Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Denis Vitez

Odkrivanje kršitev v sistemu za registracijo delovnega časa Time&Space

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Ljubljana, 2016

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Denis Vitez

Odkrivanje kršitev v sistemu za registracijo delovnega časa Time&Space

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Aleksander Sadikov

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Odkrivanje kršitev v sistemu za registracijo delovnega časa Time&Space

Tematika naloge:

Izdelajte rešitev za odkrivanje kršitev v sistemu za registracijo delovnega časa Time&Space. Rešitev naj temelji na analizi realnih podatkov in ne na vnaprej določenih pravilih. Opišite arhitekturo razvite rešitve, uporabljene metode in analizirajte učinkovitost ter uporabnost izdelane rešitve v realnem okolju.

Zahvaljujem se mentorju, viš. pred. dr. Aleksandru Sadikovu za strokovno pomoč in nasvete pri izdelavi diplomske naloge. Zahvaljujem se tudi vsem zaposlenim na podjetju Špica International d. o. o., ki so mi pomagali pri pridobivanju podatkov in razlagi delovanja sistema za registracijo delovnega časa Time&Space, ter svoji družini, ki me je tekom študija podpirala.

Diplomo posvečam svojim najbližjim.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
T&S	Time&Space	Time&Space
MAD	median absolute derivation	absolutni odklon mediane
KDE	kernel density estimate	cenilka gostote jedra
k-NN	k-Nearest Neighbors	k najbližjih sosedov
CSV	comma seperated values	z ločilom razmejene vrednosti
API	application programming interface	programski vmesnik aplikacije
RWD	responsive web design	odziven spletni design
CRUD	create, read, update and delete	ustvari, beri, posodobi in briši
JSON	JavaScript Object Notation	opis objekta JavaScript
REST	representational state transfer	arhitektura za imenjavo podatkov med spletnimi stranmi

Povzetek

Naslov: Odkrivanje kršitev s sistemu za registracijo delovnega časa Time&Space

Namen diplomskega dela je predstavitev rešitve odkrivanja kršitev v sistemu za registracijo delovnega časa T&S. Klasične rešitve odkrivanja kršitev slonijo na vnaprej definiranih pravilih, ki so pogosto del samega sistema in se težko spreminjajo. Pravila so po navadi odvisna tudi od lokalnih zakonov, ki veljajo v posameznih državah, saj težko najdemo državo, kjer bi se delovna zakonodajo popolnoma ujemala s katero izmed drugih držav. Namen tega diplomskega dela je zato predstaviti način, ki ne temelji na vnaprej definiranih pravilih, ampak na vedenjskih navadah uporabnikov in odstopanjih od običajnih vzorcev obnašanja. V sklopu diplomskega dela smo razvili spletno aplikacijo, ki na podlagi podatkov iz obstoječega T&S sistema operaterju predlaga možne kršitve.

Ključne besede: registracija delovnega časa, Time&Space, osamelci.

Abstract

Title: Detecting violations in time and attendance system Time&Space

The purpose of this thesis is to present solution for detection of violations in time and attendance system T&S. Classical solutions for detecting violations are based on predefined rules which are typically hard coded into systems and are difficult to change. Rules are usually dependant on local laws that are enforced in individual countries. It is almost impossible to find two countries that would have identical work related laws. Therefore this thesis presents a way that is not based on hard coded rules but it uses worker habits and deviations from their normal patterns. To present the findings of this thesis web application was developed that presents operator with possible violations.

Keywords: time and attendance system, Time&Space, outlier detection.

Poglavje 1

Uvod

Danes skoraj ne najdemo srednjega ali velikega podjetja, ki ne bi uporabljalo elektronskega vodenja prisotnosti na delu. V nekaterih državah je takšen način vodenja delovnega časa tudi zakonsko določen in je za podjetja obvezen. Zaradi velikosti trga obstaja za reševanje te problematike nešteto programskih in strojnih rešitev. Ena izmed teh rešitev je sistem za registracijo in vodenje delovnega časa Time&Space, ki ga na trgu ponuja podjetje Špica International d. o. o. Ta sistem je med najbolj razširjenimi v Sloveniji, uporabljajo pa ga tudi v velikih podjetjih, kot je na primer Luka Koper. V sistemu se vse registracije, ki jih uporabniki opravijo preko registracijskih terminalov, shranjujejo v podatkovno bazo, kjer nato sistem nad nad njimi opravlja različne operacije. Sistem pozna dve vrsti urnikov, ki jih podjetja najpogosteje uporabljajo za svoje zaposlene. To so nepremični urniki, ki se najpogosteje uporabljajo v proizvodnji in premični urniki, ki se uporabljajo pri pisarniškem delu. Za oba tipa urnikov ima sistem vgrajeno tudi detekcijo kršitev, ki delodajalcem ponuja enostaven vzvod za vzdrževanje reda med svojimi zaposleni. Nekatere izmed kršitev, ki jih program zaznava, so:

- pozen prihod na delo,
- kršitev obvezne prisotnosti,
- kršitev dolžine malice,

- nedosežena dneva prisotnost.

Te kršitve so zelo natančno določene glede na urnike, ki jih imajo dodeljene zaposleni in so vgrajene v sam sistem. Ker je na ta način mogoče odkriti le incidente, ki kršijo določena pravila, smo izdelali nadgardnjo sistema, ki za odkrivanje kršitev uporablja podatke same in v njih išče odklone od normalnega obnašanja zaposlenih. Pri izdelavi rešitve so bili kot testni podatki uporabljeni resnični podatki srednje velikega podjetja, ki svojo dejavnost opravlja na več lokacijah, ki so v različnih državah. Rešitev smo zasnovali tako, da je neodvisna od obstoječega sistema in tako ne moti njegovega osnovnega delovanja. Podatki se v našo rešitev uvozijo preko izmenjevalih datotek, rezultati pa se lahko tudi izvozijo nazaj v sistem T&S, kjer se posamezna kršitev vnese kot nov dogodek, ki skrbniku sistema olajša iskanje kršiteljev. Uvoz in zgradba uvoženih podatkov sta podrobneje opisana v poglavju 2.

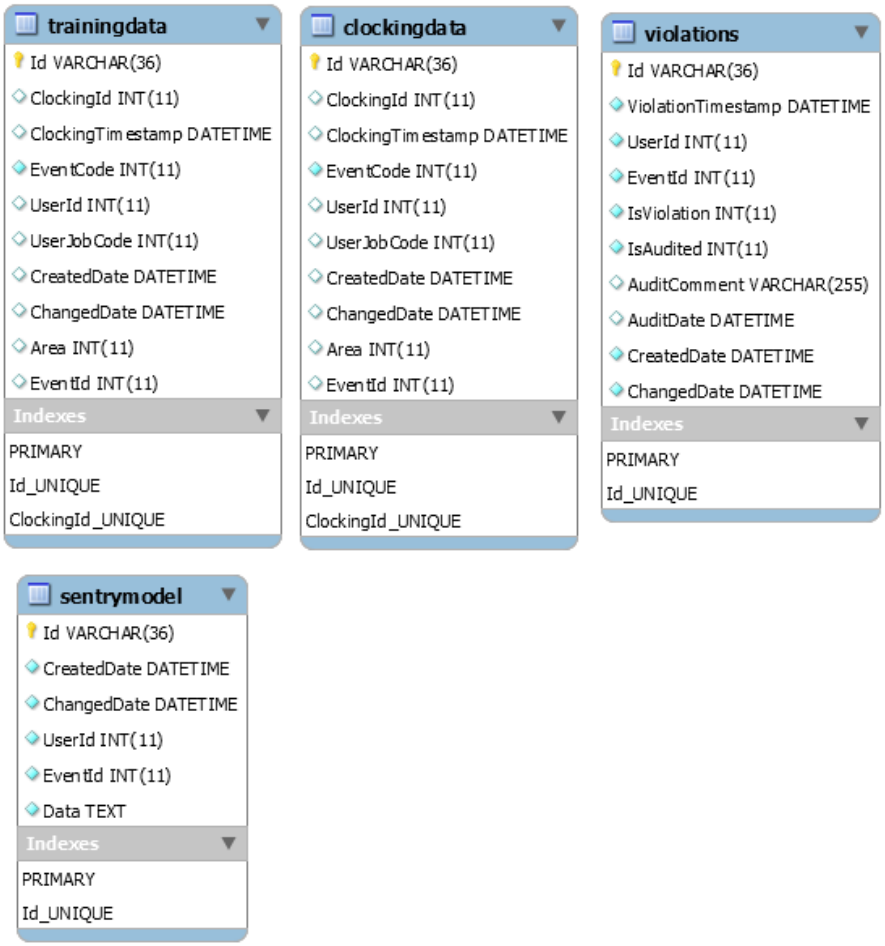
1.1 Podatkovni model

Rešitev za hranjenje podatkov izrablja podatkovno bazo MySQL. V podatkovni bazi so hranjeni tako vhodni podatki kot tudi rezulati samega odkrivanja kršitev. Podatkovni model je nazorneje prikazan na sliki 1.1.

1.2 Obdelava podatkov

Registracije se iz sistema T&S izvozijo v datoteko tipa CSV. Zaradi varovanja osebnih podatkov je treba zagotoviti, da so v izvoženih podatkih samo podatki, ki so nujno potrebni za delovanje rešitve in ne izpostavljajo nobenega tveganja za nedovoljeno deljenje osebnih podatkov zaposlenih podjetja. Obdelavo podatkov v naši rešitvi smo razdelili na pet korakov:

- uvoz podatkov,
- posodabljanje modela,



Slika 1.1: Podatkovni model rešitve.

- iskanje kršitev,
- posodabljanje učnih podatkov,
- izvoz kršitev.

Pri vzpostavitvi rešitve je treba tabelo trainingdata napolniti z veljavnimi registracijami, na podlagi katerih lahko izračunamo potrebne parametre za odkrivanje kršitev. Pozneje lahko tabelo trainingdata razširimo s predlogi kršitev, ki jih je oskrbnik sistema označil za veljavne registracije.

1.3 Spletna aplikacija

Za pregled možnih kršitev smo izdelali spletno aplikacijo v ogrodju Django. V spletni aplikaciji lahko uporabniki, ki imajo dovoljenje za dostop do aplikacije, urejajo statuse predlogov za kršitve. Tako lahko revizirajo posamezne vnose, ki se bodo pozneje vnesli v sistem T&S. Aplikacija je podrobneje opisana v poglavju 4.

Poglavje 2

Zgradba podatkov in njihov izvoz iz sistema Time&Space

Podatke iz sistema T&S izvozimo s pomočjo pogleda na tabeli, kjer so shranjene registracije. Izvoženi podatki morajo ustrezati formatu, ki je opisan v tabeli 2.1. Rezultate pogleda, ki smo ga napisali, shranimo v datoteko tipa CSV.

2.1 Registracije v sistemu Time&Space

Vse registracije, ki pridejo v sistem T&S, se shranijo v podatkovno bazo v tabelo EVENTS. To je osrednja tabela sistema, saj so prav registracije ključni podatki pri sistemu za registracijo delovnega časa in kontrolo pristopa. Pri manipulaciji podatkov iz te tabele (urejanje, dodajanje, brisanje) je treba biti dosleden, saj lahko nepazljiva sprememba podatkov povzroči nepravilno delovanje drugih komponent sistema, kot je recimo obračun plač. Registracija je v osnovi dokaj preprosta entiteta, saj za osnovno delovanje keteregakoli sistema za registracijo delovnega časa potrebujemo zgolj podateke o uporabniku, času registracije in tipu dogodka.

Ime stoplca	Tip podatka	Opis
NO	INT	Primarni ključ dogodka.
USERNO	INT	Šifrant uporabnika.
DT	DATETIME	Datum dogodka.
CODE	INT	Koda dogodka (izračuna jo sistem).
LOCATION	INT	Lokacija dogodka.
DATA	INT	Podatki o dogodku (odvisni od tipa dogodka).
TYPE	INT	Tip dogodka (registracija, odsotnost, ...).
EVENTID	INT	Šifrant dogodka.

Tabela 2.1: Struktura podatkov, ki jih izvažamo iz sistema T&S.

2.2 Izvoz podatkov

Podatke iz sistema T&S izvozimo v obliki CSV datoteke, saj bi izvoz velikega števila registracij preko spletnih vmesnikov potreboval preveč časa. Za potrebe izvoza podatkov smo nad tabelo EVENTS izvedli poizvedbo, ki nam vrne zahtevane podatke. Programsko orodje Oracle PL/SQL Developer omogoča enostavno shranjevanje rezultatov poizvedbe v datoteko. Treba je nastaviti obliko izvoza kot CSV, saj se privzeto podatki izvozijo v obliki stolpcev fiksne širine, ki je zaradi praznega prostora v datoteki precej bolj potraten.

2.3 Uvoz podatkov v našo razšitiev

Za potrebo uvoza podatkov iz datoteke CSV v podatkovno bazo naše rešitve smo napisali skripto v programskem jeziku Python, ki poskrbi za uvoz teh podatkov. Podatke iz vhodne datoteke obdeluje po vrsticah in za vsako vrstico v tabelo clockingdata vnese nov zapis. Ob klicu skripte lahko navedemo, ali želimo nove podatke dodati obstoječim, ali pa želimo pred vnosom vsebino tabele pobrisati.

Poglavje 3

Analiza registracij

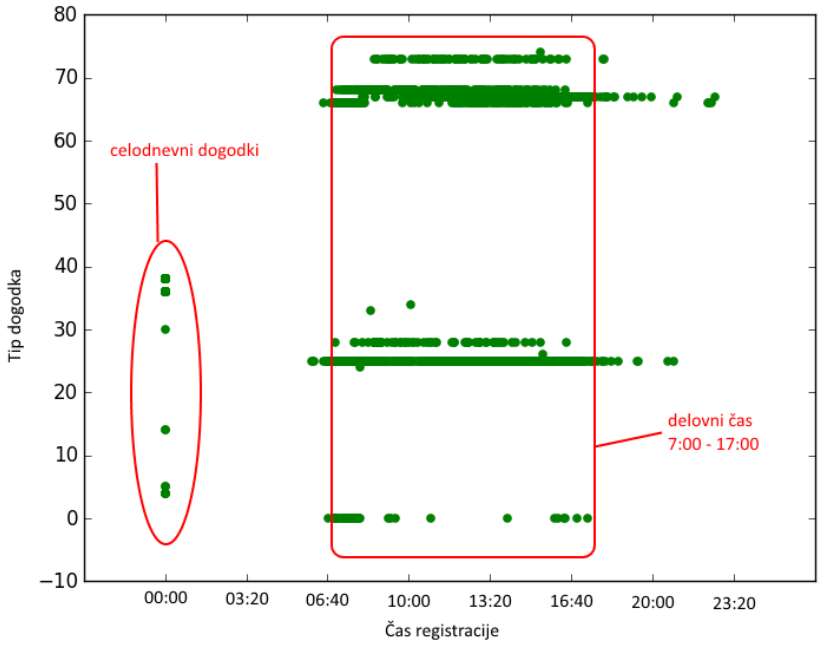
Pri izdelavi naše rešitve smo največ časa posvetili analizi podatkov in iskanju načinov, kako bi na najbolj preprost in učinkovit način poiskali možne kršitve.

3.1 Grafična analiza

Za analizo smo si podatke o registracijah naprej pripravili v grafično obliko. Tako smo izrisali več različnih grafov, preko katerih smo nato prišli do ideje za idelavo same rešitve. Čas registracije smo iz standardnega zapisa datuma pretvorili v numerično vrednost, ki predstavlja število minut v dnevu, ki so minile od polnoči pa do časa dogodka. Za ta korak smo se odločili, saj je numerična predstavitev datuma lažja za nadaljno obdelavo kot pa datumski podatkovni tip.

3.1.1 Izris porazdelitve registracij za posameznega uporabnika

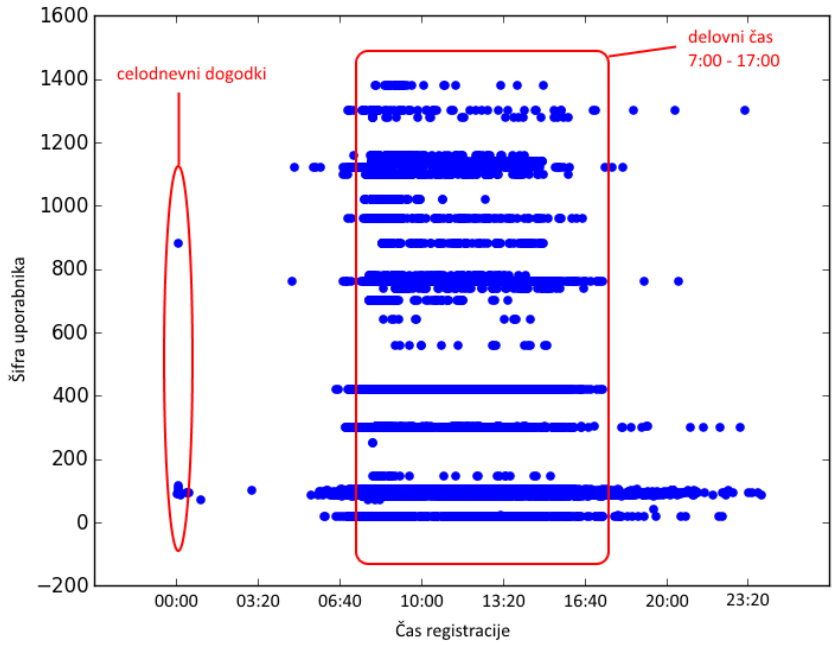
Pripravili smo prikaz porazdelitve vseh registracij za posameznega uporabnika. Podatke smo izrisali na razpršenem grafu. Za abscisno os smo uporabili čas v dnevu, ko se je registracija zgodila, za ordinatno os pa smo izbrali tip dogodka. Na podlagi rezultatov, ki so prikazani na sliki 3.1, vidimo, da se registracije porazdeljujeje znotraj delovnega časa uporabnika.



Slika 3.1: Porazdelitev registracij za uporabnika 148.

3.1.2 Izris porazdelitve registracij za posamezni dogodek

Na razpršenem grafu smo izrisali porazdelitev vseh registracij za posamezni tip dogodka. Na podlagi rezultatov prikazanih na sliki 3.2, vidimo, da se dogodki istega tipa najpogosteje zgodijo znotraj istega območja v dnevu. Tako bi lahko to porazdelitev uporabili za odkrivanje kršitev, ko se dogodki zgodijo zunaj delovnega časa, kar pa bi v določenih primerih (dežurstvo) privedlo do velikega števila napačno odkritih kršitev, zato se za to v naši rešitvi nismo odločili.



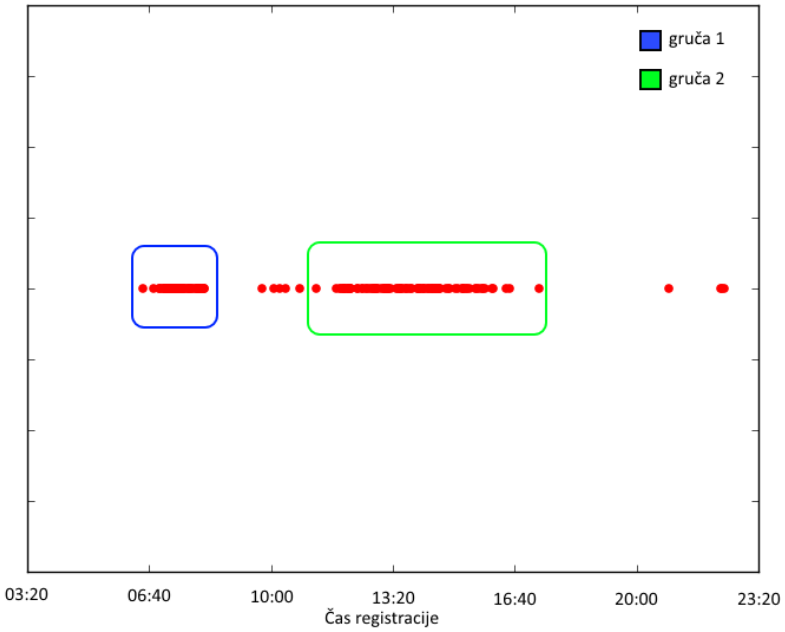
Slika 3.2: Porazdelitev registracij za dogodek Prihod.

3.1.3 Razdeljen izris porazdelitve registracij za posamezni dogodek

Za vsakega uporabnika smo na razpršenem grafu izrisali porazdelitev registracij za posameznega uporabnika. Tako smo dobili enodimenzionalno porazdelitev registracij. Na sliki 3.3 lahko vidimo, da se registracije združujejo v gruče, nekaj pa je tudi registracij, ki ne spadajo v nobeno izmed gruč. Odločili smo se, da bi lahko te registracije izločili z metodo iskanja osamelcev, ki je podrobneje opisana v poglavju 3.3.

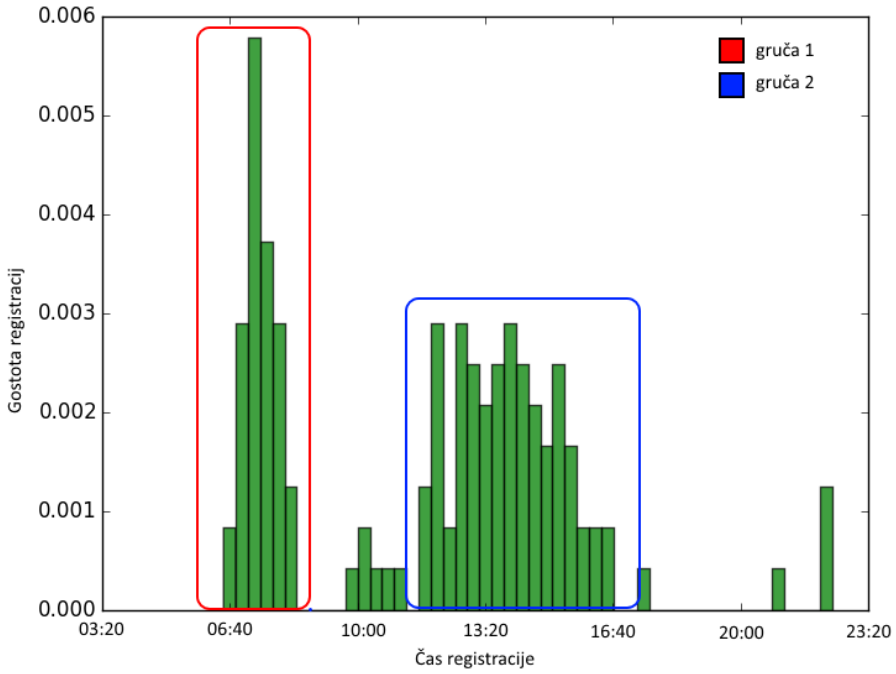
3.1.4 Izris histograma registracij

Izris porazdelitve registracij za posameznega uporabnika in tip dogodka z uporabo razpršenega grafa nam je sicer razkril dejstvo, da se registracije združujejo v gruče, ni pa na njem mogoče videti gostote te porazdelitve. Go-



Slika 3.3: Porazdelitev registracij za dogodek Prihod pri uporabniku 148.

stoto bi lahko prikazali z uporabo različno velikih pik na razpršenem grafu, vendar je takšen prikaz gostote še vedno slabši kot prikaz gostote s pomočjo histograma. Izrisali smo uravnotežen histogram porazdelitve registracij, razdeljen na 50 stolpcev. Iz grafa 3.4 lepo vidimo, da porazdelitev registracij znotraj posameznih gruč močno spominja na Gaussovo porazdelitev. Za združevanje podatkov v skupine, smo uporabili metodo KDE (kernel density estimation), ki nam iz podatkov vrne funkcijo gostote, na kateri lahko potem poiščemo minimume in maksimume. Te vrednosti v nadaljevanju uporabimo pri razdeljevanju registracij v gruče. Kot alternativo metodi KDE bi lahko uporabili algoritem k-NN (k-Nearest Neighbors) ali k-means. Za uporabo metode KDE smo se odločili, ker nam pri njej ni potrebno poznati števila gruč, ki se za vsako kombinacijo uporabnika in tipa registracije spreminja.



Slika 3.4: Normaliziran histrogram registracij za dogodek Prihod pri uporabniku 148.

3.2 Združevanje podatkov v skupine

3.2.1 KDE

KDE je neparametrična funkcija naključne spremenljivke. Uporabljamo jo za glajenje podatkov, kjer na podlagi končne podmnožice podatkov opravimo predvidevanja o obliki podatkov. Izračun za KDE je opisan v enačbi 3.1.

$$f_h(x) = \frac{1}{n}\sum_{k=1}^{n} K_h(x_i - x) \tag{3.1}$$

Spremenljivka K predstavlja jedro porazdelitve. Pri naši rešitvi smo uporabili Gaussovo porazdelitev, zato za izračun vrednosti K uporabimo enačbo 3.2.

$$K(h) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}h^2} \tag{3.2}$$

Pri enačbi za KDE je še posebej pomemben parameter h, ki predstavlja stopnjo glajenja naše funkcije. Če za parameter h izberemo premajhno vrednost, bo rezultat imel veliko šuma. V nasprotnem primeru, pa bo prevelika stopnja glajenja lahko zameglila mejo med dvema gručama, ki se bosta potem zlili skupaj. Na podlagi preizkušanja se je za našo rešitev najbolje izkazala vrednost parametra h=50. Vrednost parametra smo dobili s pomočjo metode prečnega preverjanja (Cross-validation), uporabili smo 20-kratno prečno preverjanje. Za vsako kombinacijo uporabnika in tipa registracije, ki ima več kot 100 vnešenih registracij, smo z metodo prečnega preverjanja izračunali optimalno vrednost parametra h. Vrednosti parametra so se za vsako kombinacijo uporabnika in dogodka razlikovale in so se nahajale v intervalu med 23 in 72. Odločili smo se vzeti približek srednje vrednosti parametra (h=50), ki se je v praksi izkazala za dovolj dobro. Rezultate funkcij za različne parametre h (5,50,200) imamo prikazane na sliki 3.5.

3.2.2 Razvrščanje registracij v gruče

Ko smo poiskali minimume in maksimume v funkciji KDE za množico registracij, jih lahko porazdelimo v gruče. Za porazdeljevanje v gruče smo kot primerne izbrali samo kombinacije uporabnikov in tipov dogodkov, ki imajo v učnih podatkih 100 ali več zapisov. Za to omejitev smo se odločili, ker so podatki v nasprotnem primeru preveč razpršeni znotraj delovnega časa in ni nujno, da se že oblikujejo v gruče. Razvrščanje naredimo tako, da v določeno gručo dodelimo vse registracije med dvema minimuma, ki ju vzamemo kot meje posamezne gruče. Za vrednost minimumov uporabimo tudi začetek in konec časovnega območja (začetek in konec dneva). Primer metode za razvrščanje registracij v gruče je podrobneje opisan v primeru kode 3.1.

Primer kode 3.1: Metoda, ki podatke razdeli v gruče.

```
def SplitDataIntoClusters(clockings, minimums, maximums):
    clusters = []
    for i in range(0, len(maximums)):
```

```
        clusters.append([])
    for clocking in clockings:
        date = clocking[2]
        timestamp = date.hour*60+date.minute
        for i in range(0, len(maximums)):
            left_limit = 0
            right_limit = 1440
            if i > 0:
                left_limit = minimums[i-1]
            if i < len(minimums):
                right_limit = minimums[i]
            if timestamp > left_limit and timestamp < right_limit:
                clusters[i].append(timestamp)
                break
    return clusters
```

3.3 Odkrivanje osamelcev

Osamelce v podatkih odkrivamo s pomočjo Z-ocene, ki jo izračunamo za vsak element posamezne gruče registracij. Če je absolutna vrednost izračunane Z-ocene večja od vnaprej definirane meje, je ta registracija osamelec. Za izračun Z-ocene je prej treba izračunati še vrednosti mediane in absolutnega odklona mediane za gručo, v katero spada posamezna registracija.

3.3.1 Mediana

Mediana predstavlja srednjo vrednost podatkov. Dobimo jo na način, da podatke uredimo po vrstnem redu in iz njih vzamemo vrednost, ki je točno na sredini. V primeru, da je število podatkov sodo in nimamo vrednosti, ki je točno na sredini, vzamemo kot mediano aritmetično sredino dveh vrednosti, ki sta najbližje sredini. V naši rešitvi smo mediano posamezne gruče izračunali s pomočjo metode 3.2.

Primer kode 3.2: Metoda za izračun mediane.

```
def CalculateMedianForData(data):
   data = sorted(data)
   size = len(data)
   if size % 2 == 0:
      median = float(data[size/2]+data[(size/2)-1])/2
   else:
      median = float(data[size/2])
   return median
```

3.3.2 MAD

MAD (median absolute deviation) je vrednost, ki predstavlja mediano absolutnih vrednosti odstopanj vseh vrednosti v množici od mediane te množice. Izračunamo jo tako, da najprej izračunamo mediano za podatke, nato pa za vsak podatek izračunamo absolutno odstopanje od izračunane mediane. Podatke podobno kot pri izračunu mediane uredimo po velikosti. Mediana te nove množice predstavlja vrednost MAD, ki jo bomo uporabili pri odkrivanju osamelcev. V naši rešitvi smo vrednost MAD izračunali z metodo 3.3

Primer kode 3.3: Metoda za izračun vrednosti MAD.

```
def CalculateMADForData(data):
   adjusted = []
   median = CalculateMedianForData(data)
   for clocking in data:
      adjusted.append(abs(clocking-median))
   mad = CalculateMedianForData(adjusted)
   return mad
```

3.3.3 Z-ocena

Pri odkrivanju osamelcev nam je v pomoč Z-ocena. Izračunamo jo za vsako gručo posebej. Z-ocene ne moramo izračunati za podatke, kjer je MAD vrednost 0, saj bi v nasprotnem v enačbi 3.3 imeli deljenje z 0. V naši rešitvi smatramo za osamelce vse registracije, pri katerih je vrednost Z-ocene večja od 3,5. V kolikor bi se za posamezne podatke izkazalo, da takšna meja vrne preveč osamelcev, jo lahko poljubno povečamo in tako zmanjšamo število registracij, ki bodo zaznane kot osamelci. Pri testnih podatkih se je prej omenjena vrednost izkazala za dobro, saj smo tako dobili obvladljivo število osamelcev, ki smo jih nato uporabili pri reviziji predlogov kršitev, ki je opisana v poglavju 4.

$$Z(x) = \frac{0.6745 * (x - MEDIAN)}{MAD} \tag{3.3}$$

3.4 Pospešitev hitrosti programa z uporabo večnitnosti

Ker je treba izračunati vse kombinacije uporabnikov iz množice U in dogodkov iz množice D, moramo naš algoritem izvesti natanko U * D krat. V primeru testnih podatkov smo morali algoritem izvesti približno 10000-krat, za kar je testni računalnik porabil 70 minut. Izvajanje programa smo najprej pospešili tako, da smo pri računanju izpustili vse kombinacije uporabnikov in dogodkov, ki so imele manj kot 100 registracij. Tako smo hitrost izvajanja programa uspeli znižati na 16,63 minute. Ob samem izvajanju smo opazili, da je kljub dolgotrajnemu računanju procesor na računalniku zelo slabo izkoriščen, zato smo se zaradi narave izračuna odločiti naš algoritem paralelizirati. Za paralelizacijo računanja smo uporabili knjižnico za programski jezik Python z imenom threading. Že ob prvih poskusih se je nadgrajena rešitev izkazala za boljšo, saj se je čas izvajanja algoritma precej zmanjšal. Naredili smo teste z različnim številom niti in tako prišli do optimalne rešitve, kjer se algoritem istočasno izvaja na štirih nitih. Rezultate testiranja za različna

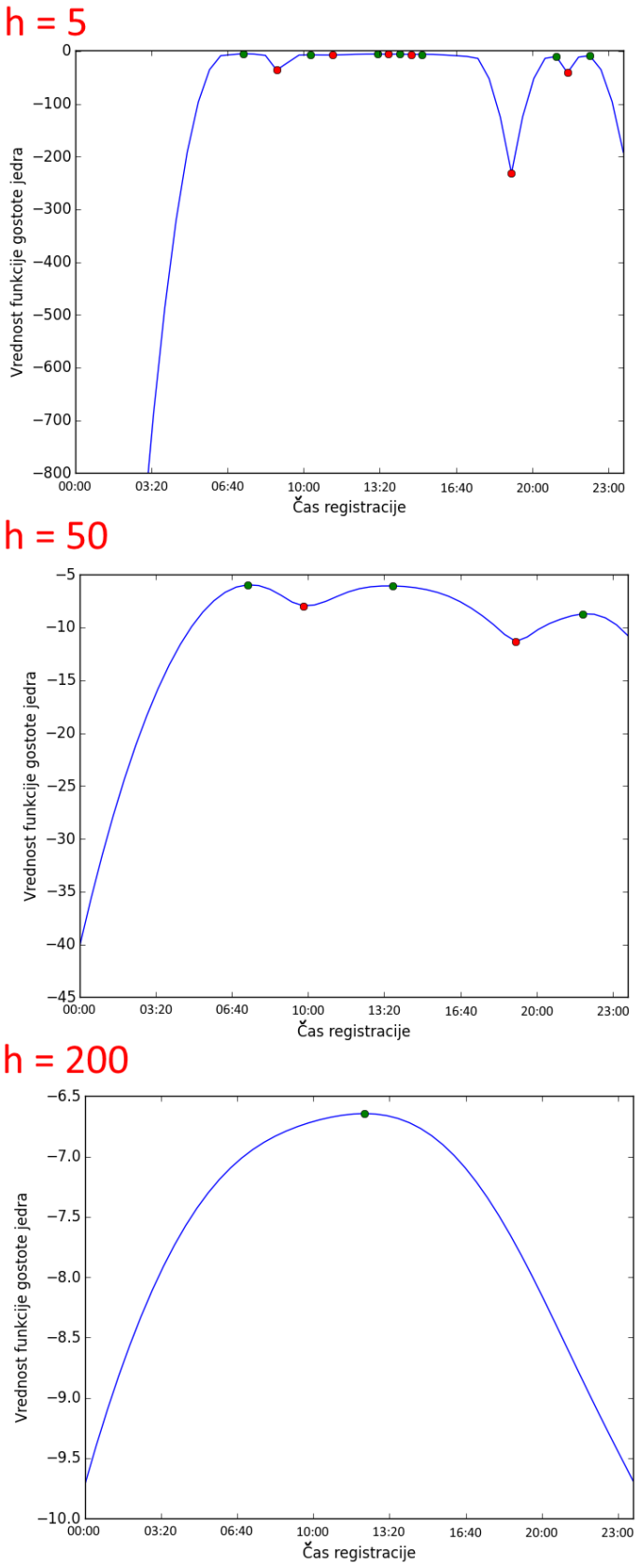
števila niti lahko vidimo v tabeli 3.2. Testiranje smo izvedli na prenosnem računalniku z naslednjo konfiguracijo strojne opreme, ki je opisana v tabeli 3.1.

Tip komponente	**Model**	**Karakteristika**
Procesor	Intel i7-4702MQ	4 jedra, 8 niti, frekvenca 2.2 GHz
Sistemski pomnilnik	Kingston DDR3L	kapaciteta 16 GB, frekvenca 800 MHz
Zunanji pomnilnik	Intel SSD 530	kapaciteta 240GB, hitrost pisanja 400 MB/s, hitrost branja 500 MB/s, število vhodno-izhodnih operacij na sekundo 80000

Tabela 3.1: Konfiguracija strojne opreme na testnem računalniku.

Število niti	**Povprečni čas izvajanja programa v minutah**
1	16,63
2	9,51
4	7,19
8	7,86
16	8,25
32	9,23

Tabela 3.2: Čas izvajanja programa na različnih številih niti.



Slika 3.5: Primerjava rezultatov funkcije za različne vrednosti parametra h.

Poglavje 4

Revizija kršitev in izvoz v sistem Time&Space

Za revizijo predlogov kršitev smo izdelali spletno aplikacijo v programskem ogrodju Django, kjer lahko izdelujemo spletne aplikacije z uporabo programskega jezika Python. Ker je preostali del aplikacije napisan v tem programskem jeziku, je iz stališča vzdrževanja rešitve vzdrževanje rešitve najlažje, če je napisana samo v enem programskem jeziku. Predpostavili smo, da je dostop do aplikacije za revizijo kršitev omejen na nivoju sistema, zato za uporabo aplikacije ni potrebna avtorizacija uporabnika. Spletna aplikacija je narejena v načinu izvedbe RWD (responsive web design), kar pomeni, da se prikaz same aplikacije prilagaja mediju, na katerem se aplikacija prikazuje. Tako uporaba aplikacije ni mogoča samo na namiznih in prenosnih računalnikih, temveč jo je mogoče uporabljati tudi na tabličnih računalnikih ali telefonih. Uporabniki aplikacije so ljudje, ki so v podjetju na vodilnih položajih. To je lahko direktor, v primeru uporabe pri delavcih v industriji pa je to lahko vodja izmene.

4.1 Pregled kršitev

Glavni pogled spletne aplikacije predstavlja pregled kršitev, na katerem se nam izpišejo vsi predlogi kršitev, ki jih je naša rešitev odkrila in vnesla v podatkovno bazo. Na pregledu so za vsak predlog kršitve prikazani naslednji podatki:

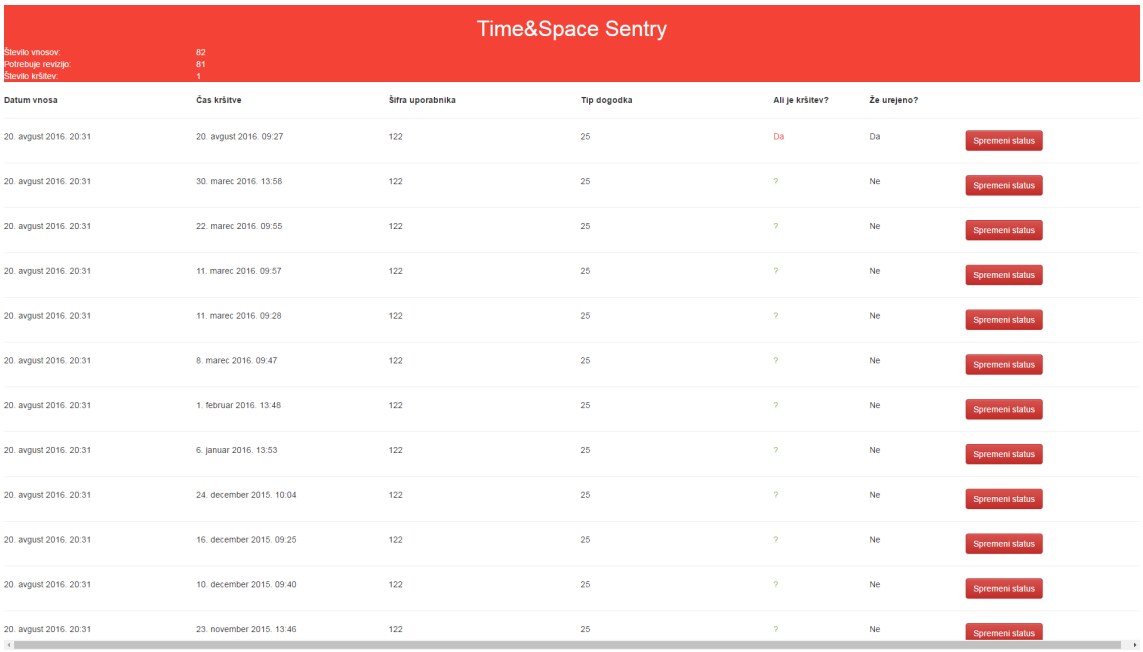
- datum vnosa v podatkovno bazo,
- datum dogodka, ki je predlagan kot kršitev,
- šifra uporabnika,
- tip dogodka,
- trenutni status predloga,
- ali je bil predlog že reviziran.

Nad izpisom seznama predlogov kršitev so na pregledu tudi števci, ki nam prikazujejo statistiko trenutnega stanja v aplikaciji. Te števce lahko vidimo na sliki 4.1, ki prikazuje videz pregleda predlogov kršitev na osebnem računalniku. Prikazani so podatki o:

- skupnem številu predlogov kršitev,
- številu predlogov kršitev, ki še niso bili revizirani,
- številu predlogov, ki so bili označeni kot dejanske kršitve.

4.2 Revizija kršitve

Ob izbiri kršitve, se odpre nov pogled, ki ponuja možnost revizije predloga o kršitvi. Uporabniku je tako na voljo, da označi, ali gre pri vnosu za dejansko kršitev ali ne, in ob tem vpiše revizijski komentar. Odločitev o stanju posameznega predloga ni končna, saj lahko uporabnik svojo odločitev kadar koli spremeni. Ravno tako lahko spremeni tudi revizijski komentar. Revizija kršitve na mobilni napravi je prikazana na sliki 4.2.



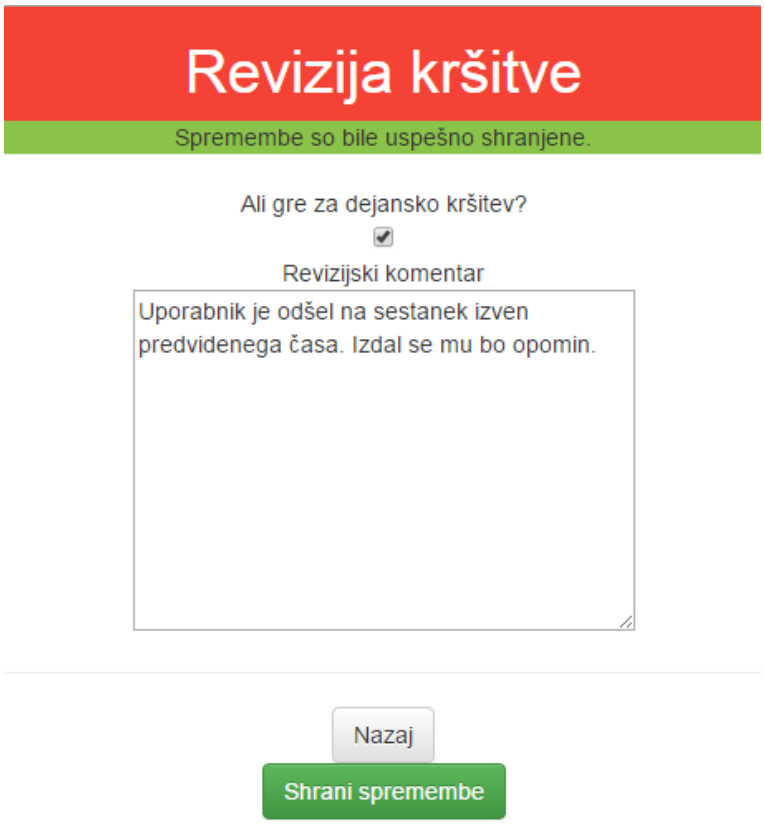
Slika 4.1: Pregled predlogov o kršitvah.

4.3 Izvoz kršitev v Time&Space

Predlogi kršitev, ki jih uporabnik spletne aplikacije označi kot dejanske kršitve, v naši rešitvi ne predstavljajo posebne dodane vrednosti, zato jih je smiselno izvoziti nazaj v sistem Time&Space. V sistemu je treba definirati nov dogodek, ki bo predstavljal kršitve, ki jih bomo izvozili iz naše rešitve. Tip novega dogodka smo definirali kot registracijo, nastavili pa smo mu tudi parameter nevtralnost dogodka, kar pomeni, da nima vpliva na saldo zaposlenega.

4.4 Datoteka CSV

Najbolj osnoven način izvoza kršitev predstavlja izvoz v obliki datoteke CSV. To datoteko lahko nato uvozimo preko bazne procedure, ki jo poženemo na strežniku T&S. Za potrebe testiranja rešitve smo izdelali bazno proceduro, ki deluje na podatkovni bazi Oracle. V primeru, da bi rešitev uporabili pri



Slika 4.2: Revizija kršitve na mobilni napravi.

podjetju, ki uporablja drugačno podatkovno bazo, bi proceduro za uvoz bilo potrebno prilagoditi.

4.5 TimeAPI

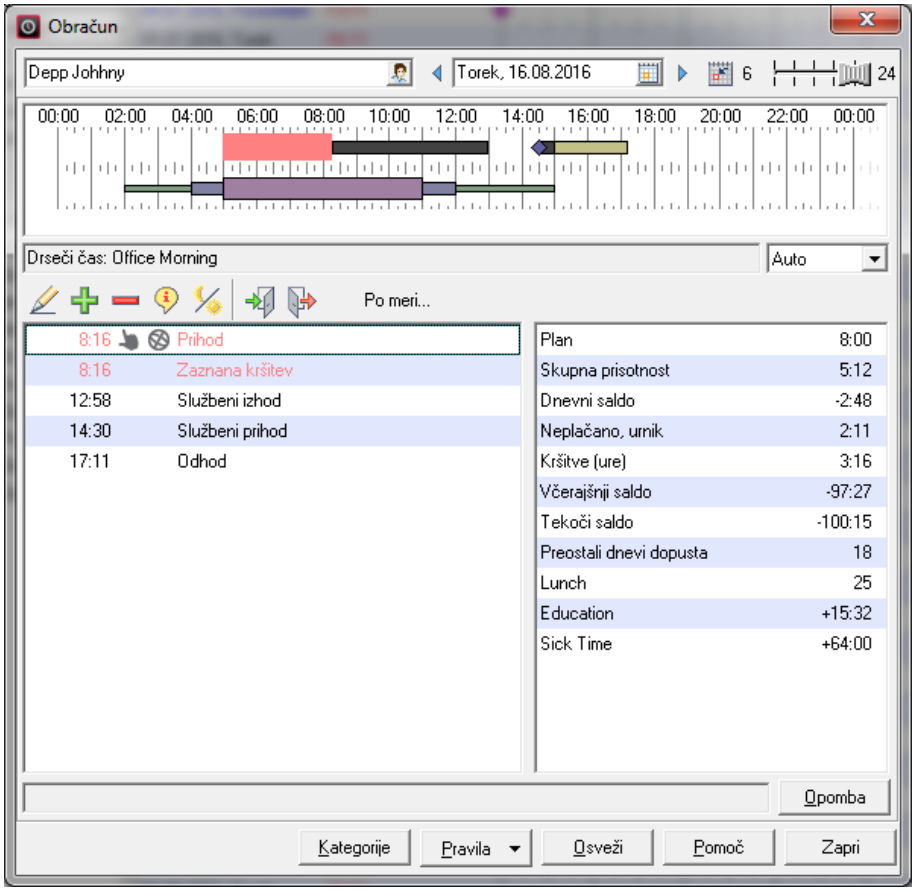
TimeAPI je REST (representational state transfer) API (application programming interface), ki ga sistem T&S uporablja za manipulacijo z vsemi

podatki, ki se tičejo delovnega časa. Programski vmesnik je napisan v jeziku C# in je na voljo kot razširitev osnovnega sistema T&S. Programski vmesnik ponuja operacije CRUD (create, read, update and delete) za naslednje entitete sistema:

- **Zaposleni**
- **Prazniki**
- **Registracije**
- **Uporabniški računi**
- **Organizacijske enote**
- **Urniki**
- **Omejitve urnikov**

4.5.1 Vnos dogodkov

Če želimo komunicirati s sistemom T&S, moramo za to imeti avtorizacijski ključ, ki ga ob vsaki zahtevi navedemo v glavi naše zahteve. Ta avtorizacijski ključ nam mora priskrbeti administrator sistema, saj ga je mogoče določiti le v glavni podatkovni bazi sistema T&S. Ravno tako je treba v sistemu T&S ustvariti nov dogodek, ki bo predstavljal odkrito kršitev. Ker so vnesene kršitve zgolj informativne narave, mora biti nov dogodek nevtralen, kar pomeni, da ne vpliva na uporabnikov saldo. Vnos opravimo preko klica POST na strežnik, kjer teče TimeAPI. Klicu metode moramo poleg avtorizacijskega ključa, ki ga vstavimo v glavo zahteve, dodati tudi parametre v formatu JSON, ki mora vsebovati vse podatke, ki so potrebni za vnos nove registracije. Struktura teh podatkov je podrobno opisana v uporabniških navodilih vmesnika TimeAPI, za vpogled v katera pa je potrebno pridobiti dovoljenje podjetja Špica International d. o. o.



Slika 4.3: Prikaz vnesene kršitve v programu Time&Space Manager.

Poglavje 5

Analiza rešitve

Dejansko uporabnost razvite rešitve smo analizirali na testnih podatkih. Podatki pripadajo srednje velikemu podjetju, v katerem se opravljajo pretežno pisarniška dela. Zaposleni imajo drseče urnike, kar pomeni, da lahko na delo prihajajo poljubno. To privede do velike razpršenosti registracij, kar posledično privede do povečanja števila osamelcev. Podjetje nima podatkov o tem, katere izmed registracij, ki smo jih uporabili, so dejanske kršitve, zato je bilo nemogoče določiti točnost naše rešitve pri odkrivanju kršitev. Da bi dobili te podatke, bi moral nekdo od vodilnih v podjetju v določenem časovnem obdobju, npr. 1 mesec, za vse registracije v podjetju voditi evidenco, ali gre pri registraciji za kršitev ali ne. V primeru podjetja, ki ima 100 zaposlenih, to v povprečju nanese 400 registracij dnevno (prihod v službo, odhod na malico, prihod z malic, odhod domov), ki bi jih moral nekdo ročno preveriti. Za beleženje te statistike bi se morala s tem v podjetju ukvarjati ena oseba za poln delovni čas, takšnega vložka pa nobeno podjetje ni pripravljeno investirati. V primeru testnih podatkov smo za 128326 registracij, ki smo jih uporabili pri izračunih, odkrili 5100 osamelcev, kar predstavlja 3,98 % vseh registracij. Če ta podatek uporabimo pri zgoraj omenjenih 400 dnevnih registracijah v srednje velikem podjetju, potem bi to dnevno naneslo 15,92 osamelca, ki bi jih nekdo v podjetju moral pregledati. Ta številka je precej bolj obvladljiva. Uvedba rešitve v dejanskem podjetju, bi na zaposlene imela

tudi psihološki vpliv, saj bi ob vedenju, da podjetje uporablja sistem za odkrivanje kršitev, svoje vedenje spremenili. S tem sem se že srečal pri podjetju, ki za svoje zaposlene aktivno vodi statistiko uspešnosti na terenu. Po začetku uporabe sistema, ki zaposlene kaznuje za nekorektno porabljen delovni čas, se je število odvečnih ur, ki jih podjetje izplača zaposlenim zmanjšalo, kar pa ni posledica odbitkov ur, ki jih na podlagi statistike opravi sistem zaradi nekorektno porabljenega časa, temveč zaradi bolj korektnega odnosa delavcev, ki je posledica strahu pred kaznovanjem.

5.1 Test v podjetju

Delovanje rešitve smo preverili na ažurnih podatkih testnega podjetja. Za test smo uporabili zadnjih 400 registracij v sistemu, ki so se zgodile v časovnem območju 30 dni. Rešitev je v podatkih odkrila 17 osamelcev, kar predstavlja 4,25% vseh registracij in ne odstopa preveč od deleža osamelcev, ki smo jih odkrili na večji množici registracij. Med kršitelji je bilo 11 zaposlenih od skupno 91, ki smo jih zajeli v testnih podatkih. Opazili smo, da se med predlogi kršitev največkrat pojavlja dogodek, ki je enostavna registracija delovnega časa. To pomeni, da nimamo podatka o tem, ali je zaposleni prišel v službo ali pa jo zapustil, saj ta podatek sistem naknadno izračuna glede na zgodovino registracij ter poslovno logiko. Ta tip dogodka se je izkazal za problematičnega, zato smo se ga odločili izpustiti. Pričeli smo nov test, kjer smo uporabili zadnjih 400 registracij, ki niso vsebovale problematičnega tipa dogodka. Tokrat je naša rešitev odkrila 10 osamelcev, kar predstavlja 2,5% vseh registracij. Tokrat je bilo med kršitelji 9 zaposlenih od skupno 70, ki smo jih zajeli v testnih podatkih. Zgoščeni rezultati obeh testiranj se nahajajo v tabeli 5.1. Izkazalo se je, da so rezultati testiranja, kjer nismo uporabili enostavnih registracij bolj pregledni. V podjetju so preučili predloge kršitev, ki jih je podala naša aplikacija in o njih podali mnenje, da sicer res gre za registracije, ki odstopajo od normalnega obnašanja zaposlenih (nadure, zdravniški pregled, ...) vendar pri nobenem izmed predlogov ni

šlo za dejansko kršitev pravil podjetja. Podjetje tudi drugače nima težav z kršitelji, saj so pravila delovnega časa precej ohlapno zastavljena ter temeljijo na pravilu poštenosti. Izkazalo se je, da rešitev nima velike dodane vrednosti za podjetje, kjer imajo zaposleni drseč delovni čas in ki nima strogih pravil glede delovnega časa. V sklopu izdelave diplomske naloge nismo uspeli pridobiti k sodelovanju nobenega podjetja, ki se sooča z rednimi kršitvami delovnega časa.

Vrsta testa	**Št. predlogov kršitev**	**Št. kršiteljev**	**Št. dejanskih kršitev**
Vsi dogodki	17	11	0
Dogodki brez enostavne registracije	10	9	0

Tabela 5.1: Rezultati testiranja.

Poglavje 6

Sklepne ugotovitve

Pri izdelavi rešitve sem poskušal izdelati način, ki bi omogočal odkrivanje kršitev v sistemu T&S. Cilj rešitve je bilo razviti sistem, ki bo neodvisen od obstoječega sistema in ne bo posegal v njegovo delovanje. Za samo izdelavo rešitve je bilo ključno poznavanje delovanja sistema T&S, s katerim sem se srečal že med delovno prakso, ki sem jo opravljal pri podjetju Špica International d. o. o. Rešitev sem izdelal v programskem jeziku Python, saj je podprt na vseh večjih platformah, kar izvajanje rešitve ne omejuje na operacijski sistem Windows, na katerega je omejen sam sistem Time&Space. Prednost izbranega programskega jezika je tudi velik nabor knjižnic za analizo podatkov, ki jih lahko vključimo v svoj program. Tako si znatno zmanjšamo čas razvoja, saj nam ni treba samim implementirati vseh algoritmov, ki jih potrebujemo. Slabost programskega jezika je predvsem počasnost izvajanja, ki ni primerljiva s hitrostjo jezikov, ki se prevajajo v strojno kodo. Ravno tako pa se programi izvajajo samo na enem jedru, kar pripelje do slabe izkoriščenosti modernih procesorjev, ki ponujajo veliko število fizičnih jeder, vsako jedro pa lahko istočasno izvaja po dve niti. Rešitev sem zato pohitril z uporabo večnitnosti, kar je privedlo do 100% izkoriščenosti procesorja. Presenetljivo pa se je rešitev hitreje izvajala na številu niti, ki je bilo enako številom fizičnih jeder procesorja in ne številu niti, ki jih lahko procesor izvaja istočasno.

Naslednji korak po spoznavanju z delovanjem sistema je predstavljal izvoz

podatkov o registracijah, ki sem jih potem uporabljal v svoji rešitvi. Četudi bi bil dostop do podatkov mogoč preko spletnega vmesnika TimeAPI, pa je ta vmesnik precej neučinkovit za prenos velikega števila podatkov. Odločil sem se za izvoz preko izmenjevalne datoteke v formatu CSV, v katero sem shranil rezultat poizvedbe, ki sem jo izvedel na tabeli, kjer se shranjujejo registracije. Posebno pazljivost sem namenil temu, da v izvoženih podatkih ni bilo nobenih osebnih podatkov.

Ker si je zgolj iz števil precej težko ustvariti sliko o problemu in priti do ideje o njegovi rešitvi, sem podatke o registracijah prikazal na razpršenih grafih. Ljudje smo presenetljivo dobri pri prepoznavanju vzorcev, kar me je privedlo do spoznanja, da se registracije združujejo v gruče, opaziti pa je bilo tudi dejstvo, da obstajajo nekatere registracije, ki na videz ne spadajo v nobeno izmed gruč. Odločil sem se, da bom kršitve v svoji rešitvi enačil z osamelci v podatkih. Čeprav se za gručenje najpogosteje uporablja metoda KNN, pa je šlo pri moji rešitvi za podatke, ki so imeli samo eno dimenzijo, zato ta algoritem ni bil najbolj učinkovit. Namesto njega sem uporabil metodo KDE, s katero se poprej nisem nikoli srečal. Menim, da bom to metodo tekom svoje karierne poti še večkrat uporabil, ko se bom srečal s podatki, ki bodo imeli samo eno dimenzijo. Pri izdelavi rešitve sem za vse uporabnike uporabil isto stopnjo glajenja funkcije KDE. Rešitev bi lahko dodelali tako, da bi za vsakega izmed zaposlenih uporabili različno stopnjo glajenja. Ravno tako bi lahko vrednost Z-ocene, ki se uporablja za določanje osamelcev, prilagodili posamezniku. Tako bi dobili manj osamelcev, kar bi sistem naredilo preglednejši. Bi pa bilo treba tudi v praksi preizkusiti, ali bi zmanjšano število osamelcev morda privedlo do izpuščenih kršitev.

Ker osamelci sami po sebi ne zagotavljajo, da gre za dejansko kršitev, služijo v izdelani rešitvi le kot predlogi možnih kršitev, ki jih uporabnik spletne aplikacije za revizijo kršitev nato po svoji nadaljni presoji in morebitnem razgovoru s kršiteljem označi kot kršitve. Predlogi, ki so označeni kot dejanske kršitve se lahko izvozijo nazaj v sistem T&S, kjer se potem obravnavajo kot klasični T&S dogodki, kar pomeni, da jih je mogoče izvažati v poročila

preko vgrajene funkcionalosti in tudi recimo upoštevati pri obračunu plač.

Izdelana rešitev predstavlja osnovni primer odkrivanja kršitev v podatkih o registracijah, ki jo bo v prihodnosti mogoče razširiti z vpeljavo dodatnih dimenzij podatkov. V primeru dovolj velikega interesa je rešitev mogoče tudi prepisati v obliko, v kateri bo potem na voljo kot komponenta samega sistema T&S.

Literatura

[1] Kaj je Responsive Web Design(RWD) oz. odziven spletni design. [Online]. Dosegljivo: https://www.pomagalnik.com/izobrazevanje/slovar/kaj-je-responsive-web-designrwd-oz-odziven-spletni-design/. [Dostopano 1. 9. 2016].

[2] Density Estimation. [Online]. Dosegljivo: http://scikit-learn.org/stable/modules/density.html. [Dostopano 14. 8. 2016].

[3] J. Palach. *Parallel Programming with Python*. Packt Publishing, 2014.

[4] A. Žiberna. *Osnovna statistična analiza v R-ju [Elektronski vir]*. Založba FDV, 2016.

[5] The Basics of Python Multithreading and Queues. [Online]. Dosegljivo: http://www.troyfawkes.com/learn-python-multithreading-queues-basics/. [Dostopano 14. 8. 2016].

[6] Median obsolute deviation. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Median_absolute_deviation. [Dostopano 7. 8. 2016].

[7] Pyplot. [Online]. Dosegljivo: http://matplotlib.org/users/pyplot_tutorial.html. [Dostopano 20. 7. 2016].

[8] NIST/SEMATECH e-Handbook of Statistical Methods. [Online]. Dosegljivo:
http://www.itl.nist.gov/div898/handbook/. [Dostopano 2. 8. 2016].