UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Dejan Pavlovič

Razvoj finančno-računovodske aplikacije z uporabo metode Scrum

DIPLOMSKO DELO

UNIVERZITEZNI ŠTUDIJSKI PROGRAM RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2014

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Dejan Pavlovič

Razvoj finančno-računovodske aplikacije z uporabo metode Scrum

DIPLOMSKO DELO

UNIVERZITEZNI ŠTUDIJSKI PROGRAM RAČUNALNIŠTVO IN INFORMATIKO

MENTOR: izr. prof. dr. Viljan Mahnič

Ljubljana, 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Razvoj finančno računovodske aplikacije z uporabo metode Scrum

Tematika naloge:

Proučite metodo Scrum in jo uporabite za razvoj finančno-računovodske aplikacije za potrebe majhnega start-up podjetja. Prikažite posamezne korake v procesu razvoja od izdelave začetnega seznama zahtev in načrtovanja izdaje do končne realizacije. Opišite uporabljene tehnologije in predstavite delovanje izdelanih programov.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Dejan Pavlovič, z vpisno številko **63060258**, sem avtor diplomskega dela z naslovom:

Razvoj finančno računovodske aplikacije z uporabo metode Scrum

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Viljana Mahniča,
- 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 29. septembra 2014 Podpis avtorja:

Zahvaljujem se mentorju izr. prof. dr. Viljanu Mahniču za pomoč pri izvedbi diplomske naloge, družini za finančno pomoč in potrpežljivost ter prijateljem za podporo in pomoč tekom študija.

Svoji družini.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AJAX	Asynchronous JavaScript and XML	asinhroni JavaScript in XML
CSS	Cascading Style Sheets	kaskadne stilske podloge
EXT JS	Extended Javascript	razširjen JavaScript
FTP	File Transfer Protocol	protokol za prenos datotek
HTML	Hyper Text Markup Language	jezik za označevanje nadbesedila
HTTP	HyperText Transfer Protocol	protokol za prenos informacij na spletu
JSON	JavaScript Object Notation	notacija JavaScript objekta
MVC	Model–View–Controller	model-pogled-krmilnik
MySQL	My Structured Query Language	sistem za upravljanje s podatkovnimi bazami
PDF	Portable Document Format	format datoteke, ki je neodvisen od računalniškega okolja in medoperacijsko prenosljiv
PHP	Hypertext Preprocessor, izvirno pa Personal Home Page Tools	orodja za osebno spletno stran
URL	Uniform Resource Locator	enolični krajevnik vira
XAMPP	X ("cross"-platform), Apache HTTP Server, MySQL, PHP and Perl	X-povezovanje večih platform, Apache http strežnik, MySql, PHP in Perl

Povzetek

V diplomski nalogi je prikazan postopek razvoja finančno-računovodske aplikacije za manjše podjetje po metodi Scrum. Najprej so predstavljene glavne karakteristike agilnih metod, sledi predstavitev metode Scrum s poudarkom na agilnem planiranju s pomočjo uporabniških zgodb. Nato je opisan podroben potek razvoja aplikacije po metodi Scrum in uporabljena spletna orodja. Na koncu je prikazano še delovanje aplikacije. Njene glavne funkcionalnosti lahko razdelimo v štiri skupine – obdelava podatkov o uporabnikih, vodenje projektov, prejetih računov in izdanih računov. Aplikacija je zasnovana za upravljanje s strani administratorja ali pa uporabnika. Prvi ima omogočene vse funkcionalnosti, uporabnik pa ima le opcijo urejanja projektov in osebnih podatkov.

Ključne besede: agilne metode, metoda Scrum, finančna aplikacija, spletne tehnologije

Abstract

The thesis presents the development of a financial accountancy application for small enterprises using Scrum method. Firstly, the thesis introduces the main characteristics of agile methods, followed by a presentation of Scrum method with the emphasis on agile planning by means of user stories. Secondly, a detailed procedure of application development with the Scrum method and online tools is depicted. Finally, the thesis highlights the application's functionalities. The main functions of the application can be divided into four categories: user data processing, project management, overview of received and issued invoices. The application is designed to be operated by an administrator or user. The former can make use of all four functions, while the latter has only the option of editing projects and personal data.

Keywords: agile methods, Scrum method, financial application, online technologies

Poglavje 1 Uvod

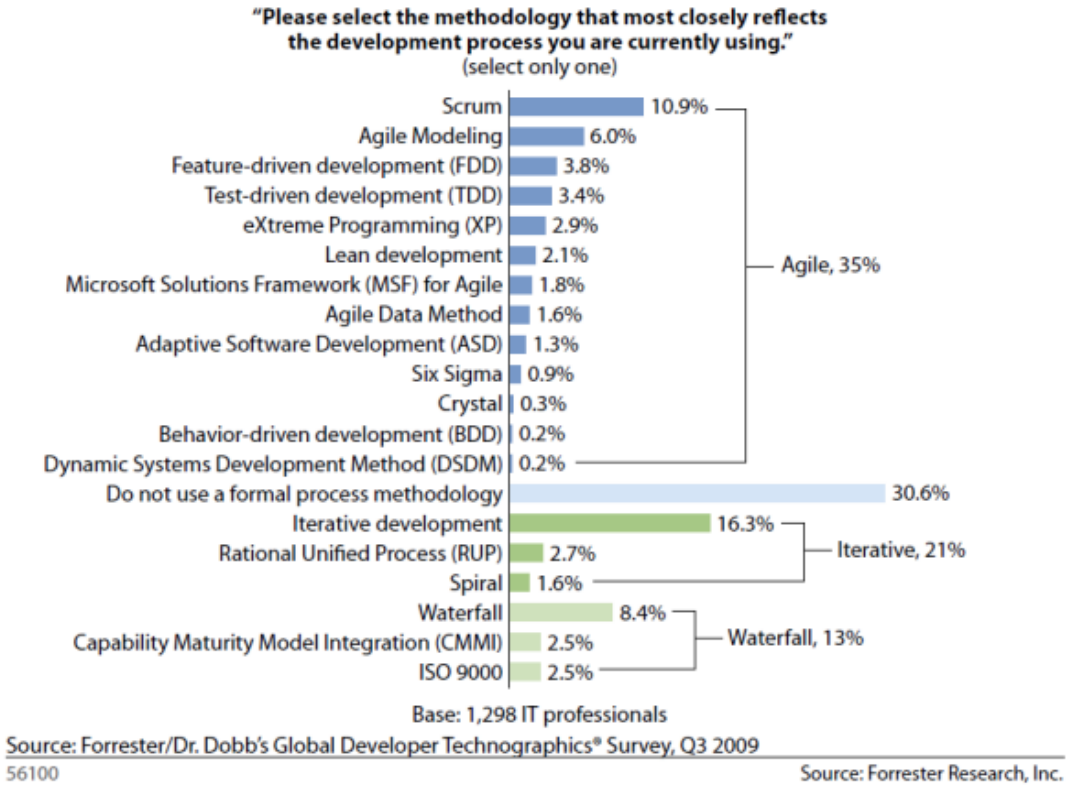
Zamisel o izdelavi finančno računovodskega sistema je prišla na podlagi ideje o morebitni ustanovitvi lastnega manjšega podjetja. Za samo delovanje potrebujemo nek način vodenja dejavnosti in denarnih tokov. V ta namen sicer obstajajo plačljive verzije spletnih aplikacij, vendar smo razmišljali o postavitvi svoje. Prednost lastnega sistema ni le v cenovno ugodnejši rešitvi, gledano dolgoročno, ampak tudi prilagoditvi lastnim potrebam poslovanja. Rešitve, ki jih najdemo na spletu, dostikrat ponujajo univerzalen nabor funkcionalnosti, med katerimi se dostikrat najde kakšna, ki je ne potrebujemo ali celo kakšna, ki jo potrebujemo, vendar ni zajeta oziroma ni implementirana v sistemu. Rezultat praktičnega dela bo torej finančna aplikacija, ki jo lahko v grobem razdelimo na dva dela:

- vodenje projektov in
- vodenje financ

Prvi del vsebuje osnovne podatke o projektu ter artikle in storitve vezane na projekt. Finančni del pa je razdeljen na prejete račune, izdane račune in pregled finančnega stanja podjetja. S temi funkcionalnostmi bi zadovoljili osnovnim potrebam delovanja bodočega podjetja.

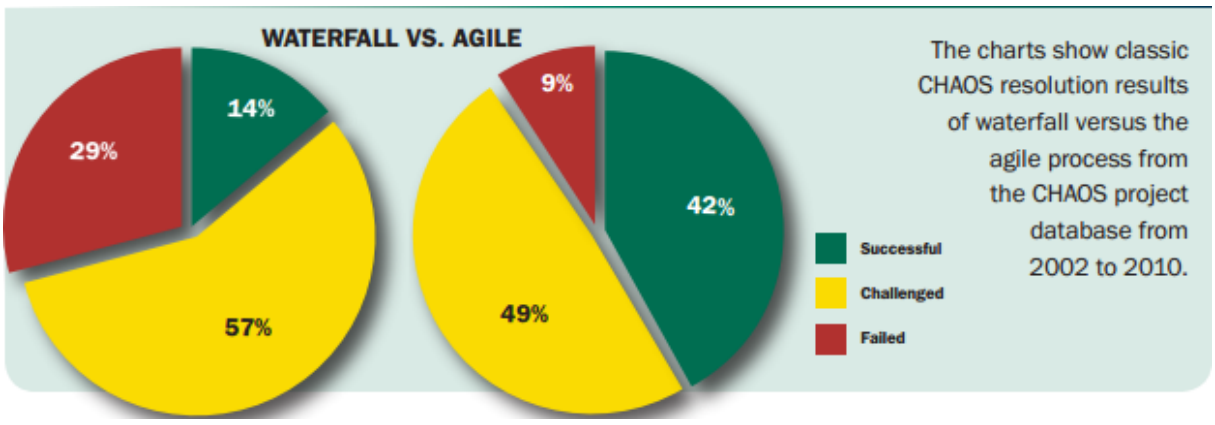
Naslednji korak je bila odločitev o razvojnem postopku, po katerem naj bi ta aplikacija nastala. Pri izdelavi programske opreme danes prevladujeta dva pristopa. Prvi je tradicionalni; njegova glavna značilnost je, da poteka izdelava programske opreme po vnaprej določenem načrtu. Njegovo nasprotje predstavlja agilni pristop. Prednost tega je v sprotnem prilagajanju zahtevam naročnika.

Pri iskanju najbolj primerne rešitve za izdelavo aplikacije smo se odločili za agilni pristop. Študija, ki jo je izdelala svetovalna hiša Forrester Reseach [6], kaže, da je delež uporabe agilnih metod trikrat večji kot pa slapovnih (slika 1.1). Najbolj razširjena metoda pri agilnem pristopu po tej raziskavi je metoda Scrum [6] in zato smo se odločili zanjo.



Slika 1.1: Slika prikazuje delež uporabe razvojnih procesov in posameznih metod glede na proces [6].

Naslednja študija prihaja iz podjetja The Standish Group [7]. V svoji raziskavi je pokazala, da je uspešnost agilnih projektov trikrat večja, neuspešnost projektov pa trikrat manjša kot pri slapovnih. (slika 1.2)



Slika 1.2: Slika prikazuje primerjavo deležev uspešnosti projektov med agilnim in slapovnim razvojnim procesom [7].

V drugem poglavju diplomske naloge sledi teoretična predstavitev agilnih metod. Tu bomo spoznali značilnosti metod na splošno ter predstavili metodo Scrum. V tretjem poglavju bomo spoznali uporabniške zgodbe. V četrtem poglavju bomo opisali potek razvoja naše aplikacije po metodi Scrum. V petem poglavju bo predstavljena implementacija finančne aplikacije, in opis uporabljenih tehnologij, zahtev naročnika in predstavitev same aplikacije. Diplomsko nalogo zaključijo sklepne ugotovitve v šestem poglavju.

Poglavje 2 Agilne metode

Pojem agilnosti se je uveljavil leta 2001 z objavo dokumenta Agile Manifesto [8]. Sedemnajst avtorjev je vanj zapisalo temeljne trditve, s katerimi so uokvirili agilnost. Te trditve so sledeče:

- Pomembnejši so posamezniki in interakcije med njimi kot pa sam proces in orodja. To pomeni, da je bolje imeti pravo razvojno ekipo, ki dobro sodeluje med sabo, kot pa zapletene procedure in drago programsko opremo, ker prava ekipa lahko z dobro komunikacijo in malce iznajdljivosti premosti probleme tudi brez slednjega.
- Delujoča programska oprema je pomembnejša kot izčrpna dokumentacija. Namreč s krajšimi iteracijami in sprotnim izdajanjem produkta se lažje prilagajamo naročniku.
- Pomembnejše je sodelovanje z naročnikom kot pa pogajanja o pogodbi. S sodelovanjem lažje dosežemo skupni cilj, saj s tem povečujemo zaupanje naročnika.
- Odzivanje na spremembe je pomembnejše od sledenja planu. V tej točki si povečamo zaupanje in zadovoljstvo naročnika, ker izdajamo dele produkta, ki so stranki pomembnejši in s tem ne sledimo slepo našemu planu.

2.1 Agilna razvojna skupina

V prejšnjem podpoglavju smo omenili razvojno ekipo oziroma skupino, zato bomo tukaj predstavili, katere karakteristike mora imeti, da zadošča pogojem agilnosti [1]. Prva taka lastnost je, da mora delati kot homogena skupina. To pomeni, da mora imeti skupno vizijo, sodelovati med sabo in pomagati drug drugemu. Druga zelo pomembna lastnost je, da mora delati v kratkih iteracijah (običajno je dolžina iteracije od dva do štiri tedne). S tem dosežejo boljšo komunikacijo z naročnikom, saj naročnik na takšen način hitreje dobi povratne informacije o napredku razvoja produkta. Naslednja karakteristika je, da mora skupina po vsaki iteraciji dostaviti nekaj delujočega. To pomeni, da bi moral biti ob zaključku iteracije nastali del produkta narejen do te mere, da bi ga naročnik lahko uporabljal. Ker pa je običajno sama iteracija prekratko obdobje za lansiranje produkta, se šele po določenem številu le-teh preide v fazo izdaje (do te stopnje se običajno pride v treh do šestih mesecih). Za skupino je pomembno tudi, da se mora usmeriti na prioritete naročnika. Tukaj je pomembno, da skupina razvija zahtevke lastnika izdelka glede na prioriteto, ki jo je lastnik določil za posamezen zahtevek.

Kot zadnjo pomembno lastnost bi omenili še prilagodljivost skupine (naročnik si lahko premisli glede nečesa, plan se med samim razvojem spreminja, itd).

Med predstavitvijo lastnosti skupine smo omenili akterje, ki sodelujejo v procesu razvoja. Zato si podrobneje oglejmo njihove vloge v procesu (predstavili jih bomo po odgovornosti na projektu hierarhično padajoče) [1]:

- Lastnik izdelka (angl. *Product Owner*) je zadolžen za vzdrževanje seznama zahtev in s tem zastopa interese naročnika.
- Naročnik (angl. *Customer*) je oseba, ki financira ali pa kupuje produkt. Pri manjših projektih vlogo naročnika in lastnika izdelka večkrat predstavlja ena oseba.
- Vodja projekta (angl. *Project Manager*) ima vlogo vodenja razvoja projekta.
- Razvijalec (angl. *Developer*) je oseba, ki razvija programsko opremo.

2.2 Agilno planiranje

Pojem agilnega planiranja se vrti okoli planiranja in ocenjevanja zahtevnosti projekta. Ti dve metriki sta zelo pomembni pri razvoju projekta, ker na njiju temeljita uspešnost in hitrost razvoja. S pomočjo njiju okvirno določimo, kdaj bo projekt končan (kreiramo urnik razvoja in rok izdaje) ter koliko nas bo proces stal. Vedno je potrebno zagotoviti optimalno število ljudi na projektu, glede na stroške in čas, ki je potreben za razvoj. Zato na tem mestu omenimo, da se vodja projekta ves čas srečuje s kompromisi, na podlagi katerih mora sprejemati odločitve. Za primer ilustrirajmo dva takšna kompromisa [1]:

- kompromis med funkcionalnostjo in vloženim naporom (na primer, ali se splača posvetiti več časa obliki uporabniškega vmesnika glede na to, da s tem ne povečamo funkcionalnost produkta);
- kompromis med stroški in časom (na primer: ali se splača najeti dva programerja več, da bo projekt končan mesec prej).

Tudi naročniki se srečujejo s kompromisi. Eden takšnih je odločitev med časom razvoja in funkcionalnostjo (na primer: naročniku povemo, da za razvoj vseh funkcionalnosti potrebujemo na primer en mesec, medtem ko bo večina le-teh razvita v polovičnem času).

Torej kakšen plan je agilen? Agilen plan ni klasičen fiksen plan, ampak se spreminja z namenom, da se izognemo napakam iz preteklih izkušenj, ki smo jih pridobili na tem področju (na primer: če se je na preteklih projektih izkazalo, da je najbolje šifrante izdelati na koncu, upoštevamo to izkušnjo tudi pri novem projektu). Zato morajo biti naši plani fleksibilni, da jih

lahko zlahka spreminjamo, ne da bi morali s tem vplivati na datum izdaje [1]. Značilnosti agilnega planiranja so sledeče [1]:

- večji poudarek na planiranju, kot na planu,
- rezultati so plani, ki jih zlahka spreminjamo,
- je razširjen čez celoten projekt,
- ocenjevanje v projektih na začetku je manj natančno kot kasneje,
- je iterativen proces (razvoj projekta je razdeljen v iteracije).

Toda zakaj je agilno planiranje uspešnejše od tradicionalnega? Razlogov je veliko, zato bomo povzeli le najpomembnejše [1]:

- Razvojne skupine se preveč usmerjajo na naloge in ne na končanje neke funkcionalnosti (kar je za naročnika dejansko najpomembnejše). Dokler so naloge narejene v roku, nimamo nobenih problemov. V nasprotnem primeru pride do prezasedenosti urnika, temu pa največkrat kljubuje več dejavnikov [1]:
  - naloge se ne zaključijo predčasno (avtor knjige [1] poda lep primer, ko nekdo zaključi nalogo predčasno, raje preostali čas porabi za nadgradnjo svojega znanja na spletu, kot pa da bi se posvetil novi nalogi);
  - zamuda vpliva na vse nadaljnje aktivnosti v načrtu;
  - aktivnosti so med seboj povezane (to pomeni, da čas za izvršitev ene naloge vpliva na čas razvoja druge naloge).
- Do zamud prihaja tudi, ko nek posameznik izmenično dela na večjem številu nalog (običajno nastane problem pri več kot dveh nalogah).
- Tradicionalno planiranje ne obrodi sadov, ker programerji ne upoštevajo prioritet funkcionalnosti, kot jih vidi naročnik, ampak delajo naloge po vrstnem redu, kot so v planu. Toda ko se čas izteče, morajo zavreči določene funkcionalnosti in pri tem lahko izpadejo pomembnejši zahtevki naročnika.
- Pri tradicionalnem planiranju razvojna ekipa meni, da so funkcionalnosti, ki jih je dobila na začetku, fiksne. Zato zanemarijo možnosti, da si naročnik lahko premisli glede zastavljenega cilja ali pa doda dodatne zahteve. Ta pomanjkljivost je pri agilnih metodah odpravljena z uporabo kratkih iteracij.

Planiranje lahko razdelimo na tri faze. Prva se imenuje planiranje izdaje, pri kateri se načrtuje poraba virov, obseg funkcionalnosti za razvoj ter razpored dela. Za to fazo je značilno, da je časovno najmanj natančno ocenjena, ker načrtujemo za daljno prihodnost. Bolj natančna je

naslednja, ki se imenuje faza iteracije. Na tem mestu načrtujemo funkcionalnosti, ki so prioritetno pomembnejše za naročnika. Zadnja, časovno najbolj natančna, se imenuje faza dnevnega načrtovanja, kjer se s pomočjo dnevnih sestankov določijo naloge, ki morajo biti končane v tekočem dnevu.

2.2.1 Planiranje izdaje

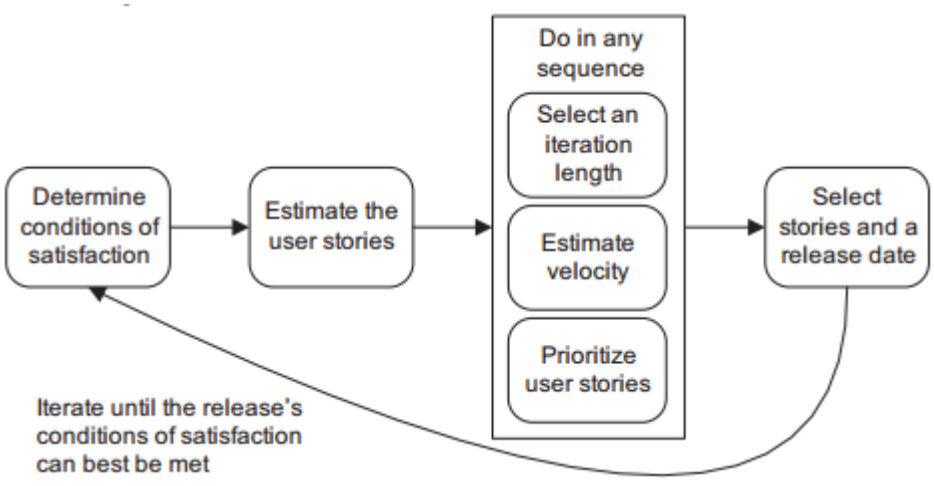
S pomočjo planiranja izdaje lastnik izdelka in razvojna skupina določita, kolikšen del projekta mora biti razvitega in v kakšnem časovnem okvirju. Običajna dolžina okvirja je tri do šest mesecev (ta interval izhaja iz dolžine in števila iteracij). Namen izdaje je povezati iteracije v zaključeno celoto.

Planiranje izdaje lahko začnemo na dva načina:

- z naborom uporabniških zgodb, na podlagi katerih določimo, koliko časa potrebujemo za razvoj;
- z izbiro roka za dokončanje in določanjem koliko uporabniških zgodb lahko končamo v tem času.

Zato lahko rečemo, da je za to stopnjo pomembno le, da se izberejo uporabniške zgodbe (z opisom funkcionalnosti), ki morajo biti dokončane do dogovorjenega datuma. Za načrtovanje nalog in delitev dela je na tem mestu še prezgodaj, ker še ne poznamo vseh podrobnosti.

Proces planiranja se izvaja na sestanku planiranja izdaje (angl. *Release Planning meeting*), kar prikazuje slika 2.1.



Slika 2.1: Slika prikazuje proces planiranja izdaje [1].

Najprej je potrebno določiti pogoje, ki jih mora izpolnjevati plan izdaje (to je kombinacija ciljev kot so urnik, pogled na projekt s strani lastnika in potrebni viri) [1]. Naslednji korak predstavlja ocenjevanje uporabniških zgodb, kar je zelo pomembno za ceno razvoja posamezne zgodbe (na podlagi ocen lastniku produkta lažje predstavimo stroške). V tretjem koraku so zajete tri aktivnosti, ki jih lahko izvedemo v poljubnem zaporedju:

- Izberemo dolžino iteracije, primerno zastavljenemu projektu.
- Ocenimo hitrost razvoja. Za to oceno lahko uporabimo pretekle izkušnje (v primeru, ko je razvojna skupina že sodelovala), tehniko napovedovanja (za vsakega posameznika se izračuna, koliko prostega časa bo imel na voljo in koliko prostih ur je na voljo v eni iteraciji. Zaradi pomanjkanja časa, ki je potreben za razvoj vseh funkcionalnosti, je potrebno uporabniške zgodbe razvrstiti po prioriteti. S tem dosežemo, da se zgodbe, ki so ključnega pomena, razvijejo najprej, manj pomembne zgodbe pa se razvijejo na koncu.

Potem, ko smo definirali vse pogoje, razporedimo uporabniške zgodbe po posameznih iteracijah in s tem dobimo datum izdaje. Če vse ustreza pogojem zadovoljivosti projekta, je plan izdaje končan. V nasprotnem primeru se vrnemo nazaj na ponovno določitev pogojev zadovoljivosti projekta.

2.2.2 Planiranje iteracije

Na sestanku planiranja iteracije (angl. *Iteration Planning Meeting*) ekipa natančneje definira, katere uporabniške zgodbe morajo biti dokončane (zgodbe se izberejo po prioriteti padajoče). V tem koraku razdelimo izbrane zgodbe na naloge, vendar te naloge na tem mestu še ne dodeljujemo posameznim razvijalcem. Izbira nalog za razvijalca se naredi šele, ko se iteracija prične. Na koncu ocenimo zahtevnost dobljenih nalog (vsaki nalogi se določi potrebno število idealnih ur) in s tem se planiranje iteracije zaključi.

Toda, kaj je glavna razlika med planom izdaje in iteracije? Poglejmo si tri ključne kriterije, v katerih se razlikujeta [1]:

- čas trajanja (iteracija lahko traja od enega do štirih tednov, za izdajo pa je potrebno od tri do devet mesecev),
- osnovna postavka (pri iteraciji je to naloga pri izdaji pa uporabniške zgodbe),
- enota za ocenjevanje (za iteracijo uporabljamo idealne ure, za izdajo pa točke uporabniških zgodb ali pa idealne dni).

2.3 Ocenjevanje zahtevnosti projekta

Zahtevnost projekta se ocenjuje na dva načina:

- s točkovanjem uporabniških zgodb,
- idealnimi dnevi.

Če hočemo uporabiti ocenjevanje, moramo zahteve naročnika najprej pretvoriti v uporabniške zgodbe (več o uporabniških zgodbah bomo povedali v naslednjem poglavju). Na tem mestu povejmo le, da so točke uporabniških zgodb enota za merjenje zahtevnosti določene zgodbe. Pomembno je, da vemo, katere uporabniške zgodbe so večje in zahtevnejše od drugih. Ocena v idealnih dneh pomeni, da bo razvijalec ves čas, ki ga bo imel na voljo, namenil razvoju določene funkcionalnosti, ne da bi mu kdo odvračal pozornost (na primer naloge, ki niso povezane s projektom, sestanki itd).

Oba pristopa imata svoje prednosti in slabost. Najprej si oglejmo prednosti točkovanja uporabniških zgodb [1]:

- Prednost v vodenju večfunkcijske skupine (razvijalci, oblikovalci, testerji itd). Ker se število točk nanaša na določeno zgodbo in ker so točke nedeljive na posameznika, se razvojna skupina bolj poglobi v to, kaj bo kdo počel. Pri idealnem številu dni pa vsakega zanima le, koliko od teh dni pripada njemu.
- Točke zgodbe ostajajo enake. Pri ocenjevanju zgodbe s točkovnim pristopom enako zgodbo (pri dveh ločenih projektih) ocenimo z enakim številom točk. Pri idealnih dnevih pa zaradi izkušenj, ki smo jih dobili na prvem projektu, običajno na drugem ocenimo zgodbo z manjšim številom idealnih dni.
- Točke so pravo merilo obsežnosti zgodbe, saj z njimi dejansko ocenimo zahtevnost zgodbe, ne da bi bili pristransko vezani na koledar, kot je to značilno za idealne dni. Zato lahko rečemo, da so točke abstraktna ocena, ki je ne moremo poenotiti z že obstoječimi merskimi enotami (na primer koledarskim časom).
- Vsakdo ima drugačno hitrost razvoja, zato merjenje v idealnih dneh lahko tu predstavlja težavo.

Sedaj pa si oglejmo še prednosti idealnih dni [1]:

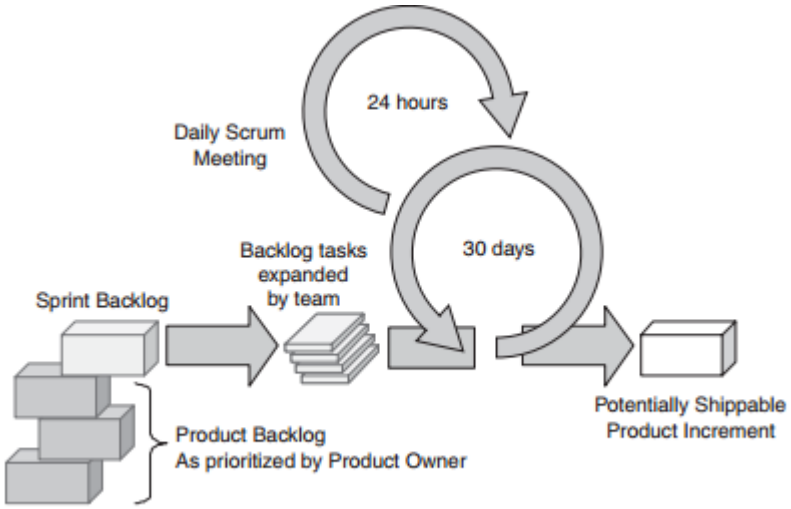
- Lažja predstava za nekoga, ki ni del razvojne skupine. Zahtevnost v točkah je težko razložiti laiku, ker so točke enota, ki je ne moremo povezati z nobeno že obstoječo enoto. Na drugi strani pa imamo idealne dni, ki jih lahko interpretiramo kot število dni, potrebnih za razvoj določene funkcionalnosti, če ves čas delamo samo na tem.
- Začetni ekipi je lažje ocenjevati v idealnih dneh.

2.4 Metodologija Scrum

Scrum je inkrementalen in iterativen proces [2]. Zanj veljajo vse značilnosti, ki smo jih do sedaj omenili v tem poglavju, le da za poimenovanje dogodkov in osebja uporablja svojo terminologijo.

Inkrementalnost procesa pomeni grajenje in izdajanje produkta po delih. Vsak del predstavlja zaključeno celoto funkcionalnosti, ki jo predamo naročniku v uporabo. Pomembno je, da je takšen del dobro testiran, da se po zaključku iteracije z njim ne ukvarjamo več. Iterativni proces lahko definiramo kot gradnjo sistema v enako dolgih intervalih (čas trajanja iteracije). V metodologiji Scrum se iteracija imenuje *Sprint*, načrtovanje le-te pa *Sprint Planning Meeting*.

Ker smo predstavili lastnosti agilnih metod že v drugem poglavju (v tretjem poglavju bomo predstavili uporabniške zgodbe, ki so prav tako del metode Scrum), se bomo tukaj osredotočili le na terminologijo in proces razvoja projekta po metodi Scrum (Slika 2.2).



Slika 2.2: Slika prikazuje proces razvoja produkta po metodi Scrum [2].

Najprej zberemo funkcionalnosti, ki jih želi imeti lastnik izdelka, in jih pretvorimo v uporabniške zgodbe (v Scrum-u se seznam zahtev imenuje *Product Backlog*). Nato sledi planiranje iteracije (angl. *Sprint Planning Meeting*), kjer razvojna skupina in lastnik izdelka določita zgodbe za implementacijo v iteraciji (nabor zgodb za iteracijo se imenuje *Sprint Backlog*). Na koncu tega sestanka skupina razbije zgodbe na naloge (angl. *Sprint Tasks*) in s tem se sestanek zaključi. Sledi izvajanje iteracije, kjer se člani skupine udeležujejo dnevnih sestankov (angl. *Daily Scrum Meeting*). Sestanek traja okoli 15 minut z namenom, da vsak član razvojne skupine odgovori na naslednja vprašanja:

- Kaj sem naredil od prejšnjega dnevnega sestanka?
- Kaj nameravam narediti do naslednjega dnevnega sestanka?
- Ali sem med razvojem naletel na kakšne težave?

Ob koncu iteracije se skupina znova sestane, da pregleda uspešnost in rezultate iteracije (angl. *Sprint Review Meeting*). Na njem skupina prikaže lastniku razviti del produkta. Gre za demonstracijo funkcionalnosti v obliki dejanske uporabe in ne v obliki poročila. Če je šel razvoj po planu, je razviti del sistema primeren za predajo naročniku. Ta trditev izhaja iz ene izmed lastnosti iteracije, ki pravi, da ob koncu iteracije dobimo novo funkcionalnost produkta, ki jo lahko lastnik neposredno uporabi (angl. *Potentially Shippable Product Increment*).

Poglejmo si še, kdo sestavlja Scrum skupino. Člani skupine imajo lahko eno izmed naslednjih vlog:

- Lastnik izdelka (angl. *Product Owner*) je oseba, ki predstavlja naročnika. Zadolžen je za vizijo projekta, postavitev prioritet uporabniškim zgodbam, določanje datumov izdaje, vodenje projekta itd.
- Skrbnik metodologije (angl. *Scrum Master*) pomaga skupini pri izvajanju procesa (skrbi za doslednost izvedbe procesa po metodi Scrum).
- Razvojna skupina (angl. *Team*) je zadolžena za implementacijo produkta. Sestavlja jo od 5 do 9 članov. Njena značilnost je samoorganizacija, kar pomeni, da se sami dogovorijo, kdo bo izvajal katero nalogo.

Poglavje 3 Uporabniške zgodbe

Uporabniška zgodba opisuje funkcionalnost, ki jo bo imel produkt na način, ki je razumljiv lastniku izdelka in razvijalcu [2]. Sestavljena je iz treh delov [2]:

- opis zgodbe, s katerim si pomagamo pri planiranju (služi tudi kot opomnik, saj nam pomaga, da česa ne pozabimo);
- komentarji, ki nastajajo ob pogovoru o zgodbi in nam pomagajo izluščiti detajle, ki so pomembni za razvijalce (z njihovo pomočjo odpravljamo nejasnosti);
- sprejemni testi nam pomagajo določiti, kdaj je zgodba zaključena.

3.1 Značilnosti uporabniških zgodb

Oglejmo si glavne karakteristike, ki jih ima dobra uporabniška zgodba. Predstavili bomo naslednjih šest atributov [2]:

- neodvisnost,
- majhnost,
- omogočati mora pogajanje,
- biti mora koristna za uporabnike ali naročnike,
- mora se dati oceniti,
- mora se dati testirati.

Neodvisnost

Pri zgodbah, ki so odvisne med sabo, lahko naletimo na probleme pri določanju prioritete in posledično pri samem planiranju. Posledice slednjega nas pripeljejo do težjega ocenjevanja zahtevnosti. Odvisnost lahko rešimo na dva načina [2]:

- z združevanjem odvisnih zgodb v eno večjo, ki bo neodvisna od ostalih zgodb (v tem primeru je potrebno paziti, da novonastala zgodba ne postane preveč obsežna in s tem porabi preveč časa za razvoj),
- zgodbi, ki sta odvisni, vstavimo v isto iteracijo.

Majhnost

»Prava« velikost zgodbe je za vsako razvojno skupino drugačnega pomena. Odvisna je od sposobnosti, časa skupnega sodelovanja skupine in tehnologije, ki jo uporablja. Poleg zgodb »pravih« velikosti poznamo še:

- Premajhne (to so zgodbe, kjer bi porabili več časa za razglabljanje, kot pa za implementacijo). Ta problem lahko rešimo z združevanjem zgodb.
- Preobsežne zgodbe običajno porabijo veliko razvojnega časa. Posledično lahko zgodba postane neobvladljiva (več zaporednih težav, ki se pojavijo v večjih zgodbah) in zato težko pravočasno zaključimo iteracijo. Temu problemu se lahko izognemo z razbitjem zgodbe na več manjših.

Omogočiti mora pogajanje

Ta karakteristika pomeni, da zgodba ni mišljena kot pogodba med razvijalci in naročnikom, ampak se med pogovorom dopolnjuje in spreminja. Zgodba, napisana na kartici (to je kos papirja z vsemi informacijami o zgodbi) predstavlja opomnik in ker je sproten pogovor med razvijalci in naročnikom pomembnejši od izčrpne dokumentacije, zgodba vedno pušča prostor za ideje in spremembe. Najbolj izrazit del kartice, ki odraža rezultat »pogajanj«, so komentarji (imajo vlogo dodatnih pojasnil k zgodbi in olajšajo nadaljevanje pogovora med naročnikom in razvijalci, saj z njihovo pomočjo vedo, kje so ostali).

Vrednosti za uporabnike ali naročnike

Najprej moramo pojasniti, da imata uporabnik in naročnik različen pogled na produkt. Kar je ključnega pomena za naročnika, ni vedno ključno za uporabnika. Za primer vzemimo uporabnost. Uporabniku je pomembno, da produkt na njegovem računalniku deluje brezhibno, naročnik pa običajno kupuje produkt za večje število uporabnikov. V tem primeru ni dovolj, da produkt deluje idealno le na enem računalniku (na primer na eni vrsti operacijskega sistem ali enemu tipu brskalnika, če gre za spletni produkt), ampak na vseh ostalih, na katerih delajo uporabniki. Iz tega sledi, da ima naročnik širši pogled na uporabnost zgodb kot uporabnik, zato je smiselno prepustiti pisanje zgodb naročniku.

Mora se dati oceniti

Za razvijalce je pomembno, da se zgodbo lahko oceni (določitev velikosti oziroma število potrebnih dni za razvoj). Če se zgodbe ne da oceniti, je to ponavadi zaradi enega izmed naslednjih razlogov [2]:

- razvijalci imajo premalo znanja o področju, na katerem razvijajo produkt (nerazumevanje zgodbe lahko odpravijo z dodatnimi pogovori z naročnikom)

- razvijalci nimajo zadostnega tehničnega znanja (predvsem je tu mišljeno nepoznavanje razvojne tehnologije, s katero razvijajo produkt);
- preobsežne zgodbe (rešitev predstavlja razbitje zgodbe na manjše).

Zgodba se mora dati testirati

S testiranjem ugotavljamo, ali je zgodba pravilno implementirana. Brez testov bi zgodba zadovoljila programerskim in ne testnim zahtevam. V ta namen se uporabljajo sprejemni testi (kratka navodila za testerje), ki so zapisani na zadnji strani kartice. Lahko jih dodajamo kadarkoli, dokler zgodba ni zaključena. Pri testiranju je treba preveriti vse pogoje (teste) in če so vsi testi uspešni, je zgodba zaključena.

3.2 Zbiranje uporabniških zgodb

V pogovoru z naročnikom poskušamo določiti uporabniške zgodbe. Na začetku se osredotočimo na velike zahtevke, saj s tem dobimo grobo predstavo o produktu. Nato pa gremo proti vedno manjšim, dokler ne pridemo do detajlov in s tem do jasnega pogleda na produkt. Za agilne projekte je značilno, da ne moremo zajeti vseh zahtev na začetku [2]. Zato se uporabniške zgodbe formirajo tudi med razvojem.

Pri zbiranju zgodb igrajo pomembno vlogo izkušnje, saj oseba, ki zbira zahtevke že dolga leta, hitreje izlušči zgodbe kot nekdo, ki je še novinec na tem področju.

Za oblikovanje zgodb obstajajo pristopi, ki jih lahko uporabljamo iterativno [2]. Pomembno je, da so nevpadljivi ter jih lahko uporabljamo kadarkoli (skozi celoten razvojni čas produkta). Oglejmo si štiri pristope za oblikovanje zgodb [2]:

- intervju z uporabniki,
- vprašalniki,
- opazovanje,
- delavnice za pisanje zgodb.

Intervju z uporabniki

Je najbolj uporabljen pristop. Ključna pri tem pristopu je izbira pravih uporabnikov za intervju (poskušamo izbrati uporabnike čim širšega spektra, saj dobimo s tem pogled na projekt z različnih zornih kotov). Paziti moramo, da vprašanja, na katera lahko odgovarja le z DA ali NE, postavljamo zelo redko, saj s tem zapremo prostor za obsežnejši odgovor. Na začetku poskušamo zastavljati vprašanja, ki niso vezana na kontekst (na primer:

- V katerih predelih aplikacije se vam zdi iskalnik najbolj primeren?

- Namesto kako pomemben se vam zdi iskalnik v pregledu projektov?),

ker na ta način lažje ugotovimo, kaj je za uporabnika pomembnejše.

Vprašalniki

Vprašalniki so primerni za zbiranje informacij o zgodbah, ki jih že imamo. Za nove zgodbe so neprimerni, ker smo z vprašalnikom omejeni z vnaprej določenimi vprašanji. S tem posledično ne moremo slediti interesu uporabnika kot pri intervjuju. Pri večjem številu uporabnikov s to tehniko zgodbam lažje postavljamo prioritete.

Pri sestavljanju vprašanj se moramo odločiti med tekstovnimi in izbirnimi odgovori. S tekstovnimi odgovori dobimo širši pogled uporabnika na določeno temo, vendar s tem otežimo grupiranje odgovorov. Pri izbirnih odgovorih dobimo nasproten efekt. Izbor tipa vprašanj je torej odvisen od cilja, ki ga želimo doseči.

Opazovanje

Gre za opazovanje uporabnikov pri uporabi produkta. Ta pristop se uporablja bolj poredko, ker je težko najti priložnost za opazovanje. Največkrat pride v poštev pri razvoju produkta za lastno podjetje. Z uporabo te tehnike lahko zmanjšamo čas, potreben za izdajo produkta (končni uporabniki so najbolj primerni testerji produkta, ker šele z opazovanjem le-teh vidimo uporabnost razvitega produkta).

Delavnice za pisanje zgodb

Gre za sestanek vseh sodelujočih akterjev pri oblikovanju zgodb (razvijalci, uporabniki, testerji, naročnik itd.). Na sestanku se najprej napišejo uporabniške zgodbe, prioritete pa jim kasneje določi naročnik. Ker je bistvo sestanka soočanje idej (angl. *brainstorming*), se posledično proizvede veliko število zgodb v zelo kratkem času (poudarek je na kvantiteti in ne na kvaliteti). Na podlagi zgodb se nato gradi prototip produkta (risanje komponent sistema in medsebojnih povezav na papir).

3.3 Uporabniške vloge

Uporabniško vlogo predstavlja skupina uporabnikov s podobnimi lastnostmi, ki za svoje delo uporabljajo skupen nabor funkcionalnosti v sistemu. Postopek definiranja uporabniških vlog lahko opišemo v štirih korakih [2]:

- Zbiranje idej za začetni nabor uporabniških vlog. Naročnik in razvijalci se sestanejo v sejni sobi. Pripravijo kartice, na katere bodo pisali uporabniške vloge. Pomembno je, da pri zbiranju idej sodelujejo vsi (s tem zmanjšamo verjetnost, da bi pozabili na katero izmed vlog). Ko nekdo definira novo vlogo, jo poimenuje in kartico odloži na sredino

mize. V tej stopnji se zbirajo le nazivi uporabniških vlog, zato ta stopnja ni namenjena razglabljanju o vlogah.

- Zbiranje začetnega nabora. Po zaključenem zbiranju uporabniških vlog je le-te potrebno kategorizirati. Skupina razvrsti kartice glede na to, koliko se vloge prekrivajo med sabo. V primeru, ko sta vlogi med sabo zelo podobni, se kartici skorajda prekrivata, če pa sta si vlogi popolnoma različni, sta kartici odmaknjeni druga od druge.
- Konsolidacija vlog, kjer poskušamo zmanjšati nabor uporabniških vlog z združevanjem podobnih. Osebe, ki so zapisale prekrivajoče vloge (kartice se skoraj prekrivajo), razložijo njihov pomen. Če gre za identične vloge ,se določi naziv, ki bo skupen vsem prekrivajočim, ostale se zavržejo. Skupina mora prediskutirati smisel vseh vlog. Če neka vloga nima pomena za sistem, jo zavrže. Ob zaključku strnjevanja skupina ponovno razvrsti vloge glede na relacije med njimi.
- Izpopolnjevanje vloge, kjer definiramo atribute posamezne uporabniške vloge. Atribut je lastnost, po kateri se uporabniška vloga razlikuje od ostalih.

Poglavje 4 Potek razvoja aplikacije po metodi Scrum

V razvojnem procesu aplikacije smo se morali postaviti v vse vloge, ki nastopajo v skupini Scrum. Zato najprej predstavimo funkcije, ki smo jih opravljali v posamezni vlogi.

4.1 Delo v posamezni vlogi

Lastnik izdelka

Naša glavna naloga v tej vlogi je bila skrb za vsebinsko plat projekta (svojo vizijo produkta smo skušali karseda natančno predstaviti razvojni skupini). V pogovoru z razvojno skupino smo ustvarili seznam zahtev, ki smo jih pretvorili v uporabniške zgodbe. Naša naslednja naloga je bila postavitev prioritet posameznim zgodbam (zgodbe, ki so bile pomembnejše za nas in smo jih želeli implementirati v zgodnji fazi razvoja, so dobile višjo prioriteto kot ostale). S prioritetami smo določili, kaj želimo imeti razvito v sledeči iteraciji. Razvojni skupini smo v fazi planiranja iteracije razložili podrobnosti zgodb, ki so bile vključene v iteracijo. Ob tem smo podali še sprejemne teste za posamezno zgodbo, na podlagi katerih je razvojna skupina ugotavljala, ki ali je določena zgodba zaključena uspešno ali ne.

Skrbnik metodologije

V vlogi skrbnika metodologije smo vodili agilno skupino (vodenje vseh sestankov, učenje ocenjevanja itd.) ter skrbeli, da je proces razvoja potekal po metodi Scrum. Nadzorovali smo napredek razvojne skupine (ali gre vse po planu ali prihaja do zaostanka zaradi določenih težav) in pomagali razvojni skupini, da je dosegla zastavljen cilj (uspešno izvedena iteracija). Za lastnika je zelo pomembna kakovost izdelka. Če bi bil namreč izdelek slabo narejen, bi bila krivda na naši strani (slaba kakovost izdelka je posledica slabo vodenega procesa, zato je pri pojavljanju pomanjkljivosti pomembno nemudoma ugotoviti, kje in zakaj se pojavljajo in jih eliminirati).

Razvojna skupina

Zadnja funkcija, ki smo jo opravljali, je bila razvojna skupina. Lastnik izdelka nam je najprej predstavil uporabniške zgodbe, ki smo jih nato ocenili (oceno uporabniških zgodb smo določili na podlagi zahtevnosti zgodbe). Pred vsako iteracijo smo se sestali z lastnikom. Ta je dodelil prioritete zgodbam, na podlagi tega pa smo potem izbirali zgodbe za posamezne iteracije. Še

pred izbiro zgodb za prvo iteracijo smo se odločili, da bo prva iteracija dolga 14 dni. Ko smo imeli seznam zahtev za iteracijo, nam je lastnik predstavil teste sprejemljivosti za vsako zgodbo. Naslednja naloga, ki smo opravili, je bila določitev nalog za posamezno zgodbo. Ker smo prvi dan iteracije definirali naloge in jim določili predviden čas razvoja, smo z dejanskim programiranjem začeli šele naslednji dan. Med iteracijo smo imeli dnevne sestanke, kjer smo poročali o napredku od prejšnjega dne, se dogovarjali za naslednji dan, ponovno ocenili naloge, ki so ostale nedokončane od prejšnjega dne, in se pogovorili o problemih, s katerimi smo se srečali med razvojem.

4.2 Predstavitev dela po iteracijah

To je bil to naš prvi projekt, ki smo ga delali po metodi Scrum, in ker nismo bili vešči v določanju hitrosti razvoja, smo se odločili, da bomo le-to določili na podlagi prve iteracije. Za prvo iteracijo smo predvideli, da bi lahko realizirali zgodbe v obsegu od 30 do 40 točk. Izbrali smo najbolj optimistično oceno in tako prišli do ocene hitrosti 40. Zato si v tem podpoglavju oglejmo potek razvoja po posameznih iteracijah.

4.2.1 Prva iteracija

V prvi iteraciji je lastnik izdelka določil najvišjo prioriteto naslednjim zgodbam (poleg vsake zgodbe bomo zapisali, s koliko točkami je bila ocenjena. Točke uporabniških zgodb so določene glede na zahtevnost zgodbe. Za ocenjevanje smo uporabili vnaprej določene vrednosti 1, 2, 3, 5, 8, 13 in 21, ki sovpadajo s Fibonaccijevimi števili:

- prijava v aplikacijo (UZ01 - to je šifra zgodbe, ki smo jo uporabili v tabeli časovnega razvoja na sliki 4.1), število točk: 8,
- dodajanje projekta (UZ02), število točk: 8,
- urejanje projekta (UZ03), število točk: 3,
- dodajanje artikla (UZ04), število točk: 5,
- urejanje artikla (UZ05), število točk: 3,
- brisanje artikla (UZ06), število točk: 1,
- dodajanje storitve (UZ07), število točk: 8,
- urejanje storitve (UZ08), število točk: 3,
- brisanje storitve (UZ09), število točk: 1,
- dodajanje uporabnikov v aplikacijo (UZ10), število točk: 8 in

- vzdrževanje šifrantov (UZ11), število točk: 3.

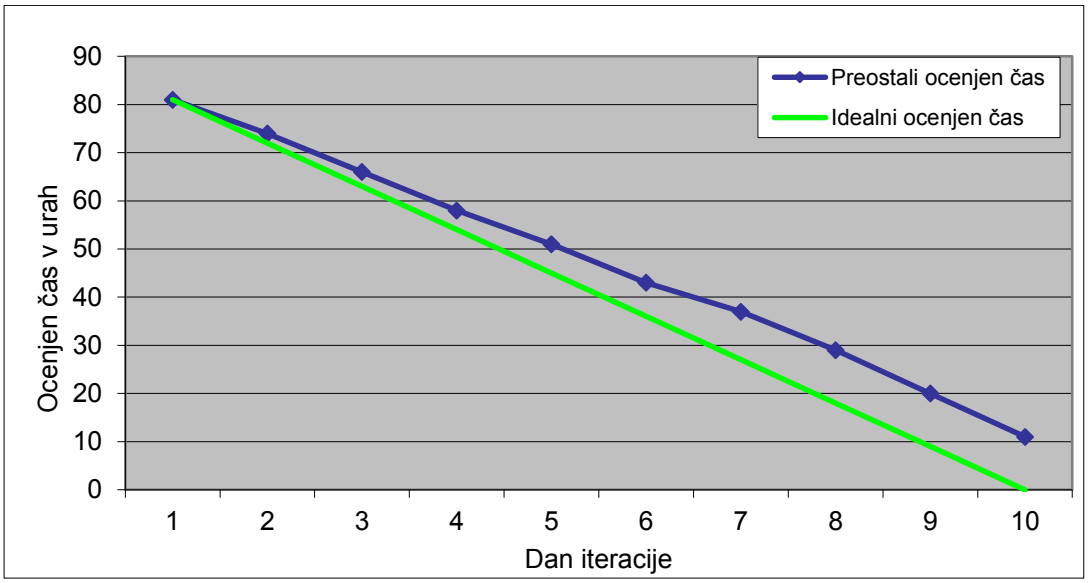
Na spodnji sliki 4.2 je prikazana tabela ocen in dejanske porabe časa za posamezno nalogo. Stolpec s številko 1 v stolpcu *Preostali ocenjen čas* predstavlja (za vsak dan posebej) oceno, koliko ur dela je še ostalo za dokončanje posameznih nalog (delitev zgodbe na naloge in ocenjevanje nalog smo izvedli prvi dan sestanka iteracije). Kljub temu, da smo izbrali dolžino iteracije 14 dni, smo v tabeli prikazali desetdnevni razvoj (delo je potekalo od ponedeljka do petka, poskušali smo se držati osem urnega delovnika). Iz tabele je razvidno, da nismo uspeli implementirati zgodbe *Dodajanje uporabnikov*. Ker ni razvidno, zakaj je prišlo do tega, smo za ilustracijo dodali še graf (slika 4.2), ki prikazuje idealno in dejansko oceno časa.

ID Zgodbe	Opis nalog	Porabljen čas v urah										Preostali ocenjen čas									
		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
UZ01	Kreiranje tabele user	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
UZ01	Izdelava prijavne forme	0	4	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
UZ01	Postavitev seje	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
UZ01	Odjava uporabnika iz sistema	0	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
UZ02	Kreiranje pregleda projektov	0	0	5,5	0	0	0	0	0	0	0	6	6	0	0	0	0	0	0	0	0
UZ02	Kreiranje povezave na formo	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
UZ02	Kreiranje forme za dodajanje projekta	0	0	1,5	5	0	0	0	0	0	0	7	7	6	0	0	0	0	0	0	0
UZ11	Kreiranje in polnjenje tabele company	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
UZ02	Validacija vnosnih polj	0	0	0	2	1	0	0	0	0	0	2	2	2	1	0	0	0	0	0	0
UZ02	Testiranje dodajanja nove projekta	0	0	0	0	3	0	0	0	0	0	3	3	3	3	0	0	0	0	0	0
UZ03	Prenos podatkov iz baze v formo	0	0	0	0	2	0	0	0	0	0	2	2	2	2	0	0	0	0	0	0
UZ03	Validacija podatkov pred shranjevanjem	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
UZ03	Testiranje urejanja projekta	0	0	0	0	1	1,5	0	0	0	0	2	2	2	2	2	0	0	0	0	0
UZ04	Kreiranje tabele project_item	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	0	0	0	0
UZ04	Kreiranje pregleda artiklov	0	0	0	0	0	3,5	0	0	0	0	3	3	3	3	3	0	0	0	0	0
UZ11	Kreiranje in polnjenje tabele item	0	0	0	0	0	1,5	0	0	0	0	1	1	1	1	1	0	0	0	0	0
UZ04	Kreiranje forme artikel	0	0	0	0	0	0,5	4	0	0	0	4	4	4	4	4	3	0	0	0	0
UZ04	Validacija vnosnih polj	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0
UZ04	Testiranje dodajanja artikla k projektu	0	0	0	0	0	0	3	1	0	0	3	3	3	3	3	3	1	0	0	0
UZ05	Prenos podatkov iz baze v formo	0	0	0	0	0	0	0	2,5	0	0	2	2	2	2	2	2	2	0	0	0
UZ05	Validacija podatkov pred shranjevanjem	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0
UZ05	Testiranje urejanja artikla	0	0	0	0	0	0	0	2	0	0	2	2	2	2	2	2	2	0	0	0
UZ06	Odstrani artikel vezan na projekt	0	0	0	0	0	0	0	0,5	0	0	1	1	1	1	1	1	1	0	0	0
UZ07	Kreiranje tabele project_service	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0
UZ11	Kreiranje in polnjenje tabele period	0	0	0	0	0	0	0	0	0,5	0	1	1	1	1	1	1	1	1	0	0
UZ11	Kreiranje in polnjenje tabele item	0	0	0	0	0	0	0	0	1,5	0	1	1	1	1	1	1	1	1	0	0
UZ07	Kreiranje pregleda storitev	0	0	0	0	0	0	0	0	2,5	0	2	2	2	2	2	2	2	2	0	0
UZ07	Kreiranje forme storitev	0	0	0	0	0	0	0	0	3	0	4	4	4	4	4	4	4	4	0	0
UZ07	Validacija vnosnih polj	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0
UZ07	Testiranje dodajanja storitve k projektu	0	0	0	0	0	0	0	0	0	2,5	3	3	3	3	3	3	3	3	3	0
UZ08	Prenos podatkov iz baze v formo	0	0	0	0	0	0	0	0	0	1	2	2	2	2	2	2	2	2	2	0
UZ08	Validacija podatkov pred shranjevanjem	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
UZ08	Testiranje urejanja storitve	0	0	0	0	0	0	0	0	0	2,5	2	2	2	2	2	2	2	2	2	0
UZ09	Odstrani storitev vezano na projekt	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
UZ10	Kreiranje pregleda uporabnikov	0	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3	3
UZ10	Kreiranje forme za dodajanje uporabnika	0	0	0	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4
UZ10	Validacija vnosnih polj	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2
UZ10	Testiranje dodajanja uporabnika glede na vlogo	0	0	0	0	0	0	0	0	0	0	3	2	2	2	2	2	2	2	2	2
Skupaj		**0**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8,5**	**8**	**81**	**74**	**66**	**58**	**51**	**43**	**37**	**29**	**20**	**11**

Slika 4.1: Tabelo porabe in ocene časov za naloge v prvi iteraciji.

Kot je razvidno iz grafa na sliki 4.2, smo se z našim ocenjevanjem vsak dan malo oddaljili od idealnega poteka iteracije. Razlog je bil slabo poznavanje uporabljenih tehnologij. Zaradi tega so se pojavljale tehnične težave. Ker smo vsak dan ponovno ocenjevali naloge, ki jih prejšnji dan nismo dokončali, so dobile na koncu posamezne naloge skupno precej višje ocenjene čase, kot pa smo jih določili na začetku iteracije. Na primer: naloga *Testiranje dodajanja projekta* je

imela četrti dan oceno 2 uri, ko pa smo peti dan porabili eno uro za testiranje, smo naslednji dan zopet ocenili, da naloga potrebuje še dve uri za testiranje, saj smo prvi dan naleteli na semantične napake pri testiranju dodajanja projekta).



Slika 4.2: Graf idealne in preostale in ocene časa v prvi iteraciji.

4.2.2 Druga iteracija

V drugi iteraciji je naročnik postavil najvišjo prioriteto izdanim računom. Hitrost te iteracije smo določili na podlagi izkušenj iz prve. Ker smo imeli v prvi iteraciji hitrost razvoja 32 točk uporabniških zgodb in ker smo predvidevali, da se hitrost razvoja v prvih treh iteracijah povečuje, smo se odločili, da bomo ocenili hitrost druge iteracije na 37 točk uporabniških zgodb. Zato smo oblikovali naslednji seznam uporabniških zgodb glede na prioriteto:

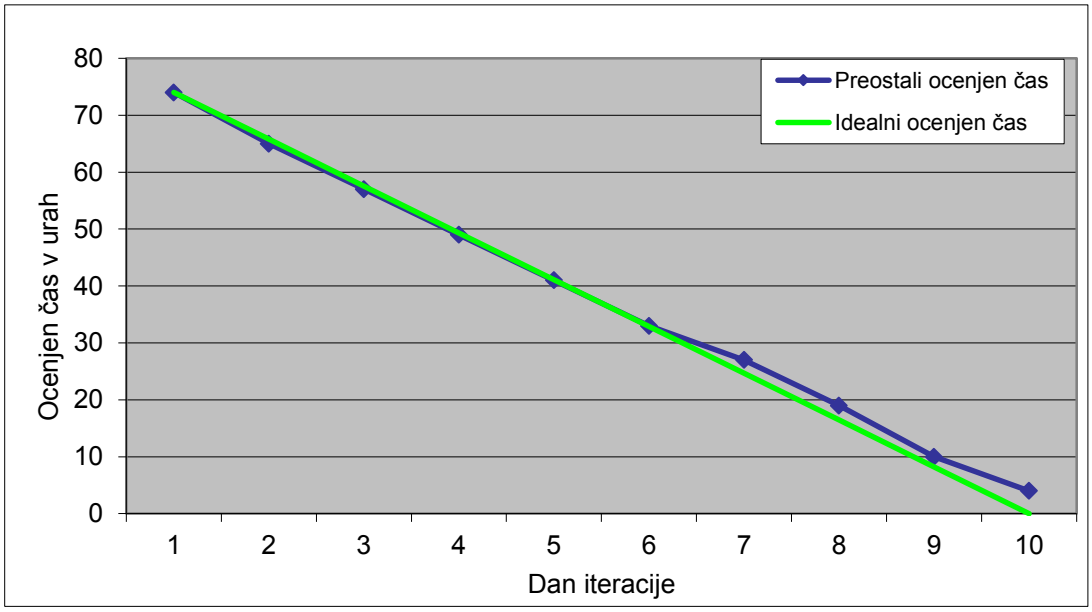
- dodajanje izdanega računa (UZ12), število točk: 13,
- urejanje izdanega računa (UZ13), število točk: 5,
- brisanje izdanega računa (UZ14), število točk: 3,
- kreiranje PDF (UZ15), število točk: 5,
- dodajanje uporabnikov v aplikacijo (UZ10), število točk: 8 in
- urejanje uporabnikov (UZ16), število točk: 3.

Na sliki 4.3, ki prikazuje tabelo porabljenega in ocenjenega časa dela na nalogah, vidimo, da smo bili v drugi iteraciji uspešnejši kot v prvi. Največ problemov smo imeli pri zgodbi kreiranja

PDF-ja, kar je lepo razvidno na sliki 4.4, ki prikazuje graf idealnega in preostalega ocenjenega časa.

ID Zgodbe	Opis nalog	Porabljen čas v urah										Preostali ocenjen čas									
		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
UZ12	Kreiranje tabele sent_invoice	0	1,5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
UZ12	Kreiranje pregleda izdanih računov	0	5	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0
UZ12	Kreiranje povezave na formo za dodajanje rač.	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
UZ12	Kreiranje forme za dodajanje računa	0	0,5	5,5	0	0	0	0	0	0	0	6	5	0	0	0	0	0	0	0	0
UZ12	Kreiranje pregleda artiklov na formi izd. rač.	0	0	2,5	0	0	0	0	0	0	0	3	3	0	0	0	0	0	0	0	0
UZ12	Kreiranje forme za dodajanje artiklov na rač.	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
UZ12	Testiranje dodajanja artiklov na račun	0	0	0	3	0	0	0	0	0	0	2	2	2	0	0	0	0	0	0	0
UZ12	Brisanje artikla iz računa	0	0	0	1,5	0	0	0	0	0	0	2	2	2	0	0	0	0	0	0	0
UZ12	Kreiranje pregleda storitev na računu	0	0	0	2,5	0	0	0	0	0	0	3	3	3	0	0	0	0	0	0	0
UZ12	Kreiranje forme za dodajanje storitev na rač.	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
UZ12	Testiranje dodajanja storitev na račun	0	0	0	0	2	0	0	0	0	0	2	2	2	2	0	0	0	0	0	0
UZ12	Brisanje storitve iz računa	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
UZ12	Testiranje dodajanja izdanega računa	0	0	0	0	4	1	0	0	0	0	5	5	5	5	1	0	0	0	0	0
UZ13	Prenos podatkov iz baze v formo	0	0	0	0	0	3,5	0	0	0	0	3	3	3	3	3	0	0	0	0	0
UZ13	Testiranje urejanja izdanega računa	0	0	0	0	0	3,5	2	0	0	0	6	6	6	6	6	2	0	0	0	0
UZ14	Brisanje izdanega računa	0	0	0	0	0	0	2,5	0	0	0	2	2	2	2	2	2	0	0	0	0
UZ15	Namestitev knjižnice za izdelavo pdf formata	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0
UZ15	Kreiranje html predloge za izdajo pdf računa	0	0	0	0	0	0	2,5	6	0	0	8	8	8	8	8	8	7	0	0	0
UZ15	Testiranje pravilnega kreiranja pdf	0	0	0	0	0	0	0	2	1	0	2	2	2	2	2	2	2	1	0	0
UZ10	Kreiranje pregleda uporabnikov	0	0	0	0	0	0	0	0	3	0	4	4	4	4	4	4	4	4	0	0
UZ10	Kreiranje povezave na formo za dodajanje	0	0	0	0	0	0	0	0	1,5	0	1	1	1	1	1	1	1	1	0	0
UZ10	Kreiranje forme za dodajanje uporabnika	0	0	0	0	0	0	0	0	2,5	1,5	4	4	4	4	4	4	4	4	2	0
UZ10	Validacija vnosnih polj	0	0	0	0	0	0	0	0	0	2	1	1	1	1	1	1	1	1	1	0
UZ10	Testiranje dodajanja uporabnika glede na vlogo	0	0	0	0	0	0	0	0	0	2,5	2	2	2	2	2	2	2	2	1	0
UZ16	Prenos podatkov iz baze v formo	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2	0
UZ16	Testiranje urejanja uporabnika	0	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3	3
UZ16	Brisanje uporabnika	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
Skupaj		**0**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**74**	**65**	**57**	**49**	**41**	**33**	**27**	**19**	**10**	**4**

Slika 4.3: Tabelo porabe in ocene časov nalog v drugi iteraciji.



Slika 4.4: Graf idealne in preostale in ocene časa v drugi iteraciji.

4.2.3 Tretja iteracija

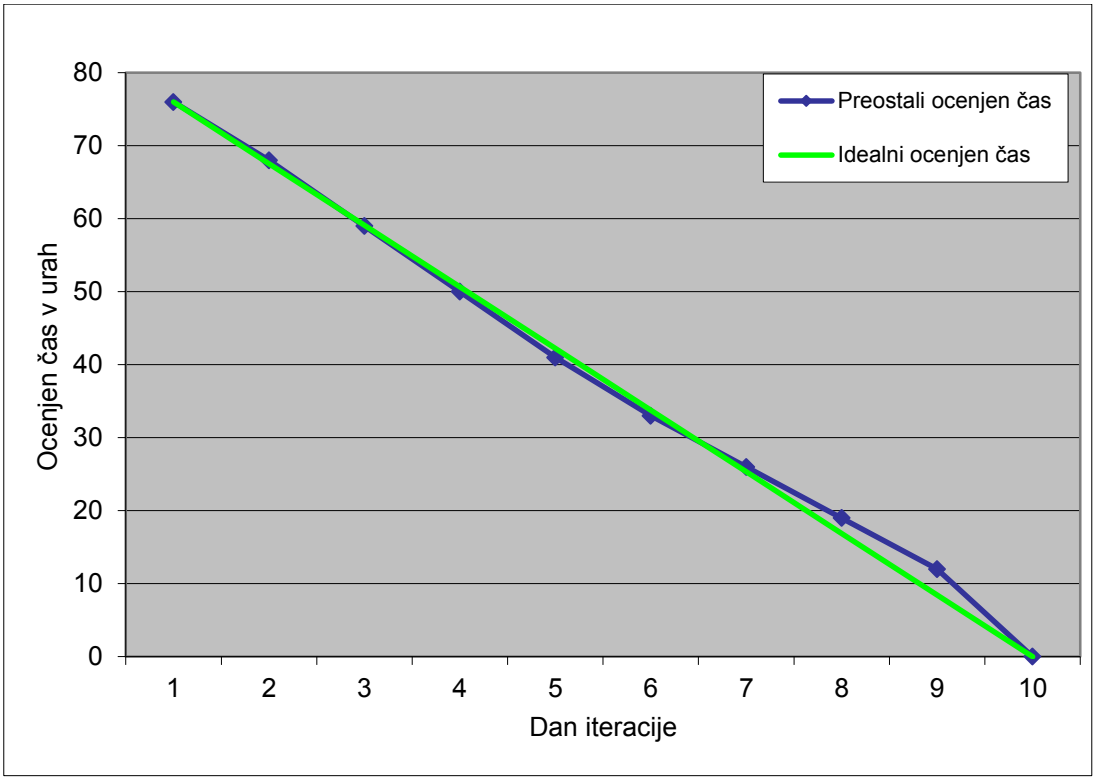
V zadnji iteraciji pred prvo izdajo je naročnik namenil najvišjo prioriteto prejetim računom. Na podlagi prve in druge iteracije smo predvideli, da bomo v tretji lahko realizirali več točk kot v drugi. Ker smo v prvih dveh iteracijah odpravili večino tehničnih težav in ker je naročnik želel imeti končane uporabniške zgodbe, ki so nam še preostale, smo se odločili, da bomo ocenili hitrost razvoja tretje iteracije z oceno 44. Tako je nastal naslednji seznam prioritetnih zgodb za tretjo iteracijo:

- dodajanje prejetega računa (UZ17), število točk: 13,
- urejanje prejetega računa (UZ18), število točk: 5,
- brisanje prejetega računa (UZ19), število točk: 3,
- povezava izdanega računa s projektom (UZ20), število točk: 3,
- povezava prejetega računa s projektom (UZ21), število točk: 3,
- urejanje uporabnika (UZ16), ki ga nismo uspeli dokončati v prejšnji iteraciji, število točk: 3,
- urejanje osebnih podatkov (UZ22), število točk: 3,
- iskalnik po pregledih (UZ23), število točk: 5,
- sortiranje po pregledih (UZ24), število točk: 3 in
- vzdrževanje šifrantov (UZ11) , število točk: 3.

Za zadnjo iteracijo lahko rečemo, da je bila najbolj uspešna. Slika 4.5 prikazuje, da smo uspeli zaključiti vse zgodbe, ki smo jih imeli v tej iteraciji. Zaradi programerskih izkušenj iz prejšnjih dveh iteracij tu nismo imeli velikih tehničnih problemov, le pri ocenjevanju urejanja osebnih podatkov smo določili premajhne ocene, kar je lepo razvidno iz grafa na sliki 4.6.

ID Zgodbe	Opis nalog	Porabljen čas v urah										Preostali ocenjen čas									
		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
UZ17	Kreiranje tabele received_invoice	0	1,5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
UZ17	Kreiranje pregleda prejetih računov	0	4,5	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
UZ17	Kreiranje povezave na formo za dodajanje rač.	0	0,5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
UZ11	Kreiranje in polnjenje tabele receive_invoice_type	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
UZ17	Kreiranje forme za dodajanje prejetega računa	0	0,5	6	0	0	0	0	0	0	0	6	6	0	0	0	0	0	0	0	0
UZ17	Kreiranje pregleda artiklov na formi pr. rač.	0	0	2	0	0	0	0	0	0	0	3	3	0	0	0	0	0	0	0	0
UZ17	Kreiranje forme za dodajanje artiklov na pr. rač.	0	0	0	1,5	0	0	0	0	0	0	2	2	2	0	0	0	0	0	0	0
UZ17	Testiranje dodajanja artiklov na pr. račun	0	0	0	2,5	0	0	0	0	0	0	2	2	2	0	0	0	0	0	0	0
UZ17	Brisanje artikla iz prejetega računa	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
UZ17	Kreiranje pregleda storitev na pr. računu	0	0	0	2,5	0	0	0	0	0	0	3	3	3	0	0	0	0	0	0	0
UZ17	Kreiranje forme za dodajanje storitev na pr. rač.	0	0	0	0,5	1	0	0	0	0	0	2	2	2	1	0	0	0	0	0	0
UZ17	Testiranje dodajanja storitev na pr. račun	0	0	0	0	2	0	0	0	0	0	2	2	2	2	0	0	0	0	0	0
UZ17	Brisanje storitve iz prejetega računa	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
UZ17	Testiranje dodajanja prejetega računa	0	0	0	0	4	1,5	0	0	0	0	6	6	6	6	1	0	0	0	0	0
UZ18	Prenos podatkov iz baze v formo	0	0	0	0	0	3	0	0	0	0	3	3	3	3	3	0	0	0	0	0
UZ18	Testiranje urejanja prejetega računa	0	0	0	0	0	4	0,5	0	0	0	5	5	5	5	5	1	0	0	0	0
UZ19	Brisanje prejetega računa	0	0	0	0	0	0	2	0	0	0	2	2	2	2	2	2	0	0	0	0
UZ20	Kreiranje povezave izdanega računa na projekt	0	0	0	0	0	0	4,5	0	0	0	3	3	3	3	3	3	0	0	0	0
UZ20	Testiranje shranjevanja spremenjenega izd. rač.	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0
UZ21	Kreiranje povezave prejetega računa na projekt	0	0	0	0	0	0	0	3,5	0	0	3	3	3	3	3	3	0	0	0	0
UZ21	Testiranje shranjevanja spremenjenega pr. rač.	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0
UZ16	Testiranje urejanja uporabnika	0	0	0	0	0	0	0	2,5	0	0	2	2	2	2	2	2	2	0	0	0
UZ16	Brisanje uporabnika	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0
UZ22	Prenos osebnih podatkov v formo	0	0	0	0	0	0	0	0	1,5	0	1	1	1	1	1	1	1	0	0	0
UZ22	Testiranje urejanja podatkov za obe vlogi	0	0	0	0	0	0	0	0	2,5	0	2	2	2	2	2	2	2	0	0	0
UZ23	Kreiranje in testiranje iskalnika v pregledu projektov	0	0	0	0	0	0	0	0	2	0	2	2	2	2	2	2	2	0	0	0
UZ23	Kreiranje in testiranje iskalnika v pregledu pr. rač.	0	0	0	0	0	0	0	0	2	0	2	2	2	2	2	2	2	0	0	0
UZ23	Kreiranje in testiranje iskalnika v pregledu izd. rač.	0	0	0	0	0	0	0	0	0	1,5	2	2	2	2	2	2	2	2	2	0
UZ23	Kreiranje in testiranje iskalnika v pregledu upor.	0	0	0	0	0	0	0	0	0	1,5	2	2	2	2	2	2	2	2	2	0
UZ24	Implementacija in testiranje sortiranja v pregledu projektov	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2	0
UZ24	Implementacija in testiranje sortiranja v pregledu pr. rač.	0	0	0	0	0	0	0	0	0	1,5	2	2	2	2	2	2	2	2	2	0
UZ24	Implementacija in testiranje sortiranja v pregledu izd. rač.	0	0	0	0	0	0	0	0	0	1	2	2	2	2	2	2	2	2	2	0
UZ24	Implementacija in testiranje sortiranja v pregledu upor.	0	0	0	0	0	0	0	0	0	1	2	2	2	2	2	2	2	2	2	0
Skupaj		**0**	**8**	**8**	**8**	**8**	**8,5**	**8**	**8**	**8**	**8,5**	**76**	**68**	**59**	**50**	**41**	**33**	**26**	**19**	**12**	**0**

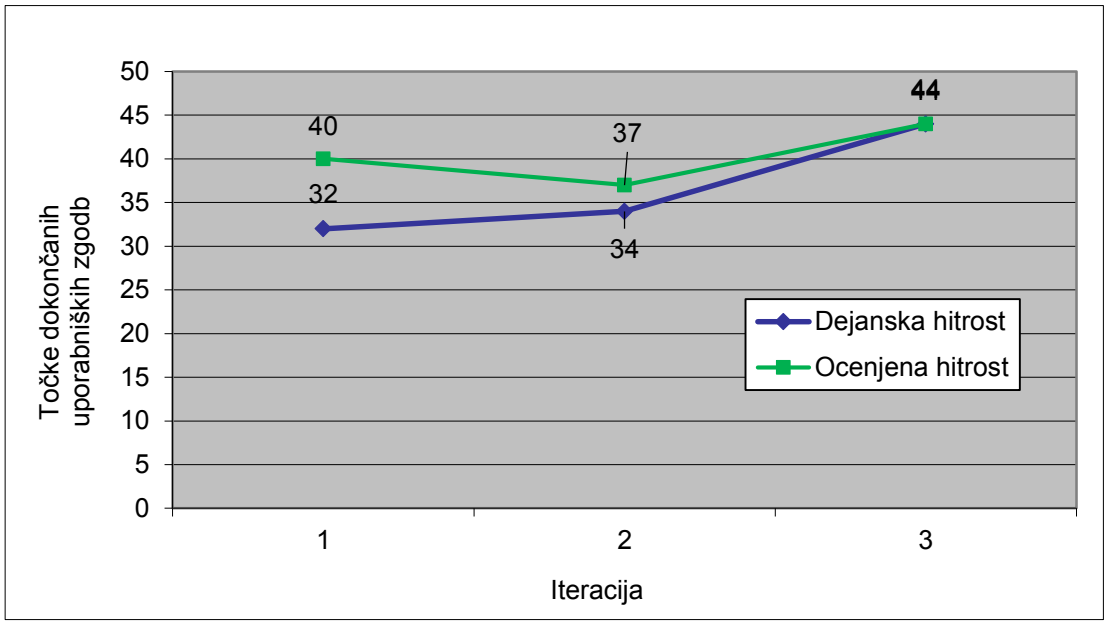
Slika 4.5: Tabelo porabe in ocene časov nalog v tretji iteraciji.



Slika 4.6: Graf idealne in preostale in ocene časa v tretji iteraciji.

4.2.4 Hitrost razvoja

S hitrostjo razvoja prikažemo, koliko točk smo uspeli realizirati v določeni iteraciji. Izračunamo jo s seštevanjem števila točk uporabniških zgodb, dokončanih v iteraciji. Na sliki 4.7 vidimo, da se je dejanska hitrost razvoja stopnjevala z zaporedno številko iteracije. To je pravzaprav normalen pojav, ker smo z vsako iteracijo pridobili nova znanja o metodologiji in samem programiranju. Medtem ko je bila ocenjena hitrost v prvi iteraciji dokaj nenatančna, smo z vsako iteracijo izboljšali to oceno. Če bi imeli še eno iteracijo, bi se dejanska hitrost razvoja verjetno še povečala, vendar se običajno po četrti iteraciji hitrost ustali, o čemer piše tudi avtor knjige *Agile Estimating and Planning* [1].



Slika 4.7: Graf hitrosti razvoja po iteracijah.

Poglavje 5 Implementacija

5.1 Uporabljene tehnologije

V današnjem svetu je pomembno, da zagotovimo dostop do podatkov neodvisno od lokacije, kjer se nahajamo. Naslednji dejavnik, ki je zelo pomemben je, ali je potrebno namestiti neko programsko opremo, da lahko dostopamo do podatkov ali pa je ta programska oprema privzeto nameščena na vsakem računalniku (spletni brskalnik). Tu se pojavi razlog, zakaj smo se odločili za spletno in ne za namizno aplikacijo. Prednost spletne aplikacije pred namizno je v tem, da je ni potrebno namestiti na namizje, ampak jo lahko uporabljamo povsod, kjer je nameščen spletni brskalnik.

Za izdelavo aplikacije smo uporabili spletne tehnologije: HTML, CSS, JavaScript, PHP, ogrodje CodeIgniter, ogrodje Ext JS, jezik MySQL in podatkovno bazo MySQL.

5.1.1 HTML

Jezik HTML (angl. *Hyper Text Markup Language*) je označevalni jezik, ki služi kot osnova za postavitev spletne strani. Njegov osnovni gradnik je oznaka, ki se začne z <oznaka> in konča z </oznaka>. Z oznakami definiramo strukturo in izgled spletne strani. Osnovna struktura HTML dokumenta je zgrajena iz naslednjih gradnikov:

```
<!DOCTYPE html>
    <HTML>
        <HEAD>
        </ HEAD >
        <BODY>
        </ BODY >
    </ HTML >
```

Kjer <html> predstavlja začetek, <head> je glava in <body> telo dokumenta.

5.1.2 CSS

S CSS-om (angl. *Cascading Style Sheets*) obogatimo izgled HTML strani tako, da elementom nastavimo oblikovne lastnosti, kot na primer velikost pisave, barva ozadja, pozicioniranje elementov, itd. Dobro je ločiti vsebino (HTML dokument) od oblike (CSS dokument), ker lahko tako obliko uporabimo v večih HTML dokumentih, preglednost vsebine pa je večja, saj imamo

stil zapisan v ločeni datoteki. Pri ločenem načinu implementacije se lahko sklicujemo na oznako (npr. *H2, P, TABLE*, itd) ali pa na enega izmed dveh atibutov oznake (*id* ali *class*).

Primer sklicevanja na ozanko *H2*:

H2 { font-size: 24px; }

Primer sklicevanja na atribut *class* elementa *<DIV class="colorRed">*:

.colorRed { color: red; }

V diplomski nalogi smo uporabili omenjen način, tako da smo za obliko uporabili datoteko z imenom main.css. Datoteke s stilom smo vključili v glavo HTML dokumenta na sledeči način:

<link rel="stylesheet" type="text/css" href=" <?php echo site_url(); ?> resources/css/main.css">

5.1.3 JavaScript

JavaScript je skriptni jezik, ki se uporablja za programiranje spletnih strani. Namen tega jezika je ustvarjanje dinamike in interaktivnosti na spletni strani. V HTML datoteki ga lahko uporabimo na 2 načina [3]:

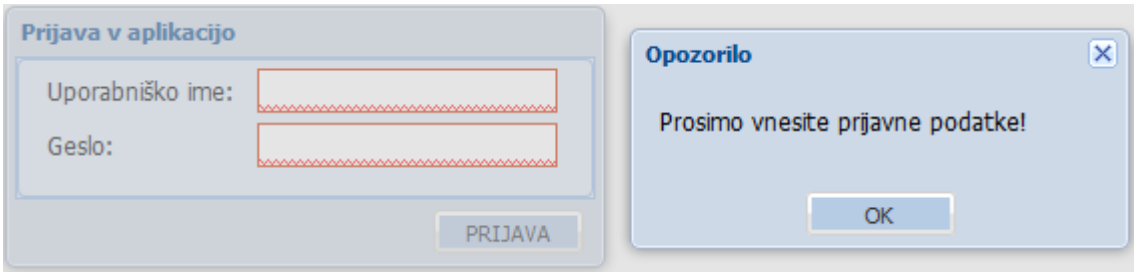
- pisanje JavaScript kode v HTML dokument (z uporabo oznake *<SCRIPT> </SCRIPT>*). Primer:

  <SCRIPT type="text/javascript">
  program
  </SCRIPT >

- sklicevanje na zunanjo datoteko v glavi HTML dokumenta. Primer:

  <SCRIPT type="text/javascript" src='<?php echo site_url("jsloader/project/form .js") ?>'></ SCRIPT >

Za ilustracijo uporabe vzemimo prijavno okno (slika 5.1).



Slika 5.1: Slika prikazuje poizkus prijave v sistem brez vnesenih podatkov.

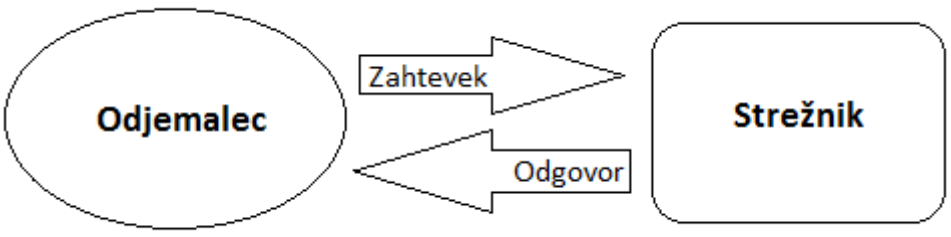
Zgornja slika je dober primer interakcije med uporabnikom in aplikacijo. V prvem koraku, ko uporabnik klikne na gumb PRIJAVA, se kliče JavaScript funkcija za validacijo polj. Validacija je narejena prav s pomočjo tega jezika in v primeru, da je katero polje prazno, ga obarvamo z rdečo opozorilno barvo. Ker se je uporabnik poskusil prijaviti v aplikacijo brez prijavnih podatkov, je sistem s pomočjo JavaScript generiral opozorilno okno, ki opozori uporabnika, da mora vnesti prijavne podatke.

5.1.4 PHP

Programski jezik PHP (angl. *Personal Home Page Tools*) je strežniški skriptni jezik [4], s pomočjo katerega dosežemo dinamiko spletnih strani. Za njegovo izvajanje potrebujemo spletni strežnik. Ker smo aplikacijo razvijali lokalno, smo uporabili spletni strežniški paket XAMPP (angl. *X ("cross"-platform), Apache HTTP Server,MySQL, PHP and Perl*) [12], ki je sestavljen iz petih modulov, od katerih smo za razvoj aplikacije uporabili prva dva:

- **Apache** je strežnik, na katerem se izvaja PHP.
- **MySql** (angl. *My Structured Query Language*), ki nam služi kot podpora pri kreiranju in manipulaciji z podatkovno bazo MySql.
- **FileZilla** se uporablja za prenos datotek preko FTP (angl. *File Transfer Protocol*) protokola.
- **Mercury** je strežnik za elektronsko pošto.
- **Tomcat** je spletni strežnik za implementacijo javinih spletnih strani.

S PHP smo definirali podatkovne formate in kreirali predloge, ki se prenašajo med strežnikom in odjemalcem (slika 5.2). V našem primeru smo za komunikacijo uporabili tehniko AJAX (angl. *Asynchronous JavaScript and XML*), ki jo bomo podrobneje predstavili v podpoglavju o tehnologiji ExtJS.



Slika 5.2: Slika predstavlja primer povezave in prenosa podatkov med strežnikom in odjemalcem.

Zahtevek je lahko tipa *POST*, kjer podatke pošljemo na strežnik ali pa *GET* kjer podatke pridobivamo s strežnika. V prvem primeru je odgovor strežnika *success:true*, kar pomeni, da je bil zahtevek uspešno izveden, ali pa *success:false* kar pomeni, da se zahtevek tipa *POST* ni izvedel uspešno zaradi določene izjeme, do katere je prišlo na strani strežnika. V drugem primeru pa kot odgovor dobimo podatke, ki smo jih zahtevali z določenimi parametri.

Za pisanje PHP v HTML dokument smo uporabili začetno oznako *<?php* in končno oznako *?>*. Primer uporabe PHP v HTML, ko preverjamo vlogo uporabnika v aplikaciji:

```
<?php if($this->session->userdata('role')==1){ ?>
    <div id="usersGridLayoutID" >
    </div>
  <?php }?>
```

5.1.5 Ogrodje CodeIgniter

CodeIgniter je odprtokodno ogrodje PHP za razvoj spletnih aplikacij [7]. Sestavljen je iz modulov, med katerimi bomo predstavili le najpomembnejše [5] (slika 5.3).

Controllers (krmilniki)

Vsebuje nabor krmilnikov, s katerimi krmilimo ukazne tokove in preko katerih se podatki prenašajo med pogledom (*views*) in podatkovno bazo (*models*). Običajno jih prožimo s klicem url zahtevka, ki je sestavljen iz dveh delov. Prvi del predstavlja naziv krmilnika, ki ga kličemo, drugi pa željeno funkcijo. Primer klica krmilnika *project* in njegove funkcije za urejanje projekta (*edit*) v AJAX zahtevku:

http://www.diploma/project/edit

Models (modeli)

Predstavlja del aplikacije, kjer skrbimo za komunikacijo s podatkovno bazo (branje, osveževanje in zapisovanje podatkov). Primer klica modela *project_model* in njegove funkcije za urejanje podatkov o projektu (*edit*):

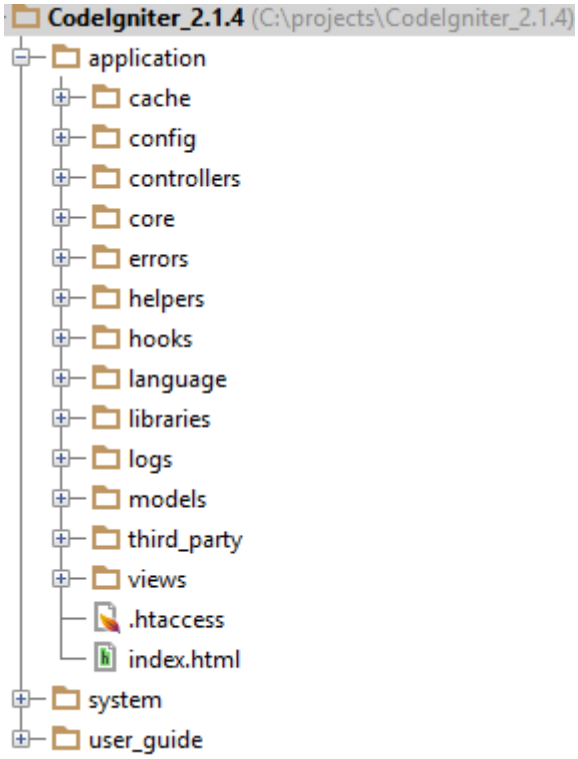
$this->ProjectModel->edit($params);

Views (pogledi)

Vsebuje predloge za prikaz informacij. Največkrat nam pogled vrača HTML kodo, če pa imamo samo prenos podatkov, ga lahko uporabimo za JSON (angl. *JavaScript Object Notation*) obliko podatkov. Njihova prednost je ta, da nam omogočajo kompleksnejšo zgradbo – gnezdenje pogledov, kjer imamo za en pogled več podpogledov. Takšna zgradba nam omogoča, da lahko nek pogled razdrobimo na manjše dele in s tem določen podpogled večkrat uporabimo. S takšno

strukturo si olajšamo delo in povečamo preglednost aplikacije. Primer klica pogleda v aplikaciji iz krmilnika:

$this->load->view('form');.

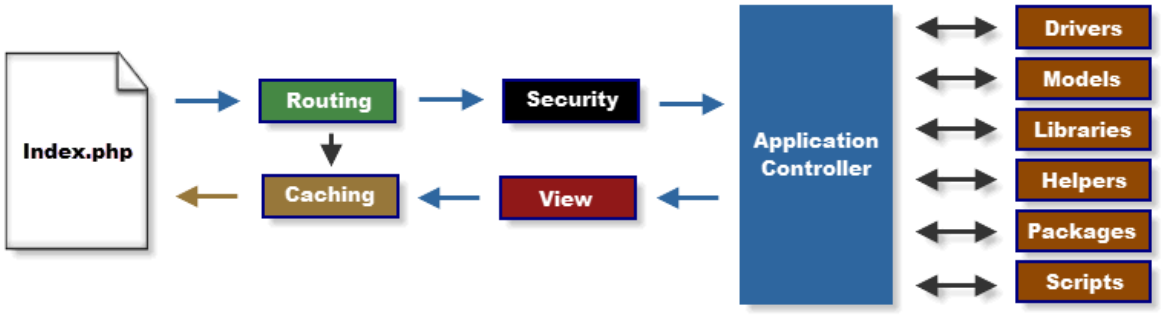


Slika 5.3: Slika prikazuje datotečno zgradbo ogrodja CodeIgniter.

Na tem mestu moramo omeniti, da struktura aplikacije temelji na oblikovnem vzorcu (angl. design pattern), ki nam pove, da je dobra praksa, da imamo za vsako entiteto podatkovne baze svoj model, krmilnik in pogled. Tako imamo na primer za entiteto *projekt* narejen svoj model (*project_model.php*), svoj krmilnik (*project.php*) in svoj pogled (*project*), pri katerem smo uporabili gnezdenje (ločen pogled za vnosno formo in pregled projektov). Ta oblikovni vzorec nam omogoča večjo preglednost in boljšo strukturiranost same aplikacije.

Kot je razvidno iz zgornje zgradbe, CodeIgniter temelji na MVC arhitekturi (angl. *Model–View–Controller*). Zanj je značilen tok dogodkov aplikacije, kot ga prikazuje slika 5.4. Za lažjo predstavo vzemimo primer, ko hoče uporabnik odpreti določen projekt za urejanje. S klikom na povezavo *Uredi* (in njenimi parametri) v pregledu projektov se proži HTTP (angl. *HyperText Transfer Protocol*) zahtevek GET, ki naslovi krmilnik *project*. Ta preuči zahtevo in parametre (v tem primeru imamo parameter *project_id*) in kliče ustrezno metodo s parametri v modelu *project_model* (v našem primeru gre za klic metode *get($projectId)*), kjer od njega zahteva

podatke o določenem projektu na podlagi podanih parametrov. Model glede na prejete parametre naredi poizvedbo na podatkovni bazi in vrne podatke o projektu nazaj v krmilnik. Ta na koncu vrne podatke v pogled, ti pa se potem prikažejo v formi za urejanje projekta.



Slika 5.4: Slika prikazuje tok dogodkov v ogrodju CodeIgniter [11].

Prednosti, ki nam jih ponuja to ogrodje, so naslednje [9]:

- ker je ogrodje odprtokodno, ga lahko vsakdo spreminja in prilagaja lastnim potrebam;
- zelo dobra dokumentacija – pri prenosu ogrodja z uradne strani dobimo tudi celotno dokumentacijo v ločeni mapi z imenom *user_guide*;
- enostavno za namestitev – za pričetek uporabe so potrebne le osnovne nastavitve v nastavitvenih datotekah, tako da se nam ni treba ukvarjati z nastavitvami preko konzole;
- temelji na MVC arhitekturi;
- ima veliko vgrajenih razredov in funkcij, s pomočjo katerih lažje implementiramo zapletene funkcionalnosti (razred za sejo, lokalizacijo, beleženje napak, itd);
- ima zelo majhen odtis (angl. small footprint), kar pomeni, da zasede zelo malo pomnilniškega prostora;
- primeren je tako za preproste kot za zmogljivejše aplikacije;
- velika podpora skupnosti, kjer lahko posamezniki prispevajo k odpravljanju napak in širjenju svojega znanja.

5.1.6 Ogrodje Ext JS

Ext JS (angl. *Extended Javascript*) je ogrodje JavaScript, ki se uporablja za razvoj interaktivnih spletnih in namiznih aplikacij [10]. Omogoča nam zelo dobro združljivost z vsemi spletnimi

brskalniki, celo z Internet Explorerjem od verzije 6 dalje. Za potrebe diplomske naloge smo uporabili Ext JS 4.2.1, ki jo je bilo mogoče prenesti z uradne strani Sencha. Določene verzije Ext JS so namreč plačljive, zato smo se odločili za zadnjo brezplačno stabilno verzijo.

Za delovanje tega ogrodja moramo v aplikacijo vključiti naslednje tri komponente:

- JavaScript datoteko *ext-all.js*, ki vsebuje vse funkcije za delovanje tega ogrodja,
- CSS datoteko *ext-all.css*, ki definira obliko izbrane teme,
- ikone, ki jih potrebuje CSS datoteka (na primer ikone za obrobe gumbov, ozadja pregledov, vnosnih form, itd).

Značilnosti ogrodja Ext JS [11]:

- Organiziran je v pakete in razrede, kar programerju omogoča, ki je vajen objektno usmerjenega programiranja, laže razumevanje.
- Podpira modularno zgradbo programske kode (vsak gradnik lahko zgradimo kot svoj objekt in ga na koncu pripnemo na nek vsebovalnik, ki je prav tako objekt zase).
- Je zelo dobra podpora tabelaričnim pregledom (ima podporo za urejanje vrstice tabele, ne da bi potrebovali le to odpreti v formi).
- Grajenje objektov v JSON notaciji, kar nam omogoča lažji nadzor nad atributi objekta. Na primer izgradnje gumba za dodajanje novega projekta:

```
var addProject = Ext.create('Ext.button.Button',{
   sytle: "font-weight:normal;",
   text: "Dodaj",
   iconCls: 'add',
   handler: function(){
      projectForm(0);
   }
});
```

- Ponuja nam bogat nabor gradnikov za gradnjo uporabniškega vmesnika (vnosna polja, spustni meniji, funkcionalnost povleci in spusti (angl. *drag and drop*), itd).

Pri ogrodju bi izpostavili razred Ext.Ajax, s katerim pošiljamo zahtevke na strežnik [12]. Imamo dva tipa zahtevkov, POST in GET. S tipom POST pošiljamo podatke na strežnik, z GET pa prejemamo podatke. Spodnja koda prikazuje primer enostavnega zahtevka GET.

```
Ext.Ajax.request({
        url:  "<?php echo site_url('project/get');?>",
        params:{
                id : projectID
        },
```

```
        method: 'GET',
        success: function(response, opts){
        },
        failure: function(response, opts){
        }
});
```

Kot vidimo iz zgornje kode, pokličemo razred Ext.Ajax v notaciji JSON. Osnovni atributi, ki jih nastavimo, imajo naslednje funkcionalnosti:

- **Url** je URL (angl. *Uniform Resource Locator*), na katerega je zahtevek naslovljen. V našem primeru kličemo krmilnik *project.php* in njegovo metodo *get()*.
- **Params** je nabor vseh spremenljivk, ki se pošljejo na strežnik.
- **Method** je atribut, ki predstavlja POST ali pa GET zahtevek.
- **Success** predstavlja parameter za obravnavo dogodkov ob uspešno izvedenem zahtevku.
- **Failure** je atribut, namenjen obravnavi dogodkov ob neuspešno izvedenem zahtevku.

5.1.7 Jezik MySql

Jezik MySql je povpraševalni jezik, ki se uporablja za komunikacijo s podatki v podatkovni bazi [4]. Da bi olajšali razumevanje, so ga izdelali na način, kot bi brali angleščino. Za lažjo predstavo si naglas preberite spodnji stavek:

SELECT name FROM table_user WHERE lastname = 'Novak';

Osnovni ukazi tega jezika so sledeči:

- **CREATE** (kreiraj) je ukaz za kreiranje tabele v podatkovni bazi.
- **INSERT** (vstavi) je namenjen vstavljanju novega zapisa v vrstico tabele.
- **SELECT** (izberi) smo uporabili za izbiranje vrstic tabel.
- **UPATE** (posodobi) je namenjen posodabljanju vrstic v tabeli.
- **DELETE** (izbriši) se uporablja za brisanje vrstice izbrane tabele.

5.1.8 Podatkovna baza MySQL

Danes skorajda ni več aplikacije, ki ne bi imela neke podatkovne baze za hranjenje podatkov. Zato lahko rečemo, da v podatkovni bazi hranimo vse podatke, ki jih obdelujemo v naši aplikaciji. Za izdelavo in manipulacijo podatkovne baze smo uporabili modul MySql strežniškega paketa XAMPP. Ko smo pognali ta modul, smo do baze lahko dostopali preko URL-ja http://localhost/phpmyadmin/.

Osnovni gradnik podatkovne baze je tabela. Vsaka tabela je sestavljena iz vrstic in stolpcev, kjer vrstice predstavljajo zapise (posamezne objekte), stolpci pa so atributi. Vsaka tabela ima običajno primarni ključ, ki je unikatni identifikator posamezne vrstice ter nič ali več tujih ključev (tipičen primer tabel brez tujega ključa so šifranti). Podrobnejšo zgradbo podatkovne baze za našo aplikacijo si bomo ogledali v enem izmed nadaljnjih podpoglavij.

5.2 Zahteve naročnika

V tem podpoglavju bomo predstavili zahteve naročnika, za katerega smo izdelali aplikacijo. Na podlagi teh zahtev smo oblikovali uporabniške zgodbe in nato s pomočjo njih razvili aplikacijo. Zaradi večje preglednosti bomo zgodbe razvrstili po sklopih. Pri zgodbi bomo zapisali naziv zgodbe in v oklepaju šifro, ki jo bomo uporabili v predstavitvi rezultatov razvoja po metodi Scrum. Pod opisom vsake zgodbe bomo predstavili še sprejemne teste, na podlagi katerih smo določili, ali je uporabniška zgodba zaključena ali ne.

5.2.1 Prijava v aplikacijo

Prijava v aplikacijo

Kot uporabnik ali administrator se želim prijaviti v sistem z vnosom uporabniškega imena in gesla.

Sprejemni testi:

- Preveri ujemanje uporabniškega imena in gesla.
- Opozori uporabnika ali administratorja ob vnosu napačnih prijavnih podatkov.

5.2.2 Delo z uporabniki

Dodajanje uporabnikov v aplikacijo

Kot administrator aplikacije lahko dodaja nove administratorje ali uporabnike, ki bodo uporabljali aplikacijo.

Sprejemni testi:

- Preveri edinstvenost novega uporabniškega imena.
- Preveri dolžino gesla (minimalna dolžina gesla je 6 znakov).
- Administrator ne more shraniti forme , ki ima prazna obvezna polja.
- Preveri pravilnost oblike vnesenega emaila.
- Če je prišlo do napačnega vnosa v katerem izmed vnosnih polj, opozori administratorja, v katerem polju je napaka.

Urejanje uporabnikov

Kot administrator aplikacije želim spreminjati podatke o že vnesenih administratorjih ali uporabnikih, da popravi napake, do katerih lahko pride pri dodajanju.

Sprejemni testi:

- Administrator pri urejanju podatkov o uporabniku ali administratorju ne sme imeti opcije spreminjanja svojega ali uporabnikovega uporabniškega imena.
- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.

Urejanje osebnih podatkov

Kot administrator ali uporabnik želim urejati svoje podatke v primeru, da se je kateri izmed njih spremenil.

Sprejemni testi:

- Preveri, da uporabnik ne more spreminjati svoje vloge v aplikaciji.
- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.
- Preveri, ali se je pri brisanju spremenila vrednost atributa *deleted* na 1.
- Preveri, ali so se ob shranjevanju zapisali spremenjeni podatki.

5.2.3 Delo s projekti

Dodajanje projekta

Kot administrator želim dodajati nove projekte, na podlagi katerih bo razvidno trenutno dogajanje v podjetju.

Sprejemni testi:

- Preveri samodejnost generiranja šifre projekta.

- Administrator ne more shraniti forme, ki ima prazna obvezna polja.
- Če je prišlo do napačnega vnosa v katerem izmed vnosnih polj, opozori administratorja, v katerem polju je napaka.
- Uporabnik ne sme imeti možnosti dodajanja novega projekta.
- Preveri pravilnost oblike vnesenih datumov.

Urejanje projekta

Kot administrator ali uporabnik želim urejati podatke o projektu, da bi lahko spreminjal podatke, ki so se spremenili med izdelavo projekta.

Sprejemni testi:

- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.
- Preveri, ali so se ob shranjevanju zapisali spremenjeni podatki.

Dodajanje artikla

Kot administrator ali uporabnik želim dodajati artikle na projekt, da imam pregled nad prodanimi artikli na projektu.

Sprejemni testi:

- Preveri samodejno zapolnjevanje polj ob izbiri artikla.
- Preveri samodejen izračun cene pri spreminjanju količine artiklov.
- Preveri samodejen izračun cene pri spreminjanju popusta na artikel.
- Če je prišlo do napačnega vnosa v katerem izmed vnosnih polj, opozori administratorja, v katerem polju je napaka.

Urejanje artikla

Kot administrator ali uporabnik želim urejati podatke o artiklu, vezanem na projekt, da lahko popravim napake pri vnosu.

Sprejemni testi:

- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.
- Preveri, ali so se ob shranjevanju zapisali spremenjeni podatki.

Brisanje artikla

Kot administrator ali uporabnik želim odstraniti artikel, vezan na projekt v primeru, da se stranka odloči, da artikla ne bo kupila.

Sprejemni testi:

- Preveri, ali se je pri brisanju spremenila vrednost atributa *deleted* na 1.

Dodajanje storitve

Kot administrator ali uporabnik želim dodajati storitve na projekt, da imam pregled nad opravljenimi storitvami na projektu.

Sprejemni testi:

- Preveri samodejno zapolnjevanje polj ob izbiri storitve.
- Preveri samodejen izračun cene pri spreminjanju časa, porabljenega za storitev.
- Preveri samodejen izračun cene pri spreminjanju popusta na storitev.
- Če je prišlo do napačnega vnosa v katerem izmed vnosnih polj, opozori administratorja, v katerem polju je napaka.

Urejanje storitve

Kot administrator ali uporabnik želim urejati podatke o storitvi, vezani na projekt, da lahko popravim napake pri vnosu.

Sprejemni testi:

- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.
- Preveri, ali so se ob shranjevanju zapisali spremenjeni podatki.

Brisanje storitve

Kot administrator ali uporabnik želim odstraniti storitev, vezano na projekt, ki je bila pomotoma vnesena.

Sprejemni testi:

- Preveri, ali se je pri brisanju spremenila vrednost atributa *deleted* na 1.

5.2.4 Delo z izdanimi računi

Dodajanje izdanega računa

Kot administrator želim dodajati nove izdane račune, na podlagi katerih bodo razvidni prihodki podjetja.

Sprejemni testi:

- Preveri samodejnost generiranja številke računa.

- Administrator ne more shraniti forme, ki ima prazna obvezna polja.
- Če je prišlo do napačnega vnosa v katerem izmed vnosnih polj, opozori administratorja, v katerem polju je napaka.
- Uporabnik ne sme imeti možnosti dodajanja novega računa.
- Preveri pravilnost oblike vnesenih datumov.
- Preveri dodajanje in brisanje artiklov na izdanem računu.
- Preveri dodajanje in brisanje storitev na izdanem računu.
- Preveri samodejno izpolnjevane vsote pri dodajanju artiklov oziroma storitev na račun.
- Pri izbiri podjetja preveri samodejno izpolnjevanje števila dni za plačilo.
- Preveri samodejno izpolnjevanje roka plačila, če nastavimo ali spreminjamo datum storitve (predpogoj je izpolnjeno polje število dni) ali če spreminjamo število dni (predpogoj je izpolnjeno polje datum storitve).

Urejanje izdanega računa

Kot administrator želim urejati podatke o izdanem računu, da bi lahko spreminjal podatke, ki so se spremenili med izdelavo izdanega računa.

Sprejemni testi:

- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.
- Preveri, ali so se ob shranjevanju zapisali spremenjeni podatki.

Brisanje izdanega računa

Kot administrator želim odstraniti izdani račun, ki je bil vnesen pomotoma.

Sprejemni testi:

- Preveri, ali se je pri brisanju spremenila vrednost atributa *deleted* na 1.
- Preveri, ali se je izbrisala povezava računa na artikle in storitve.

Povezava izdanega računa s projektom

Kot administrator želim odpreti izdani račun iz pregleda storitev ali artiklov (odvisno, na katero postavko je vezan račun), da ob spremembi podatkov o postavki ali artiklu sproti popravim podatke na računu.

Sprejemni testi:

- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.

- Preveri, ali so se ob shranjevanju popravljenega izdanega računa zapisali spremenjeni podatki.

Kreiranje PDF (angl. *Portable Document Format*) izdanega računa

Kot administrator želim izvoziti izdani račun v obliki PDF, da lahko stranki natisnem izdani račun.

Sprejemni testi:

- Preveri pravilnost podatkov na nastalem PDF-ju.

5.2.5 Delo s prejetimi računi

Dodajanje prejetega računa

Kot administrator želim dodajati nove prejete račune, na podlagi katerih bodo razvidni izdatki podjetja.

Sprejemni testi:

- Preveri samodejnost generiranja številke računa.
- Administrator ne more shraniti forme , ki ima prazna obvezna polja.
- Če je prišlo do napačnega vnosa v katerem izmed vnosnih polj, opozori administratorja, v katerem polju je napaka.
- Uporabnik ne sme imeti možnosti dodajanja novega računa.
- Preveri dodajanje in brisanje artiklov na izdanem računu.
- Preveri dodajanje in brisanje storitev na izdanem računu.
- Preveri pravilnost oblike vnesenih datumov.
- Preveri samodejno izpolnjevane vsote pri dodajanju artiklov oziroma storitev na račun.

Urejanje prejetega računa

Kot administrator želim urejati podatke o prejetem računu, da bi lahko popravil napačno vnesene podatke.

Sprejemni testi:

- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.
- Preveri, ali so se ob shranjevanju zapisali spremenjeni podatki.

Brisanje prejetega računa

Kot administrator želim odstraniti prejeti račun, ki je bil pomotoma vnesen.

Sprejemni testi:

- Preveri, ali se je pri brisanju spremenila vrednost atributa *deleted* na 1.
- Preveri, ali se je izbrisala povezava računa na artikle in storitve.

Povezava prejetega računa s projektom

Kot administrator želim odpreti prejeti račun iz pregleda storitev ali artiklov (odvisno, na katero postavko je vezan račun), da lahko hitreje preverim, ali se podatki artiklov/storitev ujemajo s podatki artiklov/postavk na prejetem računu.

Sprejemni testi:

- Preveri pravilnost prenosa podatkov iz podatkovne baze v vnosno formo.

5.2.6 Ostale uporabniške zgodbe

Vzdrževanje šifrantov

Ker v prvi fazi razvoja ni zajeta administracija za delo s šifranti, se šifrante v podatkovno bazo vnaša ročno. Tu so zajeti naslednji šifranti: podjetje, storitev in artikel.

Iskalnik po pregledih

Kot administrator ali uporabnik želim iskati zapise v pregledih s pomočjo tekstovnega filtra, da hitreje najdem iskani zapis.

Sprejemni testi:

- Preveri, ali iskalnik najde ustrezne zapise glede na vpisan iskalni niz.

Sortiranje po pregledih

Kot administrator ali uporabnik želim sortirati zapise v pregledih s klikom na izbrani stolpec v pregledu, da lahko sortiram zapise po izbranem stolpcu.

Sprejemni testi:

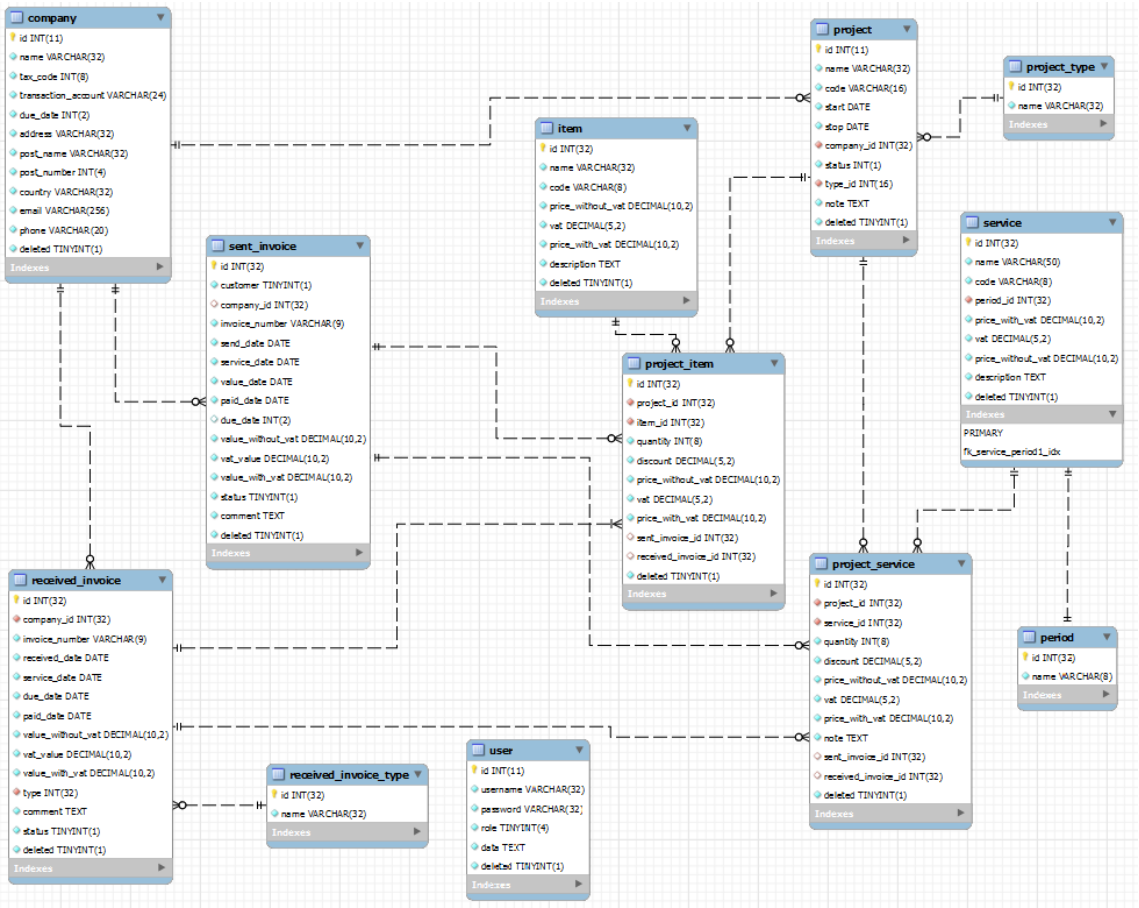
- Preveri, ali sortiranje deluje v stolpcu, kjer ga je smiselno implementirati.

5.3 Struktura podatkovne baze

Podatkovno bazo smo kreirali in upravljali preko lokalnega strežnika MySql, ki je del spletnega strežniškega paketa XAMPP. Za dostop smo uporabili URL http://localhost/phpmyadmin/index.html. Zgradba baze je prikazana na sliki 5.5.

Oglejmo si pomen posameznih tabel v bazi:

- ***User*** je tabela, ki vsebuje podatke o vseh uporabnikih in administratorjih aplikacije.
- ***Project*** se uporablja za shranjevanje osnovnih podatkov o projektu.
- ***Project_type*** je šifrant, ki služi za definiranje tipa projekta.
- ***Project_item*** vsebuje artikle, ki jih vežemo na projekt. Poleg tega se uporablja pri izdanih in prejetih računih (na artikel projekta lahko vežemo prejeti ali izdani račun).
- ***Item*** je šifrant artiklov. Uporablja se za dodajanje artiklov na projekt.
- ***Project_service*** vsebuje storitve, ki jih vežemo na projekt. Poleg tega se uporablja pri izdanih in prejetih računih (na storitev projekta lahko vežemo prejeti ali izdani račun).
- ***Service*** je šifrant storitev. Uporablja se za dodajanje storitev na projekt,
- ***Period*** predstavlja šifrant časovnih intervalov, ki jih uporablja tabela *service*.
- ***Company*** je šifrant podjetij. Uporabljamo ga pri projektih, prejetih in izdanih računih.
- ***Sent_invoice*** je tabela, ki hrani podatke o izdanih računih.
- ***Received_invoice*** je tabela, ki hrani podatke o prejetih računih.
- ***Received_invoice_type*** je šifrant, ki služi definiciji tipa prejetega računa.



Slika 5.5: Slika predstavlja strukturo podatkovne baze *financial*.

5.4 Zgradba uporabniškega vmesnika

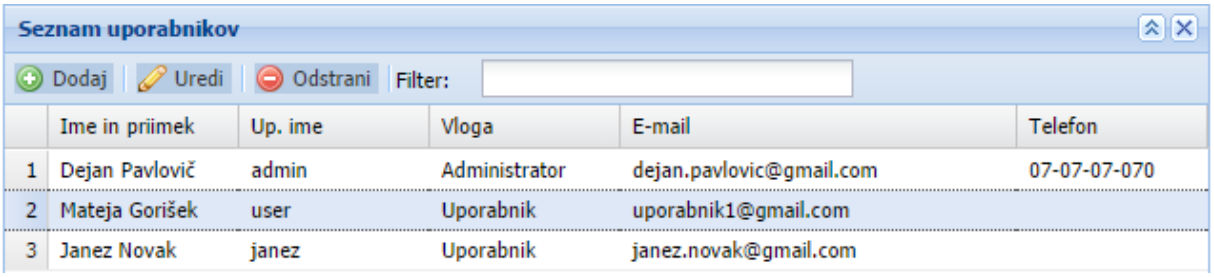
Za delo z aplikacijo uporabnik potrebuje uporabniško ime in geslo, ki ju vnese v obrazec za prijavo v sistem. Če je prijava uspešna, se uporabniku prikaže nadzorna plošča (slika 5.6), preko katere lahko upravlja celotno aplikacijo.

Slika 5.6: Slika predstavlja nadzorno ploščo aplikacije.

Prikaz funkcionalnosti, ki jih uporabnik lahko uporablja, je odvisen od njegove vloge v sistemu. Zgornja slika prikazuje nadzorno ploščo v primeru prijave administratorja. Če se v sistem prijavi navaden uporabnik, se mu prikažejo le povezave *Projekti*, *Profil* in *Odjava*. Oglejmo si funkcije, ki jih predstavlja posamezna povezava na nadzorni plošči:

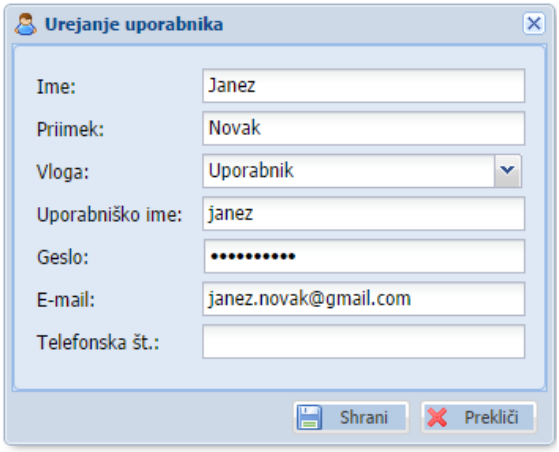
- *Uporabniki* je povezava na seznam uporabnikov. Uporablja se za pregled uporabnikov in nam omogoča dodajanje, urejanje in brisanje uporabnikov v sistemu.
- *Projekti* je povezava na seznam projektov. Preko pregleda projektov lahko dodajamo (navaden uporabnik nima te opcije) ali urejamo projekte.
- *Izdani r.* predstavlja povezavo na seznam izdanih računov. Administrator lahko dodaja, ureja in briše izdane račune.
- *Prejeti r.* predstavlja povezavo na seznam prejetih računov, do katerega lahko zopet dostopa le administrator. Omogoča nam dodajanje, urejanje in brisanje prejetega računa.
- *Profil* je namenjen urejanju osebnih podatkov.
- *Odjava* je namenjena odjavi iz aplikacije.

Ker so pregledi narejeni po podobnem principu, bomo prikazali samo pregled uporabnikov, ki je prikazan na sliki 5.7 (gre za prikaz podatkov določene entitete) in se bomo raje osredotočili na vnosne forme.



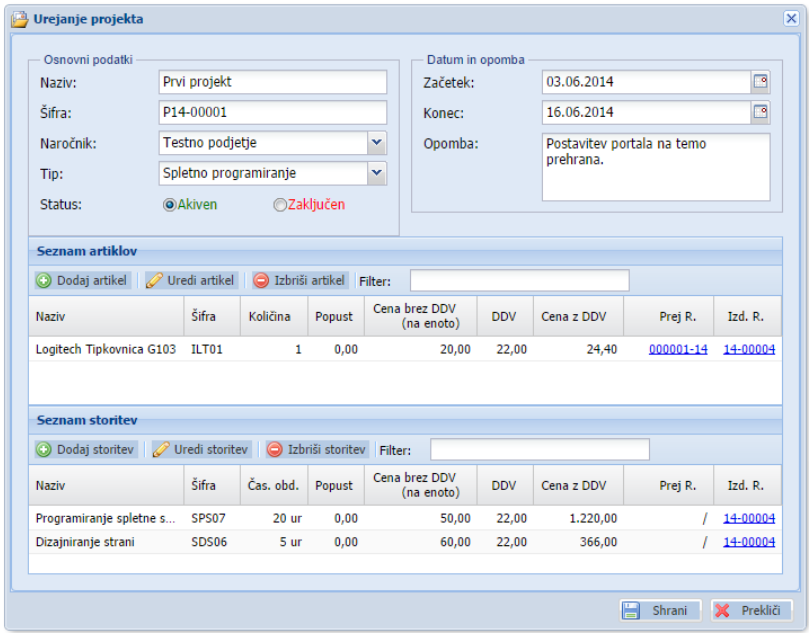
Slika 5.7: Slika predstavlja pregled uporabnikov.

Vnosni formi za delo z uporabniki in urejanje osebnih podatkov (slika 5.8) sta identični, vendar s to razliko, da pri urejanju osebnih podatkov ni opcije izbiranja vloge v sistemu. V tej formi so obvezna vsa vnosna polja razen telefonske številke.



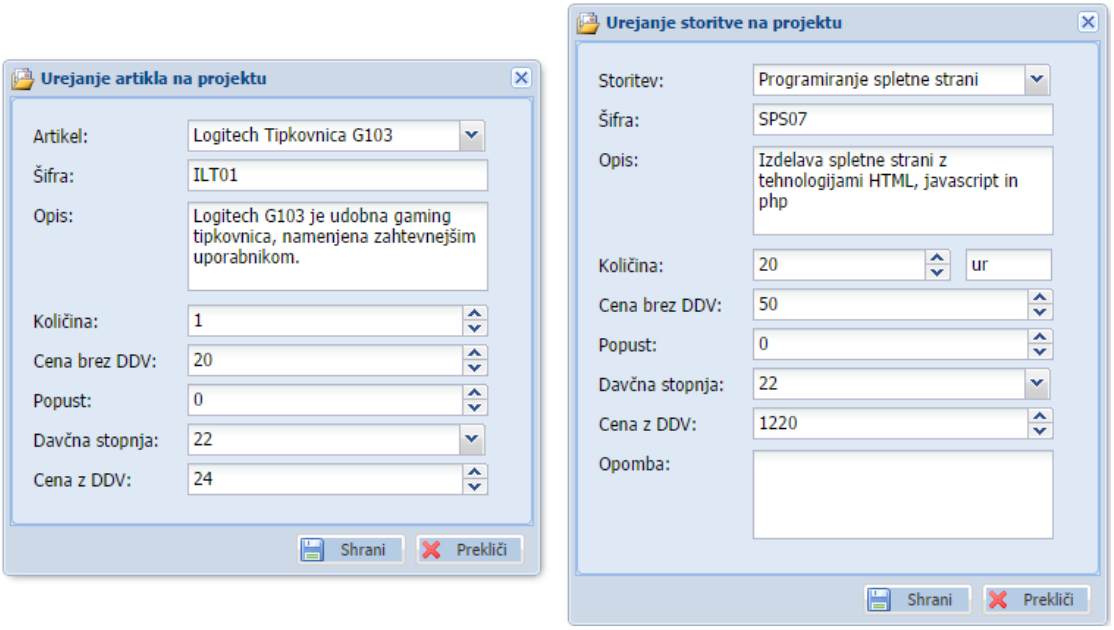
Slika 5.8: Slika prikazuje vnosno formo za dodajanje in urejanje uporabnika.

Naslednja forma, ki jo bomo predstavili, se uporablja za dodajanje in urejanje projektov (slika 5.9). Kot vidimo na spodnji sliki, forma poleg podatkov o projektu vsebuje še pregled storitev in artiklov. Pri vnašanju projekta so obvezna vsa vnosna polja razen polja *Opomba*. Posebnost, ki bi jo na tem mestu izpostavili, predstavlja polje *Šifra*, ki se nastavi avtomatično pri vsakem novem projektu (prvi del šifre P14 predstavlja leto, v katerem je bil projekt kreiran, drugi del 00001 pa predstavlja zaporedno številko projekta v tistem letu). Ker forma za dodajanje in urejanje projekta vsebuje pregled artiklov in storite, le-te lahko dodajamo, ko imamo odprto formo projektov. V pregledu artiklov vidimo povezavo na prejeti račun (s klikom na to povezavo se na nam odpre forma za urejanje prejetega računa) in povezavo na izdani račun



Slika 5.9: Slika prikazuje vnosno formo za dodajanje in urejanje projektov.

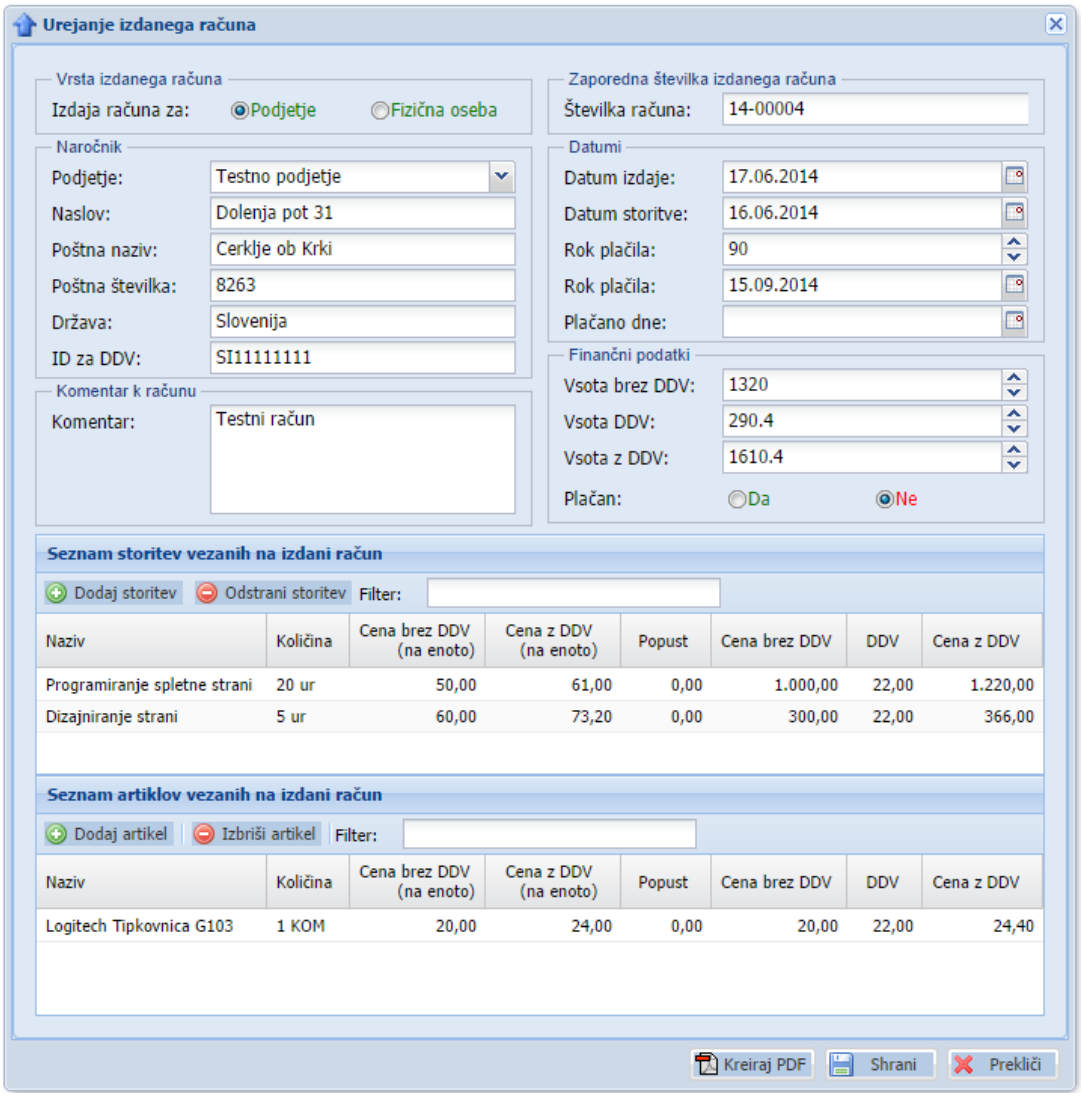
(s klikom na to povezavo se nam odpre forma za urejanje izdanega računa). Prav tako imamo v pregledu storitev dve povezavi na izdani račun. Formi za dodajanje/urejanje artiklov in storitev sta prikazani na sliki 5.10.



Slika 5.10: Slika prikazuje formo za dodajanje/urejanje artiklov (levo) in formo za dodajanje/urejanje storitev (desno).

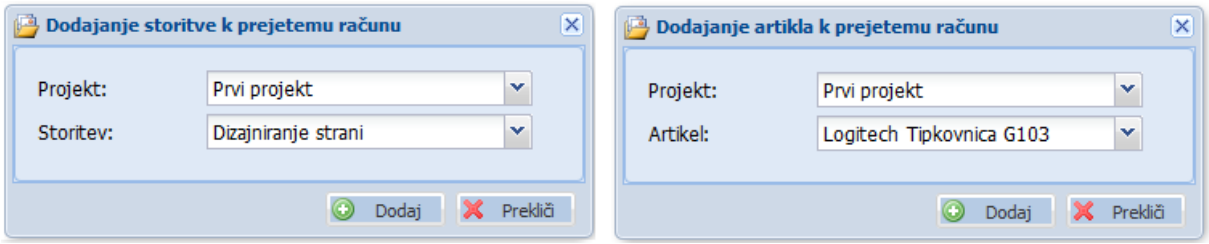
Pri dodajanju artikla izberemo le-tega v spustnem meniju, ostala polja se izpolnijo samodejno. Ob spreminjanju količine ali popusta se vrednost v polju *Cena z DDV* samodejno popravi glede na vnesene vrednosti. Enako velja za spreminjanje količine in popusta pri storitvah. V formi za obdelavo artiklov so obvezna vsa vnosna polja, pri storitvah pa le *Opomba* ni obvezno vnosno polje.

Naslednja funkcionalnost, ki nam jo omogoča aplikacija, je delo z izdanimi računi. Slika 5.11 prikazuje formo za dodajanje/urejanje izdanega računa. Forma nam omogoča, da kreiramo račun za podjetje ali pa fizično osebo. V prvem primeru so obvezna vsa polja razen polj *Plačano dne* in *Komentar*. V drugem primeru pa so poleg že omenjenih dveh neobvezna vsa polja, ki spadajo v razdelek *Naročnik*. *Številka računa* se pri kreiranju novega izdanega računa samodejno poveča za ena. Polje *Plačano dne* se izpolni samodejno, ko administrator izbere, da je račun plačan. Polja pod razdelkom *Finančni podatki* se izpolnijo samodejno ob dodajanju



Slika 5.10: Slika prikazuje formo za dodajanje/urejanje izdanega računa.

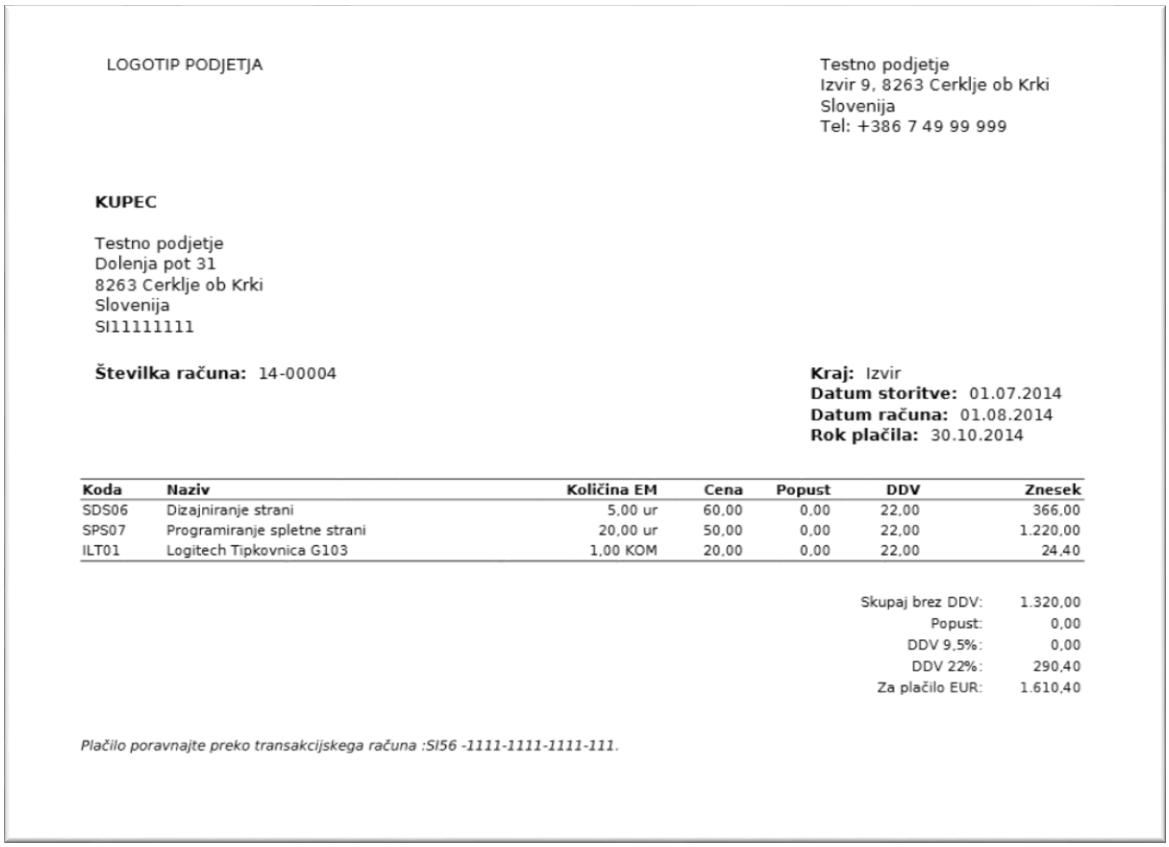
storitve/artikla. Kot vidimo na zgornji sliki, imamo v pregledu artiklov/storitev možnost dodajanja in brisanja artikla/storitve. Formi, ki se uporabljata za dodajanje storitve/artikla, sta prikazani na sliki 5.11.



Slika 5.11: Slika prikazuje formo za dodajanje storitve (levo) in formo za dodajanje artikla (desno)

Kot vidimo na zgornji sliki, imamo v pregledu artiklov/storitev možnost dodajanja in brisanja artikla/storitve.

Forma prejetih računov nam omogoča tudi kreiranje formata PDF izdanega računa. Kreiramo ga s klikom na povezavo *Kreiraj PDF*, kjer se podatki o računu najprej shranijo, nato pa se odpre predogled računa v PDF formatu v novem oknu (slika 5.12).



LOGOTIP PODJETJA

Testno podjetje
Izvir 9, 8263 Cerklje ob Krki
Slovenija
Tel: +386 7 49 99 999

KUPEC

Testno podjetje
Dolenja pot 31
8263 Cerklje ob Krki
Slovenija
SI11111111

Številka računa: 14-00004

Kraj: Izvir
Datum storitve: 01.07.2014
Datum računa: 01.08.2014
Rok plačila: 30.10.2014

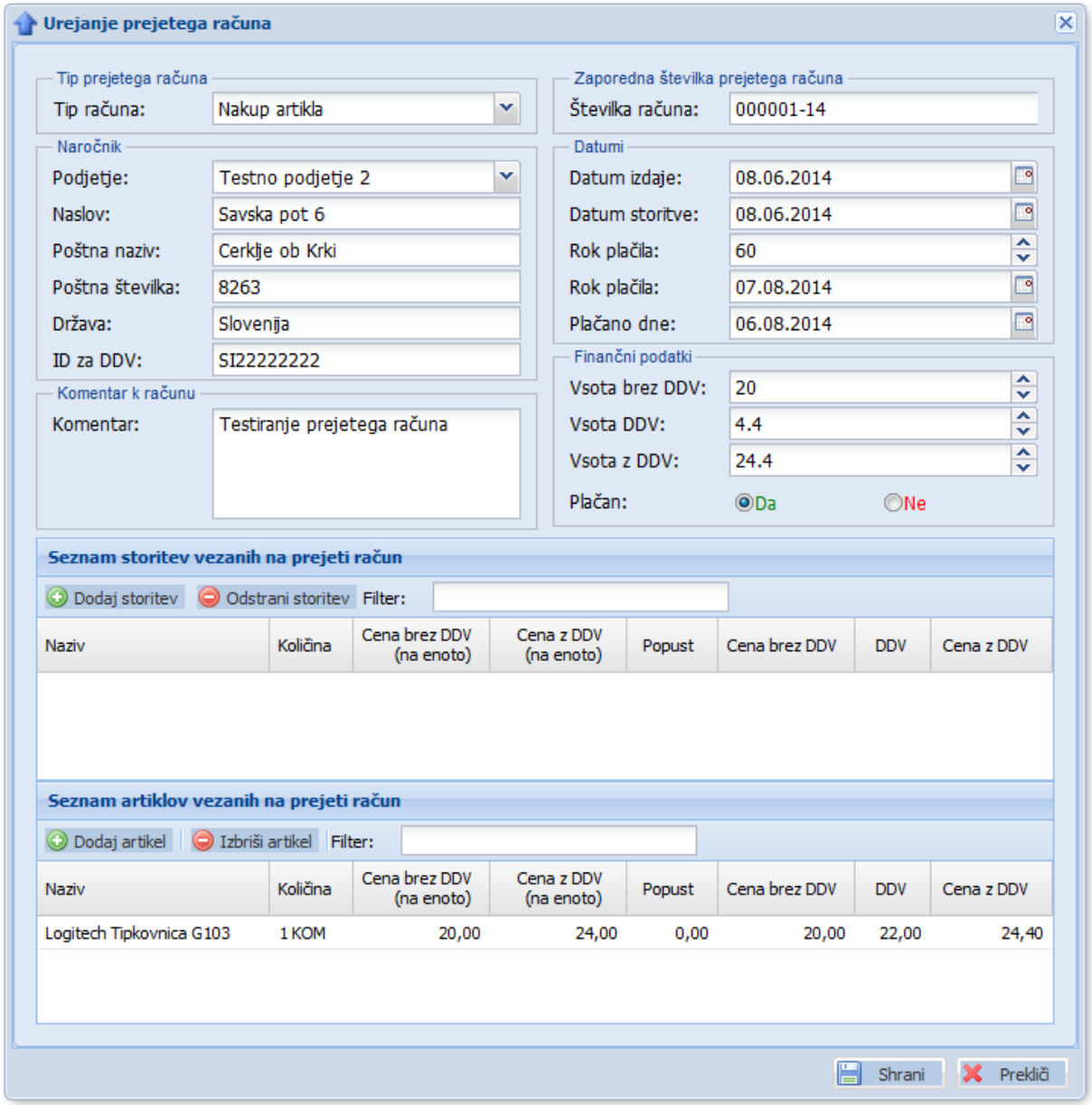
Koda	Naziv	Količina EM	Cena	Popust	DDV	Znesek
SDS06	Dizajniranje strani	5,00 ur	60,00	0,00	22,00	366,00
SPS07	Programiranje spletne strani	20,00 ur	50,00	0,00	22,00	1.220,00
ILT01	Logitech Tipkovnica G103	1,00 KOM	20,00	0,00	22,00	24,40

Skupaj brez DDV:	1.320,00
Popust:	0,00
DDV 9,5%:	0,00
DDV 22%:	290,40
Za plačilo EUR:	1.610,40

Plačilo poravnajte preko transakcijskega računa :SI56 -1111-1111-1111-111.

Slika 5.12: Slika prikazuje predlogo PDF formata računa.

Zadnja funkcionalnost, ki nam jo omogoča aplikacija, je dodajanje prejetih računov. Na formi, ki je prikazana na sliki 5.13, so obvezna vsa vnosna polja razen polj *Komentar* in *Plačano dne*. Polja pod razdelkom *Finančni podatki* se izpolnijo samodejno ob dodajanju artikla/storitve. Formi za dodajanje artikla/storitve sta enaki kot pri izdanem računu.



Slika 5.13: Forma za dodajanje/urejanje prejetega računa.

Poglavje 6 Sklepne ugotovitve

V diplomski nalogi smo uspešno razvili aplikacijo, s pomočjo katere lahko vodimo projekte ter prilive (izdani računi) in odlive (prejeti računi) podjetja. Čeprav ima aplikacija razvite le osnovne funkcionalnosti finančnega poslovanja, zadostuje osnovnim potrebam manjšega podjetja. Najbolj opazen manjkajoči del aplikacije je vmesnik za delo s šifranti, ki jih je potrebno še vedno vnašati ročno.

Tudi sam razvoj po metodi Scrum je bil uspešen, glede na to, da pred tem nismo imeli izkušenj z delom po tej metodi. Na začetku razvoja smo imeli kar nekaj težav. Prva izmed njih je bila ocenjevanje hitrosti razvoja. Zato smo pri prvi iteraciji namesto številčne ocene raje podali obseg zgodb, ki bi jih lahko realizirali med potekom iteracije. Poleg težav z ocenjevanjem hitrosti smo v prvi in drugi iteraciji naleteli na kup tehničnih (programerskih) problemov. Največ preglavic nam je povzročalo ogrodje Ext JS, s katerim smo gradili uporabniški vmesnik. S to tehnologijo smo se srečali že v preteklosti, a v starejši različici. Različica, ki smo jo uporabili za izdelavo diplomskega dela, se od omenjene razlikuje predvsem v sintaksi in obsežnejšem naboru knjižnic.

Med procesom izdelave diplomskega dela smo se veliko naučili. Najpomembnejše pridobljeno znanje predstavlja metodologija Scrum. Tu smo se preizkusili v vseh vlogah, ki nastopajo v tej metodologiji. Na začetku smo bili mnenja, da je najzahtevnejša vloga vloga razvijalcev, med procesom pa smo ugotovili, da so zahtevnosti vlog dokaj enakovredne. Namreč, če vsakdo ne poskrbi, da dobro opravi svojo nalogo, se produkt ne more uspešno razvijati. Kot lastnik izdelka smo morali skrbeti za prioritete in vizijo aplikacije, kot skrbnik metodologije za doslednost izvedbe procesa po metodi Scrum in kot razvojna skupina za uspešno implementacijo produkta.

Nadaljnji razvoj diplomskega dela vidimo v razširitvi aplikacije. Proces bi nadaljevali po metodologiji Scrum, tako da bi v aplikacijo implementirali nove funkcionalnosti. Dodali bi vmesnik za delo s šifranti, grafični prikaz finančnega stanja podjetja, beleženje spreminjanja podatkov v aplikaciji glede na uporabnika itd. Ob vseh teh dodatnih funkcionalnostih, bi razvoj usmerili na mobilno aplikacijo, saj v današnjem času prevladuje uporaba pametnih telefonov, trend razvoja pa se giblje v smeri mobilnih aplikacij.

Literatura

[1] Mike Cohn, "Agile Estimating and Planning", Practice Hall, 2006

[2] Mike Cohn, "User Stories Applied for Agile Software Development", Boston, 2004

[3] Danny Goodman with Michael Morrison, "JavaScript Bible 5th Edition", Indianapolis, Indiana, 2004

[4] Matjaž Štrancar, Simon Klemen, "PHP in MySql na spletnem strežniku Apache, Druga, dopolnjena izdaja", Ljubljana, 2005

[5] David Upton, "CodeIgniter for Rapid PHP Application Development", 2007, Dosegljivo: http://it-ebooks.info/book/2760/

[6] D. West in T. Grant, "Agile Development: Mainstream Adoption Has Changed Agility", 2010, Dosegljivo : http://www.heacademy.ac.uk/assets/bmaf/documents/events/Events_2011/agile_development_mainstream_adoption_has_changed_agility.pdf

[7] (2011) "CHAOS MANIFESTO", Dosegljivo: http://versionone.com/assets/img/files/ChaosManifest_2011.pdf

[8] Manifesto for Agile Software Development, Dosegljivo: http://agilemanifesto.org/

[9] Ogrodje CodeIgniter, Dosegljivo: https://ellislab.com/codeigniter

[10] Ogrodje Ext JS, Dosegljivo: http://docs.sencha.com/extjs/4.2.1/

[11] Prabhu Sunderaraman, "Practical Ext JS 4", Dosegljivo : http://it-ebooks.info/book/2938/
bmaf/documents/events/Events_2011/agile_development_mainstream_adoption_has_changed_agility.pdf

[12] Strežniški paket XAMPP, Dosegljivo: https://www.apachefriends.org/index.html

[13] Zahtevek Ajax, Dosegljivo: http://docs.sencha.com/core/manual/content/ajax.html