Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Peter Povirk

Aplikacija za beleženje časa

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Beleženje in evidentiranje porabljenega časa za dnevne delovne aktivnosti je pomembno za podjetja več gospodarskih branž.

Analizirajte nekaj obstoječih aplikacij za evidentiranje porabljenega časa. Na podlagi ugotovitev analize zasnujte in razvijte spletno aplikacijo za evidentiranje časa. Pri izbiri tehnologij za razvoj upoštevajte, da naj aplikacija ne deluje le v brskalnikih računalnikov, temveč naj deluje tudi na čim več različnih mobilnih napravah.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Peter Povirk, z vpisno številko **63050088**, sem avtor diplomskega dela z naslovom:

Aplikacija za beleženje časa

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika,
- 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 na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 8. septembra 2014 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

API (angl. *Application Programming Interface*) - Programski vmesnik, ki se uporablja za povezovanje različnih programov

CSS (angl. *Cascading Style Sheets*) - Kaskadne stilske podloge, preprost mehanizem za dodajanje stilov, barv, ozadij spletnih stranem

CRUD (angl. *Create Read Update Delete*) - Ustvari, poglej, posodobi in izbriši so štiri tipične operacije, ki jih izvajamo na elementih

HTML (angl. *Hypertext Markup Language*) - označevalni jezik, ki ga brskalnik prikaže kot spletno stran

IaaS (angl. *Infrastructure as a Service*) - infrastruktura kot storitev, najem fizične in virtualne računalniške opreme

JSON (angl. *JavaScript Object Notation*) - Zapis (format) objekta v jeziku JavaScript

MVC (angl. *Model View Controller*) - arhitektura Model Pogled Krmilnik, ki se uporablja za boljšo ločenost med podatki v bazo, programsko kodo in HTML kodo

PaaS (angl. *Platform as a Service*) - platforma kot storitev, najem fizične opreme s prednaloženo osnovno programsko opremo

REST (angl. *Representational state transfer*) - predstavitveni prenos stanja, enostaven protokol za izmenjavo podatkov med odjemalcem in strežnikom

SaaS (angl. *Software as a Service*) - programska oprema kot storitev, najem aplikacij in pogosto uporaba le-teh preko svetovnega spleta

SOAP (angl. *Simple Object Access Protocol*) - manj enostaven protokol za izmenjavo podatkov med spletnimi storitvami

SQL (angl. *Structured Query Language*) - strukturiran povpraševalni jezik, ki se uporablja v relacijskih bazah podatkov

Povzetek

Namen diplomskega dela je predstaviti izdelavo spletne aplikacije za beleženje časa. Beleženje časa nam omogoča spremljanje finančnega stanja naših projektov, izkoriščenosti posameznih kadrov in optimizacijo poslovanja. Še posebej pomembno je za samostojne podjetnike in manjše ekipe, ki nimajo sistema za evidenco delovnega časa z uporabo kartic. Spletna aplikacija je zaradi odzivnega oblikovanja na voljo na vseh napravah, ki imajo dostop do svetovnega spleta od pametnih telefonov do osebnih računalnikov. Uporabniki lahko beležijo svoj čas z uporabo štoparice ali pa svoje časovne vnose vnesejo za nazaj. Aplikacija omogoča pregled časovnih vnosov z uporabo različnih filtrov (po projektu, po stranki) in izvoz podatkov v različnih formatih.

Ključne besede: beleženje časa, spletna aplikacija, splet, www, symfony2, php, mysql, saas.

Abstract

We built a time tracking web application and documented every step of it in this thesis. Time tracking is of great importance to companies as it offers them insights into financial status of each project, human resources usage and enables them to optimize their operations. Freelancers and small teams would benefit most from our application because they usually don't have time tracking terminals in their offices. Our application can be used on virtually all web capable devices such as smart phones, tablets and PCs due to responsive design. Users can track their time using our stopwatch in real-time or manually enter duration for past entries. Our application offers export in PDF and XLS formats as well as custom filters (by project, by client).

Keywords: time tracking, time logging, web application, web, www, symfony2, php, mysql, saas.

Poglavje 1

Uvod

V diplomski nalogi bomo v Poglavju 1 predstavili probleme z beleženjem časa v manjših kolektivih, v Poglavju 2 si bomo pogledali, kakšne tehnologije so nam na voljo za rešitev problema ob razvoju računalništva v oblaku in kako lahko uporabimo spletne tehnologije za izdelavo aplikacij, ki so dostopne na različnih napravah.

Poglavje 3 je namenjeno opisu glavnih funkcij rešitve s stališča uporabnika, medtem ko Poglavje 4 razkriva tehnične podrobnosti rešitve in tehnologije, ki smo jih uporabili. Posebno pozornost smo posvetili avtomatskemu testiranju programske opreme, ki se prevečkrat zanemarja - več v Poglavju 5.

Zaključimo s Poglavjem 6, kjer predstavimo možne izboljšave oziroma nadaljnje možnosti za razvoj naše spletne aplikacije.

1.1 Opis problema

Beleženje časa (angl. *time tracking*) je pomembno povsod, kjer se tekom delovnega dneva srečujemo z različnimi projekti, ki jih opravljamo za različne stranke. Na ta način lahko spremljamo zaposlene (za potrebe obračuna osebnih dohodkov in izkoriščenosti) in finančno uspešnost projektov (so naše cene primerne?), kar je še posebej pomembno za vodje projektov.

V večjih podjetjih se za beleženje časa uporablja draga programska oprema, ki si je samostojni podjetniki in manjše ekipe ponavadi ne morejo privoščiti. Večja podjetja uporabljajo tudi registratorje časa, kjer zaposleni ob prihodu in odhodu

ob registrator prislonijo kartico. Ta naprava sicer beleži skupni delovni čas zaposlenega, ne zna pa ga razdeliti med projekte.

Manjša podjetja se tako zatečejo k uporabi preglednic ali pa še huje - evidence vodijo v papirnati obliki, kar je dober recept za neorganizirano poslovanje, saj ne vedo, za katere naloge njihovi zaposleni porabljajo čas. Omeniti je potrebno tudi izgubo dragocenih podatkov, saj v primeru požara vsi podatki izginejo.

Rešitev za to so aplikacije v obliki programske opreme kot storitve (angl. *Software as a service*). Omogočajo varno hrambo podatkov v oblaku (angl. *cloud*) z nizkimi stroški, primerne pa so tudi za tehnično slabše izobražene posameznike, saj imajo prijazen uporabniški vmesnik. Dostopne so preko svetovnega spleta (iz pisarne, od doma, s terena) s kateregakoli operacijskega sistema.

Cilj naše aplikacije je torej odgovoriti na vprašanje direktorja, šefa, vodje:

Kaj je moja ekipa počela včeraj?

1.2 Pregled obstoječih rešitev

Na tržišču je na voljo mnogo različnih rešitev za beleženje časa. Omenili bomo nekatere, ki so posebej primerne za samostojne podjetnike in male ekipe.

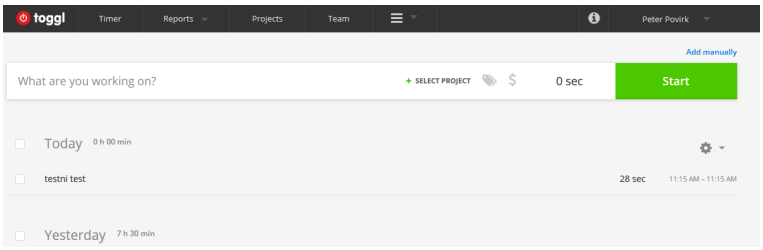
Osredotočili smo se na rešitve, ki za svoje delovanje ne potrebujejo fizičnega čitalca kartic ampak se zanašajo na ročni vnos delovnega časa. Omejili smo se tudi na rešitve, ki delujejo preko spletnega brskalnika in ki smo jih lahko brezplačno testirali.

1.2.1 Toggl

Toggl [1] je aplikacija, ki ponuja beleženje časa. Večina funkcij je brezplačna, zato je še posebej primerna za tiste, ki se s potrebo po beleženju časa srečujejo prvič, za nekatere napredne funkcije pa je potrebno plačati.

Poleg vnosa preko spletne strani omogoča tudi uporabo namiznega programa, ki deluje na vseh večjih operacijskih sistemih. Razvijalci Toggl-a so se še posebej potrudili pri uporabniški izkušnji, saj je vnos časovnih vnosov enostaven in hiter, kot je razvidno na Sliki 1.1.

Omogoča tudi enostavno ustvarjanje poročil z možnostjo filtriranja po času, projektu in stranki.



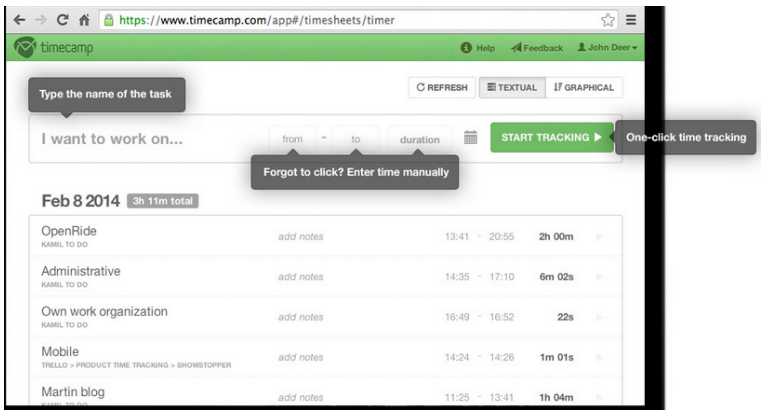
Slika 1.1: Vmesnik aplikacije Toggl

1.2.2 Timecamp

Timecamp [2] se beleženja časa loti na drugačen način. Omogoča sinhronizacijo podatkov s storitvami, ki jih zaposleni uporabljajo pri svojem delu, kot so programi za vodenje projektov in računovodski programi.

Z enostavnim uvozom projektov iz prej omenjenih programov je potrebno zaposlenim samo še dodati čas, ki je bil porabljen na posameznem projektu, Timecamp pa omogoči pregled nad vsemi porabljenimi urami in porabljenimi sredstvi.

Kot je razvidno iz uporabniškega vmes 1.2, je možen tudi ročni vnos podatkov. Timecamp omogoča tudi uporabi namiznega program, možno pa ga je uporabljati tudi na pametnih telefonih.



Slika 1.2: Vmesnik aplikacije Timecamp

Poglavje 2

Tehnologije

2.1 Računalništvo v oblaku

Računalništvo v oblaku (angl.*cloud computing*) lahko opredelimo kot storitev, ki nam omogoča obdelavo podatkov (vse vrste računanja). Pojem je povezan z deljenjem sredstev kot so procesor, pomnilnik in diskovni prostor z drugimi uporabniki. [3]

Za uporabnike je računalništvo v oblaku zanimivo, ker znatno zmanjša stroške in potrebe po usposobljenih kadrih. Manjšim podjetjem tako ni treba kupiti dragih strežnikov ampak jih preprosto najamejo pri ponudnikih storitev. S tem prihranijo tudi pri kadrovskih stroških, saj za najete strežnike skrbi ponudnik sam.

Enako velja tudi za programsko opremo: manjša podjetja si na začetku njihove poslovne poti lažje privoščijo mesečno najemnino programske opreme kot pa nakup dragih (trajnih) licenc.

Računalništvo v oblaku delimo na tri glavne tipe, ki jih bomo v nadaljevanju bolj podrobno spoznali:

- IaaS - infrastruktura kot storitev (angl. *Infrastructure as a service*),
- PaaS - platforma kot storitev (angl. *Platform as a service*) in
- SaaS - programska oprema kot storitev (angl. *Software as a service*).

2.1.1 IaaS

Infrastruktura kot storitev je osnoven model računalništva v oblaku se navezuje na ponujanje strojne opreme v najem, bodisi fizične ali virtualne. Ponudnik je lastnik in vzdrževalec strojne opreme, svojim strankam pa jo daje v najem na podlagi vnaprej dogovorjenih pogojev.

Ponudniki so navadno večja podjetja, ki imajo svoje ali najete prostore, ki so primerni za gostovanje strežnikov. Skrbijo za fizično varovanje, imajo generatorje za primer izpada električne energije in odlično internetno povezljivost.

Največkrat uporabnik najame določeno količino določenih sredstev, na primer virtualni računalnik, ki ima 2 GHz procesor, 2 GB pomnilnika in 20 GB trdega diska. Uporabnik lahko v večini primerov svoja sredstva na enostaven način poljubno nadgrajuje ali znižuje glede na potrebe oziroma zasedenost, kar še posebej potrebujejo nova, majhna podjetja, ki se hitro razvijajo.

Ta model storitve zajema samo strojno opremo, kot je strežnik ali omrežna oprema. Primer takega ponudnika je na primer ponudnik gostovanja, ki ponuja najem virtualnega strežnika.

2.1.2 PaaS

Platforma kot storitev je nadgradnja infrastrukture kot storitev z osnovno programsko opremo in okoljem. Ponudnik na svojo strojno opremo namesti in vzdržuje ter posodablja operacijski sistem, bazo podatkov, spletni strežnik in knjižnice, ki jih posamezen programski jezik potrebuje.

Namenjena je razvijalcem, ki nimajo znanja administracije sistemov ali pa želijo svoj čas posvetiti razvoju aplikacije in se ne želijo obremenjevati z vzdrževanjem okolja. Različni ponudniki podpirajo različne programske jezike, vsi pa zagotavljajo več kot 99.9% čas delovanja (angl. *uptime)*.

Znana primera IaaS ponudnikov sta Google s storitvijo App Engine [4] in Microsoft s storitvijo Azure [5].

2.1.3 SaaS

Programska oprema kot storitev opisuje poslovni model, kjer ponudnik storitve strankam ponuja uporabo aplikacije. Ponudnik skrbi za hrambo in varnost podat-

kov strank.

Ponudnik strankam pogosto zaračunava uporabo storitve glede na število uporabnikov, stranke pa večinoma niso vezane za dalj kot en mesec. Gre torej za mesečno naročnino, ponudnik pa strankam običajno ponuja več različnih paketov, na primer: bronastni, srebrni, zlati. Paketi z naprednimi funkcijami so seveda dražji, stranke pa lahko, glede na svoje potrebe in razvoj, prehajajo med paketi.

Model SaaS je čedalje bolj priljubljen pri razvijalcih, saj omogoča hiter razvoj z nizkimi stroški. Še ena prednost je delovanje v brskalniku - če uporabimo odzivno oblikovanje ni važno, katero napravo oziroma kateri operacijski sistem ima uporabnik. Tako razvijalcu ni potrebno skrbeti, če bo program enako deloval na sistemu Windows kot na sistemu Mac, ni pa se mu treba tudi odreči delu trga.

Majhne prototipe izdelka, ki so namenjeni hitremu preizkusu tržišča, imenovane MVP (angl. *Minimum viable product* je mogoče razviti že v nekaj tednih. S tem se razvijalci izognejo dolgotrajnemu razvoju izdelka, ki ga stranke na koncu ne bodo hotele kupiti.

Za razliko od prejšnjih dveh modelov računalništva o oblaku, ki sta bila namenjena tehnično bolje izobraženim posameznikom, je ta model namenjen tudi tehnično manj veščim posameznikom. Uporabnik dobi uporabniško ime in geslo, s katerima se prijavi v sistem. Arhitektura sistema končnemu uporabniku ni znana, je transparentna, ponavadi pa gre za porazdeljen sistem.

Primer SaaS storitve je poleg sistema, ki smo ga razvili v okviru diplomske naloge, Toggl [1].

2.2 Spletne tehnologije

V tem poglavju bomo predstavili glavne spletne tehnologije, ki so bile uporabljene za izvedbo naše aplikacije. Za naslednje spletne tehnologije smo se odločili predvsem zaradi dobrega poznavanja, primernosti in enostavnega ter hitrega razvoja, ki ga omogočajo.

2.2.1 PHP

PHP je popularen, objektno orientiran skriptni jezik [7], ki je bil razvit prav za spletno programiranje, zato omogoča hiter razvoj spletnih aplikacij. Njegove ko-

renine segajo v leto 1994, danes pa uporabljamo njegovo peto verzijo (PHP5).

Jezik PHP je v januarju 2013 uporabljalo več kot 240 milijonov strani [8], zato ga podpira večina podjetij, ki ponujajo spletno gostovanje. Zaradi enostavnosti in razširjenosti je primeren tudi za učenje programiranja, na svetovnem spletu pa obstaja mnogo brezplačnih tečajev.

2.2.2 Ogrodje Symfony2

V jeziku PHP je razvito tudi ogrodje Symfony2, ki je skupek neodvisnih komponent [9]. Kot nakazuje že ime, gre za drugo verzijo. Začetki ogrodja sodijo v leto 2005, razvilo ga je podjetje SensioLabs. Uporablja se v popularnih odprotokodnih projektih kot sta Drupal [10] in phpBB [11].

Za ogrodje Symfony smo se odločili, ker je omogoča hiter razvoj in ponuja stabilno, dobro definirano arhitekturo. Celotno ogrodje temelji na principu MVC (angl. *Model-View-Controller*).

Vstavljanje odvisnosti

Poseben pomen posvetimo konceptu vstavljanje odvisnosti [12] (angl. *dependecy injection*), ki se v razvoju v jeziku PHP zelo redko pojavi, je pa za nas zelo pomemben koncept, ki je tudi vplival na izbiro ogrodja Symfony2.

Pri vstavljanju odvisnosti vse zunanje odvisnosti v objekt prenesemo kot parameter konstruktorja objekta ali pa odvisnost nastavimo kot lastnost objekta. Z drugimi besedami - namesto, da si objekt odvisnosti ustvari sam, jih prejme kot parametre konstruktorja.

To nam močno olajša testiranje enot, o katerem bomo več govorili v Poglavju 5.

Komponente ogrodja Symfony2

Predstavili bomo nekaj najbolj popularnih, ključnih komponent ogrodja Symfony2:
Console - komponenta, ki omogoča razvoj vmesnikov za ukazno vrstico
DependencyInjection - komponenta, ki omogoča razvoj vmesnikov za ukazno vrstico
EventDispatcher - komponenta, ki zagotavlja sistem za dogodke med izvajanjem

aplikacije (angl. *events*) (obveščanje, možnost prijave na obveščanje)
Form - komponenta, ki vsebuje orodja za enostavno kreiranje in ponovno uporabo HTML obrazcev
Routing - komponenta, ki omogoča enostavno prevedbo lepih naslovov URL na krmilnike
Security - komponenta, ki vsebuje vso potrebno infrastrukturo za razvoj sistema za avtorizacijo in avtentikacijo

2.2.3 MySQL

MySQL je odprtokodna relacijska baza [13], ki je na drugem mestu po uporabi med odprtokodnimi relacijskimi bazami. [14].

Skupaj s PHP-jem in spletnim strežnik Apache [6] tvorijo na operacijskem sistemu Linux odprtokodni razvojni paket (angl. *solution stack*) LAMP, ki je zelo popularen med razvijalci spletnih rešitev.

Uporablja dialekt jezika SQL, ki omogoča hitre poizvedbe. Ključni ukazi so: `SELECT`, `INSERT`, `UPDATE` in `DELETE`.

2.2.4 JavaScript in knjižnica jQuery

JavaScript je dinamičen skriptni jezik [15], ki navadno teče v brskalniku, lahko pa se izvaja tudi na strežniku. Uporablja ga večina spletnih strani, saj poskrbi za interaktivnost in omogoča dinamično vsebino brez ponovnega nalaganja strani.

jQuery je knjižnica za JavaScript, ki razvijalcem omogoča hitro implementacijo pogosto uporabljenih funkcionalnosti [16] kot so na primer:

- validacijo podatkov,
- skrivanje in prikazovanje elementov (na primer: obvestil),
- asinhrone zahteve na strežnik,
- animacije elementov (na primer: menjavanje oglasnih pasic)
- in še bi lahko naštevali.

2.2.5 REST

REST (angl. *Representational state transfer*) je arhitektura, na kateri temelji delovanje svetovnega spleta [17]. Loči med odjemalcem in strežnikom in nima stanj. Sicer ga nezavedno uporabljamo ob večini zahtev na svetovnem svetu, namenoma pa se omenja pri spletnih servisih, kjer je glavni način za izmenjavo podatkov.

Za SaaS aplikacije je pomemben zaradi integracij, saj ga uporablja veliko vmesnikov API (angl. *Application Programming Interface*). Uspešna SaaS aplikacija omogoča enostavno izmenjavo podatkov v standardnih izmenjevalnih formatih kot sta XML in JSON.

Poglavje 3

Opis funkcionalnosti aplikacije

V tem poglavju bomo opisali glavne funkcionalnosti aplikacije, čemu služijo in kakšni problemi so se pojavili pri razvoju le-teh.

Uporabniški vmesnik je za uspešnost spletne aplikacije zelo pomemben, sploh če je na trgu že veliko konkurentov. Poleg uporabnikov so entitete, s katerimi se ukvarjamo še Stranka, Projekt in Časovni vnos.

3.1 Uporabniki

Za dostop do aplikacije se je potrebno registrirati. Obrazec za registracijo je prikazan spodaj na Sliki 3.1.

Želeli smo, da ima obrazec za registracijo minimalno število polj. Tako povečamo število registracij in s tem število potencialnih strank, morebitne druge podatke, ki jih želimo pridobiti, pa pridobimo naknadno preko urejanja podatkov uporabnika ali preko elektronske pošte.

Lahko bi odstranili še polje uporabniško ime, saj je elektronska pošta unikaten identifikator, vendar se za ta korak nismo odločili, saj se v tem primeru lahko pojavijo težave ob spremembi elektronskega naslova uporabnika.

3.1.1 Facebook

Glede na razširjenost socialnega omrežja Facebook in dejstva, da želijo imeti uporabniki čim manj uporabniških imen in gesel smo se odločili, da omogočimo upo-

Slika 3.1: Obrazec za registracijo

rabnikom poleg standardne registracije in prijave tudi prijavo s podatki, ki jih že uporabljajo na Facebook-u.

Ustvarili smo Facebook aplikacijo, da smo dobili zahtevana parametra `facebook_app_id` in `facebook_app_secret`. Nastavili smo tudi naslov, na katerega Facebook preusmeri uporabnika po uspešni avtentikaciji. Nato smo uporabnika ob kliku na gumb "Prijavi se s Facebook-om"preusmerili na Facebook kjer se je moral, če že ni bil, prijaviti.

Po uspešni prijavi je bil uporabnik preusmerjen nazaj na našo stran, prikazala pa se je prva stran za prijavljene uporabnike.

3.1.2 PayPal

Cilj naše aplikacije za beleženje časa je seveda tudi poslovna uspešnost, to pa ni mogoče brez vsaj enega načina za sprejemanje plačil. Klasična načina, kot sta plačilo po povzetju ali bančno nakazilo, na spletu ne prideta v poštev, ker nista takojšnja. Ponudnik plačila ne prejme tako, zato mora uporabnik na odobritev dostopa čakati tudi po več dni.

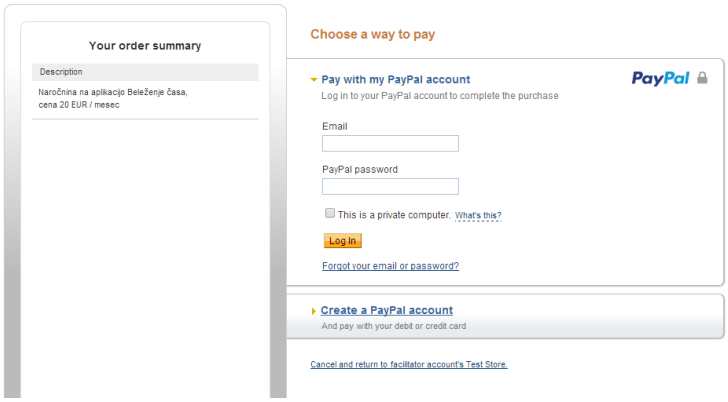
Zato ne preseneča dejstvo, da je med uporabniki spletnih storitev popularen

sistem za plačila PayPal [19], ki omogoča sprejemanje in pošiljanje v več kot 200 državah. Omogoča varno nakupovanje, saj uporabniku ni treba vpisovati številke kreditne kartice na vsaki spletni strani posebej, ampak jo vpiše samo v PayPalu, spletna stran pa ga ob nakupu samo preusmeri na PayPal, kjer uporabnik odobri transakcijo.

PayPal omogoča tudi sprejemanje plačil za naročnine, kar je še posebej pomembno za SaaS aplikacije. Sam postopek je zelo podoben avtentikaciji z omrežjem Facebook.

Uporabnika ob nakupu paketa preusmerimo na PayPal, kjer sklene naročnino (Slika 3.2). Po uspešnem plačilu je uporabnik preusmerjen nazaj na našo stran, naša aplikacija pa je s strani PayPal-a obveščena o uspešni transakciji, da lahko uporabniku dodamo novo ali podaljšamo obstoječo naročnino.

PayPal nas o vseh spremembah statusa naročnine obvešča preko servisa IPN (angl. *Instant Payment Notification*) na naslov URL, ki smo ga vnesli v administraciji. Tako smo takoj obveščeni v primeru, da uporabnik naročnino prekliče ali pa v sistemu nima dovolj stanja za plačilo mesečne naročnine.



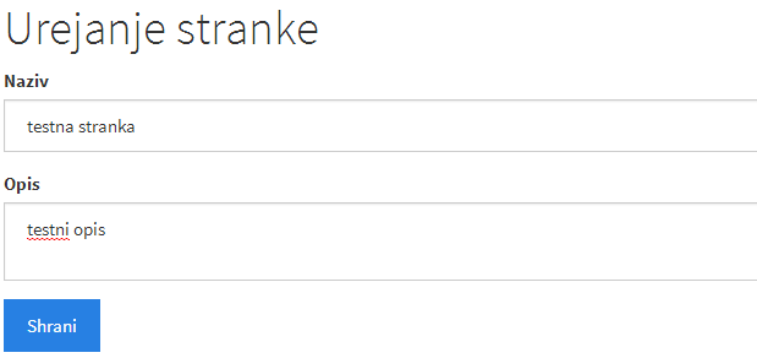
Slika 3.2: Sklenitev naročnine v sistemu PayPal

3.2 Stranke

Stranka je entiteta v sistemu, kateri se dodeli projekte. Na Sliki 3.3 je razvidno, da ne vsebuje veliko polj - samo naziv in opis.

Entiteto Stranka bi lahko razširili s poljem obračunski interval, ki bi, če bi bil nastavljen na na primer 30 minut, vse časovne vnose navzgor zaokrožil na 30 minut. Za izboljšanje uporabniške izkušnje bi lahko vsaki stranki dodali tudi barvo, da bi se uporabnik lažje znašel v pregledu že opravljenih časovnih vnosov.

Smiselno bi bilo tudi, da bi uporabnik lahko stranke uvažal iz svojega programa za elektronsko pošto.



Slika 3.3: Obrazec za urejanje stranke

3.3 Projekti

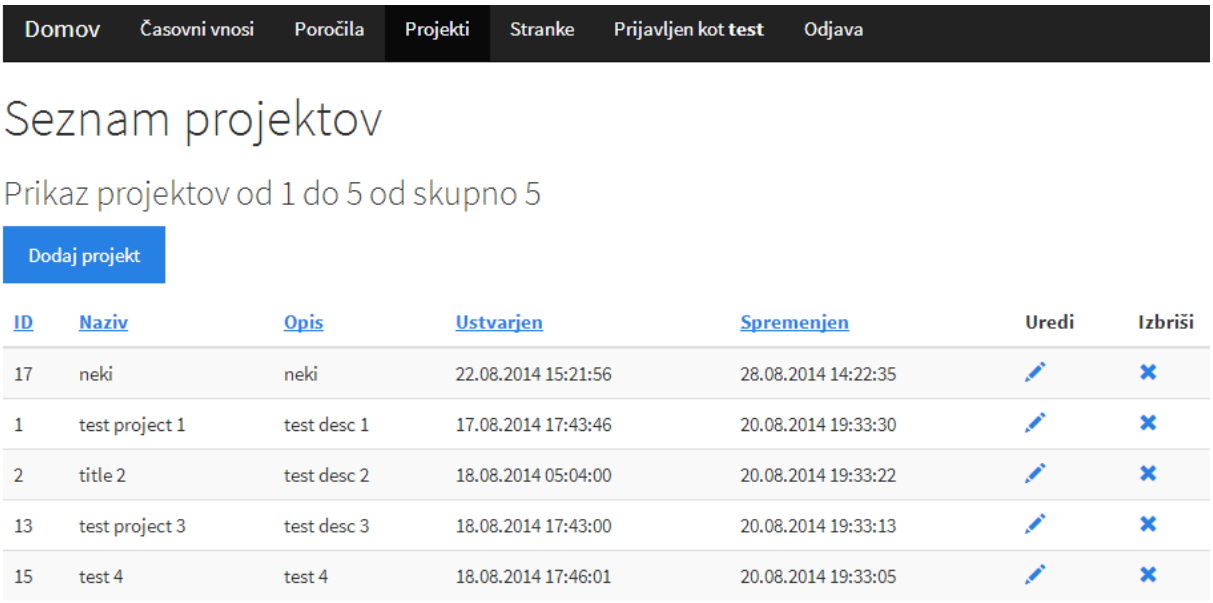
Vsaka stranka ima lahko več projektov, projektu pa lahko poleg naziva in opisa nastavimo tudi stranko.

Projekte lahko, kot je razvidno s Slike 3.4, poleg naziva in opisa razvrščamo po datumu ustvarjanja ali spremembe.

Pri časovnih vnosih je ime projekta združeno z imenom stranke. Projekt `Testni projekt`, ki pripada stranki `Testna stranka`, je prikazan kot `Testni projekt (Testna stranka)`. Na ta način omogočimo uporabnikom boljšo preglednost pripadnosti posameznega časovnega vnosa.

3.4 Časovni vnosi

Časovni vnos je osnovna entiteta v sistemu. Pripada projektu in posledično tudi stranki.

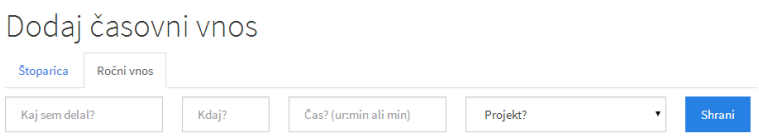


Slika 3.4: Razvrščanje projektov

Zaradi enostavnosti vnosa je polje projekt neobvezno. To nam omogoča hiter vnos časovnih vnosov tudi za nove projekte, ki jih še nimamo v sistemu, povzroči pa nam težave pri poročilih in izvozih, saj ne vemo, kam naj štejemo ta porabljen čas.

Časovni vnos lahko vnesemo ročno (Slika 3.5) ali pa uporabimo štoparico (Slika 3.6).

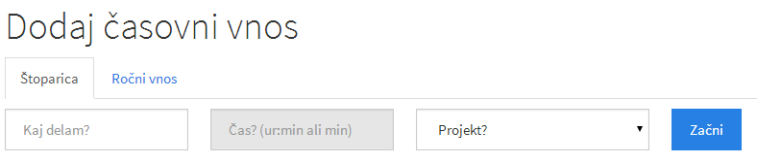
Ročni vnos je namenjen predvsem preteklim časovnim vnosom ali pa tistim, ki jih zaradi različnih okoliščin (na primer: bili smo na terenu brez dostopa do spleta) nismo mogli vnesti. Pri ročnem vnosu lahko izberemo tudi datuma vnosa. Polje za vnos časa sprejme vnos oblike `ure:minute` ali pa samo `minute`.



Slika 3.5: Ročni vnos časovnega vnosa

Štoparica pa je idealna za vnos tistih nalog, ki jih opravljamo v pisarni, saj ob začetku naloge enostavno pritisnemo gumb **Začni** ob koncu pa gumb **Končaj**. Medtem, ko nalogo opravljamo, nam ni treba imeti odprtega brskalnika ali pa sploh prižganega računalnika, saj je za merjenje časa zadolžen strežnik.

Posebno pozornost je potrebno nameniti izbiri časovnega pasu. Ta se ne sme spreminjati medtem, ko je nek časovni vnos uporabnika aktiven, saj bi v tem primeru lahko prišli v situacijo, ko bi končali z merjenjem časa kasneje kot smo začeli.



Slika 3.6: Vnos časovnega vnosa preko štoparice

Ob pritisku na gumb **Začni** se uporabniku trenutni časovni vnos nastavi kot aktiven, zabeleži se tudi trenutni strežniški čas. Če uporabnik zapre stran, se ob ponovnem obisku aktivni časovni vnos prebere iz baze, štoparica pa se nastavi na pravilen čas in merjenje časa se nadaljuje.

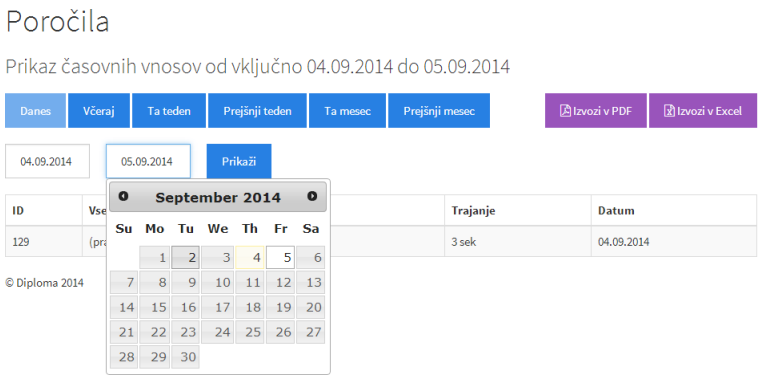
3.5 Poročila in izvozi

Pomemben del aplikacije so tudi poročila, saj brez ne vemo, koliko moramo določeni stranki zaračunati oziroma koliko ur je bilo porabljenih na določenem projektu.

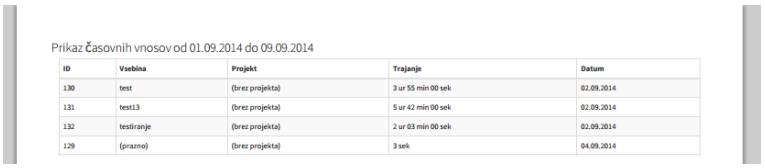
Slika 3.7 prikazuje vmesnik za izbiro časovnega okvirja izvoza. Vsebuje šest vnaprej določenih možnosti (danes, včeraj, ta teden, prejšnji teden, ta mesec in prejšnji mesec), omogoča pa tudi poljuben začetka in konca časovnega okvirja.

Za izvoze smo podprli dva popularna formata: PDF in XLS.

Za generiranje PDF dokumentov smo uporabili odprtokodno orodje `wkhtmltopdf` [22], ki kodo HTML pretvori v format PDF. Naredili smo preprosto HTML tabelo, uporabili smo enak izgled kot pri poročilih in tudi tukaj je bilo uporabljeno ogrodje Twitter Bootstrap. Večjih težav nismo imeli, problem je bil samo s šumniki. Rezultat si lahko ogledate na Sliki3.8.



Slika 3.7: Vmesnik za določanje časovnega okvirja poročil



Slika 3.8: Poročilo v formatu PDF

Za generiranje XLS dokumentov smo uporabili odprtokodno knjižnico PHPExcel [21], katere uporaba je zelo enostavna:

```
$phpExcelObject = $this->get('phpexcel')->
    createPHPExcelObject();
// Set active sheet index to the first sheet, so Excel
    opens this as the first sheet
$phpExcelObject->setActiveSheetIndex(0);
// title row
...
// header row
...
$i = 4;
foreach($entries as $entry)
{
        // add entries
```

```
            ...
        $activeSheet
                ->setCellValue('A' . $i, $entry->getId())
                ->setCellValue('B' . $i, $content)
                ->setCellValue('C' . $i, $project)
                ->setCellValue('D' . $i, PericeExtension::
                    formatDurationFilterStatic($entry->
                    getDuration(), $this->get('translator'))
                    )
                ->setCellValue('E' . $i, $entry->
                    getStartedAt()->format(self::
                    DATE_TIME_FORMAT));
        $i++;
}
```

Od odprtju datoteke sicer Microsoft Excel prikaže sporočilo, da je datoteka lahko nevarna, ker izvira s spletne lokacije, česar žal ne moremo preprečiti. Izgled dokumenta XLS si lahko ogledate na Sliki3.9.

	A	B	C	D	E
1	Prikaz časovnih vnosov od 01.09.2014 do 09.09.2014				
2					
3	ID	Vsebina	Projekt	Trajanje	Datum
4	130	test	(brez projekta)	3 ur 55 min 00 sek	02.09.2014 00:00:00
5	131	test13	(brez projekta)	5 ur 42 min 00 sek	02.09.2014 00:00:00
6	132	testiranje	(brez projekta)	2 ur 03 min 00 sek	02.09.2014 00:00:00
7	129	(prazno)	(brez projekta)	3 sek	04.09.2014 15:22:45

Slika 3.9: Poročilo v formatu XLS

3.6 Večjezičnost

Že od začetka smo planirali, da bi aplikacija lahko podpirala več jezikov, če bi bilo to potrebno. Vsa besedila, imena polj, napisi na gumbih in ostali teksti so zbrani v ločeni datoteki v obliki asociativne tabele:

...

```
'customer.title' => 'Naziv stranke',
'customer.description' => 'Opis stranke',
...
```

Ogrodje Symfony za določitev jezika za prevode uporablja parameter `locale`. Tabela s prevodi se nahaja v datoteki `messages.[locale].php`, za slovenščino torej `messages.sl.php`.

Za dodajanje novega jezika je tako potrebno prevesti samo to datoteko, programska in HTML koda pa ostaneta nespremenjeni. To tudi pomeni, da za prevajalca ni potrebno znanje jezika HTML.

Poglavje 4

Arhitektura

4.1 Opis uporabljene programske opreme

Za razvojno okolje smo uporabili okolje za razvoj spletnih aplikacij WAMP (*Windows Apache MySQL PHP*) [24].

Sestavljajo ga štiri komponente, ki smo jih bolj podrobno predstavili že v Poglavju 2:

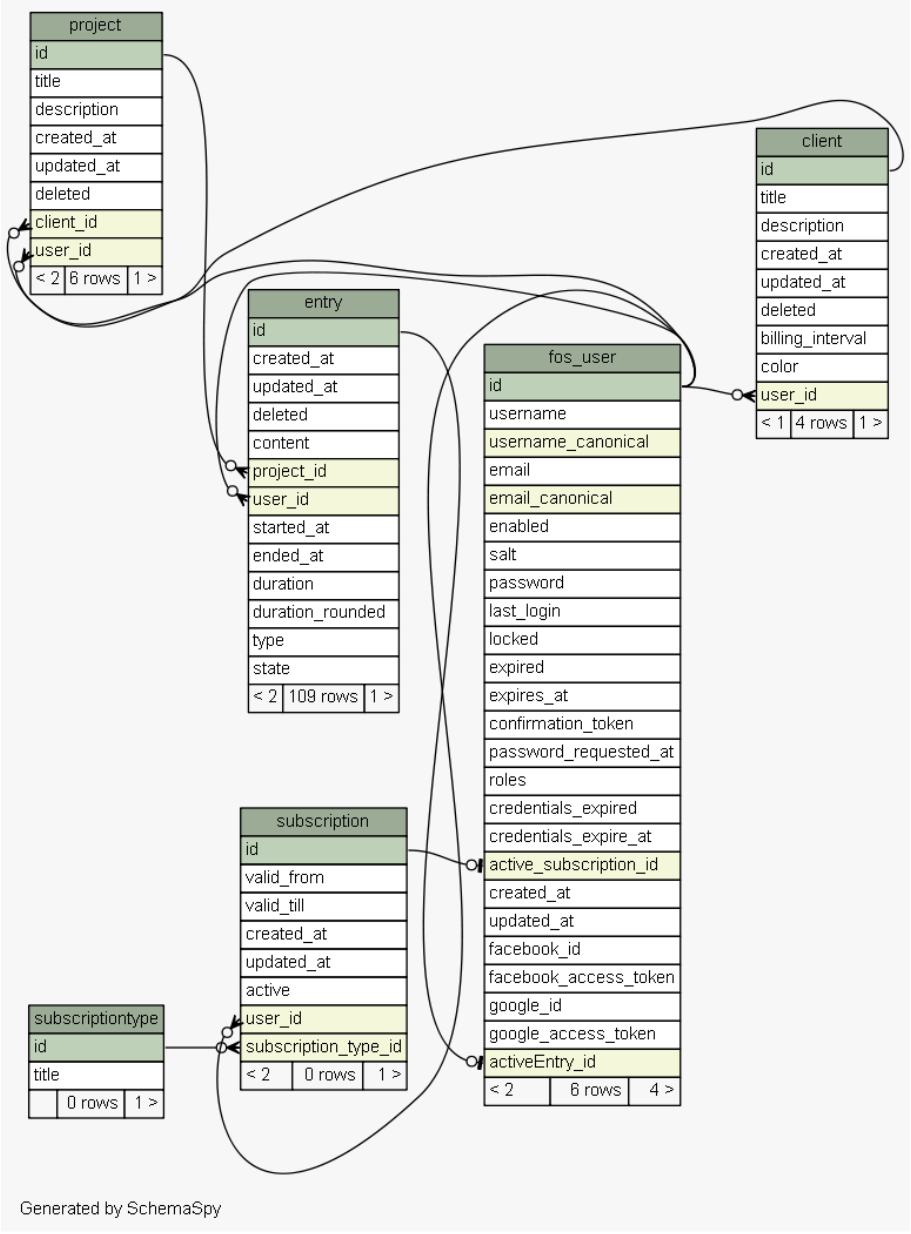
- operacijski sistem Windows,
- odprtokodni spletni strežnik Apache,
- odprtokodna relacijska baza MySQL in
- programski jezik PHP.

4.2 Arhitektura baze

Symfony za delo z bazo uporablja Doctrine ORM (*Object relational mapper*) [25]. Programerju omogoča, da piše poizvedbe v jeziku DQL (angl. *Doctrine Query Language*). Aplikacija je tako prenosljiva med različnimi bazami podatkov (MySQL, PostgreSQL, Microsoft SQL).

Doctrine dela z objekti ne glede na to, koliko poizvedb je za to potrebnih v ozadju. Nudi nam hitre bližnjice kot so `->find($id)`, ki nam vrne objekt s ključem `$id`, ki omogočajo izjemno hiter razvoj.

Struktura baze je zasnova na osnovnih relacijah med objekti in večuporabniškim okoljem *angl. multi-tenancy*. Uporabnik ima lahko več strank, vsaka stranka ima lahko več projektov, vsak projekt ima lahko več časovnih vnosov.



Slika 4.1: Arhitektura baze

4.3 Zagotavljanje varnosti

Zagotavljanje varnosti je pri spletnih aplikacijah še posebej pomembno. Potencialne stranke je potrebo prepričati, da so njihovi podatki na spletu enako varni kot tisti, ki se nahajajo na njihovih osebnih računalnikih ali internih strežnikih podjetja.

Odločili smo se za uporabo 128 bitnega certifikata SSL (angl. *Secure Sockets Layer*) [26]. Vsi podatki, ki se izmenjujejo preko protokola HTTPS (angl. *Hypertext Transfer Protocol Secure*), so šifrirani. [27] Podatke šifrirata strežnik in brskalnik na podlagi vnaprej dogovorjenega sejnega ključa, ki ga vzpostavita v postopku, ki ga imenujemo rokovanje (angl. *handshake*).

Uporaba protokola HTTPS nam omogoča tudi enostavnejšo implementacijo vmesnika API, saj je za avtentikacijo uporabnik dovolj le ključ API, ki si ga, ob morebitnem razkritju, uporabnik lahko ponovno ustvari v administrativnem vmesniku.

4.4 Vmesnik API

Vmesnik API smo zasnovali zato, da bi imela naša spletna aplikacija zaradi enostavnega načina integracije čim večje možnosti za uspeh na tržišču. Odločili smo se za protokol REST, ki je v primerjavi s protokolom SOAP (angl. *Simple Object Access protocol*) bolj enostaven za implementacijo.

Za dostop do vmesnika API potrebuje uporabnik ključ API, ki mu ga dodelimo ob registraciji. Ključ API je niz naključnih znakov dolžine 32. Uporabnik ga lahko kadarkoli spremeni. Vsi klici vmesnika API potrebujejo obvezen parameter ključ API, ki ga uporabimo za avtentikacijo uporabnika.

Preko vmesnika API smo izpostavili vse ključne entitete (Stranka, Projekt, Časovni vnos) in njihove CRUD (angl. *Create Read Update Delete*) operacij. V skladu s protokolom REST je struktura zahtev zelo preprosta.

Uporabili smo angleška imena entitet (client, project, entry) in angleška imena operacij (list, add, edit, delete).

Vmesnik API sprejme vse zahteve samo v obliki zahteve HTTP metode POST. Metoda POST je v primerjavi z metodo GET veliko bolj varna, saj parametrov zahteve ne shranjuje v dnevniških zapisih strežnika.

Zahteve imajo naslednjo strukturo URL:

- prikaz seznama entitet: `/api/{entiteta}/list`,
  primer: `/api/client/list`
- dodanje nove entitete: `/api/{entiteta}/add`,
  primer: `/api/project/add`
- urejanje obstoječe entitete: `/api/{entiteta}/edit/{id}`
  primer: `/api/entry/edit/6`
- brisanje entitete: `/api/{entiteta}/delete/{id}`,
  primer: `/api/client/delete/8`

Različne zahteve sprejmejo tudi različne parametre kot so polja in in njihove vrednosti pri dodajanju in urejanju ter maksimalno število rezultatov pri prikazu seznama.

Primer odgovora vmesnika API za poizvedbo `/api/entry/list`:


```
[
  {
    "id": 123,
    "created_at": "2014-08-28T12:26:03+0200",
    "updated_at": "2014-08-28T12:26:03+0200",
    "started_at": "2014-08-28T12:26:03+0200",
    "ended_at": "2014-08-28T12:26:03+0200",
    "duration": 1380,
    "duration_rounded": 1380,
    "type": 1,
    "state": 3,
    "content": "vsebina test"
  },
  {
    "id": 122,
    "project": {
      "id": 1,
      "client": {
```

```
        "id": 2,
        "title": "testclientno2",
        "description": "testdesc2",
        "billing_interval": 30
      },
      "title": "testproject1",
      "description": "testdesc1"
    },
    "created_at": "2014-08-28T12:19:38+0200",
    "updated_at": "2014-08-28T12:22:30+0200",
    "started_at": "2014-08-28T12:19:38+0200",
    "ended_at": "2014-08-28T12:22:30+0200",
    "duration": 172,
    "duration_rounded": 172,
    "type": 1,
    "state": 3,
    "content": "testni vnos abc"
  },
  {
    "id": 121,
    "created_at": "2014-08-28T12:19:21+0200",
    "updated_at": "2014-08-28T12:19:23+0200",
    "started_at": "2014-08-28T12:19:21+0200",
    "ended_at": "2014-08-28T12:19:23+0200",
    "duration": 2,
    "duration_rounded": 2,
    "type": 1,
    "state": 3,
    "content": ""
  }
]
```

4.5 Opravila CRON

Za potrebe avtomatskega generiranja poročil smo uporabili komponento Console ogrodja Symfony. Omogoča nam enostavno izdelavo ukazov za ukazno vrstico (angl *command line*). Še posebej uporabni so za periodična opravila kot so uvozi in izvozi.

Za uporabnike smo pripravili enostavne povzetke o tedenskem delu, ki se pošiljajo vsak ponedeljek za prejšnji teden in vsak peti dan v mesecu za prejšnji teden. Tako omogočimo tudi vnose za nazaj, saj prvega v mesecu nekateri sodelavci morda še nimajo za nazaj vpisanih vseh vnosov.

Klica komponente sta enostavna: `php app/console reports:generate weekly` in `php app/console reports:generate monthly`

Uporabniki prejmejo poročila na email naslov s priponkama v formatih PDF in XLS.

Poglavje 5

Testiranje

Za zagotavljanje stabilnosti spletne aplikacije smo se odločili, da za pomembnejše funkcionalnosti uporabimo dve različni tehniki avtomatskega testiranja:

- testiranje enot in
- testiranje funkcionalnosti.

Pri testiranju enot (angl. *unit testing*) testiramo najmanjši, samostojen del kode, ki ga je možno testirati, pogosto je to metoda ali funkcija.

Uporabili smo tudi testiranje funkcionalnosti (angl. *functional testing*). Za razliko od testiranja enot testiranje funkcionalnosti testira večjo akcijo kot je na primer prijava uporabnika, kjer testiramo več komponent.

Za skriptni jezik PHP je bilo razvito ogrodje za testiranje enot PHPUnit[28], ki ga bomo v nadaljevanju podrobneje predstavili. Izbrali smo ga predvsem zaradi razširjenosti, zrelosti in enostavne integracije z ogrodjem Symfony2.

5.1 Testiranje enot in PHPUnit

Avtomatsko testiranje enot nam omogoča:

- prihranek časa, saj test napišemo enkrat, poženemo pa ga lahko ob vsaki večji spremembi kode,
- natančnost, ki je pri ročnem testiranju ne moremo zagotoviti in

- hitrost, saj lahko v nekaj minutah testiramo celotno (manjšo) aplikacijo.

Funkcije testiramo tako, da jim podamo zahtevane parametre in ocenimo (angl. *assert*) rezultat. Za oceno pogosto uporabljamo funkcije kot so: manjše, večje, je enako oziroma pravilno / nepravilno (angl. *true / false*).

Preprost primer, ki preveri, če računalo vrne 42, ko kličemo funkcijo `seštej` z argumentoma 30 in 12, si lahko ogledate na Sliki 5.1.

```php
// src/Acme/DemoBundle/Tests/Utility/CalculatorTest.php
namespace Acme\DemoBundle\Tests\Utility;

use Acme\DemoBundle\Utility\Calculator;

class CalculatorTest extends \PHPUnit_Framework_TestCase
{
    public function testAdd()
    {
        $calc = new Calculator();
        $result = $calc->add(30, 12);

        // assert that your calculator added the numbers correctly!
        $this->assertEquals(42, $result);
    }
}
```

Slika 5.1: Primer testiranja enot

PHPUnit poženemo iz ukazne vrstice s preprostim ukazom:

```
phpunit -c app/
```

V primeru uspeha nam izpiše:

```
.....

Time: 1.43 minutes, Memory: 74.00Mb

[30;42mOK (21 tests, 21 assertions)[0m
```

Če pa kakšen test ne uspe, pa dobimo ustrezno sporočilo o napaki:

```
.....[41;37mF[0m.....
```

```
Time: 1.4 minutes, Memory: 74.25Mb

There was 1 failure:

1) Perice\TimetrackerBundle\Tests\Controller\entryControllerTest::testEdit
Failed asserting that false is true.

D:\www\diploma\src\Perice\TimetrackerBundle\Tests\Controller\
EntryControllerTest.php:70
[37;41m                                                        [0m
[37;41mFAILURES!                                               [0m
[37;41mTests: 21, Assertions: 21, Failures: 1.[0m
```

5.2 Testiranje funkcionalnosti

Testiranje funkcionalnosti uporabimo za testiranje zaključene akcije, kot je na primer urejanje časovnega vnosa. Testiramo naslednje komponente:

- usmerjanje (angl. *routing*),
- baza podatkov,
- krmilnik (angl. *controller*) in
- pogled (angl. *view*).

Poleg urejanja časovnega vnosa testiramo tudi prijavo, saj moramo biti za urejanje časovnega vnosa, ki nam pripada, prijavljeni v spletno aplikacijo. Kodo testa si lahko ogledate na Sliki 5.2, kjer je tudi jasno razvidno, kako hitro lahko v okolju Symfony2 razvijemo teste.

Postopek funkcionalnega testa je sledeč:

- prijavi se v sistem,
- naloži stran za urejanje časovnega vnosa številka 1,

```php
public function testEdit()
{
    $client = static::createAuthenticatedClient();

    $crawler = $client->request('GET', '/dashboard/entry/edit/1');

    $buttonCrawlerNode = $crawler->selectButton('entry[save]');

    $form = $buttonCrawlerNode->form(array(
        'entry[content]'         => 'test entry 777'
    ));

    $crawler = $client->submit($form);

    $assertions = array
    (
        $crawler->filter('html:contains("Prikaz časovnih vnosov od 1 do 1 od skupno 1")')->count() > 0,
        $crawler->filter('html:contains("test entry 777")')->count() > 0,
        $crawler->filter('html:contains("Uspešno shranjeno")')->count() > 0,
    );

    $this->assertTrue(self::assertArray($assertions));
}
```

Slika 5.2: Primer testiranja urejanja časovnega vnosa

- poišči obrazec, ki ima gumb za shranjevanje z imenom t entry[save]
- v polje za vsebino časovnega vnosa vpiši *test entry 777*,
- pošlji obrazec,
- na strani, ki se prikaže, preveri ali vsebuje vse tri tekste (”Prikaz časovnih vnosov od 1 do 1 od skupno 1”, ”test entry 777”in ”Uspešno shranjeno”).

Test je uspešen, če je uspešna zadnja točka (ocenimo ga z `assertTrue`).

Poglavje 6

Možne izboljšave

6.1 Dodatne funkcionalnosti

Ob razvoju spletne aplikacije smo ves čas dobivali nove ideje za izboljšanje le-te, žal pa vseh zaradi pomanjkanja časa nismo mogli implementirati. V tem poglavju bomo nekaj najboljših podrobneje predstavili.

6.1.1 Prijava z ...

Facebook ni edina razširjena storitev, ki omogoča uporabo svojega avtentikacijskega sistema drugim. V prihodnosti bi lahko podprli še druge take servise kot so: Google, Instagram, Linkedin in drugi.

6.1.2 Izboljšanje uporabniške izkušnje

Beleženje časa ne spada med najbolj priljubljena opravila, zato smo želeli aplikacijo izdelati tako, da bo uporabniku omogočala čim hitrejši in čim bolj prijazen vnos časovnih vnosov. Za dodatno izboljšanje uporabniške izkušnje, bi bi bilo potrebno dodati še naslednje funkcionalnosti:

- vnos novih strank in projektov pri vnosu časovnega vnosa - če stranka (ali projekt) ne obstaja, lahko uporabnik vpiše naziv stranke, v ozadju pa se samodejno ustvari stranka. Dodatne podrobnosti stranke lahko uporabnik uredi kasneje.

- možnost shranjevanja poljubnih, pred nastavljenih časovnih okvirjev. Primer: tromesečje.

6.1.3 Uporabniške vloge

Trenutno so vsi naši uporabniki med sabo enakovredni, vidijo in urejajo lahko vse entitete. Smiselno bi bilo, da imeli več nivojev uporabnikov, na primer: administrator, vodja oddelka, kadrovska služba, delavec.

Administrator bi lahko dostopal do vseh podatkov. Vodja oddelka bi lahko urejal samo tiste projekte, ki spadajo pod njegov oddelek. Kadrovska služba bi imela možnost urejati in pregledovati podatke o zaposlenih. Navaden delavec bi videl samo projekte, ki mu jih je dodelil vodja oddelka.

6.1.4 Integracije

Naša aplikacija sicer vsebuje vmesnik API, ki ga lahko drugi programi uporabljajo, nismo pa naredili nobene integracije z že obstoječimi programi. Za tržno uspešnost naše aplikacije bi bilo koristno, če bi podpirala katerega od bolj znanih slovenskih računovodskih programov, kot so: Pantheon, Birokrat.

6.2 Skaliranje

V primeru, da bi bila naša spletna aplikacija zelo uspešna, bi lahko prerasli zmogljivosti virtualnega strežnika. Lastnosti sistema, ki ga je mogoče prilagoditi na veliko večjo obremenitev, pravimo skaliranje (angl. *scalability*).

Poznamo dva glavna načina skaliranja [29]:

- vertikalno skaliranje (angl. *vertical scaling*) in
- horizontalno skaliranje (angl. *horizontal scaling*).

6.2.1 Vertikalno skaliranje

Vertikalno skaliranje je pristop, pri katerem obstoječemu sistemu povečamo zmogljivosti. Za primer lahko uporabimo spletni strežnik (fizični ali virtualni), ki ga ob povečanju obiska nadgradimo - na primer: procesor zamenjamo z zmogljivejšim.

Ta vrsta skaliranja je pomembna predvsem za aplikacije, ki niso primerne za porazdeljeno računalništvo. Dodati je potrebno še, da smo pri tej vertikalnem skaliranju omejeni, saj nam hitro zmanjka zmogljivejših procesorjev ali mest za dodatni spomin.

6.2.2 Horizontalno skaliranje

Horizontalno skaliranje je pristop, pri katerem k sistemu dodajamo vozlišča (angl. *node*). Za primer lahko uporabimo spletni strežnik (fizični ali virtualni), ki mu ob povečanju obiska dodamo še enega, promet med njima pa enakomerno razdelimo. Tak sistem lahko ob predpostavki, da sta strežnika enaka, sedaj streže še enkrat več prometa.

Vse aplikacije niso primerne za porazdeljeno računalništvo, spletne pa večinoma so. Možni pristopi pri spletnih aplikacijah po naraščajočem številu zahtev:

- razdelitev spletnega strežnika in podatkovne baze na ločena strežnika,
- razdelitev podatkovne baze na več strežnikov (angl. *cluster*),
- vzpostavitev večih spletnih strežnikov,
- vzpostavitev sistema predpomnjenja (na primer: Redis [30]).

6.2.3 Skaliranje naše aplikacije

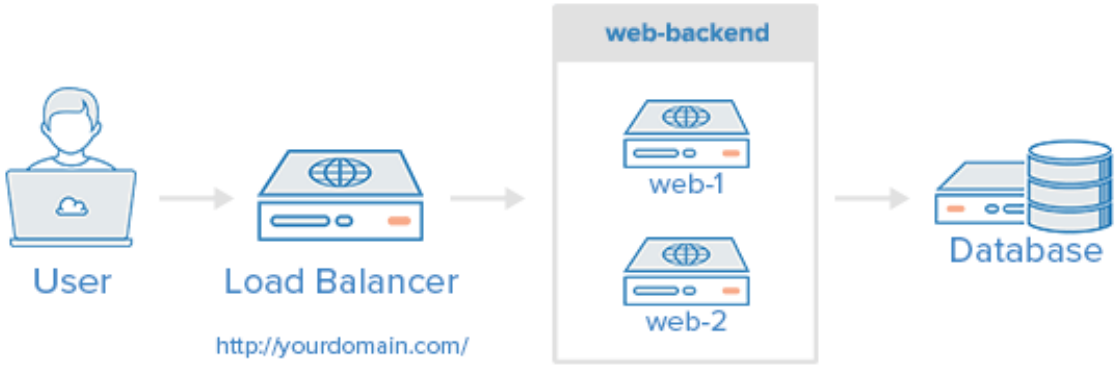
Ob velikem uspehu naše aplikacije bi se raje odločili za horizontalno skaliranje, saj je cenejše in manj omejeno. Večina ponudnikov gostovanja v oblaku ponuja enostaven načina podvojitve ali nadgradnje enote.

Pregledali smo ponudbo ponudnika DigitalOcean [31], ki nam omogoča avtomatično postavitev arhitekture, ki je prikazana na Sliki 6.1. Uporabnika ob dostopu na spletno stran prestreže izenačevalec obremenitve (angl. *load balancer*), ki ga usmeri na naključni spletni strežnik (*web-backend*). Spletni strežniki podatke pridobivajo iz baze, ki jo lahko sestavlja več strežnikov.

DigitalOcean nam omogoča avtomatsko dodajanje novih instanc spletnih in podatkovnih strežnikov s preprostimi ukazi API. S sistemskimi orodji lahko spremljamo izkoriščenost procesorja in če se le-ta konstantno giblje nad 80%, ustvarimo

novo instanco na podlagi predloge spletnega strežnika. Izenačevalec obremenitve bo tako namesto na dva pošiljal zahteve na tri spletne strežnike. Na podoben način lahko instanco, če število obiskovalcev upade, tudi uničimo.

Visok nivo avtomatizacije nam omogoča prihranek časa in denarja, saj za zmogljivosti plačujemo samo takrat, ko jih dejansko uporabljamo.



Slika 6.1: Primer skaliranja spletne aplikacije

Poglavje 7

Sklepne ugotovitve

V diplomski nalogi smo razvili aplikacijo za beleženje časa, ki je posebej primerna za samostojne podjetnike in mala podjetja. Aplikacija je zahvaljujoč odzivnemu oblikovanju dostopna z vseh naprav, ki omogočajo dostop do svetovnega spleta.

Cilj aplikacije je uporabniku olajšati beleženje časa. Aplikacija omogoča dodajanje strank, projektov in časovnih vnosov. Časovne vnose lahko uporabnik vnaša preko štoparice ali ročnega vnosa, za kar skrbi intuitiven vmesnik.

Za razvoj aplikacije smo uporabili odprtokodna orodja PHP (Symfony2), MySQL in Apache za ozadje ter Twitter Bootstrap (HTML, CSS) in jQuery (JavaScript) za uporabniški vmesnik. Aplikacijo je možno skalirati tako vertikalno kot horizontalno. Omogočili smo tudi izvoz podatkov v razširjenih formatih PDF in XLS. Raziskali smo tudi možnost plačil s sistemom PayPal, za slovenski trg pa bi lahko uporabili še kakšno domače podjetje.

Med razvojem aplikacije nismo imeli večjih težav, ugotovili pa smo, da se za funkcionalnost štoparice nikoli ne smemo zanašati na odjemalca ampak moramo začetni in končni čas časovnega vnosa vedno hraniti na strežniku, kjer tudi opravimo preračun porabljenega časa.

Aplikacija je seveda še zelo mlada, nekatere možne izboljšave pa smo navedli že v prejšnjih poglavjih. Izboljšali bi lahko predvsem uporabniško izkušnjo (izgled grafičnega vmesnika in poenostavitev nekaterih postopkov) in integracije z drugimi servisi.

Če bi aplikacijo prodajali kot programsko opremo kot storitev, bi verjetno dobili veliko odziva od prvih rednih uporabnikov - to bi uporabili za izboljšanje

uporabniške izkušnje in nove funkcionalnosti. Nenazadnje bi lahko tudi prevedli uporabniški vmesnik v kakšen tuj jezik, predvsem angleščino.

Literatura

[1] Aplikacija Toggl, dostopna na:
https://www.toggl.com/

[2] Aplikacija Timecamp, dostopna na:
http://www.timecamp.com/

[3] Računalništvo v oblaku, dostopno na:
http://en.wikipedia.org/wiki/Cloud_computing

[4] Google-ova Iaas storitev App Engine, dostopna na:
https://appengine.google.com/

[5] Microsoft-ova Iaas storitev Azure, dostopna na:
https://azure.microsoft.com/en-us/

[6] Odprtokodni spletni strežnik Apache, dostopen na:
http://www.apache.org/

[7] Skripti jezik PHP, dostopen na:
http://www.php.net/

[8] Razširjenost jezika PHP, dostopno na:
http://php.net/usage.php

[9] Ogrodje Symfony2, dostopna na:
http://www.symfony.com/

[10] Odprtokodni sistem za urejanje vsebin Drupal, dostopen na:
https://www.drupal.org/

[11] Sistem za spletni forum, dostopen na:
https://www.phpbb.com/

[12] Vstavljanje odvisnosti, dostopno na:
http://stackoverflow.com/questions/130794/what-is-dependency-injection

[13] Odprtokodna relacijska podatkovna baza MySQL, dostopna na:
http://www.mysql.com/

[14] Podatki o tržnem deležu baze MySQL, dostopna na:http://en.wikipedia.org/wiki/MySQL

[15] Skriptni jezik JavaScript, dostopen na:
http://en.wikipedia.org/wiki/JavaScript

[16] Knjižnica za skriptni jezik JavaScript jQuery, dostopna na:
http://www.jquery.com/

[17] Arhitektura REST, dostopna na:
http://en.wikipedia.org/wiki/Representational_state_transfer

[18] Socialno omrežje Facebook, dostopno na:
http://www.facebook.com/

[19] Sistem za spletno plačevanje PayPal, dostopen na:
http://www.paypal.com/

[20] Program za zaporedna dela Cron, dostopen na:
http://en.wikipedia.org/wiki/Cron

[21] Odprtnokodna knjižnica za generiranje datotek Excel za jezik PHP
https://phpexcel.codeplex.com/

[22] Odprtokodno ogrodje wkhtmltopdf, dostopno na:
http://wkhtmltopdf.org/

[23] Ogrodje za testiranje enot PHPUnit, dostopno na:
https://phpunit.de//

[24] Okolje za razvoj spletnih aplikacij WAMP, dostopna na:
http://www.wampserver.com/en/

[25] Doctrine ORM, dostopen na:
http://www.doctrine-project.org/projects/orm.html

[26] Opis protokolov SSL in TLS, dostopen na:
http://en.wikipedia.org/wiki/Transport_Layer_Security

[27] Opis protokola HTTPS, dostopen na:
http://en.wikipedia.org/wiki/HTTP_Secure

[28] Ogrodje za testiranje enot PHPUnit, dostopno na:
https://phpunit.de/

[29] Skaliranje, dostopno na:
http://en.wikipedia.org/wiki/Scalability

[30] Odprtokodna podatkovna baza Redis, dostopna na:
http://redis.io/

[31] Ponudnik gostovanja v oblaku DigitalOcean, dostopen na:
https://www.digitalocean.com/