSTAVITEV OBSTOJEČIH APLIKACIJ

Glavni razlog za razvoj aplikacije je avtomatizacija procesa generiranja poročil, ki se navezujejo na račun. Kot že omenjeno, je za generiranje poročila potrebno zbrati časovne in finančne podatke, zato se je razvila tudi ideja, da bi del aplikacije bil namenjen zgolj pregledovanju časovnih in finančnih podatkov, saj se velikokrat zgodi, da vodja skupine razvijalcev zahteva časovne metrike svoje skupine, da lahko tako kadarkoli ovrednoti njeno uspešnost in se na podlagi podatkov odloči, ali so potrebne določene spremembe.

3.1.1. ZAHTEVE PREGLEDA ČASOVNIH

Kot že omenjeno, je potrebno za dober pregled časovnih podatkov imeti informacije o računu, imenu projekta, tipu projekta, skupini razvijalcev, vodji podskupine, razvijalcu. Vsi ti podatki so medsebojno povezani in se jih lahko predstavi kot šest različnih dimenzij, ki pripomorejo k optimalni oceni kvalitete dela. Glede na informacijo, ki jo uporabnik želi pridobiti iz podatkov, je torej podatke potrebo pravilno razvrstiti in med seboj povezati. Uporabnik lahko na primer zahteva, da se mu izpiše ves čas, ki ga je neka skupina razvijalcev porabila na določenem računu, ali pa le skupni čas, ki ga je opravil en razvijalec v določenem časovnem obdobju. Uporabnik mora imeti tako na izbiro vse dimenzije, med katerimi si lahko izbere tiste, ki ga zanimajo. Pri njih pa je pomemben tudi vrstni red, saj delo enega razvijalca na projektu ni enako delu, ki so ga na projektu opravili vsi razvijalci. Potrebna je torej implementacija dinamičnega računanja, povezovanja ter izpisovanja rezultatov glede na želje uporabnika.

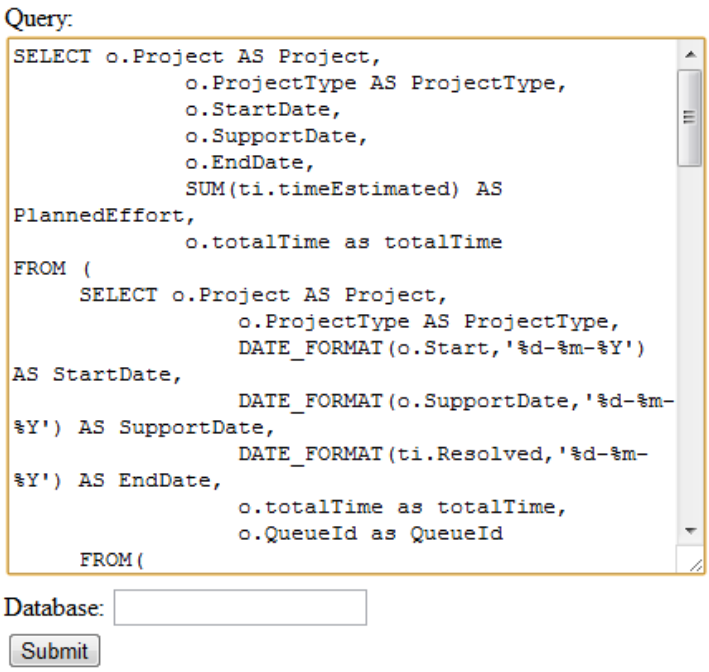
Oseba, ki zahteva določene časovne metrike navadno postavi tudi omejitve. Te omejitve se nanašajo na najvišji nivo, torej na najvišjo dimenzijo, ki jo oseba zahteva, npr.: »Pokaži vse delo, ki so ga opravili razvijalci na točno določenem projektu«. Potreben je nek filter, ki ga uporabnik lahko spreminja in s katerim se filtrira po najvišji izbrani dimenziji, v danem primeru bi bil to filter za imena projektov.

Navadno osebe, ki zahteva časovne metrike, ne zanimajo podatki za celotno zgodovino dela, ampak le za nek določen časovni interval. Uporabnik mora tako imeti možnost, da določi časovni interval, na katerem ga metrike zanimajo.

Glede na to, da so se pred razvojem te aplikacije zahteve po časovnih podatkih izvrševale preko poizvedb SQL, se shranjevale v preglednico in nato predstavile osebi, ki je te podatke zahtevala, je potrebna tudi možnost, da se podatke izvozi v obliko, ki je primerna nadaljnji obdelavi v preglednici. Potreben je torej nek način za izvoz podatkov v datoteko CSV.

V splošnem je potek dela, da je uporabnik prišel do želene informacije, zgledal tako:

- odprtje spletne strani, ki omogoča pisanje in izvajanje poizvedb SQL;
- pripravljanje zahteve SQL s pravilno nastavljenim datumom, da ustreza zahtevam;
- izvrševanje zahteve SQL (slika 5);



Slika 5: Vnosni obrazec za poizvedbo SQL po časovnih podatkih

- kopiranje rezultata v program za obdelavo preglednic;
- generiranje vrtilne tabele;
- izbira želenih dimenzij;
- izročitev generirane tabele osebi, ki je podatke zahtevala.

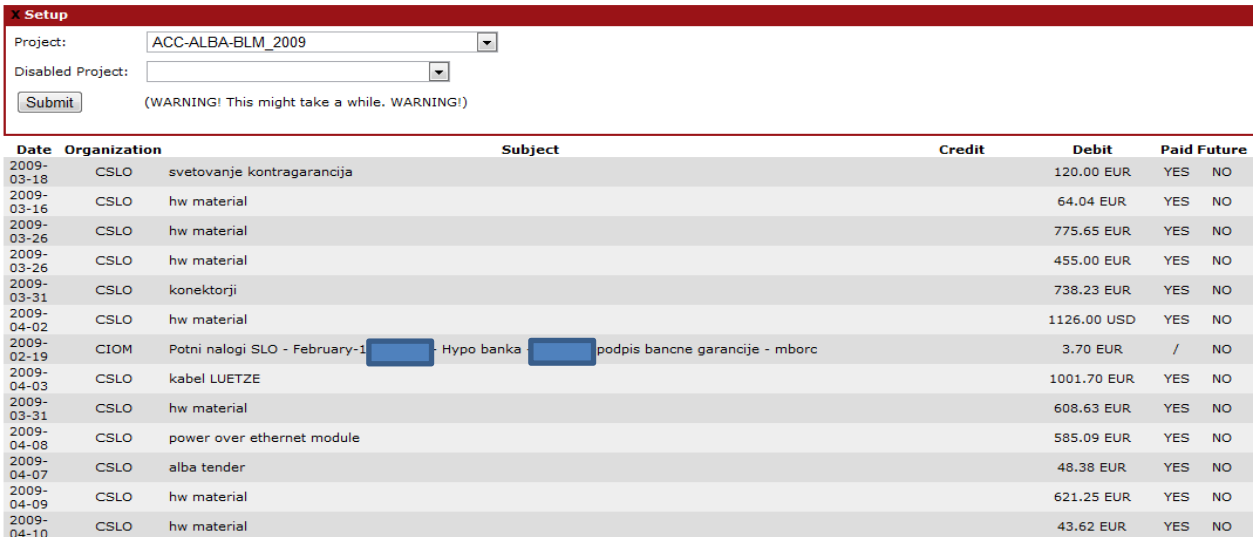
Aplikacija mora torej omogočati polno funkcionalnost zgoraj navedenega postopka, vendar v poenostavljeni obliki, tako da se razbremeni uporabnika in se zmanjša čas od pojava zahteve po podatkih do izročitve preglednice osebi, ki je podatke zahtevala.

3.1.2. ZAHTEVE PREGLEDA FINANČNIH

Pri prikazu finančnih metrik so stvari enostavnejše. Prikaz finančnih transakcij je namreč le vmesni produkt, do katerega se lahko pride z zbiranjem informacij za generiranje poročila. Predstavlja pa tudi združitev različnih orodij v eno samo orodje in s tem olajšuje delo.

Za prikaz mora imeti uporabnik možnost med izbiranjem računov in projektov na teh računih. Potrebno je torej filtriranje projektov glede na izbrani račun. Uporabnik lahko zahteva finančne podatke vseh projektov, projektov na določenem računu ali le podatke enega projekta. Ločiti je potrebno tudi projekte, ki so še odprti in projekte, ki so se že zaprli, zaradi boljšega pregleda. Tudi tukaj lahko uporabnik določi časovni interval, na katerem ga zanimajo finančni podatki projekta, če pa želi dobiti podatke o vseh transakcijah na projektu, mora imeti možnost, da datuma ne vnese, saj obstaja velika verjetnost, da ne ve, kdaj se je projekt začel – to bi vodilo do morebitnih napak. V tem primeru se datum ignorira in iz podatkovne baze se preberejo vsi podatki za določen projekt ali za projekte na določenem računu.

Pred razvojem aplikacije je bilo pridobivanje finančnih podatkov zamudno. Uporabnik je lahko pregledoval finančne podatke le za en projekt hkrati. Če je želel dobiti informacije o financah na več projektih, je moral iz spustnega seznama izbrati nov projekt in podatke nato kopirati v preglednico (slika 6).



Date	Organization	Subject	Credit	Debit	Paid	Future
2009-03-18	CSLO	svetovanje kontragarancija		120.00 EUR	YES	NO
2009-03-16	CSLO	hw material		64.04 EUR	YES	NO
2009-03-26	CSLO	hw material		775.65 EUR	YES	NO
2009-03-26	CSLO	hw material		455.00 EUR	YES	NO
2009-03-31	CSLO	konektorji		738.23 EUR	YES	NO
2009-04-02	CSLO	hw material		1126.00 USD	YES	NO
2009-02-19	CIOM	Potni nalogi SLO - February-1 [redacted] - Hypo banka - [redacted] podpis bancne garancije - mborc		3.70 EUR	/	NO
2009-04-03	CSLO	kabel LUETZE		1001.70 EUR	YES	NO
2009-03-31	CSLO	hw material		608.63 EUR	YES	NO
2009-04-08	CSLO	power over ethernet module		585.09 EUR	YES	NO
2009-04-07	CSLO	alba tender		48.38 EUR	YES	NO
2009-04-09	CSLO	hw material		621.25 EUR	YES	NO
2009-04-10	CSLO	hw material		43.62 EUR	YES	NO

Slika 6: Poizvedba po finančnih podatkih na projektih

Tudi vrnitev rezultatov po tem, ko jih je uporabnik zahteval, je bila počasna, saj je bil algoritem zgrajen tako, da se je naprej pridobilo identifikacijske številke vseh transakcij, nato pa se je za vsako dodatno informacijo izvedel klic v podatkovno bazo. Prikazovanje podatkov je lahko tako trajalo več minut in je bilo za uporabnika zelo moteče.

3.1.3. GENERIRANJE POROČILA

Glede na to, da je cilj aplikacije poenostavitev generiranja poročila, pomeni, da to poročilo že obstaja. Zahteve za izdelavo poročila so torej takšne, da bo poročilo generirano s pomočjo aplikacije ustrezalo obstoječemu poročilu. Tu poudarek ni na obliki, temveč na podatkih.

Poročilo vsebuje časovne in finančne podatke, ki so prebrani iz podatkovnih baz ter določene izračune iz teh podatkov, kar skupaj ustvari pregled nad računom, tako da je za uporabnika in osebe, na katere se nanaša, kar najbolj informativen. Problem nastane pri finančnih podatkih, saj ti ne vsebujejo informacije o tipu stroška (material, potovanja itd.), tako da mora uporabnik te tipe vnesti ročno, ob enem pa tudi potrditi, katere transakcije so veljavne in katere ne. Za dosega tega je torej potrebno dodati nek pogled, v katerem lahko uporabnik upravlja s finančnimi podatki. Poleg tega pa je za izračun vrednosti v poročilu potrebno tudi ročno vnesti določene spremenljivke.

Te spremenljivke so:

- ocena dela do zaključka naročila;
- ocena stroškov do zaključka naročila;
- uradna urna postavka, ki je prodana naročniku;
- ocena vodje računa, koliko prihodka bi lahko v prihodnosti še prišlo s strani naročnika;
- cena tedenskega dela enega razvijalca.

Prve tri informacije uporabniku posreduje vodja računa, saj je on najbolj seznanjen z razmerami na računu, ostali dve pa pridobi uporabnik sam, glede na povprečne plače ljudi, ki na računu delajo.

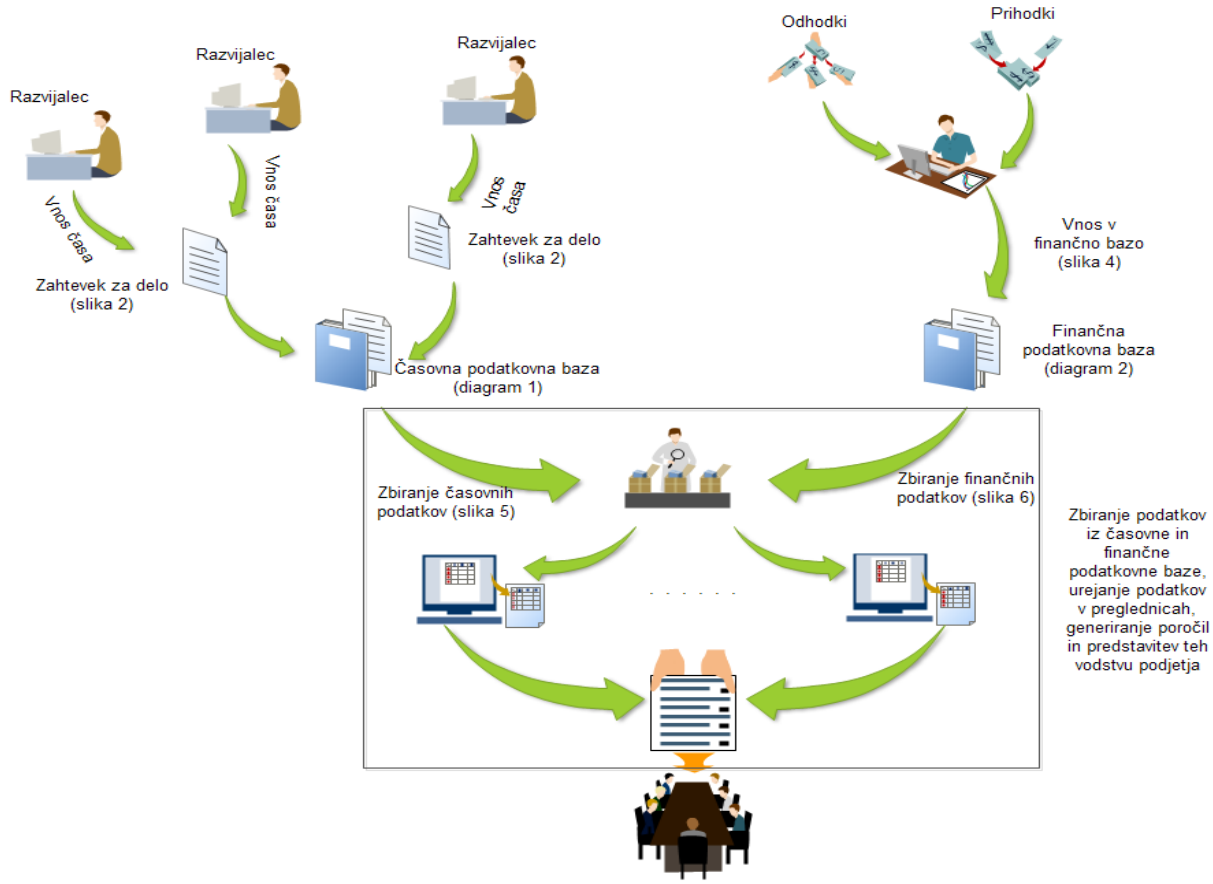
Uporabnik mora tako imeti možnost vnosa tipov stroškov in vrednosti spremenljivk ter možnost potrjevanja transakcij. Vse, kar je uporabnik nastavil, se mora nato shraniti v podatkovno bazo, da se delo ne podvaja. Po shranjevanju v bazo pa mora dobiti nek pregled transakcij, ki so že shranjene.

Ko so podatki enkrat shranjeni, mora imeti uporabnik na razpolago izbiro datuma ter računa, za katerega bi rad generiral poročilo. Po izbiri se poročilo generira in obstajati mora način za izvoz poročila v datoteko PDF, saj se tako poročilo nato predstavi vodjem računov.

Pred pojavom aplikacije je bilo opisano delo zelo zamudno. Uporabnik je moral najprej pridobiti vse potrebne podatke z uporabo orodij opisanih v poglavjih 3.1.1. in 3.1.2. ter zatem nastaviti tipe stroškov. Po končanem vnašanju tipov stroškov je moral napisati razne funkcije za pravilen prikaz metrik ter v njih nastaviti vse potrebne spremenljivke. Od tu se je razvila ideja po poenotenem programu, v katerem bi imel uporabnik minimalno količino delo in kjer bi bili vsi podatki na enem mestu in ne v več različnih spletnih orodjih.

3.1.4. KONČNO STANJE

Zahtevan končni rezultat je razbremenitev osebe zadolžene za zagotavljanje kakovosti. Na sliki 7 je prikazan celoten tok podatkov od uporabnika (v primeru časovnih podatkov) ali priliva in odliva denarja (v primeru finančnih podatkov) do predstavitve poročila vodstvu podjetja in vodjem računa.



Slika 7: Celoten tok podatkov od vnosa v podatkovno bazo do izročanja poročila

Cilj aplikacije je poenostavitev uokvirjenega dela, tako da lahko oseba zadolžena za zagotavljanje kakovosti storitev z nekaj kliki pride do istega rezultata kot prej, vendar v veliko krajšem času. Mora pa imeti tudi možnost pregleda podatkov brez generiranja poročila.

3.2. ARHITEKTURA APLIKACIJE

Pravilno zastavljena arhitektura je temeljni kamen vsake aplikacije. Pogoj za pravilno postavitev pa je natančno poznavanje zahtev naročnika.

3.2.1. RAZVOJ IDEJE

Glavno vodilo pri razvoju programske arhitekture je, da se vnaprej predvidi, kaj bo potrebno storiti, kateri podatki so potrebni, ali so isti podatki potrebni na več mestih itd. V dotičnem primeru je glavni problem organizacija časovnih podatkov. Podatke je potrebno organizirati tako, da so med seboj povezani, saj so medsebojno odvisni, ter da se lahko uporabnik odloči za kakršenkoli pogled podatkov. Uporabnik mora torej imeti proste roke za izbiro dimenzij, ki jih želi prikazati, in njihovega vrstnega reda. To povzroči kar nekaj omejitev pri načrtovanju arhitekture.

Ko so podatki prebrani iz podatkovnih baz in shranjeni v podatkovne strukture, morajo biti urejeni tako, da so uporabni tudi za generiranje poročila. V poročilu so namreč tako časovni kot finančni podatki, torej bi bilo idealno, če bi bila zgrajena neka podatkovna struktura, ki bi jo lahko uporabljali za pregled časovnih metrik kot tudi za generiranje poročila. Enako velja za finančne metrike.

3.2.2. ARHITEKTURA IN KONČNO STANJE

Med razvojem arhitekture se je pojavilo nekaj problemov, ki so zahtevali podrobno raziskovanje obstoječih rešitev in iskanje možnosti za njihovo realizacijo. Glavni problemi, ki so se pojavili med razvojem, bodo predstavljeni v tem razdelku, v naslednjem pa bo predstavljena in utemeljena izbira rešitev.

- **Kakšna naj bo aplikacija? Spletna, namizna?**

  Prvo vprašanje, na katerega je potrebno odgovoriti je, kakšen naj bo tip aplikacije. Za odločitev o tem, ali bo aplikacija namizna ali spletna, je potrebno poznati potrebe naročnika. Tu je predvsem potrebno biti pozoren na število uporabnikov in varnost, saj sta ta dva parametra glavna pri odločanju.

Lastnosti spletne aplikacije:

- Dostopna izven podjetja. Uporabnik lahko aplikacijo uporablja kjerkoli, če je strežnik pravilno nastavljen.
- Težje zagotavljanje varnosti. Če je aplikacija vidna v svet, je veliko bolj dovzetna za napade. Problem je v tem, da aplikacija vsebuje občutljive informacije o financah in zaposlenih.
- Večja možnost za nedelovanje ob problemih na strežniku.

Lastnosti namizne aplikacije:

- Dostopna je iz računalnika, kjer je naložena.
- Lažje zagotavljanje varnosti. Za napad je potreben fizičen dostop do računalnika uporabnika. Dovolj je osnovna avtentikacija z uporabniškim imenom in geslom.

Kot že večkrat omenjeno, je aplikacija v večini namenjena enemu samemu uporabniku. Zato ni smiselno, da je izpostavljena na spletu. To, da jo ima uporabnik naloženo na svojem računalniku, zagotavlja večjo varnost, manjše stroške razvoja in vzdrževanja, saj je pomoč administratorja skorajda nepotrebna, ter večjo zanesljivost.

- **Kako urediti časovne podatke, da ima uporabnik na voljo čim večjo svobodo, ter obenem optimizirati čas prikaza rezultatov?**

  Urejanje podatkov ter časovna optimizacija je že dolgo znan problem v računalništvu, literature o tem je zato veliko. Gledano na zahteve, torej, da ima uporabnik možnost izbirati dimenzije in njihov vrstni red, ter podobnosti z vrtilno tabelo, se je kot ena izmed možnosti ponudila implementacija OLAP analize [3], lahko pa se uporabi poizvedovanje za vsako zahtevo uporabnika.

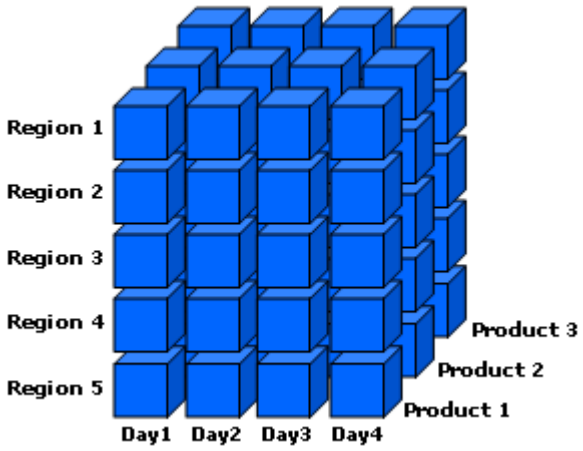
  OLAP

  Uporaba OLAP (OnLine Analytical Processing) je eden izmed možnih pristopov k reševanju problema z ogromno količino podatkov, nad katerimi se zaradi potreb uporabnika izvajajo zelo zahtevne poizvedbe. Glavna lastnost OLAP je njegova hitrost [3]. Če mora ob OLTP bazi uporabnik za različne podatke zahtevati pomoč razvijalca in se tudi prenos in računanje podatkov po navadi zavlečeta, lahko sedaj

uporabnik zahteva in dobi podatke v realnem času, torej instantno. To je omogočeno z pravilno strukturo podatkovne baze ter večdimenzionalnimi poizvedbami.

V primeru, da želi razvijalec poseči po OLAP analizi, mora imeti pravilno strukturirano podatkovno bazo. Če tega nima, torej če ima OLTP podatkovno bazo, je potrebna uvedba nove podatkovne baze, ki bo primerna za uporabno OLAP.

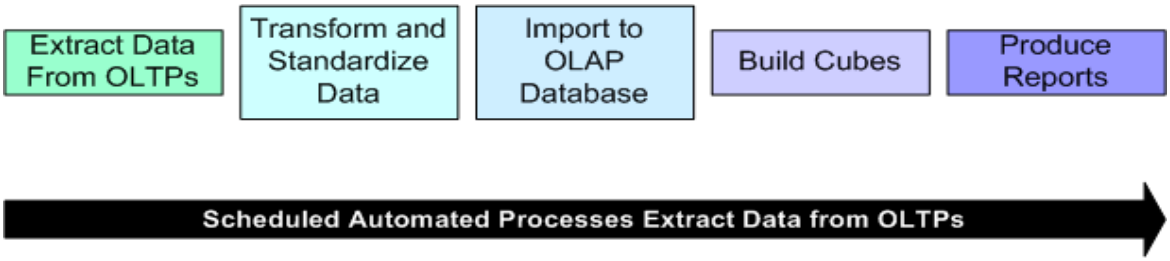
Nad pravilno urejeno podatkovno bazo lahko uporabnik na primer izvrši zahtevo po informaciji o tem, katerega produkta se je največ prodalo v nekem mestu, v določenem letu. Da lahko do tega podatka dostopa v realnem času, je potrebno iz podatkov zgraditi OLAP kocko (slika 8).



Slika 8: Primer 3-dimenzionalne OLAP kocke

OLAP kocka ima lahko poljubno dimenzijo. Kocka se lahko gradi glede na zahteve uporabnikov ali pa je vedno zgrajena iz vseh mogočih dimenzij. OLAP kocka ja v bistvu neka podatkovna struktura, ki ima vse podatke med seboj logično povezane. Do nekega podatka se torej dostopa tako, da se navede dimenzije in njihove vrednosti. Nato se v kocki poišče presečišče vseh vrednosti in vrnjen podatek ustreza zahtevam uporabnika. V kocki so vse vrednosti že izračunane, zato dodatnih računov na vrnjenih rezultatih ni potrebno opravljati, kot bi to bilo pri navadni podatkovni bazi.

Kot omenjeno, je OLAP podatkovna baza običajno narejena »na vrhu« OLTP podatkovne baze. Podatki morajo torej biti sinhronizirani, ob enem pa se mora nenehno graditi tudi OLAP kocka. Običajni proces sinhronizacije prikazuje slika 9.



Slika 9: Pot podatkov od OLTP baze do uporabnika

Izvajanje poizvedbe SQL za vsako zahtevo uporabnika

Drugi možen način za realizacijo uporabniške svobode pri uporabi programa je, da se za vsako zahtevo uporabnika ustvari nova poizvedba SQL. Število možnih kombinacij je končno, zato je ta način načeloma izvedljiv. Enostavnost pa prinaša veliko problemov. Zaradi zgradbe podatkovne baze ni mogoče narediti algoritma, ki bi lahko na podlagi podanih dimenzij, ki jih je izbral uporabnik, generiral poizvedbe SQL. Ostane torej le še možnost, da so vse možne kombinacije poizvedb spisane in se nato ob uporabnikovi zahtevi izvede ena izmed njih. Če je vseh možnih dimenzij šest, pomeni, da lahko uporabnik izbere število dimenzij za prikaz na intervalu od 1 do 6, dimenzije pa so lahko v kakršnemkoli vrstnem redu. To je občutno preveč kombinacij. Tudi v primeru, da se ustvari nek algoritem, ki dinamično ustvarja poizvedbe SQL, ostaja še vedno problem velikega odzivnega časa. Uporabnik namreč mora počakati, da se za vsako spremembo, ki jo je opravil, prenesejo novi podatki iz podatkovne baze ter se mu prestavijo. Kot že rečeno, to predstavlja slabo prakso, saj s tem učinkovitost uporabnika pada, ker lahko izgubi »nit misli« med čakanjem na izpis rezultatov.

Ti dve rešitvi sta na prvi pogled najbolj očitni in večina razvijalcev bi se odločila za implementacijo OLAP analize, saj je ima hitrejši odzivni čas in je vsesplošno zelo močno orodje. V aplikaciji je bila uporabljena rešitev, ki je kombinacija poizvedb nad podatkovno bazo in simulacijo OLAP kocke znotraj spomina aplikacije.

Izvajanje poizvedbe SQL le ob spremembi časovnega intervala, ob zahtevi podatkov pri enakem časovnem intervalu pa simulacija OLAP kocke znotraj spomina aplikacije.

Za prikazovanje želenih informacij na ta način ni potrebno poseganje v obstoječo arhitekturo podatkovne baze, ravno tako ni potrebno uvajati nove podatkovne baze za podporo OLAP in skrbeti za sinhronizacijo z OLTP bazo. Seveda pa to prinaša nekoliko počasnejše delovanje. Delovanje je počasnejše le v primeru, ko uporabnik želi zamenjati časovni interval, na katerem ga zanimajo podatki, saj se v tem primeru naredi dostop do podatkovne baze in se preberejo vsi časovni podatki, ki se nanašajo na izbrani časovni interval. Branje podatkov iz OLTP podatkovne baze lahko torej enačimo s prepisom podatkov iz OLTP baze v OLAP bazo. Prebrani podatki morajo biti nato združeni v neko podatkovno strukturo, do katere se lahko dostopa dovolj hitro in omogoča prikaz vseh možnih kombinacij dimenzij, ki jih lahko uporabnik izbere.

- **Kje naj bodo shranjeni začasni podatki potrebni za pregled časovnih metrik?**

  Zaradi narave analize časovnih podatkov se velikokrat izvajajo zahteve po prikazu nad isto množico podatkov. Nesmiselno bi bilo torej, da se vsakič znova isti podatki preberejo iz podatkovne baze časovnih podatkov, saj je branje zamudno, ker je potrebno pregledati vse zapise v podatkovni bazi in izločiti tiste, ki ne ustrezajo izbranim pogojem. Pojavita se torej dve možnosti shranjevanja začasnih podatkov:

  - shranjevanje v novo podatkovno bazo;
  - shranjevanje v podatkovne strukture znotraj spomina dodeljenega aplikaciji.

  Pri shranjevanju podatkov v novo podatkovno bazo imamo problem latence. Uporabnik bo torej še vedno moral ob vsaki zahtevi čakati, da se podatki preberejo iz podatkovne baze, razlika je le v tem, da se v tem primeru izvede le osnovna poizvedba po vrnitvi vseh rezultatov v določeni tabeli. Ker je rezultatov manj kot v izvorni tabeli, saj so ti zapisi podmnožica izvorne tabele, je poizvedba hitrejša (diagram 3).

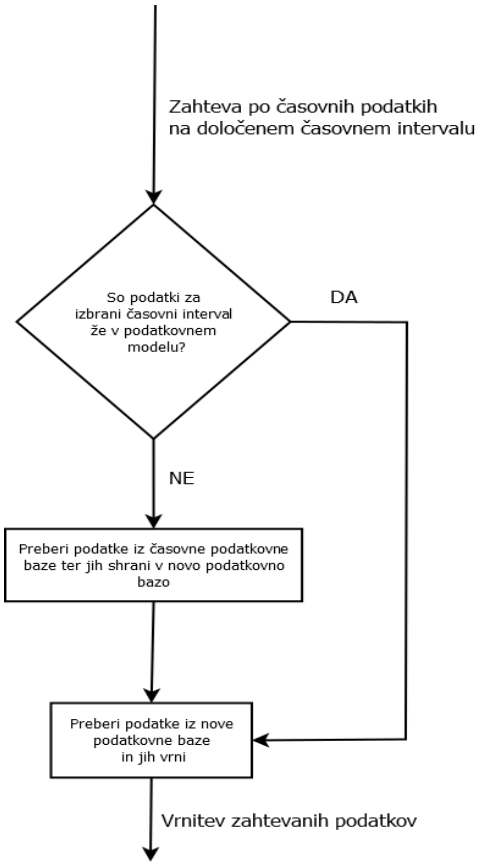


Diagram 3: Diagram postopka za pridobivanje podatkov, če so začasni podatki shranjeni v podatkovni bazi

Druga možnost je shranjevanje začasnih podatkov v spomin, ki je dodeljen aplikaciji (diagram 4). Tu pride do problemov, če je količina informacij večja od količine spominskega prostora, ki je aplikaciji dodeljen. V tem primeru navadno pride do nepričakovane zaustavitve programa, čemur se je potrebno izogniti. Temu se lahko izogne z zagotovilom, da je največja količina podatkov shranjena v spominu dodeljenem aplikaciji dovolj majhna ali tako, da se poveča količina spomina aplikacije. Pri ogromni količini podatkov torej ta možnost ne pride v poštev, saj spomina aplikacije ne moremo povečevati v neskončnost. Aplikacija postane tudi počasna in preveč požrešna glede spomina in temu se je potrebno izogniti.

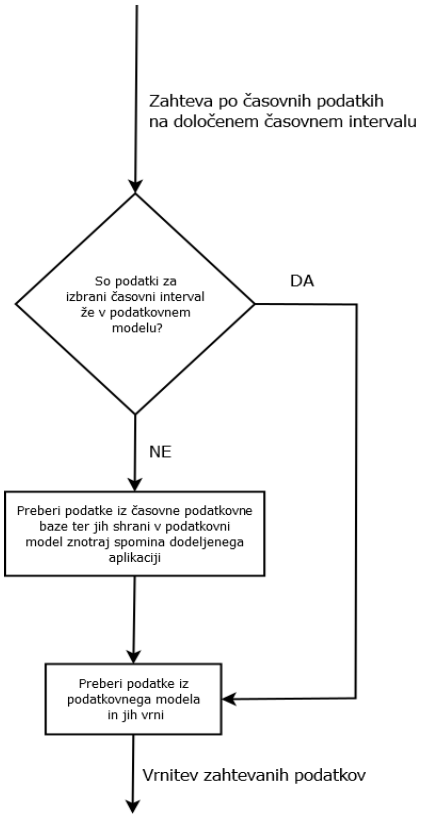


Diagram 4: Diagram postopka za pridobivanje podatkov, če so podatki shranjeni znotraj spomina dodeljenega aplikaciji

- **Kako urediti časovne podatke, da bo do njih mogoče dostopati iz pogleda za pregled časovnih metrik ter pogleda za generiranje poročila?**

  Ob izbiri odgovora na prvi dve vprašanji, je potrebno pozornost usmeriti tudi na to, da se podatki berejo iz izvorne podatkovne baze in shranjujejo v novo bazo ali v spomin na tak način, da ne ustreza zgolj pregledu časovnih metrik, ampak da se jih lahko uporabi tudi za generiranje poročila. To je zelo pomembno dejstvo, saj prepreči pisanje več metod, ki služijo istemu namenu. Odgovor na to vprašanje mora biti skrbno in premišljeno izbran ter pogojuje odgovoru na prvi dve vprašanji.

- **Kako dodati nova polja, ki so potrebna za pravilno delovanje aplikacije, torej kam naj se dodajo podatki, ki jih v podatkovni bazi še ni?**

  Za generiranje poročila so potrebni dodatni podatki, ki jih v izvorni podatkovni bazi še ni. Del teh podatkov je opisan v poglavju 3.1.3. in jih uporabnik vnese ročno po tem, ko mu jih posreduje vodja računa ali jih pridobi sam. Za pravilno analizo časovnih metrik pa so potrebni tudi dodatni podatki o razvijalcih. Ti podatki so:

  - vodja skupine, v kateri se nahaja razvijalec;
  - vodja podskupine, v kateri se nahaja razvijalec;
  - razpoložljivost razvijalca.

  Ti podatki morajo torej biti nekje shranjeni, da se lahko ob generiranju poročil ali ob prikazu časovnih metrik uporabijo.

3.2.3. IZBIRA ARHITEKTURNIH REŠITEV

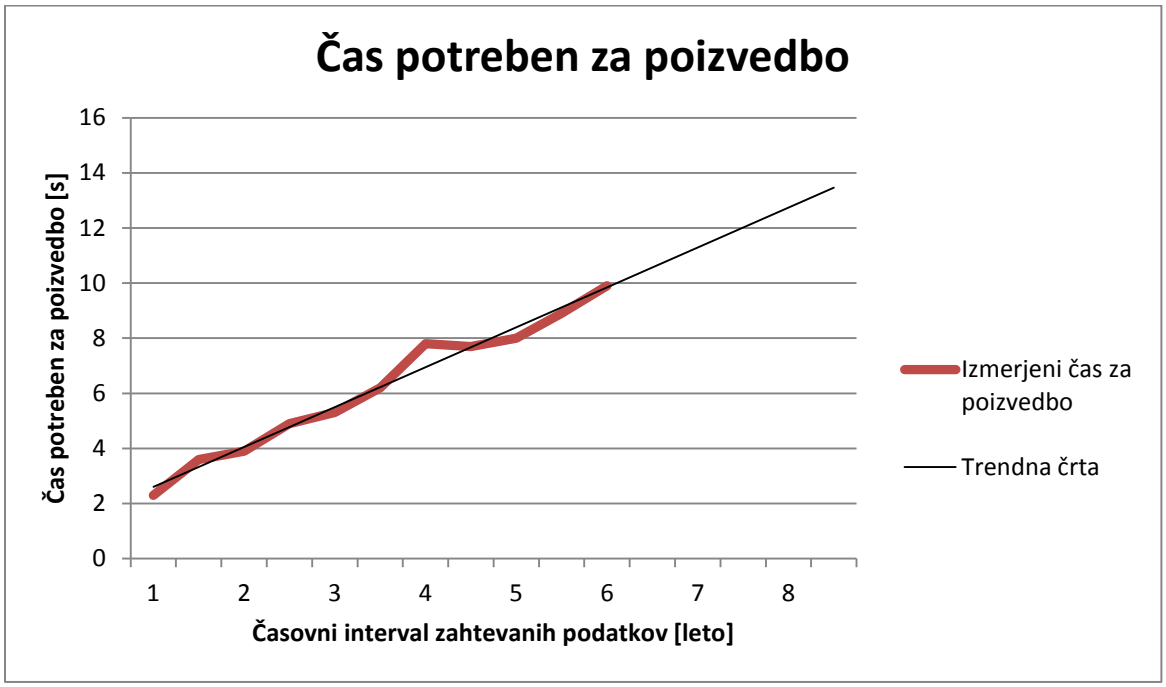
Za izbiro arhitekturnih rešitev je potrebno dobro premisliti odgovore na vsa vprašanja postavljena v razdelku 3.2.2., saj so med seboj tesno povezani.

Za prikazovanje časovnih metrik je v aplikaciji uporabljena rešitev, ki je kombinacija poizvedb med podatkovno bazo in simulacijo OLAP kocke. Na prvi pogled bi bilo bolje uporabiti OLAP analizo, saj knjižnice za to že obstajajo, poleg tega pa je o tem napisane veliko literature. OLAP je tudi preverjen princip reševanja problemov, ki so podobni danemu, saj je bil razvit za hitro poizvedbo po različnih dimenzijah. Dejstvo pa je, da je OLAP orodje za pregledovanje ogromne količine podatkov in iskanje veliko možnih povezav med njimi. V podjetjih, ki analizirajo na primer prodane produkte, ki jih je navadno ogromno, se ta izbira nedvomno splača.

Glede na to, da je za aplikacijo potrebno prikazati relativno malo število podatkov ter da je število dimenzij majhno in se ne spreminja, se zdi uporaba OLAP analize preveč močno orodje. To, da je preveč močno samo po sebi, ne predstavlja problema, do problema pride pri času, ki se ga porabi za implementacijo. Potrebno se je podrobno seznaniti z OLAP principom, imeti obširno znanje o podatkovnih bazah ter se naučiti izdelave večdimenzionalnih poizvedb. Če bi bila to torej aplikacija namenjena zgolj pregledovanju ogromne količine podatkov in njihove povezanosti, bi se implementacija OLAP izplačala, v mojem primeru pa se ne, saj prednosti ne odtehtajo časa, ki bi bil za implementacijo porabljen.

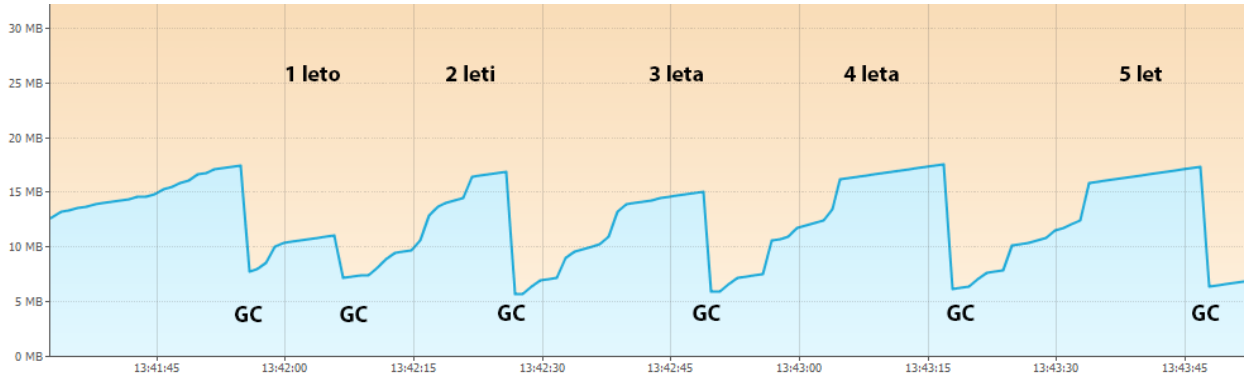
Uporabljena arhitektura je dovolj hitra, da uporabnik nemoteno dostopa do podatkov. V prid izbiri pa govori tudi dejstvo, da se večina zahtev po prikazu časa nanaša na časovni interval, ki je

manjši od enega leta. Slika 10 prikazuje naraščanje časa za poizvedbo ob povečevanju časovnega intervala želenih podatkov.



Slika 10: Graf časa porabljenega za poizvedbo v odvisnosti od izbire časovnega intervala podatkov

Drug problem, ki se lahko pojavi pri branju iz podatkovne baze in nadaljnjo manipulacijo s podatki znotraj spomina dodeljenega aplikaciji, je prekoračitev dodeljenega spomina. Privzeta vrednost, nastavljena ob kreiranju aplikacije na platformi Java, je 256 kilobajtov spominskega prostora. Ta vrednost ne sme biti prekoračena. Slika 7 prikazuje zasedenost pomnilniškega prostora s povečevanjem časovnega intervala za prikaz časovnih podatkov.



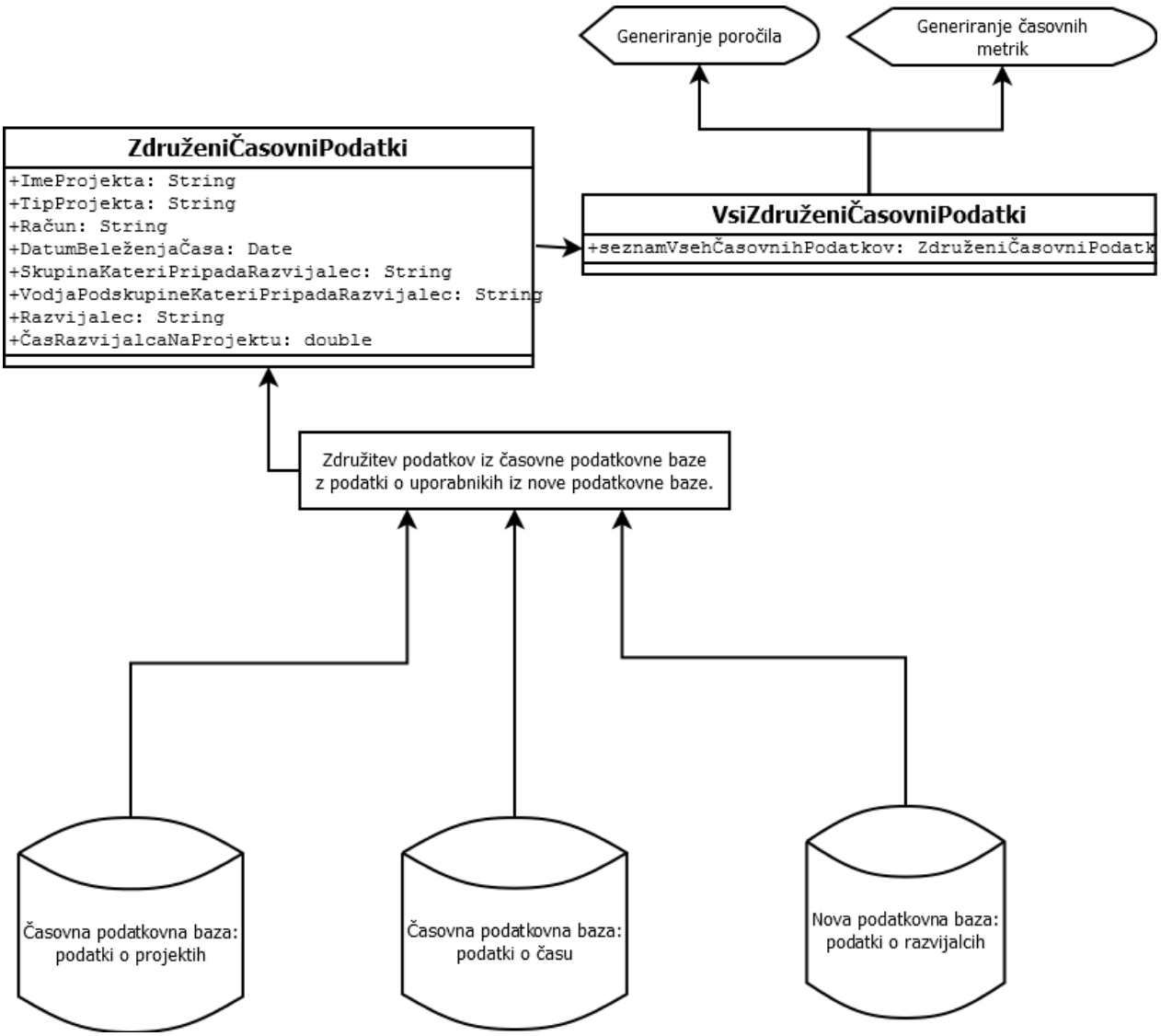
Slika 11: Zasedenost pomnilnika v odvisnosti od izbire časovnega intervala

Na sliki 11 je označena zasedenost pomnilnika za izbiro različnih časovnih intervalov. Med testiranjem sem ročno zaganjal smetarja, na sliki 11 označen z »GC«, tako se prikaže realna poraba pomnilnika. Kot je razvidno, se zasedenost pomnilnika povečuje zelo počasi, zato ni pričakovati, da bo presegla zgornjo mejo. Za primer, če je izbran časovni interval petih let, se med izvajanjem poizvedbe porabi približno 18 kilobajtov pomnilnika. Ko je poizvedba končana in podatki združeni, pa se ta poraba še zmanjša, saj se lahko sprostijo podatkovne strukture, v katerih so bili začasni podatki. To nalogo opravi smetar ob primernem trenutku. Ker na zagon smetarja na varen način ni mogoče vplivati, lahko to teoretično pomeni, da se smetar ne bo nikoli zagnal in presežena bo zgornja meja dodeljenega pomnilnika. Gledano iz te perspektive je to lahko problem pri vsaki aplikaciji, tako da tukaj ne predstavlja resnejše grožnje.

V aplikaciji so vsi začasni časovni podatki shranjeni v pomnilniku aplikacije. Kot že omenjeno, je shranjevanje podatkov v novo podatkovno bazo zamudno dejanje. Podatke je potrebno najprej prebrati iz izvorne podatkovne baze ter jih shraniti v novo podatkovno bazo. Vnos v novo podatkovno bazo in prikaz lahko sicer teče vzporedno, a je vseeno potrebno podatke ob vsaki spremembi zahtev uporabnika ponovno brati iz podatkovne baze, kjer vzporednost ni mogoča, saj podatkov ne moremo prikazati, dokler jih ne preberemo.

Za dostop do podatkov je potrebno zagotoviti takšno arhitekturo, da bo obstajala enaka podatkovna struktura tudi za dostop do podatkov iz pogleda za generiranje poročila. Podatkovna struktura mora biti tako nekoliko bolj univerzalna in ne omejena zgolj na prikaz finančnih ali časovnih metrik. Pri finančnih podatkih taka izvedba ni mogoča, a je zbiranje finančnih podatkov nezahtevno, zato to ne predstavlja velike ovire. Poenoten podatkovni model pri finančnih podatkih ni mogoč, ker potrebujemo za generiranje poročila dodatne podatke, kot je tip stroška. Ker tega podatka v izvorni podatkovni bazi ni, ga ne moremo uporabiti. Podatek o tipu stroškov uporabnik vnese ročno in ta vnos se skupaj z ostalimi informacijami o transakciji shrani v novo bazo. Podatki iz te podatkovne baze se nato uporabijo za generiranje poročila.

Pri časovnih podatkih teh razlik ni. Podatki so enaki za prikaz časovnih metrik kot za generiranje poročila. Arhitekturna rešitev, kjer imamo le en podatkovni model za uporabo podatkov, je torej tu mogoča. Kot omenjeno, v izvorni podatkovni bazi ni vseh podatkov, so pa ti podatki že shranjeni v novi podatkovni bazi. Potrebna je le združitev podatkov iz obeh podatkovnih baz v neko smiselno strukturo. Poudariti je potrebno, da mora biti struktura taka, da lahko uporabnik pri pregledu časovnih metrik spreminja število in vrstni red dimenzij. Prikaz podatkovne strukture za časovne podatke je na siki 12.



Slika 12: Podatkovni model časovnih podatkov

Kot je razvidno iz slike 12, so časovni podatki brani iz dveh podatkovnih baz, iz časovne baze, kjer so shranjeni podatki o času, projektih, uporabnikih, ki so delali na projektih itd., in iz nove podatkovne baze, kjer so dodatni podatki, ki jih v časovni bazi ni, so pa za analizo potrebni. Ko so podatki prebrani, se obdelajo znotraj programa, tako da se za uporabnika za določen časovni zapis poišče dodatne podatke v novi podatkovni bazi. Od tu se lahko razbere, kateri skupini in kateremu vodji podskupine pripada. Ostali podatki se preberejo iz časovne baze in tako so vsa potrebna polja izpolnjena. Ustvari se nov objekt, ki vsebuje vse potrebne informacije za prikaz ene časovne transakcije. Vsi objekti se shranijo v neko polje objektov in se nato lahko uporabijo za prikaz časovnih metrik ali za generiranje poročila.

Objekt časovne transakcije torej vsebuje vse potrebne informacije za oba pogleda, kjer ga potrebujemo. Grajenje objektov je vedno enako, zato lahko uporabimo isti podatkovni model za obe predstavitvi.

3.2.4. TESTIRANJE

Med implementacijo je potrebno temeljito testiranje implementiranih metod. Testiranje je ključni del dobrega programiranja. Za vsako funkcionalnost mora dober programer napisati test. Ker je aplikacija zgrajena iz logičnega in grafičnega dela, sem testiral le logični del. Z uporabo dodatnih knjižnic, kot je JFCUnit [4], je sicer mogoče simulirati tudi uporabo grafičnega vmesnika, vendar se v testiranje tega s pomočjo testov nisem spuščal.

Logični del aplikacije je bil testiran z uporabo knjižnice JUnit 4.0 [5]. Knjižnica omogoča veliko število raznih testov, s katerimi se da večino implementacije simulirati in preveriti rezultate. Problem je še vedno testirati klice v podatkovno bazo, kjer se za različne parametre vrne različno število podatkov. V tem primeru je potrebno najprej preveriti število zapisov, ki se morajo vrniti, in s tem primerjati število dejansko vrnjenih zapisov. Testi so bili napisani najpodrobneje za implementacijo povezave z MySQL podatkovnim strežnikom, to je namreč zelo kritičen del arhitekture, ki lahko onemogoči delo, če se izkaže, da je bil narobe implementiran.

Grafični vmesnik je bil testiran iz strani osebe, ki je aplikacijo zahtevala. Vmesnik se je tako prilagajal zahtevam, ki so se lahko pojavile ob testiranju. Ta način sem uporabil, ker oseba, ki zahteva aplikacijo najbolj ve, kako naj bodo stvari postavljene in kako naj funkcionirajo, saj tega z unit testi ni mogoče zagotoviti.

3.3. IZBIRA TEHNOLOGIJ

3.3.1. PROGRAMSKI JEZIK

Celotna aplikacija je napisana v programskem jeziku Java. Odločitev za ta programski jezik je bila delno predlagana iz strani podjetja, saj je večina aplikacij v podjetju napisanih v Javi, sem pa tudi sam podprl ta predlog, saj imam s programiranjem v Javi največ izkušenj. Za Javo smo se odločili, ker ima zelo veliko skupnost. Obstaja torej veliko literature, s katero si lahko uporabnik pomaga. Dodatna prednost, ki je za podjetje zelo pomembna, je, da je Java prosto dostopna platforma, ki deluje na Linux in Windows operacijskih sistemih. Tudi razvojna orodja so prosto dostopna, tako da z uporabo Jave stroškov ni, dokler se ne uporablja plačljivih dodatnih knjižnic. Velika skupnost se odraža tudi v tem, da je napisanih veliko dodatnih knjižnic, ki se jih lahko vključi v projekt, in velikokrat olajšajo delo. Po mojem mnenju je največja slabost Jave uporaba javax.swing paketa. Problem ni v neuporabnosti paketa, ampak v tem, da ima ogromno zbirko metod, med katerimi je težko izbrati tisto, ki jo potrebujemo.

3.3.2. RAZVOJNO OKOLJE

Kot razvojno okolje ima razvijalec na voljo veliko orodij, kjer ima vsaka svoje prednosti in slabosti. V mojem primeru sem izbral Eclipse, saj ponuja vrsto uporabnih orodij, je hiter in zanesljiv. Orodje, ki mogoče najbolj konkurira Eclipse je Net Beans. Net Beans je usmerjen predvsem v grafično oblikovanje aplikacij. Ima namreč graditelj grafičnega vmesnika, ki uporabniku omogoča, da grafične komponente na aplikacijo ne dodaja s programiranjem, ampak le s »povleci in spusti« metodo. Glede na to, da je aplikacija v celoti napisana ročno, se za Net Beans nisem odločil.

3.3.3. DODATNE KNJIŽNICE

Pregled izbire dodatnih knjižnic:

- XLAF (Look and Feel podjetja X, kjer sem opravljal študentsko delo): knjižnica je uporabljena, da aplikacija zgleda kot vse ostale Java aplikacije znotraj podjetja. S tem se doseže neko kontinuiteto, uporabnikom pa se ni potrebno vedno znova privajati na nov izgled.
- OPENCSV: to je knjižnica, ki omogoča zelo enostavno generiranje datotek CSV [7]. V aplikaciji je uporabljena za generiranje datoteke CSV iz časovnih podatkov.
- ITEXT: je prosto dostopna knjižnica za ustvarjanje in spreminjanje datotek PDF. Je zelo enostavna za porabo in ima veliko podpore v obliki literature. Obstaja velik nabor knjižnic za generiranje datotek PDF, a se ta zdi najbolj enostavna in od uporabnika zahteva najmanj [2].
- SWINGX: knjižnica, ki vsebuje veliko grafičnih elementov, kot so sortiranje, filtriranje, koledar, iskanje itd. [6]. V aplikaciji je uporabljen paket za izgradnjo koledarja, da lahko uporabnik grafično izbere datum.

3.3.4. PODATKOVNA BAZA

Glede podatkovne baze ni bilo veliko izbire. V podjetju se uporablja MySQL. Na tem podatkovnem strežniku so shranjeni vsi podatki. Smiselno je, da se tudi novo bazo naredi v MySQL, saj to zahteva najmanj dela za razvijalca in administratorja baze.

3.4. IMPLEMENTACIJA

Ko arhitektura ustreza zahtevam, se je mogoče posvetiti implementaciji. Ta ne bi smela predstavljati večjega problema, če je zamišljena arhitektura pravilna. Do velikega problema pa lahko pride, če se pri arhitekturi kaj spregleda, saj se lahko to ugotovi že globoko v razvoju kode, kar lahko pride do točke, ko ni več poti naprej in začeti je potrebno znova.

3.4.1. IMPLEMENTACIJA ČASOVNIH METRIK

Implementacija časovnih metrik si zasluži posebno pozornost, saj je tu implementiran algoritem, ki simulira OLAP kocko. Da se doseže polno funkcionalnost OLAP kocke, je potrebno implementirati algoritem, ki je sposoben izpolniti vse zahteve uporabnika glede dimenzij in njihovega vrstnega reda. Kot omenjeno v poglavju 3.2.3., je ob zahtevi uporabnika za prikaz podatkov na določenem časovnem intervalu potrebno prebrati podatke iz dveh podatkovnih baz ter jih med seboj povezati. Tako povezani podatki so nato shranjeni v polju časovnih transakcij. Od tu naprej je potrebno zgraditi tak algoritem, ki bo na zahtevo uporabnika prikazoval vse možne kombinacije dimenzij.

Uporabnik po izbiri datuma in kliku na gumb za prenos podatkov izbere dimenzije ter njihov vrstni red. Dimenzije so v pravilnem vrstnem redu poslane metodi za grajenje urejenega drevesa podatkov, kjer vsak nivo drevesa ustreza dimenziji z enako zaporedno številko. Za gradnjo drevesa potrebujemo vozlišča in liste. V tem primeru so listi tudi vozlišča, s to razliko, da nimajo otrok. Ustvarjen je bil nov objekt, ki predstavlja vozlišče drevesa. Podatki, ki jih vsebuje so vrednost tega vozlišča in polje otrok vozlišča. Vrednost vozlišča lahko pomeni ime projekta, ime računa, tip projekta itd. Predstavlja torej vrednost zahtevane dimenzije na tem nivoju. Podatkovnega tipa vrednosti zato ni mogoče definirati, saj se ne ve, kateri dimenziji bo pripadal. Na sliki 13 je predstavljena koda omenjenega algoritma.

```
/**
* Builds a time data tree for the time interval of this object
*
* @param dimensions ArrayList of dimensions that the user would like to see
*     The dimensions are ordered and the tree is built so that it corresponds
*     to the order
*/
public void buildTree(ArrayList<String> dimensions) {
      this.dimensions = dimensions;
      parent = new DataTreeNode();
      int dimIndex = 0;
      for(TimeDataRecord temp: timeData) {
            dimIndex = 0;
            fillDataForTree(temp, parent, dimIndex);
      }
}

/**
 * Recursive method that populates time data tree
 *
 * @param temp Current {@link TimeDataRecord} that is being processed
 * @param node Parent node
 * @param dimIndex Index of the dimension being processed. That is used so
that
 * it can calculate the depth of tree
 */
private void fillDataForTree(TimeDataRecord temp, DataTreeNode node, int
dimIndex) {
      String dim = dimensions.get(dimIndex);

      Object dimVal = null;
      boolean containsNode = false;

      /*
       * Check which dimension is currently in process
       */
      if(dim.equals("Group")) {
            dimVal = temp.getGroup();
      } else if(dim.equals("Account")) {
            dimVal = temp.getAccount();
      } else if(dim.equals("Project")) {
            dimVal = temp.getProject();
      } else if(dim.equals("Developer")) {
            dimVal = temp.getDeveloper();
      } else if(dim.equals("Project Type")) {
            dimVal = temp.getProjectType();
      } else if(dim.equals("Team Coach")) {
            dimVal = temp.getTC();
      }
```

```
	/*
	* Loop through all the node children and check if children belong to
	*currently processed dimension. If so set parent node to this node
	*/
	for(DataTreeNode tempNode:node.getChildren()) {
		if(tempNode.getAttribute().equals(dimVal)) {
			node = tempNode;
			containsNode = true;
		}
	}

	/*
	* If children are the objects of currently processed dimension and are
	*not already part of the children list, build new node for each child
	*and add it to parent node
	*/
	if(!containsNode) {
		DataTreeNode newNode = new DataTreeNode();
		newNode.setAttribute(dimVal);
		node.setChild(newNode);
		node = newNode;
	}

	dimIndex++;

	/*
	* If this is not the last dimension desired by user, make a recursive
	*call, else sum all the times of children, append them as one new node
	*and return
	*/
	if(dimIndex < dimensions.size())
		fillDataForTree(temp, node, dimIndex);
	else {
		DataTreeNode newNode = new DataTreeNode();
		dimVal = temp.getTimeWorked();
		newNode.setAttribute(dimVal);
		node.setChild(newNode);
		return;
	}

}
```

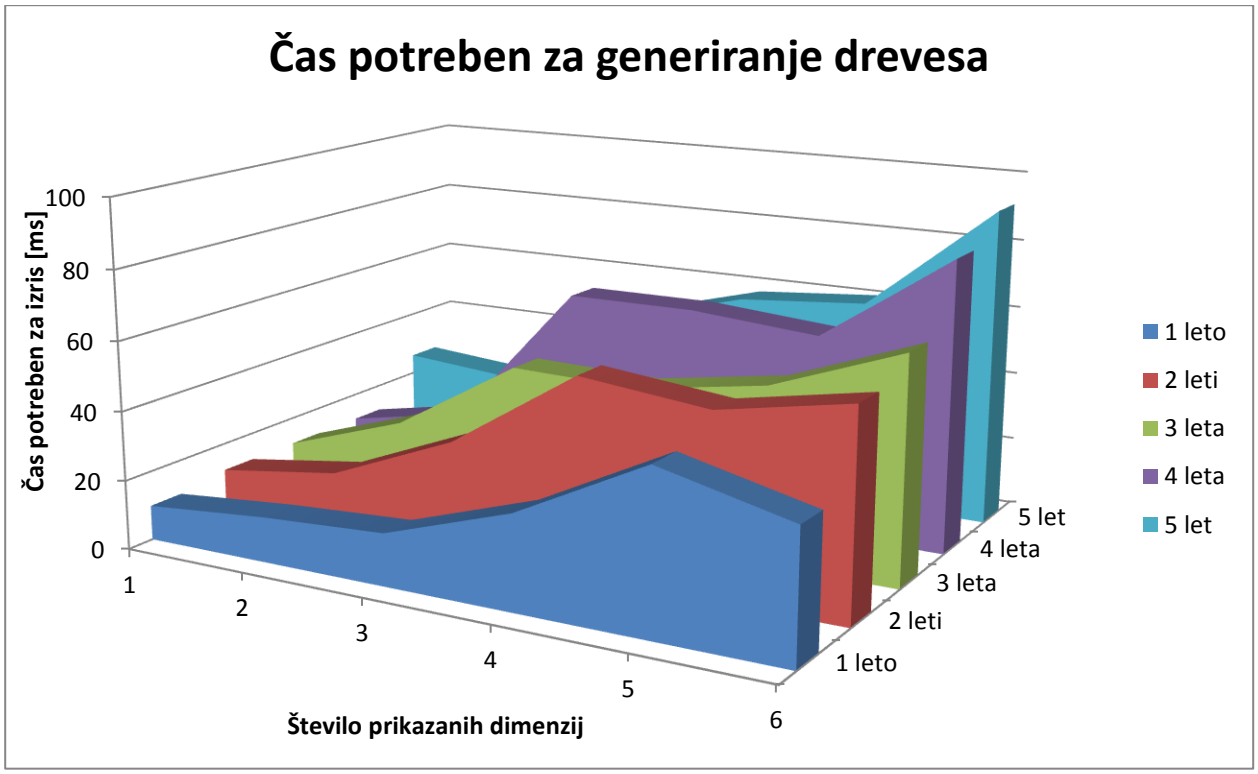
Slika 13: Izrezek programske kode za gradnjo časovnega podatkovnega drevesa

Algoritem deluje tako, da vsako časovno transakcijo umesti v pravilni del drevesa. Ker vsaka transakcija vedno vsebuje vse izbrane dimenzije, je potrebno le, da se vrednosti dimenzij v pravilnem vrstnem redu berejo iz zapisa o časovni transakciji. V začetku je ustvarjeno starševsko vozlišče drevesa. Na to vozlišče se dodajajo vsi časovni zapisi. Ob branju vsakega novega časovnega zapisa se algoritem postavi na prvo izbrano dimenzijo. Nato se preveri, če je prebrana vrednost dimenzije iz časovnega zapisa že v drevesu. Če je, se kot novo vozlišče nastavi tisto vozlišče, ki to vrednost vsebuje. S tem se prepreči podvajanje enakih vozlišč. V primeru, da

nobeno od obstoječih vozlišč na tem nivoju ne vsebuje vrednosti dimenzije trenutno branega časovnega zapisa, se ustvari novo vozlišče in se doda v seznam otrok starša. Sledi rekurzivni klic iste metode, kjer se pregleduje naslednja zaporedna dimenzija definirana iz strani uporabnika, starševsko vozlišče pa je ali že obstoječe vozlišče, ki ima enako vrednost, ko je vrednost dimenzije v časovnem zapisu, ali pa na novo ustvarjeno vozlišče. Čas je vedno zadnja dimenzija, uporabnik je tudi nima na razpolago za izbiro, saj bi bila nesmiselna, ker nima pomena kjerkoli drugje kot na zadnjem mestu. Algoritem zato, ko doseže zadnjo dimenzijo, doda novo vozlišče, če to še ne obstaja v drevesu. Temu vozlišču nato v seznam otrok zapiše čas trenutno brane časovne transakcije in se rekurzivno vrača do starševskega vozlišča.

Kot rezultat gradnje podatkovnega drevesa se vedno vrne le najvišje vozlišče. Vsa ostala vozlišča so namreč pripeta na to vozlišče. To vozlišče je uporabljeno za gradnjo grafičnega drevesa, ki bo predstavljeno uporabniku. Kot omenjeno v poglavju 3.1.1., lahko uporabnik nastavi tudi filter, ki se nanaša na prvo dimenzijo. Ta filter se upošteva pri gradnji grafičnega drevesa. Gradnja ni nič drugega kot sprehajanje po seznamih otrok vseh vozlišč. Poteka rekurzivno in se najprej razteza v globino, ko pa doseže zadnji nivo, začne z raztezanjem v širino. Ko pride do zadnjega nivoja, sešteje vse otroke, ki predstavljajo čas, in seštevek doda kot list k trenutnemu vozlišču. Če je uporabljen filter, se na prvem nivoju drevesa izriše le vozlišče, ki je zahtevano.

Slika 14 prikazuje časovno zahtevnost gradnje drevesa v odvisnosti od količine podatkov in številom izbranih dimenzij.



Slika 14: Graf časa za gradnjo drevesa v odvisnosti od števila dimenzij in količine podatkov

Iz slike 14 je razvidno, da poteka grajenje podatkovnega in grafičnega drevesa dovolj hitro, da uporabnika to med delom ne moti. Odločitev glede arhitekturnih rešitev za časovne metrike je bila torej pravilna.

Z izbiro poljubnih dimenzij in njihovega vrstnega reda je torej mogoče zgraditi poljubno drevo, ki ustreza željam uporabnika. Če bi bila namesto drevesa uporabljena tabela, bi lahko govorili o vrtilni tabeli. Vrtilna tabela je namreč orodje, ki omogoča različne poglede nad enako podatkovno množico. Množica podatkov se torej ne spreminja, spreminja se le zahteva uporabnika po načinu prikaza podatkov iz te množice [1]. Zgoraj predstavljeno podatkovno drevo je ravno to. Možnosti za prikaz podatkov iz iste podatkovne množice je ogromno in uporabnik si lahko izbere, katere podatke bi imel rad predstavljene. Razlika je le v tem, da se v primeru te aplikacije za predstavitev uporabi drevo in ne tabela.

3.4.2. IMPLEMENTACIJA FINANČNIH METRIK

Kot omenjeno v prejšnjih poglavjih, implementacija finančnih metrih ne predstavlja večjega problema. Potrebno je le pridobiti vse finančne transakcije za izbran račun ali projekt in jih izpisati v preglednico znotraj aplikacije. Glavni cilj finančnih metrik je, da se uporabnik lahko seznani s finančnimi transakcijami in ugotovi, koliko odliva in priliva je bilo na določenem projektu v določenem časovnem obdobju ter kaj je bil razlog za to. Zaželeno je tudi, da se po izpisu zahtevanih podatkov naredi nek pregled nad vsemi prikazanimi finančnimi transakcijami. Potreben je torej seštevek vseh odlivov in prilivov, kjer se loči med transakcijami, ki še niso bile plačane, in tistimi, ki so označene, da bodo izvršene v prihodnosti. Uporabnik lahko na tak način dobi občutek, koliko je v finančnem smislu nek projekt uspešen.

Velikokrat se zgodi, da uporabnik ni prepričan o začetku in koncu projekta. Da bi dobil natančne podatke, bi zato moral najprej ugotoviti datum začetka in, v primeru, da je projekt končan, datum zaprtja projekta, da bi pri dobil pravilne podatke. Da se temu uporabnik izogne, se lahko odloči, da ne bo uporabil polja za vnos datuma, in izpišejo se vsi finančni podatki za izbrani projekt/račun.

3.4.3. IMPLEMENTACIJA POROČILA

Za generiranje poročila so potrebne časovne in finančne metrike. Podatkovni model za časovne metrike je lahko isti kot za pregled časovnih metrik. Pri finančnih podatkih pa je potrebno dodatno polje za vnos tipa stroška. Potrebne so tudi spremenljivke za pravilno analizo podatkov.

Realizacija vnosa tipa stroška je bila narejena tako, da je bil dodan nov pogled. Pogled je podoben tistemu za pregledovanje finančnih transakcij, le da se tu prikažejo vedno vse finančne transakcije za celoten račun. To je pomembno zato, da ima uporabnik pregled nad vsem

finančnim dogajanjem na računu. Preglednica prikazanih transakcij ima dodatno polje, kjer lahko uporabnik izbere tip stroška. To transakcijo mora tudi potrditi s kljukico v zadnjem polju tabele. Ko je uporabnik opravil vse potrebne spremembe, obkljukane podatke shrani v novo podatkovno bazo. Za generiranje poročila so uporabljeni le ti finančni podatki, zato je nujno pred generiranjem poročila pripraviti vse potrebne finančne podatke. Ob enem pa so potrebna tudi polja za vnos spremenljivk. Spremenljivke so globalne in veljajo za celoten račun, zato niso datumsko omejene. Uporabnik jih lahko vnese v za to namenjeno okno in jih shrani v novo podatkovno bazo.

Za generiranje poročila se izbere časovni interval in račun, za katerega naj se poročilo generira. Časovni podatki se zbirajo na enak način kot pri pregledu časovnih metrik, najprej se torej izvede poizvedba v podatkovno bazo in nato pride do gradnje podatkovnega drevesa, kjer je izbrana le ena dimenzija, ki je račun, in je filter nastavljen na ime računa. Finančni podatki se preberejo iz nove podatkovne baze, preberejo se tudi spremenljivke. Preostane le še izračun želenih vrednosti in predstavitev poročila uporabniku. Sledi še možnost izvoza poročila v datoteko PDF, v primeru, da je uporabnik s poročilom zadovoljen.

4. UPORABA APLIKACIJE

Aplikacija je namenjena osebi, ki je v podjetju odgovorna za zagotavljanje kvalitete storitev in pregledom nad financami. Aplikacija je bila grajena na predpostavki, da ta oseba ve, kaj dela, in tako dodatni mehanizmi za zaščito podatkov niso potrebni. Ravno tako niso potrebne razne obrazložitve poteka dela, zato aplikacija teh ne vsebuje.

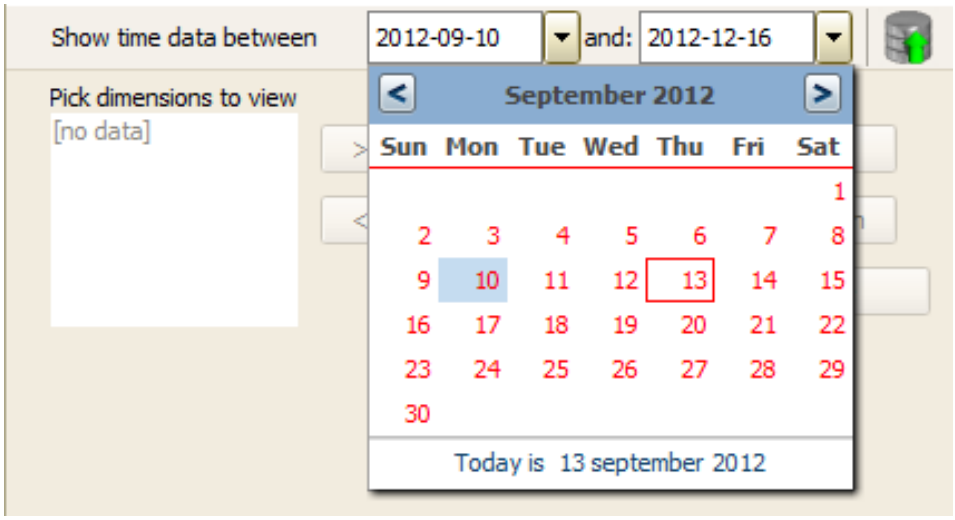
Ker aplikacija vsebuje več pogledov, so ti med seboj ločeni tako, da je vsak postavljen v svoj zavihek (glej sliko 15).

Slika 15: Vrstica z zavihki

Uporabnik lahko tako prosto preklaplja med zavihki, ne da bi pri tem izgubil podatke v ostalih pogledih.

4.1. UPORABA ČASOVNEGA DELA APLIKACIJE

Časovni del aplikacije je namenjen zgolj pregledovanju časa, ki so ga razvijalci porabili na projektih, računih itd. Uporabnik najprej izbere časovni interval, za katerega bi rad, da se mu prikažejo podatki (slika 16). To je omogočeno s pomočjo orodja za izbiro datuma. Ko sta začetni in končni datum izbrana, uporabnik zahteva podatke iz podatkovne baze s pritiskom na gumb za izvrševanje poizvedbe SQL.



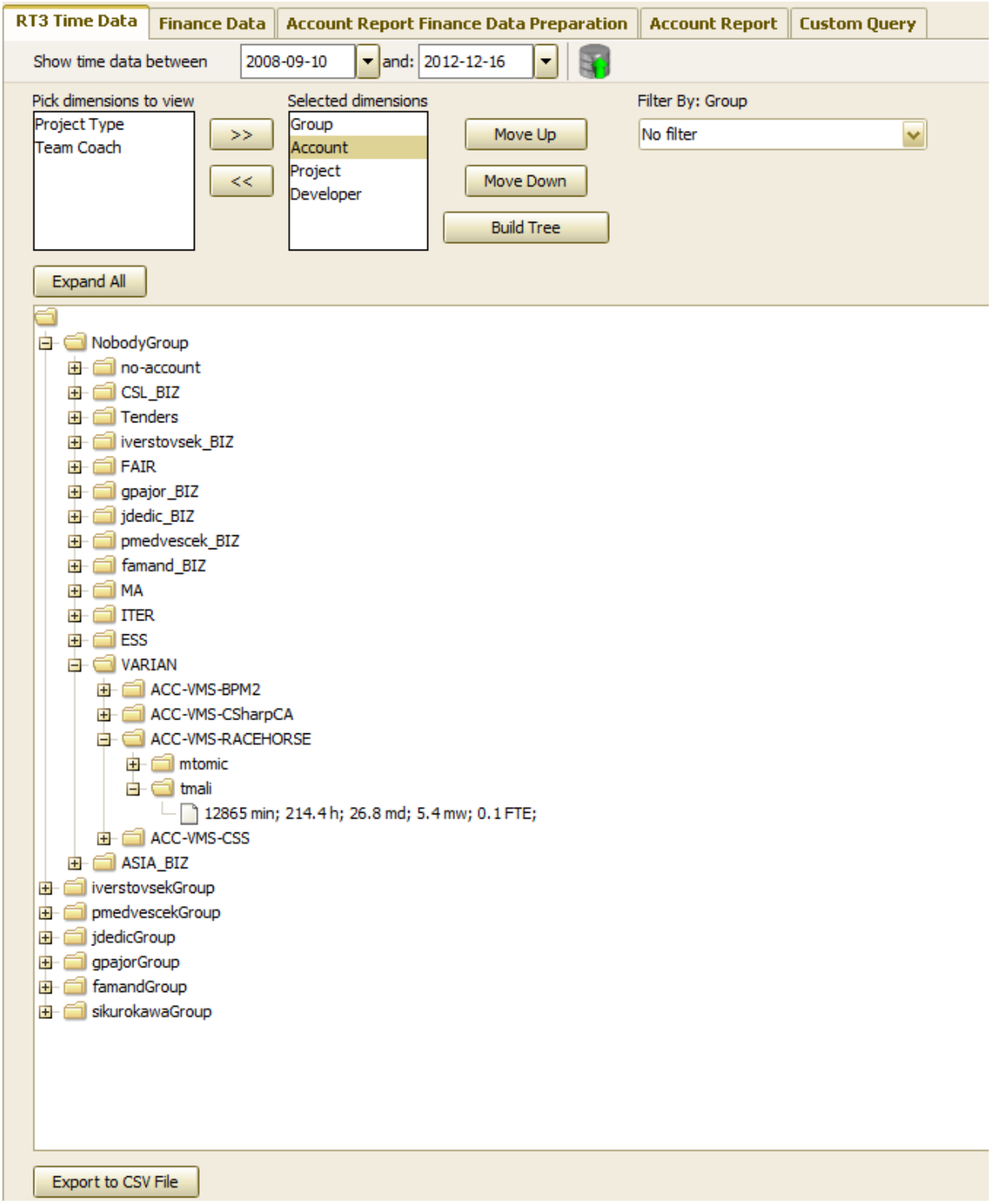
Slika 16: Izbira datuma za poizvedbo po časovnih podatkih

Ko uporabnik pritisne gumb za poizvedbo po podatkih, rezultati niso vidni takoj, saj poizvedba traja nekaj časa (glej sliko 10). Med tem časom mora uporabnik dobiti neko povratno informacijo, da je njegov zahtevek v obdelavi. Če te informacije ni, je uporabnik negotov o tem,

ali je sploh pritisnil na gumb ali je kriv nek drug problem. Uporabnik je obveščen o izvajanju njegove zahteve tako, da se mu vsi elementi na zaslonu onemogočijo in se mu pojavi animacija, ki uporabniku sporoči, da je zahteva v obdelavi (slika 17).

Slika 17: Animacija, ki obvesti uporabnika o procesiranju njegove zahteve

Ta animacija je uporabljena skozi celotno aplikacijo vedno, ko mora uporabnik čakati na rezultate. Ko so podatki prebrani iz podatkovne baze, se vsi grafični elementi omogočijo in uporabnik ima na izbiro vse dimenzije, s katerimi lahko manipulira (slika 18). Dimenzije določi tako, da izbere dimenzije iz levega seznama in jih doda na desni seznam. Če z vrstnim redom dimenzij ni zadovoljen, ga lahko spreminja z gumboma na desni strani seznama. Ob spreminjanju vrstnega reda dimenzij se spreminja tudi vsebnost spustnega seznama, ki se ga lahko uporabi kot filter za najvišjo izbrano dimenzijo. Ko je uporabnik zadovoljen s postavitvijo dimenzij, klikne na gumb »Build Tree« in drevo, na podlagi izbranih dimenzij, se mu izriše v za to namenjenem delu okna.

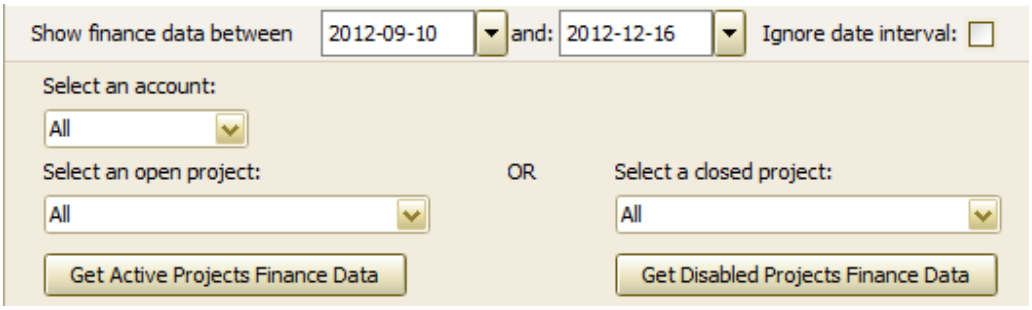


Slika 18: Primer uporabe pregledovalnika časovnih metrik

Čas je predstavljen v obliki minut, ur, števila delovnih dni, števila delovnih tednov, kolikšen del celotnega časa enega razvijalca v enem letu predstavlja ta čas. Če želi, lahko uporabnik drevo izvozi v datoteko CSV in s podatki naprej manipulira.

4.2. UPORABA FINANČNEGA DELA APLIKACIJE

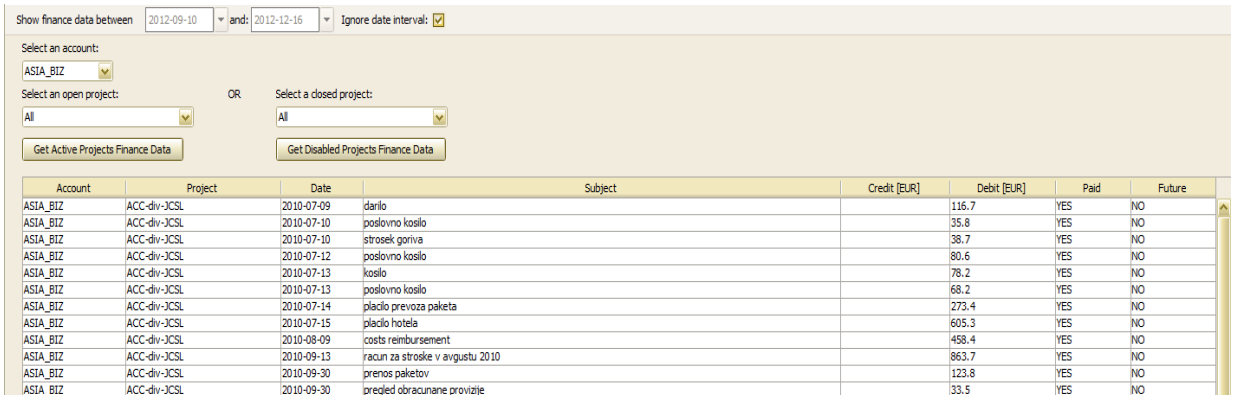
V pogledu za pregled finančnih podatkov lahko uporabnik izbere časovni interval, na katerem ga podatki zanimajo. Izbere lahko tudi projekt ali račun, za katerega ga podatki zanimajo, lahko pa tudi vse podatke na računu ali vse podatke v podjetju (slika 19).



Slika 19: Izbira parametrov za prikaz finančnih podatkov

Uporabnik lahko opcijsko izbere tudi, da se datum ignorira in se izpišejo vsi finančni podatki za izbran račun ali projekt. Ravno tako lahko izbira med odprtimi in zaprtimi projekti na računu. Ob menjavi računa se oba spustna seznama posodobita, tako da ne more priti do napak.

Ko je uporabnik zadovoljen z nastavljenimi parametri, odvisno od tega, katere projekte bi rad videl, klikne na enega izmed gumbov. Med branjem podatkov iz baze se vse grafične komponente onemogočijo in uporabniku je predstavljena animacija (glej sliko 17), ki mu pove, da je njegova zahteva v obdelavi. Ko so podatki zbrani, se grafični elementi ponovno omogočijo in rezultat se izpiše v preglednico (slika 20).


Show finance data between 2012-09-10 and: 2012-12-16 Ignore date interval:

Select an account: ASIA_BIZ

Select an open project: All OR Select a closed project: All

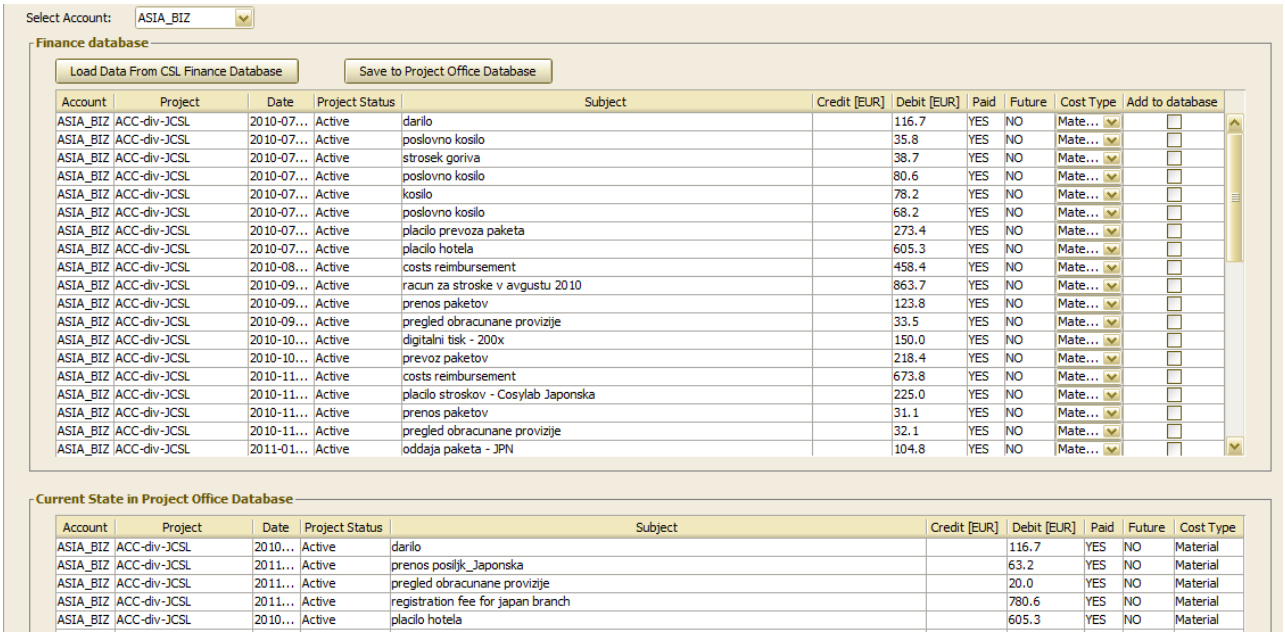
Get Active Projects Finance Data | Get Disabled Projects Finance Data

Account	Project	Date	Subject	Credit [EUR]	Debit [EUR]	Paid	Future
ASIA_BIZ	ACC-div-JCSL	2010-07-09	darilo		116.7	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-07-10	poslovno kosilo		35.8	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-07-10	strosek goriva		38.7	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-07-12	poslovno kosilo		80.6	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-07-13	kosilo		78.2	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-07-13	poslovno kosilo		68.2	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-07-14	placilo prevoza paketa		273.4	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-07-15	placilo hotela		605.3	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-08-09	costs reimbursement		458.4	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-09-13	racun za stroske v avgustu 2010		863.7	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-09-30	prenos paketov		123.8	YES	NO
ASIA_BIZ	ACC-div-JCSL	2010-09-30	pregled obracunane provizije		33.5	YES	NO

Slika 20: Prikaz rezultatov poizvedbe po finančnih podatkih

4.3. GENERIRANJE POROČILA

Pred generiranjem poročila je potrebno finančne transakcije potrditi in jim dodati tip stroška. Za to obstaja poseben pogled, kjer jih uporabnik lahko nastavi, potrdi in shrani v podatkovno bazo. Po shranitvi se mu shranjeni podatki prikažejo v tabeli, ki se bere iz nove podatkovne baze in ne iz finančne (slika 21).

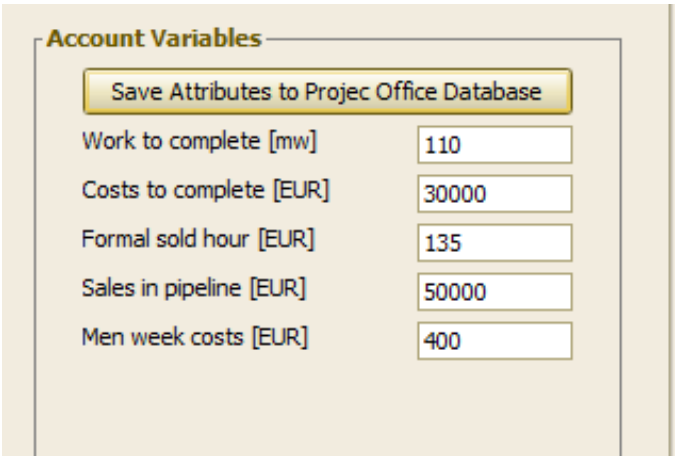


Account	Project	Date	Project Status	Subject	Credit [EUR]	Debit [EUR]	Paid	Future	Cost Type	Add to database
ASIA_BIZ	ACC-div-JCSL	2010-07...	Active	darilo		116.7	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-07...	Active	poslovno kosilo		35.8	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-07...	Active	strosek goriva		38.7	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-07...	Active	poslovno kosilo		80.6	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-07...	Active	kosilo		78.2	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-07...	Active	poslovno kosilo		68.2	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-07...	Active	placilo prevoza paketa		273.4	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-07...	Active	placilo hotela		605.3	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-08...	Active	costs reimbursement		458.4	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-09...	Active	racun za stroske v avgustu 2010		863.7	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-09...	Active	prenos paketov		123.8	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-09...	Active	pregled obracunane provizije		33.5	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-10...	Active	digitalni tisk - 200x		150.0	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-10...	Active	prevoz paketov		218.4	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-11...	Active	costs reimbursement		673.8	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-11...	Active	placilo stroskov - Cosylab Japonska		225.0	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-11...	Active	prenos paketov		31.1	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2010-11...	Active	pregled obracunane provizije		32.1	YES	NO	Mate...	
ASIA_BIZ	ACC-div-JCSL	2011-01...	Active	oddaja paketa - JPN		104.8	YES	NO	Mate...	

Account	Project	Date	Project Status	Subject	Credit [EUR]	Debit [EUR]	Paid	Future	Cost Type
ASIA_BIZ	ACC-div-JCSL	2010...	Active	darilo		116.7	YES	NO	Material
ASIA_BIZ	ACC-div-JCSL	2011...	Active	prenos posiljk_Japonska		63.2	YES	NO	Material
ASIA_BIZ	ACC-div-JCSL	2011...	Active	pregled obracunane provizije		20.0	YES	NO	Material
ASIA_BIZ	ACC-div-JCSL	2011...	Active	registration fee for japan branch		780.6	YES	NO	Material
ASIA_BIZ	ACC-div-JCSL	2010...	Active	placilo hotela		605.3	YES	NO	Material

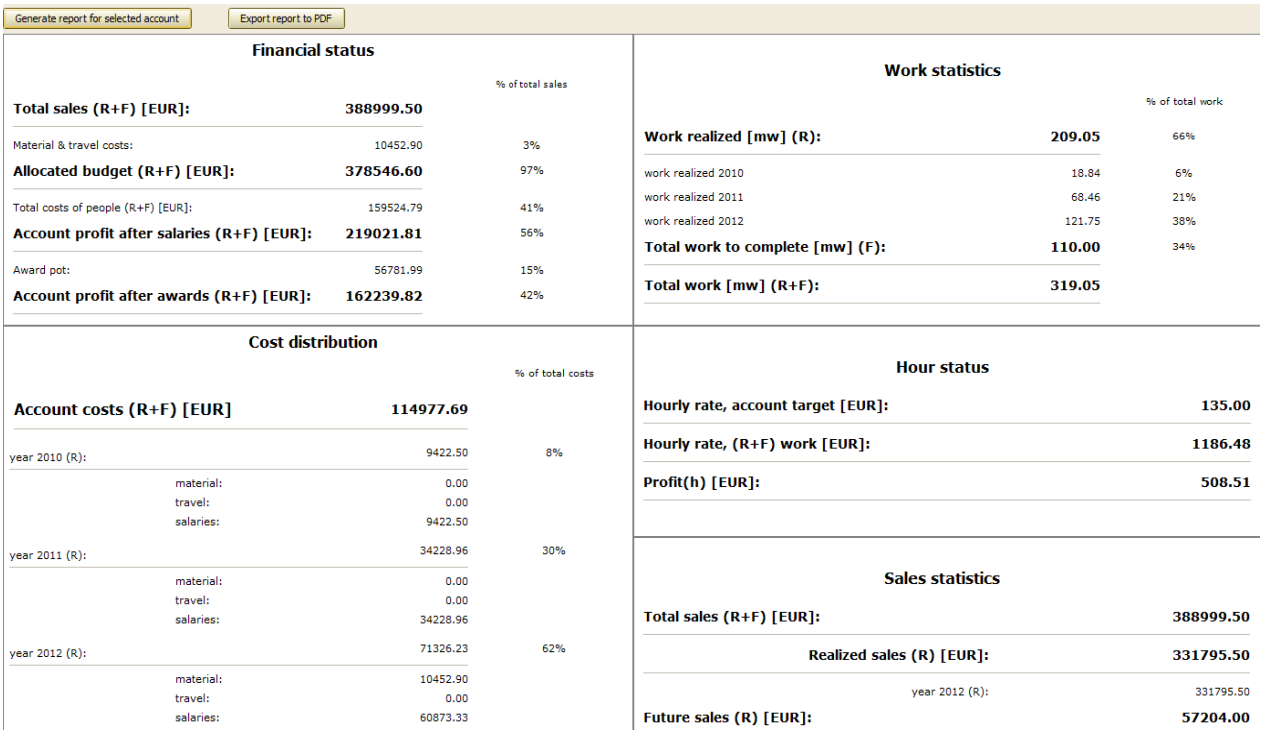
Slika 21: Prikaz vseh podatkov iz finančne podatkovne baze ter nove podatkovne baze

Potrebna so tudi dodatna polja za vnos spremenljivk, ki se uporabljajo za računanje metrik pri generiranju poročila (slika 22).



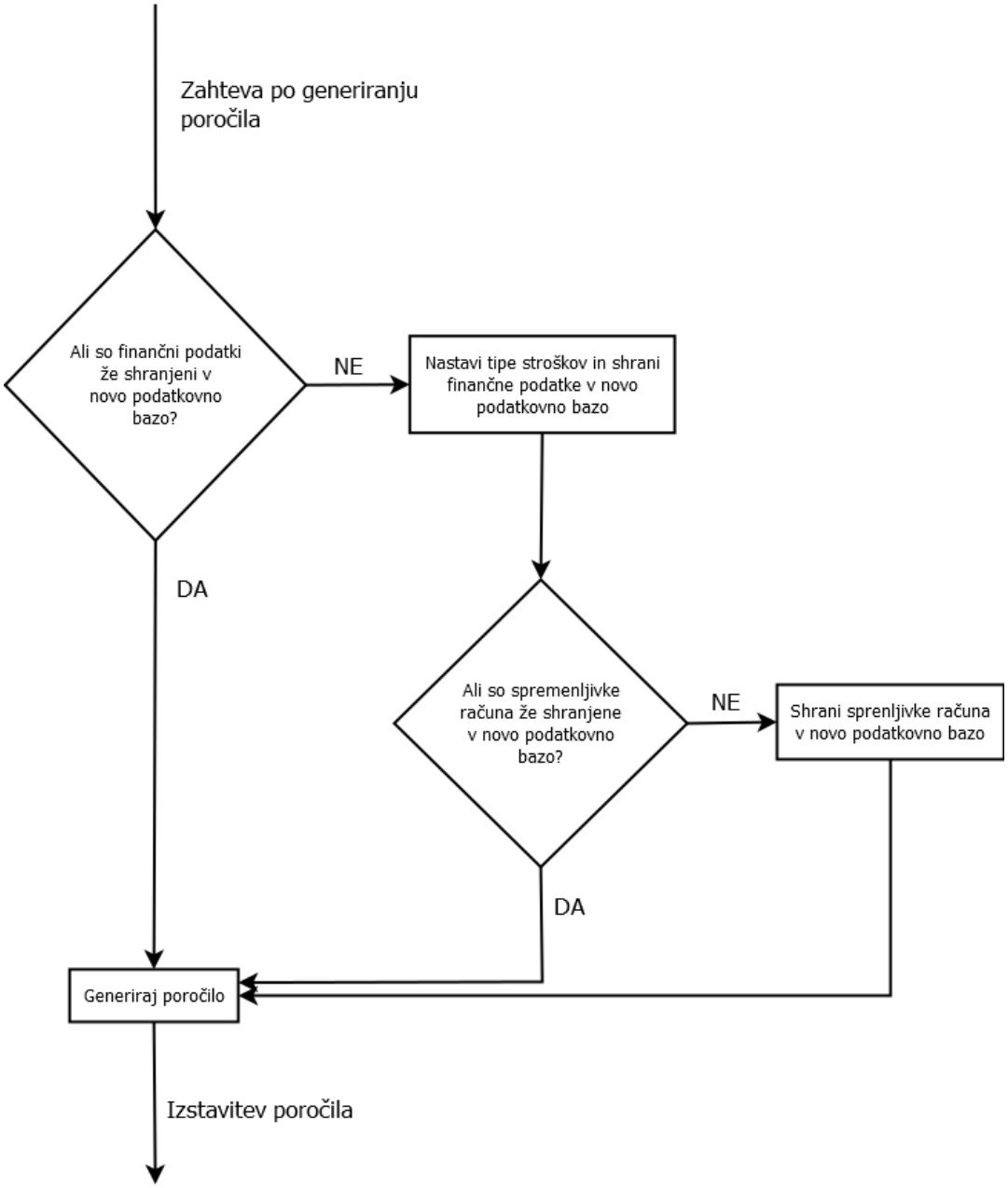
Slika 22: Prikaz polj za vnos spremenljivk računa

Ko so vsi podatki pravilno shranjeni, se lahko začne generiranje poročila. Za to obstaja drug pogled, kjer si uporabnik lahko izbere časovni interval ter račun, za katerega bi rad ustvaril poročilo. Po pritisku na gumb za generiranje poročila se to ustvari in prikaže. Če je uporabnik s poročilom zadovoljen, ga lahko izvozi v datoteko PDF in predstavi vodjem računa ter vodstvu podjetja (slika 23).



Slika 23: Primer končnega poročila za obdobje od 2010 do 2012

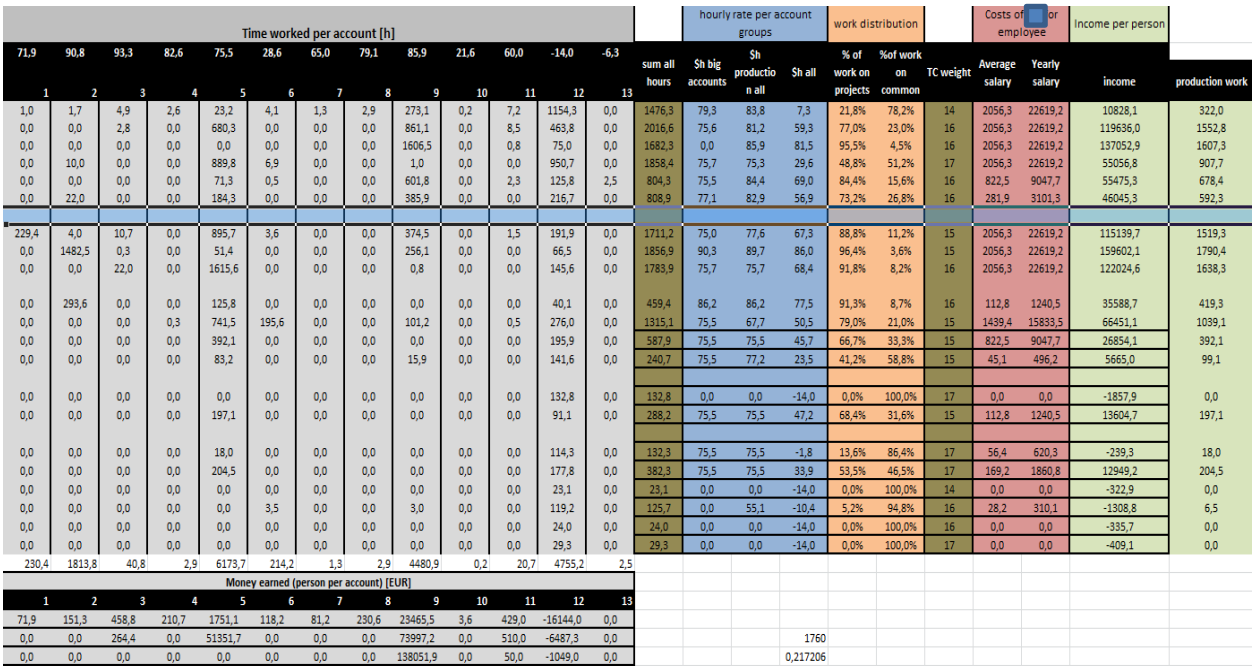
Postopek generiranje poročila od zahteve pa do izdelave poročila je prikazan na sliki 24.



Slika 24: Postopek generiranja poročila

5. ZAKLJUČEK IN MOŽNOSTI ZA NADALJNJI RAZVOJ

Pred pojavom aplikacije za pregled časovnih in finančnih metrik ter generiranje poročila je morala oseba zadolžena za nadzor nad kvaliteto storitev podatke zbirati s pomočjo spletnih orodij, ki niso bila poenotena in so praviloma delovala počasi. Za generiranje poročila je bilo potrebno zbrati vse podatke jih vpeljati v program za manipuliranje s preglednicami in podatke pravilno urediti (slika 24). Za kar je bilo potrebno veliko časa, proces manipuliranja z veliko količino podatkov pa je nagnjen k napakam, ki so prispevale k dodatnem podaljšanju dela.



Time worked per account [h]														hourly rate per account groups			work distribution			Costs of employee		Income per person	
71,9	90,8	93,3	82,6	75,5	28,6	65,0	79,1	85,9	21,6	60,0	-14,0	-6,3	sum all hours	$h big accounts	$h production all	$h all	% of work on projects	%of work on common	TC weight	Average salary	Yearly salary	income	production work
1	2	3	4	5	6	7	8	9	10	11	12	13											
1,0	1,7	4,9	2,6	23,2	4,1	1,3	2,9	273,1	0,2	7,2	1154,3	0,0	1476,3	79,3	83,8	7,3	21,8%	78,2%	14	2056,3	22619,2	10828,1	322,0
0,0	0,0	2,8	0,0	680,3	0,0	0,0	0,0	861,1	0,0	8,5	463,8	0,0	2016,6	75,6	81,2	59,3	77,0%	23,0%	16	2056,3	22619,2	119636,0	1552,8
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1606,5	0,0	0,8	75,0	0,0	1682,3	0,0	85,9	81,5	95,5%	4,5%	16	2056,3	22619,2	137052,9	1607,3
0,0	10,0	0,0	0,0	889,8	6,9	0,0	0,0	1,0	0,0	0,0	950,7	0,0	1858,4	75,7	75,3	29,6	48,8%	51,2%	17	2056,3	22619,2	55056,8	907,7
0,0	0,0	0,0	0,0	71,3	0,5	0,0	0,0	601,8	0,0	2,3	125,8	2,5	804,3	75,5	84,4	69,0	84,4%	15,6%	16	822,5	9047,7	55475,3	678,4
0,0	22,0	0,0	0,0	184,3	0,0	0,0	0,0	385,9	0,0	0,0	216,7	0,0	808,9	77,1	82,9	56,9	73,2%	26,8%	16	281,9	3101,3	46045,3	592,3
229,4	4,0	10,7	0,0	895,7	3,6	0,0	0,0	374,5	0,0	1,5	191,9	0,0	1711,2	75,0	77,6	67,3	88,8%	11,2%	15	2056,3	22619,2	115139,7	1519,3
0,0	1482,5	0,3	0,0	51,4	0,0	0,0	0,0	256,1	0,0	0,0	66,5	0,0	1856,9	90,3	89,7	86,0	96,4%	3,6%	15	2056,3	22619,2	159602,1	1790,4
0,0	0,0	22,0	0,0	1615,6	0,0	0,0	0,0	0,8	0,0	0,0	145,6	0,0	1783,9	75,7	75,7	68,4	91,8%	8,2%	16	2056,3	22619,2	122024,6	1638,3
0,0	293,6	0,0	0,0	125,8	0,0	0,0	0,0	0,0	0,0	0,0	40,1	0,0	459,4	86,2	86,2	77,5	91,3%	8,7%	16	112,8	1240,5	35588,7	419,3
0,0	0,0	0,0	0,3	741,5	195,6	0,0	0,0	101,2	0,0	0,5	276,0	0,0	1315,1	75,5	67,7	50,5	79,0%	21,0%	15	1439,4	15833,5	66451,1	1039,1
0,0	0,0	0,0	0,0	392,1	0,0	0,0	0,0	0,0	0,0	0,0	195,9	0,0	587,9	75,5	75,5	45,7	66,7%	33,3%	15	822,5	9047,7	26854,1	392,1
0,0	0,0	0,0	0,0	83,2	0,0	0,0	0,0	15,9	0,0	0,0	141,6	0,0	240,7	75,5	77,2	23,5	41,2%	58,8%	15	45,1	496,2	5665,0	99,1
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	132,8	0,0	132,8	0,0	0,0	-14,0	0,0%	100,0%	17	0,0	0,0	-1857,9	0,0
0,0	0,0	0,0	0,0	197,1	0,0	0,0	0,0	0,0	0,0	0,0	91,1	0,0	288,2	75,5	75,5	47,2	68,4%	31,6%	15	112,8	1240,5	13604,7	197,1
0,0	0,0	0,0	0,0	18,0	0,0	0,0	0,0	0,0	0,0	0,0	114,3	0,0	132,3	75,5	75,5	-1,8	13,6%	86,4%	17	56,4	620,3	-239,3	18,0
0,0	0,0	0,0	0,0	204,5	0,0	0,0	0,0	0,0	0,0	0,0	177,8	0,0	382,3	75,5	75,5	33,9	53,5%	46,5%	17	169,2	1860,8	12949,2	204,5
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	23,1	0,0	23,1	0,0	0,0	-14,0	0,0%	100,0%	14	0,0	0,0	-322,9	0,0
0,0	0,0	0,0	0,0	0,0	3,5	0,0	0,0	3,0	0,0	0,0	119,2	0,0	125,7	0,0	55,1	-10,4	5,2%	94,8%	16	28,2	310,1	-1308,8	6,5
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	24,0	0,0	24,0	0,0	0,0	-14,0	0,0%	100,0%	16	0,0	0,0	-335,7	0,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	29,3	0,0	29,3	0,0	0,0	-14,0	0,0%	100,0%	17	0,0	0,0	-409,1	0,0
230,4	1813,8	40,8	2,9	6173,7	214,2	1,3	2,9	4480,9	0,2	20,7	4755,2	2,5											

Money earned (person per account) (EUR)												
1	2	3	4	5	6	7	8	9	10	11	12	13
71,9	151,3	458,8	210,7	1751,1	118,2	81,2	230,6	23465,5	3,6	429,0	-16144,0	0,0
0,0	0,0	264,4	0,0	51351,7	0,0	0,0	0,0	73997,2	0,0	510,0	-6487,3	0,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	138051,9	0,0	50,0	-1049,0	0,0

1760	
0,217206	

Slika 25: Ena izmed mnogih preglednic potrebnih za generiranje poročila

Razvoj aplikacije pomeni združitev vseh orodij za pregled časovnih in finančnih podatkov znotraj ene aplikacije, ki še vedno omogoča izvoz podatkov v program za manipuliranje s preglednicami, funkcionalnost prejšnje metode pa se s tem ne zmanjša. Omogoča tudi generiranje poročil zgolj z izbiro datuma ter pritiskom na gumb. V primerjavi s prejšnjim načinom, ko je bilo potrebno manipulirati z ogromno podatki v preglednicah, je ta rešitev mnogo hitrejša in, ob pravilnem potrjevanju finančnih transakcij ter pravilno vnesenim spremenljivkam, odporna na napake, saj se povsem izključi človeški faktor. Končni rezultat aplikacije je torej poenotenje orodij za zbiranje časovnih in finančnih podatkov ter veliko hitrejše generiranje poročil, ki je manj nagnjeno k napakam. Vse to vodi k razbremenitvi osebe zadolžene za kontrolo kvalitete storitev. Velik

doprinos k podjetju je torej, da ima ta oseba sedaj več časa za opravljanje ostalih zadolžitev in je zato veliko bolj efektivna.

Aplikacija je zastavljena tako, da omogoča možnosti nadgradnje Ena izmed možnosti bi bila denimo lahko prikaz časovnih podatkov v obliki grafov ali pa izris grafa, ki prikazuje stroške na določenem računu. Predlog s strani podjetja je tudi, da dodajo nova poročila; s tem popolnoma poenoti sistem poročil. Ta poročila bi obsegala:

- Poročilo o skupinah razvijalcev:
    - število ljudi v skupini ter njihov status (zaposleni, študent …);
    - število ur skupine opravljenih na računih;
    - finančni podatki skupine.
- Poročilo o vseh računih:
    - izpis osnovnih časovnih in finančnih podatkov;
    - primerjava časovnih in finančnih metrik med računi.
- Poročilo o vseh skupinah:
    - izpis osnovnih časovnih in finančnih podatkov;
    - primerjava časovnih in finančnih metrik med skupinami;
    - primerjava uspešnosti skupin v primerjavi z prejšnjimi obdobji.

VIRI:

[1] Bill Jelen, Michael Alexander, *Pivot Table Data Crunching*, ZDA: Que Publishing, 2006, pogl. 1

[2] Bruno Lowagie, *iText in Action, Second Edition*, ZDA: Manning Publications Co, 2011, pogl. 1

[3] Erik Thomsen, *OLAP Solutions: Building Multidimensional Information Systems*, Canada: John Wiley & Sons, 2002, pogl. 1

[4] (2004) JFCUnit. Dostopno na: http://jfcunit.sourceforge.net/

[5] (2011) JUnit Documentation. Dostopno na: http://junit.sourceforge.net/javadoc/

[6] (2011) SwingX. Dostopno na: http://swingx.java.net/

[7] (2011) What is open CSV. Dostopno na: http://opencsv.sourceforge.net/#what-is-opencsv