UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andrej Stivičević

Spletna aplikacija za vodenje projektov s pomočjo metodologije Scrum

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: prof. dr. Viljan Mahnič

Ljubljana, 2018

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge: Spletna aplikacija za vodenje projektov s pomočjo metodologije Scrum

Proučite metodologijo Scrum in jo primerjajte z ostalimi agilnimi metodami za razvoj programske opreme. Na podlagi izkušenj z njeno uporabo v praksi definirajte zahteve, ki jih mora izpolnjevati ustrezno orodje za vodenje projektov po metodologiji Scrum, in to orodje tudi realizirajte. Opišite postopek izdelave in tehnologije, ki ste jih pri tem uporabili, ter primerjajte svojo rešitev z nekaterimi obstoječimi orodji.

Rad bi se zahvalil mentorju prof. dr. Viljanu Mahniču za strokovno pomoč in usmerjanje pri izdelavi diplomske naloge. Zahvalil bi se tudi mojim sodelavcem, ki so mi dajali koristne povratne informacije pri uporabi izdelanega orodja. Še posebej pa bi se zahvalil družini in prijateljem za podporo in vzpodbudne besede med študijem in pisanjem diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AJAX	Asynchronous JavaScript and XML	Asinhrono izvajanje JavaScript in XML
CSRF	Cross-Site Request Forgery	Ponarejanje zahteve med različnimi spletnimi stranmi
CSS	Cascading Style Sheets	Kaskadne stilske predloge
DOM	Document Object Model	Objektni model dokumenta
IDE	Integrated Development Environment	Integrirano razvojno okolje
HTML	HyperText Markup Language	Označevalni jezik za izdelavo spletnih strani
HTTP	HyperText Transfer Protocol	Protokol za prenos podatkov preko spleta
HTTPS	HyperText Transfer Protocol – Secure	Varen protokol za prenos podatkov preko spleta
MVC	Model-View-Controller	Model-pogled-krmnilnik
OOP	Object Oriented Programming	Objektno usmerjeno programiranje
PDF	Portable Document Format	Prenosna oblika dokumenta
PHP	Hypertext Preprocessor	Skriptni programski jezik za izdelavo spletnih strani
SQL	Structured Query Language	Strukturirani povpraševalni jezik
URL	Uniform Resource Locator	Enolični krajevnik vira – unikatno definira lokacijo vira na spletu
XP	eXtreme Programming	Ekstremno programiranje
XSS	Cross-Site Scripting attack	Napad z vstavljanjem zlonamerne kode v spletno stran

Povzetek

V diplomski nalogo bomo na začetku podrobno predstavili metodologijo Scrum. Naredili bomo primerjavo Scruma z drugimi razširjenimi metodologijami za razvijanje programske opreme. Pri vsaki metodologiji bomo opisali glavne pojme in razlike z metodologijo Scrum.

Nato bomo definirali naše zahteve, ki jih potrebujemo od načrtovanega orodja. Zapisali bomo tudi predpogoje za uporabo orodja. Razložili bomo, zakaj smo se odločili prav za uporabo metodologije Scrum, in ne za eno od preostalih agilnih metodologij. Pri razvoju smo uporabili programski jezik PHP z ogrodjem Laravel, za hranjenje podatkov pa smo uporabili sistem za upravljanje baz podatkov MySQL, ki temelji SQL. Predstavili bomo te, in druge uporabljene tehnologije in orodja. Predstavili bomo način izvajanja dela. Temu sledi opis rešitve.

V nadaljevanju bomo predstavili strukturo podatkovne baze za naše orodje in pomen posameznih entitenih tipov. Zapisali bomo posamezne sklope orodja in katere funkcionalnosti zajemajo. Funkcionalnosti so razdeljene v več različnih krmilnikov, ki predstavljajo organizacijsko strukturo v ogrodju Laravel.

Po vsem tem bomo naredili primerjavo med najbolj popularnimi obstoječimi orodji za vodenje projektov. Primerjali bomo njihove prednosti, slabosti in pomanjkljivosti ter obrazložili, zakaj nobena od obstoječih rešitev ni ustrezala našim zahtevam.

V zaključku bomo predstavili s katerimi težavami smo se srečali pri izdelavi in nekaj možnih izboljšav našega orodja.

Rezultat diplomske naloge je delujoče orodje za vodenje projektov s pomočjo metodologije Scrum. Ker je orodje dostopno preko spletne strani, ga lahko uporabljamo kjerkoli se nahajamo. Z njim lahko vodimo celoten proces vodenja projektov.

Ključne besede:

vodenje projektov, metodologija Scrum, spletno orodje, ogrodje Laravel

Abstract

In this thesis we will first present the Scrum methodology in detail. We will compare it with other popular methodologies. For each methodology we will describe their main concepts and differences with the Scrum methodology.

We will then define our requirements which we need from the planned tool. We will also specify the preconditions that we need in order to use the tool. We will explain why exactly have we decided to use the Scrum methodology, and not one of the other agile methodologies. During the development we used the PHP programming language with Laravel framework. For data storage we used the MySQL database management system. We will present these and other technologies and tools we used.

Hereinafter we will present the structure of the database for our tool and the purpose of individual entity types. We will introduce parts of our tool and the functionalities that they cover. Functionalities are divided into several different controllers that represent the organizational structure in the Laravel framework.

Afterwards, we will make a comparison between the most popular existing project management tools. We will compare their strengths and weaknesses and explain why none of the existing solutions met our requirements.

In conclusion, we will present some difficulties we encountered during development and some of the possible improvements to our tool.

The result of the thesis is a functioning tool for project management using the Scrum methodology. Because the tool is accessible through the website, we can use it anywhere we are. With it we can manage the entire process of project management.

Keywords:

project management, Scrum methodology, web-based tool, Laravel framework

Poglavje 1 Uvod

V poslovnem razvoju programske opreme se neprestano srečujemo s projekti, ki morajo biti dobro načrtovani in izpeljani v dogovorjenih časovnih rokih, drugače so lahko naročniki projekta nezadovoljni, pri tem pa smo vezani s pogodbami. Če je tako označeno v pogodbi, smo lahko tudi denarno odgovorni za nastalo zamudo in škodo.

Zaradi tega se moramo lotiti razvoja programov premišljeno in z naročnikom dobro načrtovati, kakšno funkcionalnost bo programska oprema zajemala, ter naročniku sproti predstavljati napredek. S tem zmanjšamo možnosti za drastične spremembe ob koncu projekta, poleg tega pa si ne ustvarjamo obilo nepotrebnega dela, ki bi ga potem mogoče celo zavrgli.

Ker se v našem podjetju ukvarjamo z razvojem programske opreme, potrebujemo primerno orodje za vodenje projektov. Sami vodimo projekte po metodologiji Scrum. Do sedaj smo preskusili že veliko obstoječih rešitev, vse pa so bile preobsežne ali pa niso zajemale tistih funkcionalnosti, ki smo jih mi potrebovali. Zato sem se odločil, da v sklopu diplomske naloge razvijem lastno orodje za vodenje projektov po metodologiji Scrum.

V diplomski nalogi bom predstavil, zakaj sem se odločil ravno za metodologijo Scrum, ter primerjal moje orodje z najbolj razširjenimi podobnimi profesionalnimi in odprtokodnimi orodji.

Poglavje 2 Uporabljena metodologija

Pri razvoju orodja za vodenje projektov sem se odločil, da uporabim metodologijo Scrum, saj je najbolj priljubljena pri razvijanju programske opreme [12], z njo pa sem bil že prej seznanjen z večletno uporabo pri izvajanju projektov v podjetju. V nadaljevanju bomo opisali proces uporabe metodologije Scrum in najbolj značilne razlike z drugimi metodologijami.

2.1 Metodologija Scrum

Scrum (gruč [2]) je metodologija za upravljanje z delom pri razvoju projekta od začetka do konca. Razvoj poteka v iteracijah, ki se imenujejo sprinti. Naročnik je prisoten celoten čas razvoja, sproti prejema na vpogled delujoč izdelek z zaključenimi enotami dela, ekipa pa prejema povratne informacije, na katere se lahko odzovejo v zgodnji fazi razvoja. Enote dela se zapišejo v obliki uporabniških zgodb, ki se lahko nato razčlenijo na manjše enote za lažjo razporeditev v posamezen sprint ali razdelitev dela na več članov ekipe. Uporabniške zgodbe se dodajajo na seznam zahtev (angl. *product backlog*), te pa se ob vsakem začetku sprinta izbirajo za vključitev v tekoči sprint na seznam nalog sprinta (angl. *sprint backlog*).

2.1.1 Vloge

Scrum v delovni proces vključuje vodje podjetja, naročnika in vse vpletene, ki doprinašajo h končnemu izdelku. Med njih sodijo člani ekipe z različnih področij (angl. *cross-functional teams*), kot so programerji, grafični oblikovalci in uporabniki, ki preskušajo stabilnost programske opreme (angl. *quality assurance*).

Scrum opredeljuje uporabo uporabniških vlog. Vloge lahko poljubno določamo po lastni presoji, metodologija pa določa tri glavne vloge, ki so:

- skrbnik izdelka (angl. *product owner*),
- skrbnik procesa (angl. *Scrum Master*),
- člani ekipe

Skrbnik izdelka je zadolžen za povezovanje naročnika z razvijalsko ekipo in razvrščanje uporabniških zgodb na seznamu zahtev po prioriteti. Zadolžen je za to, da ekipi daje podrobna pojasnila o posameznih uporabniških zgodbah.

Skrbnik procesa upravlja s sprintom, ima nalogo učenja metodologije Scrum vseh, ki sodelujejo na projektu, upravljati izvajanja sprintov ter voditi dnevne sestanke. Ekipo ščiti pred birokracijo in ji pomaga pri odstranjevanju čim več ovir pri delu.

Člani ekipe so vsi preostali zaposleni, ki so zadolženi za izvajanje dela na projektu. Pri razvoju programske opreme je to programiranje, grafično oblikovanje, testiranje in objava končnega izdelka.

2.1.2 Uporabniške zgodbe

Uporabniška zgodba je kratek opis funkcionalnosti sistema, ki bo imela vrednost za naročnika. Sestavljena je iz treh komponent:

- opis zgodbe za namene planiranja in opominjanja,
- diskusije o zgodbi, ki služijo kot pomoč pri pisanju podrobnosti zgodbe,
- sprejemni testi, ki sporočajo in dokumentirajo podrobnosti in se lahko uporabijo za določanje, kdaj je zgodba zaključena

Vsaka zgodba mora slediti merilom »INVEST« (vlaganje) [2]. Merilo je dobilo ime po inicialkah besed, ki ga določajo:

- **Independent** (neodvisna): vsaka zgodba mora biti samostojna in neodvisna od drugih zgodb.
- **Negotiable** (spremenljiva): dokler se zgodba ne zaključi, mora omogočati spremembe.
- **Valuable** (dragocena): prinašati mora dejansko vrednost h končnemu izdelku.
- **Estimable** (ocenljiva): mora biti dovolj opredeljena, da se lahko oceni čas in kompleksnost naloge.
- **Small** (kratka): ne sme biti preobširna, drugače je težko oceniti čas in vrednost glede na ostale zgodbe, lahko bi se zgodilo, da zaradi dolžine celotne zgodbe ne bi mogli vmestiti v noben sprint.
- **Testable** (preverljiva): omogočati mora, da se lahko preveri ali je zgodba dokončana.

Uporabniške zgodbe pišejo naročniki in skrbnik izdelka, saj so v najboljšem položaju za opisovanje obnašanja izdelka.

Za pisanje zgodb je priporočljivo, da se določijo uporabniške vloge na zgodbah. Uporabniška vloga je nabor opredelitvenih atributov, ki označujejo sklop uporabnikov in njihove predvidene interakcije s sistemom. Teh vlog ne smemo mešati z vlogami članov ekipe na našem projektu. To so vloge uporabnikov, ki bodo uporabljali končni izdelek. Vloge določimo, še preden se lotimo izdelave uporabniških zgodb, saj jih potrebujemo pri pisanju zgodb.

Zgodba naj ne bo označena z zaporedno številko, ampak s kratkim razločljivim naslovom, na katerega se lahko sklicujemo v diskusijah. Glavni del zgodbe je opis, ki naj bo zapisan v obliki stavka [4]:

Kot <vloga uporabnika> **želim** <določena akcija>, **zato da** <kaj želim da se zgodi kot rezultat>.

Primer: Kot predstavnik podjetja želim imeti možnost vnosa zaposlitvenega oglasa, zato da pridem v stik s čim več iskalci dela.

Poleg opisa zgodbe lahko zapišemo še seznam sprejemnih testov. Sprejemno testiranje je proces verifikacije, da so bile zgodbe implementirane točno tako, kot si jih je zamislil naročnik. En od razlogov za pisanje sprejemnih testov je namen izražanja čim več podrobnosti o zgodbi. Namesto pisanja dolgih seznamov v obliki stavkov, kot so »Sistem bo omogočal...«, naj testi zapolnijo manjkajoče podrobnosti o zgodbi. Testi zajemajo naročnikova pričakovanja o delovanju sistema. Naročnik naj napiše toliko testov, da še predstavljajo dodano vrednost in razjasnitev zgodbe. Cilj testov je sporočati dodatno informacijo o zgodbi, zato da razvijalci vedo, kdaj je zgodba končana. Zaželjeno je, da so sprejemni testi napisani pred pričetkom sprinta, lahko pa se dodajajo in odstranjujejo tudi kasneje. Primeri sprejemnih testov:

- Preveri vnos s praznim opisom dela.
- Preveri vnos z zelo dolgim opisom dela.
- Preveri vnos z manjkajočo urno postavko.
- Preveri vnos s šest-mestno urno postavko.

Za zgodbo zapišemo še grobo oceno za izpolnjevanje cilja, kar imenujemo točke zgodbe (angl. *story points*). Te se nato upoštevajo pri planiranju sprinta. Točke ne predstavljajo časovne zahtevnosti, ampak obsežnost posamezne zgodbe ter služijo kot merilo za njihovo medsebojno primerjavo. Dobra ocena za eno točko je idealen dan dela brez prekinitev [14]. Oceno naj določi ekipa skupaj, saj več ljudi poda boljše povprečje zahtevnosti zgodbe kot samo ena oseba. Vsak član ekipe poda prvo oceno za zgodbo, nato pa se ocene primerjajo. Če

se ocene članov precej razlikujejo, tisti z najmanj in največ točkami obrazložijo svojo odločitev, nato pa ponovi ocena. V drugem krogu ponavadi pride do veliko manjših odstopanj. Za točke izberemo poljuben nabor vrednosti. Uporabimo lahko številke iz Fibonaccijevega zaporedja, ki med seboj niso enakomerno oddaljene (1, 2, 3, 5, 8, 13). Neenakomerno razdelitev uporabimo zato, ker je zahtevnosti večjih zgodb težje primerjati med seboj. Težko rečemo, da je ena zgodba točno dvakrat bolj zahtevna kot druga.

Namesto izdelave celotnega načrta in odločitev na začetku projekta porazdelimo odločitve čez celotno izvajanje projekta. Na začetku lahko nekatere funkcionalnosti, ki jih ne potrebujemo takoj na začetku, pišemo v epskih zgodbah (angl. *epic*). Tako zgodbo lahko kasneje razdelimo na dve ali več manjših zgodb. Epska zgodba nam služi kot začasen zapis in je ni potrebno definirati v podrobnosti, saj jo dopišemo takrat, ko jo potrebujemo. Izhodiščna točka za velikost zgodbe je čas izvedbe in testiranja v obsegu pol dneva do dveh tednov.

Zgodbe morajo biti dovolj natančne, da so izvedljive, vendar ne smejo določati samega načina izvedbe. Ekipa se sama odloči, kako bo prišla do rešitve zgodbe.

2.1.3 Sprinti

Delo je razdeljeno na sprinte (iteracije), ki ponavadi trajajo med 1 in 2 tednoma, redkokdaj pa več kot 1 mesec. Ker so sprinti časovno omejeni, moramo oceniti hitrost sprinta (angl. *sprint velocity*). Hitrost sprinta je število točk zgodb, ki jih lahko razvojna ekipa realizira v enem sprintu. Če želimo dovolj natančno oceniti hitrost, potrebujemo veliko izkušenj z ekipo, s katero bomo delali na projektu. Z ekipo, s katero nimamo preteklih izkušenj, lahko podamo bolj točno oceno hitrosti sprinta šele po nekaj zaključenih sprintih. Za določitev hitrosti sprinta lahko torej uporabimo:

- zgodovinske vrednosti, če jih imamo na voljo,
- izvedba začetnega sprinta in uporaba njegove hitrosti,
- ugibanje

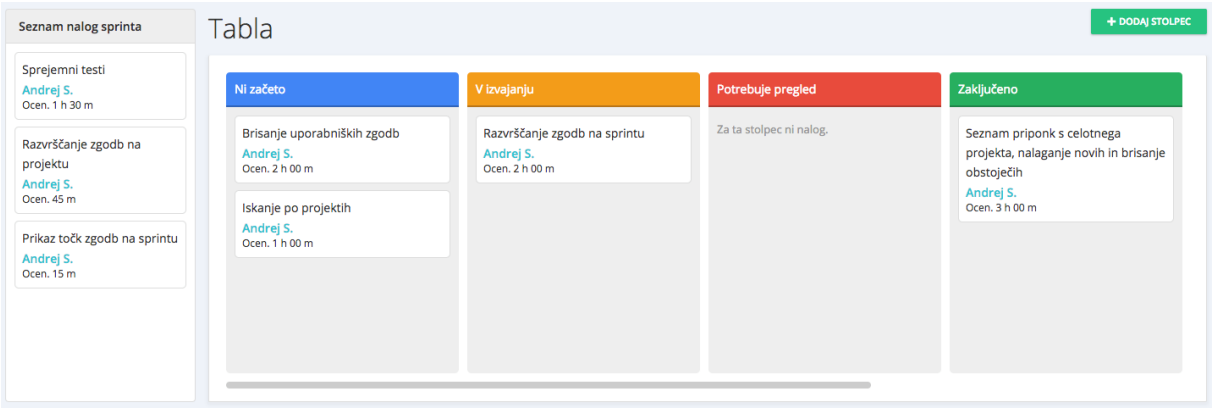
2.1.4 Proces uporabe Scruma

Na začetku izvajanja projekta mora vodja projekta najprej sestaviti dobro ekipo. Ekipa ne sme biti prevelika, saj potem ni več obvladljiva. Najbolj primerna velikost ekipe je 5-9 oseb [2].

Naročnik skupaj s skrbnikom izdelka določi seznam zahtev, ki jih oblikuje v obliki uporabniških zgodb. Te uporabniške zgodbe se zapišejo v seznam zahtev. Skrbnik izdelka nato prioritizira seznam.

Vsak sprint se začne s sestankom za načrtovanje sprinta (angl. *Sprint Planning Meeting*). Udeležijo se ga skrbnik izdelka, skrbnik procesa in celotna ekipa razvijalcev. Prisotni so lahko tudi zainteresirani člani vodstva in naročniki. Na sestanku določijo hitrost sprinta. Sestanek se izvede v dveh fazah. V prvi fazi skrbnik izdelka predstavi zgodbe z najvišjo prioriteto. Potem se prisotni skupaj odločijo, katere zgodbe bodo prestavili s seznama zahtev na seznam nalog sprinta. V drugi fazi sestanka se nato dogovorijo, koliko dela lahko opravijo in označijo, katere zgodbe bodo upoštevali glede na določeno hitrost sprinta. Če je to prvi sestanek, morajo določiti še dolžino sprinta (npr. 2 tedna), katero potem uporabljajo za vse nadaljnje sprinte.

Po sestanku se začne izvajati delo na zgodbah. Razvijalci lahko razdelijo posamezno zgodbo na manjše enote, ki se imenujejo naloge. Zgodbo se deli na več nalog iz več razlogov. Če lahko naloge v zgodbi enostavno izvaja več razvijalcev sočasno, se zgodbo razdeli na več nalog. Razdelitev na naloge nam pomaga odkriti del zgodbe, ki bi ga lahko pozabili implementirati. Če obstaja korist, da vemo koliko zgodbe je že končane, potem se jo razdeli na več nalog [14].



Slika 2.1. Izpis Scrum table

Med delom člani ekipe uporabljajo Scrum tablo, kot je prikazana na sliki 2.1. Tabla vizualno predstavlja potek dela. Na levi strani imamo seznam nalog sprinta. Vsebina seznama je bila definirana med sestankom za načrtovanje sprinta. Nato člani ekipe prestavijo zgodbe (in naloge, če so zgodbo razdelili) na prvi stolpec znotraj table. Za strukturo table se odloči ekipa sama, v našem primeru pa smo ustvarili stolpce »ni začeto«, »v izvajanju«, »potrebuje pregled« in »zaključeno«. Stolpec »ni začeto« vsebuje zgodbe, ki jih člani planirajo za trenutni dan. Stolpec »v izvajanju« vsebuje zgodbe, ki so trenutno v delu. Člani prestavijo implementirane zgodbe v stolpec »potrebuje pregled«, ki je namenjen testiranju. Ko je bila implementacija zgodbe stestirana, jo prestavijo v stolpec »zaključeno«. Tako imajo vsi člani in zainteresirani pregled nad trenutnim stanjem sprinta.

Seznam nalog sprinta se med izvajanjem sprinta ne sme spreminjati. Tabla je v lasti ene ekipe, ekipa pa se sama odloči za vrstni red izvajanja zgodb s seznama. Skrbnik izdelka sedaj ne sme več dodajati ali odvzemati zgodb s seznama nalog sprinta, lahko pa ureja in prioritizira seznam zahtev. Zgodbe se lahko izjemoma doda na seznam nalog sprinta v primeru, da je ekipa predčasno dokončala planirano delo. Takrat lahko ekipa zaprosi skrbnika izdelka za dodatno zgodbo.

Med izvajanjem sprinta vsak dan vodimo dnevni sestanek, na katerem morajo biti prisotni vsi člani ekipe. Pri dnevnih sestankih naročniki ali vodje v podjetju niso prisotni, oz. ne smejo postavljati vprašanj, ali kako drugače preusmerjati pozornosti. Sestanek se mora izvajati vsak dan ob isti uri in ne sme preseči 15 minut. Če potrebujemo več časa, se zahteva za podaljšek zabeleži, zainteresirani pa se nato v ožjem številu sestanejo po dnevnem sestanku. Sestanek vodi skrbnik procesa. Člani ekipe morajo biti obveščeni, na čem delajo drugi člani. Tako si lahko člani svetujejo, če so naleteli na podobne težave, in tako lažje skupaj premagujejo ovire. Na dnevnih sestankih se člani ekipe pomenijo o treh bistvenih vprašanjih:

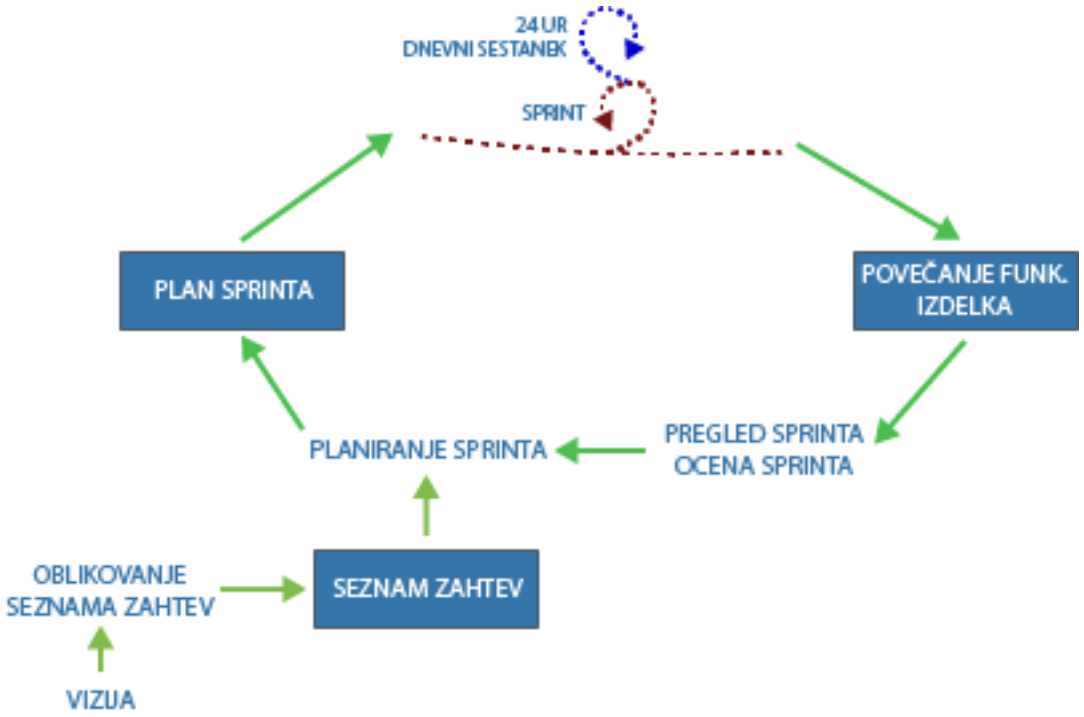
- Kaj ste naredili od zadnjega sestanka (včeraj)?
- Kaj boste storili do prihodnjega sestanka (danes)?
- Kaj vas ovira pri vašem delu oz. doseganju ciljev?

Skrbnik procesa vsakega člana ekipe vpraša ta tri vprašanja in si zabeleži njihove odgovore. S pomočjo dnevnih sestankov odkrijemo ovire, skrbnik procesa pa ima nalogo, da te ovire odpravi. Ovira je lahko neizkušenost člana ekipe z določenim problemom, ali pa npr. pomanjkanje papirja. Ovire lahko poskusi odpraviti sam skrbnik procesa, ali pa za to obvesti skrbnika izdelka oz. direktorja podjetja.

Ob koncu sprinta mora obstajati delujoča programska koda, pripravljena za predajo v produkcijo ter da lahko izdelek pokažemo naročniku. Izvede se sestanek za pregled sprinta (angl. *Sprint Review Meeting*). Na tem sestanku razvojna ekipa predstavi rezultate zadnjega sprinta naročniku in ostalim zainteresiranim. Izvede se izračun dejanske hitrosti sprinta. Pri izračunu seštejemo samo točke tistih zgodb, ki so bile opravljene v celoti in so pripravljene za uporabo. Delno končanih zgodb ne moremo upoštevati, saj je zelo težko oceniti, v kolikšnem deležu je bila zgodba že opravljena, nedokončano delo pa ponavadi ne predstavlja nobene vrednosti za naročnika [14].

Za tem sledi ocena sprinta (angl. *Sprint Retrospective Meeting*), v katerem skrbnik procesa skupaj z ekipo oceni potek dela v končanem sprintu in skupaj predlagajo morebitne izboljšave procesa. Tako se proces konstantno izboljšuje.

Če s projektom še nismo zaključili, se cikel ponovi. Začne se nov sprint s sestankom za načrtovanje sprinta. Slika 2.2 prikazuje celoten proces [3].



Slika 2.2. Diagram izvajanja metodologije Scrum

2.2 Primerjava Scruma z drugimi metodologijami

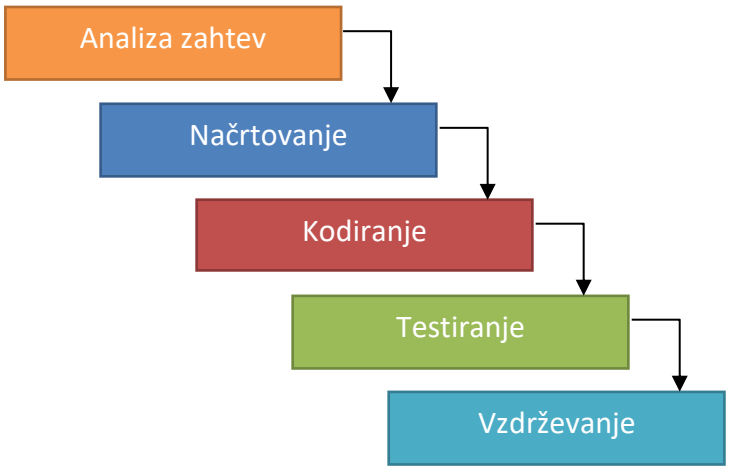
Za primerjavo smo vzeli nekaj najbolj razširjenih metodologij za razvijanje programske opreme. V naslednjih podpoglavjih bomo predstavili osnovne lastnosti vsake metodologije, ter njihove poglavitne razlike s Scrumom.

2.2.1 Klasični razvojni model

Klasični, linearni ali kaskadni življenjski cikel je najstarejši postopek razvoja programske opreme [21]. Imenuje se tudi metoda slapov (angl. *waterfall* model) [4], ki izvira iz značilne oblike strukture izvajanja projekta. Vse aktivnosti si sledijo zaporedno, na predhodne faze pa se ni možno vračati. Zaradi tega morajo biti zahteve popolno in nedvoumno definirane že na samem začetku. Po vsaki fazi je smiselno poleg verifikacije rezultatov izvesti tudi validacijo.

Verifikacija je preverjanje pravilnosti rezultatov, ki jih dobimo pri izvajanju posamezne razvojne faze [21]. **Validacija** upošteva širši vidik in preverja, če se rezultati posamezne faze ujemajo z osnovnimi zahtevami, ki so bile postavljene na začetku razvoja.

Slika 2.3 prikazuje vseh pet korakov in prehode med njimi. Ti predstavljajo celoten proces izvršitve projekta od začetka do konca.



Slika 2.3. Grafični prikaz klasičnega razvojnega modela

Analiza zahtev je prvi korak k uspešni izpeljavi razvoja programske opreme. V tej fazi moramo podrobno identificirati in dokumentirati zahteve naročnika. Zahteve podamo v dokumentu, ki se imenuje specifikacija zahtev. Specifikacija mora zajemati zahteve naročnika in spoznanja analitika. Ta dokument predstavlja osnovo za preostale faze in se uporablja kot kriterij za preverjanje uspešnosti izdelave projekta.

V **načrtovanju** izdelamo načrt programske opreme. Za osnovo uporabimo specifikacijo zahtev. Kvaliteta programske opreme je v največji meri odvisna od izdelanega načrta. Slabo načrtovano programsko opremo je težko testirati in vzdrževati. Na dolgi rok se izkaže, da je lahko nekvalitetno izdelan program veliko dražji za vzdrževanje. Načrt mora zajemati:

- načrt programske arhitekture – delitev problema na manjše module,
- načrt podatkovnih struktur – shema podatkovne baze,
- načrt posameznih sklopov programa oz. modulov – postopki morajo biti nedvoumno določeni, ponavadi pa jih zapišemo v obliki psevdokode (angl. *pseudocode*) ali z diagramom poteka

V fazi **kodiranja** prenesemo načrt programske opreme v poljuben programski jezik. Če je načrt izdelan dovolj podrobno, je programski del predvsem mehansko opravilo. Pri

programiranju je zelo pomembno, da kodo sproti dokumentiramo. Če bi kodo dokumentirali šele ob zaključku projekta, bi imeli na koncu veliko več dela z izdelavo komentarjev.

Testiranje je pri razvoju programske opreme osrednja aktivnost, ki preverja kvaliteto programa. Napake v razvoju moramo že vnaprej preprečevati in sproti odkrivati pomanjkljivosti. Čim kasneje v razvoju se izvajajo spremembe, tem dražja je sprememba. Izvajanje sprememb v fazi vzdrževanja je lahko do stokrat dražje, kot če bi spremembo zajeli v fazi analize zahtev. Testiranje preverja delo, ki je bilo opravljeno med analizo, načrtovanjem in kodiranjem. Če testiranje izvajajo isti ljudje, ki izvajajo delo v analizi, načrtovanju ali kodiranju, lahko pride pri tem do konflikta interesov. Izberejo lahko lahke testne primere, ali pa izvajajo testiranje po istih korakih, kot so opravljali razvoj.

Faza **vzdrževanja** programske opreme zajema nenehno prilagajanje programa različnim zunanjim spremembam. Te spremembe so lahko spremembe v zakonodaji, razširitev poslovanja podjetja, ali pa tehnična zastarelost. Včasih smo primorani zamenjati celoten sistem z novim, če nov operacijski sistem ali novejša strojna oprema ne podpirata več našega programa.

Vzdrževanje se pojavlja iz različnih razlogov:

- razširitev funkcionalnosti (50%),
- prilagoditev obstoječih funkcionalnosti (25%),
- popravki (20%),
- drugo (5%)

Naročnik pri klasičnem razvojnem modelu tekom izvajanja ne more podati predlogov ali sprememb. Ko pa na koncu dobi v roke končan izdelek, lahko da ni tak, kot si ga je zamislil, ali pa ga v taki obliki ne potrebuje več. Scrum rešuje ta problem s kratkimi sprinti, s katerimi se najprej lotimo reševanja uporabniških zgodb, ki največ doprinesejo h končnemu izdelku, naročnik pa po vsakem zaključenem sprintu sproti dobiva rezultat.

Klasičen razvojni model je primeren za rutinske projekte, še posebej takrat, ko ima razvojna ekipa že dovolj izkušenj na določenem aplikacijskem področju [21].

2.2.2 Ekstremno programiranje – XP

Metodologija ekstremno programiranje (angl. *Extreme Programming*) je disciplina poslovnega razvoja programske opreme, ki usmerja celotno ekipo v doseganje skupnih ciljev. Ime je dobila po tem, da uporablja principe razvoja v ekstremni obliki [3].

Z ekstremnim programiranjem izvajamo samo procese, ki nam pomagajo ustvarjati dodano vrednost za naročnika. XP se prilagaja nejasnim ali hitro spreminjajočim se zahtevam. Ukvarja se z vrednotami, principi in praksami [20].

Vrednote prinašajo pomen praksam. Pri razvoju moramo ceniti sprejemanje najboljše odločitve za celotno ekipo. Vsak član ima lahko lastne predstave vrednot. Če vsi člani ne sodelujejo kot enotna ekipa, to ne pomaga ekipi pri uspevanju. XP zajema pet vrednot za usmerjanje razvoja:

- **Komunikacija** – za ekipo predstavlja najpomembnejšo vrednoto. Ko se v razvoju pojavijo težave, ponavadi nekdo že pozna rešitev.
- **Preprostost** – predstavlja najbolj intelektualno intenzivno vrednoto. Oblikovanje sistema, da je dovolj preprost in rešuje samo trenutni problem, predstavlja težko delo. Razvijalci si preprostost določene rešitve razlagajo različno, saj imajo lahko drugačno tehnično podlago in znanje.
- **Pogum** – pojavlja se v obliki potrpežljivosti. Če ne vemo kaj je razlog za nastanek težave, potrebujemo pogum za čakanje, da pride težava jasno do izraza.
- **Spoštovanje** – če člani ekipe ne cenijo drug drugega, potem XP ne more delovati.

Principi so smernice za življenje na posameznih področjih. XP predstavlja nekaj osnovnih principov za usmerjanje razvoja programske opreme:

- **Človečnost (angl. *Humanity*)** – programsko opremo razvijajo ljudje, zato je pomembno, da se zavedamo človeškega faktorja. Ljudje se morajo počutiti del ekipe, imeti osnovno varnost, priložnost za razširjanje znanja in občutek dosežka.
- **Ekonomija (angl. *Economics*)** – naše delo mora imeti poslovno vrednost in služiti poslovnim namenom.
- **Skupne koristi (angl. *Mutual Benefit*)** – vsaka vrsta aktivnosti v razvoju mora koristiti vsem vključenim. Predstavlja najpomembnejši princip XP in se ga je najtežje držati. Preprečevati je potrebno rešitve, ki služijo enemu razvijalcu in škodijo drugemu. Primer take aktivnosti je pisanje obsežne interne dokumentacije. Služi potencialnemu razvijalcu v prihodnosti, ne pa trenutnemu razvijalcu, kateremu se razvoj precej upočasni zaradi pisanja obsežne dokumentacije. Skupno korist dosegamo npr. s pisanjem testov, ki bodo koristili tudi drugim programerjem, ki pridejo za nami.
- **Samo-podobnost (angl. *Self-Similarity*)** – strukturo ene rešitve poskušamo kopirati na nov kontekst. Ko najdemo del kode ali prakso, ki deluje, se je držimo in jo poskusimo ponoviti na drugih kontekstih.

- **Izboljšava (angl. *Improvement*)** – v razvoju ni popolnega procesa ali popolne uporabniške zgodbe. Proces in zgodbo pa lahko izpopolnjujemo. Za razvoj danes moramo dati vse od sebe in biti pri tem pozorni na naše slabosti, da smo lahko jutri še boljši.
- **Raznolikost (angl. *Diversity*)** – razvojne ekipe, kjer so si vsi podobni, niso učinkovite. Ekipa mora vsebovati več različnih znanj, odnosov in perspektiv, da vidimo težave z več zornih kotov, in najdemo čim več možnih načinov za reševanje težav.
- **Refleksija (angl. *Reflection*)** – naše delo moramo stalno analizirati, da ugotovimo, zakaj nam stvari uspevajo ali ne. Na napakah se učimo.
- **Tok (angl. *Flow*)** – tok v razvoju programske opreme pomeni stalno dostavo vrednosti s hkratnim vključevanjem vseh aktivnosti razvoja (izdajanje, testiranje, povratne informacije). Proces razvoja naj se izvaja v dostavljanju vrednosti v majhnih intervalih z vedno večjo pogostostjo.
- **Priložnosti (angl. *Opportunity*)** – na težave moramo gledati kot možnosti za spremembe. Na težavah se moramo učiti in se izpopolnjevati.
- **Redundanca (angl. *Redundancy*)** – težke probleme rešujmo na več različnih načinov. Če odpove ena rešitev, bo druga rešitev preprečila katastrofo.
- **Neuspeh (angl. *Failure*)** – če imamo težave z uspevanjem, ni nič narobe, če ne uspemo. Iz neuspeha se naučimo nekaj dragocenega.
- **Kvaliteta (angl. *Quality*)** – projekti se ne bodo izvedli hitreje, če zmanjšamo kvaliteto. Ponavadi pridemo z dvigovanjem kvalitete hitreje do rezultatov.
- **Majhni koraki (angl. *Baby Steps*)** – izvajanje velikih sprememb naenkrat je nevarno. Majhni koraki pridejo do izraza pri testiranju, kjer razvoj poteka test za testom.
- **Sprejeta odgovornost (angl. *Accepted Responsibility*)** – odgovornosti ne smemo dodeliti članom ekipe, ampak jo lahko člani samo sami sprejmejo. Če nam nekdo dodeli odgovornost za neko nalogo, se lahko samo mi odločimo, če jo sprejmemo ali ne. Kdor se javi za sprejem odgovornosti za nalogo, naj jo tudi sam časovno oceni.

Prakse so tisto, kar izvajamo dan za dnem. So jasne in objektivne. Za prakse se odločamo glede na dane situacije. Prakse uporabimo po potrebi in ne zato, ker se želimo pohvaliti, da upoštevamo čim več praks iz ekstremnega programiranja. Prakse predstavljajo preskušene metode, ki nam lahko pomagajo pri izboljšavi procesa razvoja programske opreme. Delimo jih na primarne in posledične. Primarne so samostojne, za njihovo uporabo pa se lahko odločimo individualno:

- **Primeren prostor** – razvoj naj poteka v dovolj velikem prostoru, ki lahko zajema celotno ekipo. Prostor pregradimo na manjše zasebne prostore, ali pa omejimo delavnik, saj ljudje včasih potrebujemo zasebnost.

- **Celotna ekipa** – v ekipo vključimo člane z različnih področij. Vključimo nabor ljudi, ki jih potrebujemo na projektu.

- **Informativni delovni prostor** – zainteresiran opazovalec mora ob vstopu v prostor ekipe v petnajstih sekundah ugotoviti, kako napreduje projekt. Pri tem si pomagamo s predstavitvijo uporabniških zgodb ali nalog na stenski tabli.

- **40-urni delavnik (angl. *Energized Work*)** – delavnik naj bo dolg samo toliko, da smo še produktivni na dolgi rok. Če se izgorimo danes in s tem pokvarimo naslednje dva dni, to ni dobro za ekipo.

- **Programiranje v parih** – zajema dva programerja, ki si delita eno tipkovnico in en zaslon. S skupnim znanjem pišeta programsko kodo. Prvi programer vnaša kodo preko tipkovnice in razmišlja za nekaj vrstic kode naprej, drugi programer pa gleda kodo in ugotavlja, kje se lahko pojavijo težave in na kaj vse lahko spremembe vplivajo [14].

- **Uporabniške zgodbe** – planiranje enot funkcionalnosti, ki naročniku prinesejo dodano vrednost.

- **Tedenski cikel ali iteracija** – delo izvajamo v iteracijah, dolgih en teden. Na začetku iteracije vodimo sestanek, na katerem preverimo napredek, naročniki izberejo primerno količino uporabniških zgodb za en teden, zgodbe pa razdelimo na manjše naloge. Člani prevzamejo naloge in jih časovno ocenijo.

- **Četrtletni cikel ali izdaja** – med planiranjem izdaje identificiramo ozka grla, planiramo zahteve in izberemo uporabniške zgodbe. Razmislimo o ekipi, projektu, njegovem napredovanju in usklajujemo projekt z višjimi cilji.

- **Odpad (angl. *Slack*)** – v vsakem načrtu vključimo nekaj manjših nalog, ki jih lahko zavržemo, če nam zmanjka časa za njihovo izdelavo. Te naloge nam dopuščajo variabilnost, v primeru, da nam ostane nekaj odvečnega časa.

- **Deset minutno prevajanje kode (angl. *Ten-minute Build*)** – avtomatizirani testi naj se ne izvajajo več kot 10 minut. Testi, ki trajajo dlje, se bodo izvajali veliko redkeje, kar pa zmanjša možnosti za pridobitev povratnih informacij.

- **Nepretrgana integracija (angl. *Continuous Integration*)** – ko zaključimo s pisanjem dela kode in spremembe shranimo v repozitorij, se na strežniku izvede prevajanje kode in testiranje sprememb. Če sistem pri tem naleti na napako, bo razvijalec o tem sproti obveščen po elektronski pošti.

- **Testno voden razvoj (angl. *Test-First Programming*)** – pred pisanjem kode najprej napišemo avtomatiziran test. S tem se omejimo pri količini odvečne kode, pridobimo večje zaupanje v avtorja kode in imamo jasna navodila za nadaljnje programiranje.

- **Postopna zasnova (angl. *Incremental Design*)** – v zasnovo sistema investiramo vsak dan. Zasnovo sistema nadgrajujemo po potrebi, najbolje pa tik pred implementacijo. Na primer, uporabniške zgodbe napišemo na začetku samo na grobo, bolj podrobno pa jo opišemo šele tik preden se lotimo implementacije rešitve. V tej praksi zajema XP tudi stalno preoblikovanje programske kode (angl. *refactoring*). Preoblikovanje kode zajema prestrukturiranje in spremembo programske kode. Ekstremno programiranje zagovarja stalno pozornost na preoblikovanje kode. Kadarkoli uporabnik spreminja kodo, ni le želeno, da se koda preoblikuje, ampak je to celo zahtevano [14].

Posledične prakse se težko izvedejo brez obvladovanja primarnih praks. Zajemajo pravo vključitev naročnika v proces razvoja, postopno izdajanje programa, ohranjanje strukture ekip, krčenje ekip ob povečanju učinkovitosti ekipe, analiza glavnega vzroka za težavo, deljenje kode s celotno ekipo, ohranjanje samo programske kode in testov, vzdrževanje ene same različice aktualne programske kode, dnevno izdajanje produkcijskih različic, zmanjševanje tveganja s podpisovanjem krajših poslovnih pogodb z omejenim obsegom in izdelava sistemov s plačilom po uporabi (angl. *pay-per-use*).

Uporabniške zgodbe izvirajo iz ekstremnega programiranja [14], uporabljajo pa se tudi pri Scrumu.

Scrum se z vrednotami in principi ne ukvarja neposredno. Obe metodologiji sta si podobni v nekaterih praksah, kot je inkrementalni razvoj z izdajami in iteracijami (sprinti), uporabniške zgodbe, vključevanje naročnika v razvoj in uporaba vizualne table za predstavitev trenutnega stanja izvajanja iteracije.

XP ne določa fiksnih vlog, saj je naloga vsakega člana ekipe, da prispeva najboljše kar premore za skupen uspeh celotne ekipe. Pri Scrumu so glavne vloge jasno definirane (naročnik, skrbnik procesa, skrbnik izdelka in član ekipe), lahko pa jih dodajamo po potrebi.

2.2.3 Kanban

Kanban je vitka metoda za urejanje in izboljšavo dela, ki ne predstavlja življenjskega cikla ali procesa vodenja projektov, temveč pristop za uvajanje sprememb na obstoječem razvoju programske opreme [15]. Ne pričakuje takojšnje revolucije v načinu dela, ampak spodbuja postopne spremembe.

Pri Kanbanu je količina dela v izvajanju (angl. *work in progress*) omejena. Čim se katerakoli delovna enota zaustavi, ta začne zavirati sistem. Če je zaustavljenih dovolj delovnih enot, lahko to zaustavi izvajanje celotnega sistema. Zaustavitev spodbudi celotno ekipo ali podjetje k reševanju blokade, da ponovno vzpostavijo normalen potek dela. Kanban se od Scruma razlikuje v tem, da izvaja omejitev dela neposredno na tabli, Scrum pa delo omejuje posredno s hitrostjo sprinta. Pri Kanbanu se z omejitvijo količine delovnih enot na posameznih korakih izdelave poveča predvidljivost časa izvajanja nalog, s tem pa so dostave rezultatov bolj zanesljive. To spodbuja tudi k večji osredotočenosti in jasno izpostavi omejitve sistema.

Kanban izpostavi ozka grla, čakalne vrste in odpadek – vsi vplivajo na to, koliko dragocenega dela bo na koncu narejenega.

Kanban svetuje omejevanje nedokončanega dela v izdelavi, kar zmanjša odpadke zaradi večopravilnosti in preklopa konteksta, izpostavi operativne težave in spodbuja sodelovanje z namenom izboljševanja sistema. Scrum se ne ukvarja posebej z večopravilnostjo, dokler se delo opravlja samo na nalogah, ki so zajete v določenem sprintu.

Če se v podjetju ukvarjamo predvsem s podporo uporabnikom ali vzdrževanjem sistema, se naloge pojavijo v obliki prijavljenih težav. Te težave se lahko pojavljajo vsakodnevno, na njih pa je potrebno hitro ukrepati. Ne moremo čakati na naslednjo iteracijo, zato za tako vrsto dela Scrum ni primeren. V tem primeru uporabimo Kanban, ki ne omejuje delo na iteracije [15].

Če želimo v podjetju vzpostaviti bolj efektiven proces, je za to bolj primeren Scrum, če pa imamo v podjetju že delujoč delovni proces, pa lahko uporabimo Kanban za izboljšavo čez čas, ne da bi s tem spremenili celoten delovni proces [16].

Kanban ne predpisuje, kako naj bo naročnik vključen, medtem ko je naročnik globoko integriran v Scrumu pri začetnemu planiranju sprinta in zaključku sprinta, kjer lahko naročnik dobi v vpogled delujoč izdelek z opravljenim delom s tekočega sprinta.

V eni Kanban tabli je lahko zajetih več različnih ekip, lahko pa vključujejo kar celotne oddelke in organizacije. Scrum je veliko bolj strog in dovoljuje na eni tabli zgolj zgodbe ali naloge določene ekipe. Slika 2.4 prikazuje primer Kanban table. Številka »3« pri stolpcu »V

testiranju« pomeni, da v ta stolpec hkrati ne smemo dati več kot tri enote dela. Če se pojavijo 3 ali več enot dela, je potrebno čimprej razrešiti ozko grlo, saj se v nasprotnem primeru začnejo nabirati naloge na levi strani table.

Slika 2.4. Primer Kanban table

Večje razlike med Scrumom in Kanbanom so prikazane v tabeli 2.1.

Scrum	Kanban
Izvajanje dela v sprintih	Iteracije so neobvezne
Določa dnevne sestanke	Dnevni sestanki niso določeni
Ekipa se zaveže za izvajanje določene količine dela na iteracijo	Zavezanost ni obvezna
Hitrost sprinta je privzeta enota za planiranje sprinta in izboljšavo procesa	**Povprečen čas izvajanja naloge** je enota za planiranje in izboljšavo procesa
Člani z različnim področjem znanja	Neobvezno, ekipe so lahko specializirane
Zgodbe je potrebno razčleniti, da jih lahko	Velikost naloge ni določena

vmestimo v sprint	
Burn-down diagram	Ni predvidenega diagrama
Delo v izvajanju **omejeno posredno** (preko sprinta)	Delo v izvajanju **omejeno neposredno** (stanje na tabli)
Ocena kompleksnosti zgodb s točkami	Ocene niso obvezne
Ni dovoljeno dodajati zgodb na začeti sprint	Naloge se lahko dodajajo kadarkoli, če to dopušča kapaciteta na tabli
Seznam nalog sprinta in Scrum tabla sta v lasti določene ekipe	Kanban tabla je lahko deljena med več ekipami
Določa vloge skrbnika izdelka, skrbnika procesa in članov ekipe	Ne določa vlog
Scrum tabla se ponovno zgradi z vsakim novim sprintom	Kanban tabla se ne ponastavlja
Skrbnik izdelka prioritizira seznam zahtev	Prioritizacija je neobvezna

Tabela 2.1. Razlike med Scrumom in Kanbanom

2.2.4 Scrum z ekstremnim programiranjem

Scrum z ekstremnim programiranjem sam po sebi ni metodologija, temveč združitev moči obeh metodologij. Skupaj se dobro dopolnjujeta, saj sta oba agilni metodologiji, vendar prihajata z drugih koncev agilnega razvoja.

Ekipe se lahko same odločijo, kako bodo v svoji ekipi definirale Scrum z ekstremnim programiranjem. Idealna pot je, da se v podjetju najprej uvede Scrum, nato pa se po potrebi dodaja principe ekstremnega programiranja [5]. Ekipa se lahko npr. odloči, da uvede poleg Scruma še testno voden razvoj z avtomatiziranimi testi, programiranje v parih in vzpostavljene kodirne standarde.

2.2.5 Metoda dinamičnega razvoja sistemov – DSDM

Filozofija DSDM je, da mora imeti vsak projekt jasno opredeljene strateške cilje ter se osredotočiti na zgodnjo dostavo dejanskih koristi podjetju, pri tem pa so naloge določene zelo natančno že v uvodni fazi [7]. Scrum pri samih sestankih za načrtovanje sprinta ne opredeljuje vseh zgodb v podrobnosti, saj je sprint omejen s hitrostjo in se posvetimo samo tistim zgodbam, ki jih bomo upoštevali v tekočem sprintu. Preostale zgodbe se lahko pusti v obliki epskih zgodb. Te se nato opredelijo bolj natančno šele takrat, ko jih moramo implementirati. DSDM ima lahko tu težave, če so naloge opredeljene preveč natančno in se nato izkaže, da so temeljile na neutemeljenih predpostavkah.

DSDM je podoben Scrumu v tem, da oba spodbujata sodelovanje med naročnikom in izvajalcem projekta, z iterativnim razvojem pomagata pri pravočasnem doseganju rezultatov in izvajata postopno gradnjo na trdnih temeljih.

Poglavje 3 Zajem zahtev

3.1 Uvod

V poslovnem svetu moramo biti organizirani, če želimo uspešno dokončati delo v predvidenih časovnih rokih. Napredna podjetja se zato poslužujejo različnih agilnih metodologij, ki pa jih lahko vodijo na papirju, brez uporabe računalnika. Papirji se lahko založijo, popravke besedil pa je težko izvajati. Če želimo deliti papirje z drugimi, jih je potrebno kopirati. V primeru, da želimo opravljati delo na oddaljenih lokacijah, potrebujemo izvod seznama nalog vedno s seboj.

Da se izognemo tem težavam, si pri tem pomagamo z orodji za vodenje projektov. Orodja nam olajšajo celoten proces razvoja izdelka. Če je orodje dostopno preko interneta, imamo na oddaljeni lokaciji na voljo celotno zgodovino dela, enostavno posodabljamo in razvrščamo uporabniške zgodbe po prioriteti. Scrum tablo imamo na voljo vedno posodobljeno. Vse spremembe so takoj vidne vsem članom ekipe.

Tudi sami smo v našem podjetju preskusili več takih orodij, do sedaj pa so se vsa izkazala za preobsežna, ali pa niso nudila dovolj funkcionalnosti za naše potrebe. Sam sem bil zadolžen, da raziščem to področje in najdem za nas najbolj primerno orodje. Pri tem je predstavljala faktor za izbor tudi cena najema ali nakupa obstoječe rešitve. Po raziskavi smo dognali, da obstoječa orodja niso primerna, zato smo se odločili izdelati lastno rešitev.

Zahteve našega podjetja so bile sledeče:

- Razvijalci in vodstvo morajo imeti možnost uporabljati več kot en projekt naenkrat.
- Projekt bomo vodili po metodologiji Scrum z uporabniškimi zgodbami, sprinti, dnevnimi sestanki, seznamom zahtev in Scrum tablo.
- Na projekt moramo imeti možnost pripenjati datoteke, ki služijo za dopolnilno razlago ali dokumentacijo.
- Vsebovati mora sistem za izmenjavo sporočil za lažjo komunikacijo.
- Naročnik mora imeti dostop za branje vsebin in komuniciranje z ekipo na dodeljenem projektu – v nadaljevanju je zajet v vlogi »navaden uporabnik«.

- Člani ekipe morajo imeti možnost beleženja časa dela, na voljo pa mora biti priročen časovni števec, ki nam olajša vnos časa.
- Orodje mora biti dostopno preko spleta.

V sklopu diplomske naloge smo si zadali cilj dodati še diagram preostalega dela in diagram opravljenega dela.

3.2 Naloge orodja

Pri izdelavi orodja smo se odločili, da se lotimo dela po metodologiji Scrum. Najprej smo morali razmisliti o tem, kaj želimo s orodjem doseči. Spisali smo zgodbe, ki predstavljajo določene akcije, katere mora orodje omogočati uporabnikom za uspešno vodenje projekta s Scrumom.

Najprej smo definirali uporabniške vloge sistema:

- **Navaden uporabnik** – ima pravice za branje vsebin in komuniciranje z ekipo preko sporočil, ne more pa dodajati ali spreminjati ničesar drugega.
- **Član ekipe** – lahko pregleduje vsebine, dodaja in ureja naloge na zgodbah za razčlenitev dela, pripenja datoteke, razporeja naloge na Scrum tabli in vnaša porabljen čas na nalogah.
- **Skrbnik procesa** – lahko pregleduje vsebine, dodatno pa lahko upravlja s sprinti in dnevnimi sestanki.
- **Skrbnik izdelka** – lahko pregleduje vsebine, dodatno pa lahko ureja z zgodbami in pripenja datoteke.
- **Vodja projekta** – administrator, ki ima popoln dostop.

V nadaljevanju sledijo vse zgodbe z opisi funkcionalnosti orodja. Razdelili smo jih po vsebinskih sklopih. Če je zgodba namenjena vsem vlogam uporabnikov, bomo zapisali to kot »katerikoli uporabnik«.

Za začetek moramo imeti možnost ustvarjanja projekta. Projekt lahko ustvarimo za določenega naročnika ali pa za interno uporabo, ki ni vezana na nobenega naročnika. Ko se nam čez čas nabere večje število projektov, pride prav tudi funkcionalnost iskanja po vsebinah projektov:

- **Ustvarjanje novega projekta**
  Kot vodja projekta želim imeti možnost ustvarjanja novega projekta, zato da lahko začnemo vodenje projekta po metodologiji Scrum.

- **Iskanje po projektih**
  Kot katerikoli uporabnik si želim imeti možnost iskanja po priponkah, naslovih nalog, uporabnikih in opisih ovir, zato da lažje najdem tisto kar iščem.

Najbolj pomemben del projekta so njegovi člani. Na projekt moramo imeti možnost dodajati tako člane ekipe, kot tudi zunanjega naročnika, da ima na voljo pregled nad trenutnim stanjem:

- **Dodajanje članov in strank na projekt**
  Kot vodja projekta želim imeti možnost dodajanja članov in naročnika na projekt, zato da jih vključim v proces izdelave projekta.

- **Povabilo novih uporabnikov**
  Kot vodja projekta si želim, da lahko povabim novega uporabnika na projekt, ki še nima uporabniškega računa, zato da ga lahko vključim v proces razvoja, saj registracija ni odprta za javnost.

- **Spreminjanje uporabnikovih pravic**
  Kot vodja projekta želim imeti možnost, da lahko uporabnikom spremenim obstoječe pravice na projektu, zato da lahko na primer sami nadzirajo projekt, ko sem odsoten, ali pa jim omejiti dostop zaradi njihovega zmanjšanja obsega dela.

Ko imamo enkrat pripravljeno osnovno strukturo na projektu pridejo na vrsto sprinti. Po metodologiji Scrum moramo projekt voditi v iteracijah. Za to moramo imeti možnost ustvarjanja novega sprinta, dodajanje uporabniških zgodb na sprint, pregled nad zgodovino in zaključek sprinta:

- **Ustvarjanje novega sprinta**
  Kot skrbnik procesa si želim možnost ustvarjanja novega sprinta ko se je trenutni zaključil, zato da lahko nadaljujemo z naslednjim ciklom izvajanja projekta, če ta še ni zaključen.

- **Urejanje sprinta**
  Kot skrbnik procesa želim urejati sprint, zato da ga lahko zaključim, ali spremenim skrbnika procesa ali skrbnika izdelka.

- **Pregled podrobnosti sprinta**

Kot katerikoli uporabnik si želim imeti pregled podrobnosti sprinta, zato da lahko vidim katere uporabniške zgodbe in naloge vsebuje, kdaj se je sprint začel, kdaj se konča ter kakšno je trenutno stanje izvajanje sprinta.

- **Dodajanje uporabniške zgodbe**
  Kot skrbnik izdelka si želim, da lahko dodam uporabniško zgodbo, zato da jo lahko določim zahtevo.

- **Dodajanje uporabniške zgodbe na sprint**
  Kot skrbnik procesa si želim, da lahko dodam uporabniško zgodbo na sprint, zato da jo lahko upoštevamo pri naslednjem sprintu.

- **Določanje točk uporabniške zgodbe**
  Kot skrbnik procesa ali član ekipe si želim, da lahko na uporabniški zgodbi določim število točk zgodbe (angl. *story points*), zato da jo lahko vključim v trenutni sprint glede na določeno hitrost.

- **Odstranjevanje uporabniške zgodbe s sprinta**
  Kot skrbnik procesa si želim, da lahko odstranim uporabniško zgodbo s sprinta, zato da lahko spremenimo seznam nalog na sprintu, ali pa jo upoštevamo v kasnejšem sprintu v procesu planiranja sprinta.

- **Prenos nedokončanih uporabniških zgodb na nov sprint**
  Kot skrbnik procesa si želim, da lahko ob ustvarjanju novega sprinta prenesem nedokončane uporabniške zgodbe z zadnjega zaključenega sprinta, zato da jih lažje upoštevamo v novem sprintu.

- **Zaključek sprinta**
  Kot skrbnik procesa si želim, da ob koncu sprinta lahko označim sprint kot zaključen, zato da lahko nastavim naslednjemu stanje »v izvajanju«.

- **Zgodovina sprintov na projektu**
  Kot skrbnik procesa, skrbnik izdelka ali član ekipe si želim imeti pregled nad zgodovino sprintov, ki prikazuje napredek izvajanja sprintov z zaključenimi točkami zgodb, zato da hitro presodim, kako napreduje izvajanje projekta.

Na projektu moramo imeti možnost upravljanja z uporabniškimi zgodbami in sprejemnimi testi na zgodbah:

- **Dodajanje nove uporabniške zgodbe**
  Kot skrbnik izdelka želim imeti možnost dodajanja nove uporabniške zgodbe na projekt, zato da lahko opišem moje zahteve članom ekipe.

- **Urejanje uporabniške zgodbe**
  Kot skrbnik izdelka želim imeti možnost urejanja uporabniških zgodb, zato da jih lahko dopolnim ali popravim.

- **Izpis podrobnosti uporabniške zgodbe**
  Kot katerikoli uporabnik si želim imeti podroben izpis podatkov o uporabniški zgodbi, zato da pridobm vse informacije v povezavi z njo.

- **Dodajanje sprejemnih testov na uporabniško zgodbo**
  Kot skrbnik izdelka si želim, da lahko na uporabniško zgodbo dodam enega ali več sprejemnih testov, zato da bolje opišem zgodbo in da bo jasno, kdaj je zgodba končana.

- **Odstranjevanje sprejemnih testov z uporabniške zgodbe**
  Kot skrbnik izdelka si želim, da lahko odstranim sprejemni test z uporabniške zgodbe, zato da počistim odvečne teste.

- **Označevanje sprejemnega testa kot upoštevanega**
  Kot član ekipe si želim, da lahko sprejemni test označim kot upoštevan, zato da sporočim skrbniku izdelka, kateri del zgodbe je že narejen.

Uporabniške zgodbe lahko razčlenimo na več nalog, da si jih lahko posamezni člani ekipe lažje porazdelijo med seboj:

- **Dodajanje nove naloge**
  Kot član ekipe si želim, da lahko dodam novo nalogo na uporabniško zgodbo, zato da lahko razdelim zgodbo na manjše kose.

- **Urejanje naloge**
  Kot član ekipe si želim imeti možnost, da lahko uredim prioriteto ali opis naloge, zato da lahko bolj natančno opišemo nalogo.

- **Spreminjanje pooblaščenca za izvajanje dela na nalogi**
  Kot člane ekipe si želim možnost spreminjanja pooblaščenega za izvedbo naloge, da lahko dodelim nalogo sebi.

- **Pregled seznama nalog s prekoračenim rokom izvedbe**
  Kot član ekipe si želim imeti seznam nalog, ki zamujajo rok izvedbe, zato da se posvetim najprej njim.

- **Podroben zaslon s podatki o nalogi**
  Kot katerikoli uporabnik si želim podroben pregled naloge, ki vsebuje opis, naložene datoteke in zabeleženo delo s časom, zato da imam vse podatke na enem mestu in s tem boljši pregled nad izvajanjem dela na nalogi, ter koliko časa je bilo pri tem porabljenega.

- **Ročni vnos porabljenega časa na nalogi z opisom dela**
  Kot član ekipe si želim možnost vnosa časa in opisa opravljenega dela na nalogi, zato da ostane na nalogi evidenca že opravljenega in kaj je še potrebno postoriti, ter koliko časa je bilo vloženega v izvajanje naloge.

- **Vnos časa na nalogi s časovnim števcem**
  Kot član ekipe si želim možnost začetka časovnega števca za posamezno nalogo, zato da ne pozabim, kdaj sem začel z delom, in lahko po zaključku dela ustavim števec, ki pa mi samodejno preračuna dolžino izvajanja dela.

- **Hitri pregled nad odprtimi nalogami vseh projektov**
  Kot član ekipe si želim, da bi imel na enem zaslonu pregled nad trenutnim stanjem projekta, zato da lažje ocenim, koliko je še dela na posameznem projektu.

Največjo preglednost nad trenutnim stanjem projekta imamo s Scrum tablo. Scrum tabla mora omogočati vizualno predstavitev stanja projekta in uporabniku prijazno interakcijo s tehniko povleci-in-spusti:

- **Izpis Scrum table**
  Kot član ekipe ali skrbnik procesa si želim imeti interaktivno Scrum tablo, zato da imam boljši pregled nad izvajanjem sprinta.

- **Premik nalog med stolpci Scrum table**

Kot član ekipe si želim možnost, da lahko premaknem nalogo med različnimi stolpci s pomočjo tehnike povleci-in-spusti, zato da lahko naznanim, da sem začel opravljati delo na nalogi, jo prestavim v zaključeno in na sploh lahko izvajam proces po metodologiji Scrum.

- **Dodajanje novega stolpca v Scrum tablo**
  Kot skrbnik procesa si želim, da lahko dodam nov stolpec na Scrum tablo, zato da lahko naredim dodaten korak za potovanje naloge oz. preurejanje strukture table po želji.

- **Urejanje stolpcev Scrum table**
  Kot skrbnik procesa želim, da lahko preuredim zaporedje stolpcev s tehniko povleci-in-spusti, ali pa uredim naslov, barvo naslovne vrstice ali izbrišem celoten stolpec (kar prestavi naloge nazaj v seznam zahtev), zato da lahko preuredim stukturo po potrebi.

- **Razvrščanje nalog v stolpcu Scrum table**
  Kot član ekipe si želim možnosti, da lahko naloge v določenem stolpcu Scrum table razvrstim po imenu, prioriteti, roku izvedbe in datumu ustvarjanja naloge, zato da lažje najdem določeno nalogo in se odločim, katere naloge se bom najprej lotil izvajati.

Na sprintu imamo lahko več dnevnih sestankov, odvisno od dolžine posameznega sprinta. Idealno bo na sprintu dnevni sestanek za vsak delovni dan:

- **Pregled nad dnevnimi sestanki določenega sprinta**
  Kot skrbnik procesa si želim imeti seznam dnevnih sestankov na pregledu podatkov o sprintu, zato da lahko pregledam kakšne so bile ovire članov ekipe pri izvajanju nalog.

- **Podrobnosti o dnevnem sestanku**
  Kot skrbnik procesa si želim pregled podatkov z dnevnega sestanka, kot so vprašanja članom ekip (»Kaj si naredil od zadnjega sestanka?«, »Kaj boš naredil do prihodnjega sestanka?« in »Kaj te pri delu ovira«), zato da imam boljši pregled nad tekočim delom in evidenco nad ovirami.

- **Dodajanje prisotnega na dnevni sestanek**
  Kot skrbnik procesa želim imeti možnost dodajanja prisotnih na dnevnem sestanku, zato da lahko zabeležim njihove napredek in ovire.

- **Vklop števca poteka dnevnega sestanka**
  Kot skrbnik procesa si želim, da lahko vključim časovni števec, ki šteje minute izvajanja dnevnega sestanka, zato da mi ni potrebno vklapljati štoparice in ročno vnašati porabljenega časa.

Za pomoč pri analiziranju napredovanja projekta smo ustvarili nekaj uporabniških zgodb, ki vključujejo grafe preostalega dela in narejenega dela in razne analitične izpise:

- **Pregled vnosa porabe časa in narejenega dela**
  Kot skrbnik procesa si želim, da lahko pregledam nazadnje vnešeno porabo časa in narejenega dela na nalogah in posledično tudi zgodbah, zato da lahko lažje sledim ekipi in imam pregled nad izvajanjem dela.

- **Pregled zadnje aktivnosti**
  Kot skrbnik procesa si želim enostaven pregled nad zadnjmi aktivnostmi vseh članov v ekipi, zato da imam lažjo predstavo kdo je delal na katerih zgodbah ali nalogah.

- **Pregled napredka z grafom preostalega dela**
  Kot skrbnik procesa ali skrbnik izdelka želim imeti možnost pregleda grafa preostalega dela (angl. *Burn-down chart*), zato da lahko ocenim, koliko dela je še preostalo neopravljenega na sprintu.

- **Pregled napredka z grafom narejenega dela**
  Kot skrbnik procesa ali skrbnik izdelka želim imeti možnost pregleda grafa narejenega dela (angl. *Burn-up chart*), zato da lahko ocenim, ali se na projekt dodaja preveč novih zgodb.

Na koncu ostanejo še dodatne funkcionalnosti, ki nam olajšajo izmenjavo podatkov med člani ekipe. To zajema pripenjanje priponk na naloge in projekte, ter izmenjava sporočil.

- **Pripenjanje datoteke na projekt**
  Kot skrbnik izdelka ali član ekipe si želim, da lahko pripnem datoteko na nalogo oz. projekt, zato da bolje razložim zgodbo, ali pa zagotovim gradivo za projekt.

- **Pripenjanje datoteke na nalogo**
  Kot član ekipe si želim, da lahko pripnem datoteko na nalogo, zato da bolje zagotovim gradivo za nalogo.

- **Prenos pripete datoteke**
  Kot katerikoli uporabnik si želim, da lahko prenesem naloženo datoteko, zato da lahko pregledam njeno vsebino.

- **Pregled vseh datotek na projektu**
  Kot katerikoli uporabnik si želim, da lahko na enem zaslonu vidim seznam vseh priponk celotnega projekta, zato da mi ni potrebno brskati čez vse zgodbe in naloge.

- **Izmenjava sporočil z ekipo**
  Kot katerikoli uporabnik si želim, da lahko ustvarim novo sporočilo na projektu, katerega bodo lahko prebrali vsi uporabniki na projektu, zato da si lahko lažje izmenjamo informacije, ki bi nam lahko prišle prav tudi kot zaznamki.

3.3 Predpogoji

Orodje samo po sebi ne more opraviti dela namesto človeka. Za uspešno uporabo orodja, mora imeti uporabnik določeno predznanje:

- znati mora uporabljati računalnik,
- poznati mora osnove metodologije Scrum,
- zavedati se mora različnih vlog, kot so skrbnik procesa, skrbnik izdelka, vodja podjetja in član ekipe (na primer programer),
- uporabnik mora imeti dostop do spleta ter moderni brskalnik, ki temelji na novejših tehnologijah, kot je HTML 5. Orodje se lahko uporablja preko računalnika, tablice ali mobilnega telefona.

Poglavje 4 Razvoj aplikacije za vodenje projektov po Scrumu

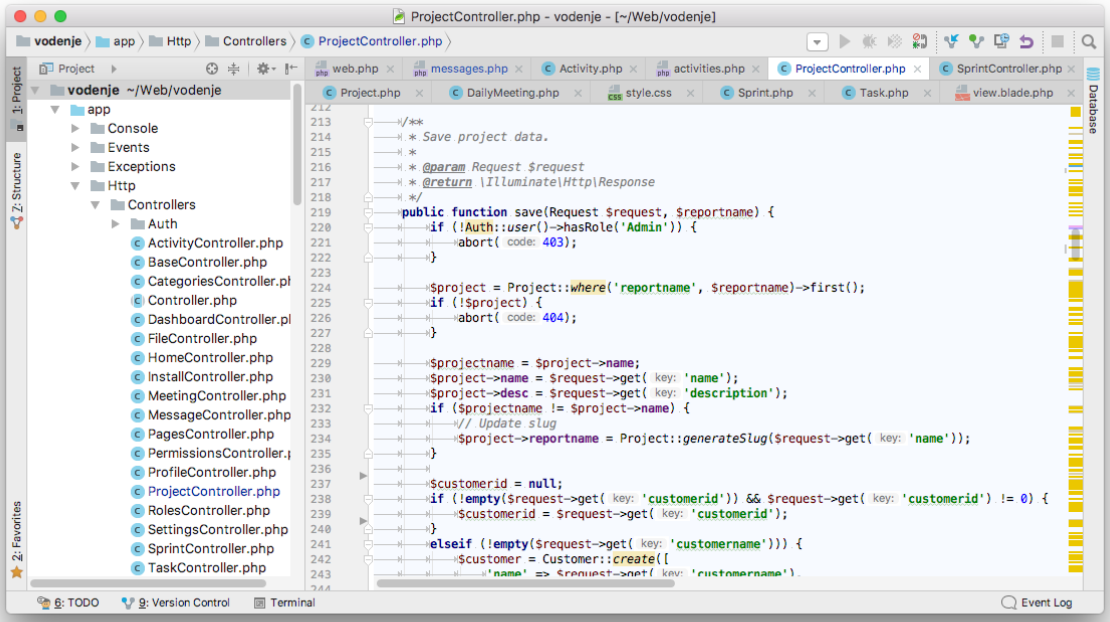
Pri razvoju lastnega orodja za vodenje projektov smo uporabili metodologijo Scrum. Najprej smo načrtovali podatkovno bazo v MySQL s pomočjo odprtokodnega orodja phpMyAdmin. Dokler nismo imeli dodelanega lastnega orodja, da bi lahko preko njega vnašali uporabniške zgodbe in naloge, smo podatke vnesli ročno preko orodja phpMyAdmin.

Za ogrodje spletne strani smo uporabili ogrodje Laravel [9], ki temelji na programskem jeziku PHP. Za to ogrodje smo se odločili, ker ga že več let uporabljamo v našem podjetju. Laravel je trenutno tudi eno najbolj popularnih ogrodij za izdelavo spletnih strani v programskem jeziku PHP [1]. Zanj je na voljo obilo odprtokodnih vtičnikov (angl. *plugins*) za razširjanje funkcionalnosti, kot na primer modul za ustvarjanje dokumentov PDF, uvoz podatkov iz preglednic Excel, ali obdelovanje slik.

Uporabili smo tudi nekaj odprtokodnih knjižnic napisanih v JavaScript programskem jeziku. Od tega je nekaj večjih:

- jQuery [17], za moderno upravljanje s strukturo HTML DOM,
- Bootstrap [8], za odzivni grafični vmesnik, da se prikaz spletne strani enostavno prilagaja različnim velikostim zaslonov,
- Dragula [18], za funkcionalnost vleči-in-spusti na Scrum tabli,
- jQuery File Upload [19], za nalaganje datotek in menjave profilne slike

Programsko kodo smo napisali z integriranim razvojnim okoljem PhpStorm (IDE), ki je za študente zastonj za čas študija. Slika 4.1 prikazuje primer grafičnega vmesnika IDE. Orodje je prilagojeno prav za programski jezik PHP in ima dobro podporo za ogrodje Laravel in njegovo orodje za delo s predlogami (angl. *templating engine*) Blade. Sistem Blade nam omogoča ločitev poslovne logike od definicije uporabniškega vmesnika. Zelo uporabno je tudi samodejno zaključevanje besed (angl. *autocomplete*), ki nam olajša delo in zmanjšuje možnosti za tipkarske napake. Zaključevanje besed deluje tako dobro, da najde imena razredov v celotni strukturi projekta in samodejno doda kodo za sklicevanje na razred.



Slika 4.1. Grafični vmesnik PhpStorm IDE

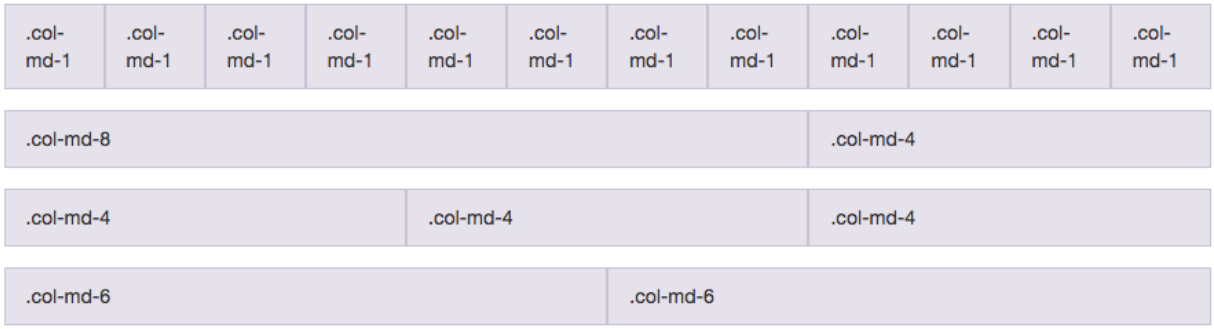
Za varnostno arhiviranje programske kode in beleženje zgodovine sprememb smo uporabili tehnologijo Git v povezavi s storitvijo GitHub. Git nam je omogočil, da smo lahko hranili zgodovino sprememb datotek, z GitHub spletno storitvijo, na katero lahko naložimo celotno zgodovino zapisano v Git skladišču (angl. *repository*), pa smo naložili varnostno kopijo na oddaljen sistem. Za lažje upravljanje z Git skladiščem smo se poslužili orodja Git-Tower.

Delo smo razčlenili v več sprintov, za vsak sprint pa smo si določili določen obseg števila točk s hitrostjo. V sprint smo nato dodali toliko uporabniških zgodb, da njihova vsota točk ni presegla hitrosti sprinta.

4.1 Opis rešitve

Pripraviti moramo orodje, kamor bodo uporabniki lahko vpisovali podatke o projektu, sprintih, dnevnih sestankih, uporabniških zgodbah in nalogah. Odločili smo se za izdelavo orodja v obliki spletne strani, saj je razvoj enega sistema časovno in cenovno ugodnejši, kot razvoj za vsako platformo posebej (npr. Windows, macOS, Linux, iOS, Android, itd.). Dandanes imajo že skoraj vsi uporabniki mobilne telefone z naročniškimi paketi, ki vsebujejo zadostno količino prenosa podatkov, zato uporaba sistema, ki je na voljo zgolj preko interneta ne predstavlja večjih ovir. Zaradi podpore različnih naprav mora biti stran prilagodljiva (angl.

responsive design), da se bo funkcionalno pravilno prikazovala na računalnikih, tablicah in mobilnih telefonih. V ta namen smo uporabili tehnologijo Bootstrap, ki vključuje osnovno temo CSS z navodili, kako naj se prikaže stran na določenih dimenzijah zaslona. Bootstrap uporablja tehniko, ki se imenuje mrežni sistem (angl. *Grid System*). Širino strani razdeli na največ 12 enako širokih stolpcev. Stolpce lahko združujemo skupaj, tako da dobimo manjše število stolpcev. Tako lahko stran razdelimo na več smiselnih ločenih enot. V primeru, da se stran prikaže na manjšem zaslonu, pa se lahko dva stolpca ločita in prikažeta eden nad drugim. Primer razdelitve stolpcev je nakazan na sliki 4.1.



Slika 4.2. Razdelitev vsebine strani s predlogo Bootstrap

4.2 Načrtovanje

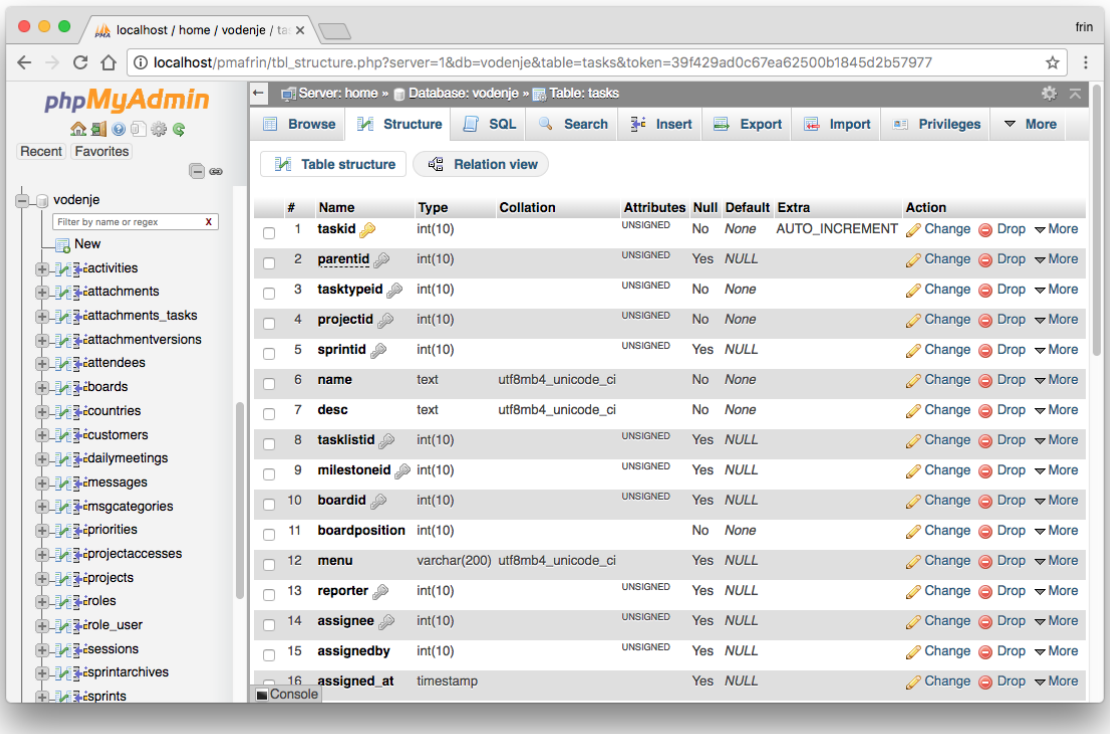
Programsko logiko smo načrtovali z ogrodjem Laravel [9], ki temelji na programskem jeziku PHP. Laravel omogoča hitro gradnjo aplikacij na solidnih temeljih. Navigacija je zasnovana po mehanizmu usmerjevalnika (angl. *router*). Programska logika je razdeljena v več različnih sklopov, imenovanih krmnilniki (angl. *controllers*). V datoteki routes/web.php zapišemo seznam pravil, ki določajo, katera povezava URL se nanaša na kateri krmilnik ter katero funkcijo znotraj tega krmilnika. Laravel deluje popolnoma na objektno orientiranem programiranju (angl. *OOP – Object Oriented Programming*). Za ločevanje segmentov kode med seboj uporablja razrede. Pri tem si dodatno pomaga še s PHP tehnologijo imenskega prostora (angl. *namespace*). Imenski prostor zagotavlja, da se lahko v programski kodi nahajata dva razreda z enakim imenom (vendar ne v istem imenskem prostoru), pri tem pa ne pride do kolizije imen.

Krmnilnik deluje v povezavi z modeli (entitetni tipi) in pogledi (angl. *views*). Ta koncept se imenuje model-pogled-krmilnik (angl. *MVC – Model-view-controller*) in ločuje posamezne sklope kode med seboj. Model vsebuje programsko logiko, ki komunicira s podatkovno bazo. Od krmilnika prejema novosti in spremembe, ki jih mora zapisati v podatkovno bazo, ter vrača podatke iz podatkovne baze. Krmilnik nato posreduje prejete podatke naprej v pogled.

Naloga pogleda je uporabniku predstaviti podatke v grafični obliki. Pri spletni strani je to grafični vmesnik spletne strani v HTML in dodatno oblikovan s CSS.

Naše orodje smo načrtovali z uporabo vseh zgoraj omenjenih pojmov. Za celotno orodje smo definirali lasten imenski prostor (Vodenje) ter sklop krmilnikov, kjer vsak opravlja funkcije posameznega dela orodja. Vsak pogled je definiran kot ločena PHP datoteka. Pogledi so zgrajeni z orodjem za delo s predlogami Blade. Pogledi so tako pripravljeni po tehniki MVC, saj programska logika ni zapisana v pogledu, temveč je zgolj opisana z osnovnimi programskimi koncepti kot so pogoj (if) in zanka (foreach).

Podatkovno bazo smo načrtovali s pomočjo orodja phpMyAdmin. Njegov enostaven in pregleden uporabniški vmesnik prikazuje slika 4.2. Orodje nam omogoča hitro zasnovo podatkovne baze. Izdelava dinamičnih povezav med entitetnimi tipi nam je omogočila verižno brisanje zapisov (angl. *cascade delete*). Primer takega verižnega izbrisa je samodejna odstranitev seznama udeležencev dnevnega sestanka ob izbrisu dnevnega sestanka. Seznam udeležencev brez dnevnega sestanka namreč nima nobenega pomena.



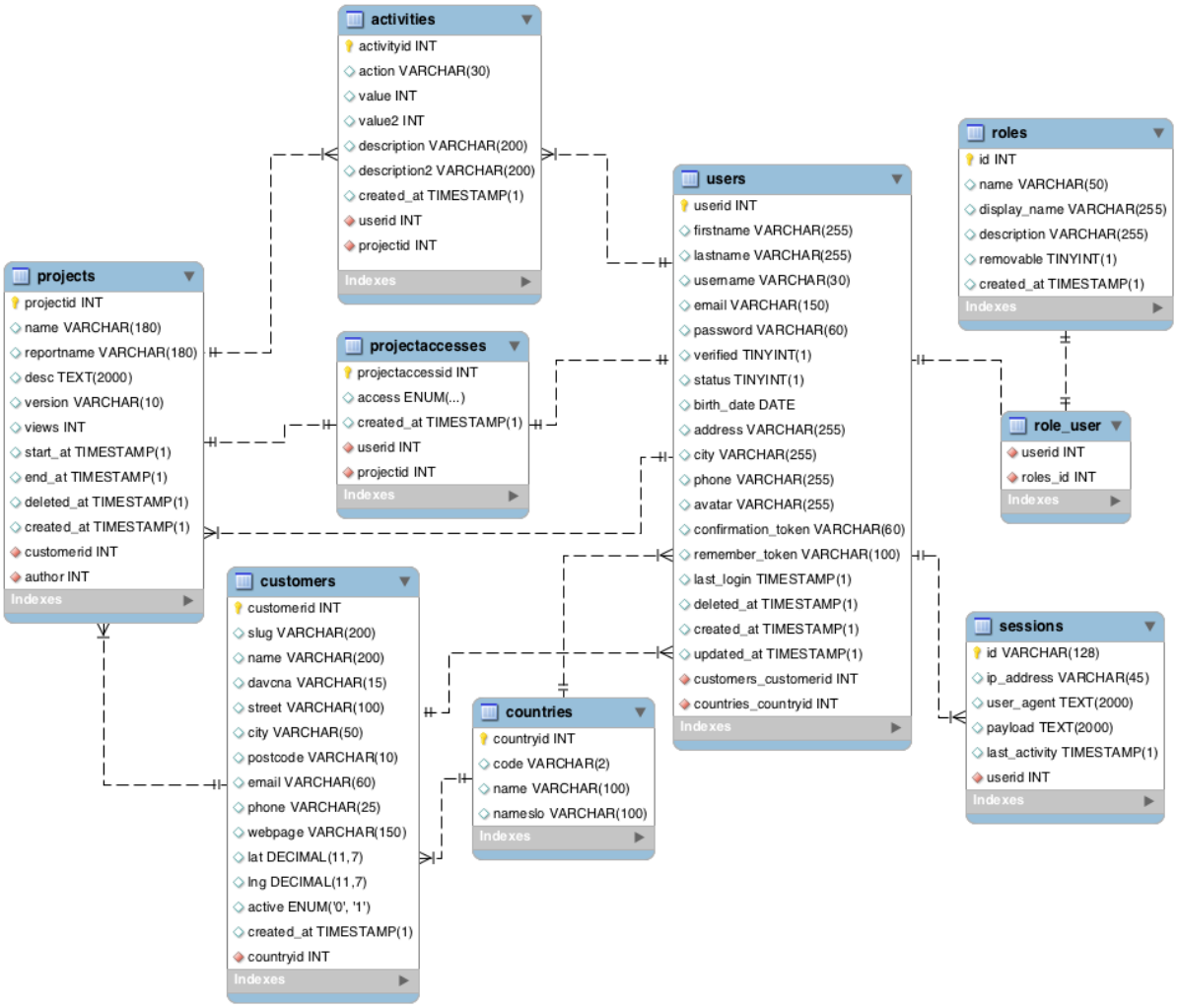
Slika 4.3. Grafični vmesnik orodja phpMyAdmin

Shemo podatkovne baze (Slika 4.4, Slika 4.5 in Slika 4.6) smo izrisali z orodjem MySQL Workbench. Orodje sicer omogoča tudi izvoz strukture podatkovne baze, vendar smo jo zaradi različnih podatkovnih tipov in poimenovanj raje pripravili ročno.

4.2.1 Struktura podatkovne baze

Zaradi obsežnosti strukture podatkovne baze smo razdelili prikaz strukture na več vsebinskih sklopov. Celotna struktura vsebuje 24 entitetnih tipov, od tega 4 služijo povezovanju.

Slika 4.4 prikazuje entitetne tipe in povezave med njimi, ki se uporabljajo pri preverjanju dostopa v sistem, kateri uporabnik ima pravico dostopanja do projekta in beleženje njihove aktivnosti.



Slika 4.4. Logični model podatkovne baze – dostop in pravice

Tabela **countries** (države) vsebuje šifrant držav.

Tabela **projects** vsebuje šifrant projektov. V njej so zapisani splošni podatki o samem projektu (naziv, kratek opis, začetek izvajanja projekta in rok dokončanja). Posamezen projekt je dodeljen samo eni stranki.

Tabela **users** vsebuje seznam uporabnikov sistema. V njej so zapisani osnovni podatki o uporabniku (naslov, kraj, telefon) ter podatki za preverjanje prijave (uporabniško ime ali elektronski naslov ter geslo).

Sistemski administrator je določen s tabelama **roles** (vloge) in **role_user** (povezovalna tabela med vlogami in uporabniki). Tabela **roles** vsebuje šifrant vlog, kot je npr. vodja projekta (administrator). V tabelo **role_user** shranimo povezavo med vlogo in uporabnikom, in s tem določimo, kakšen sistemski dostop ima uporabnik.

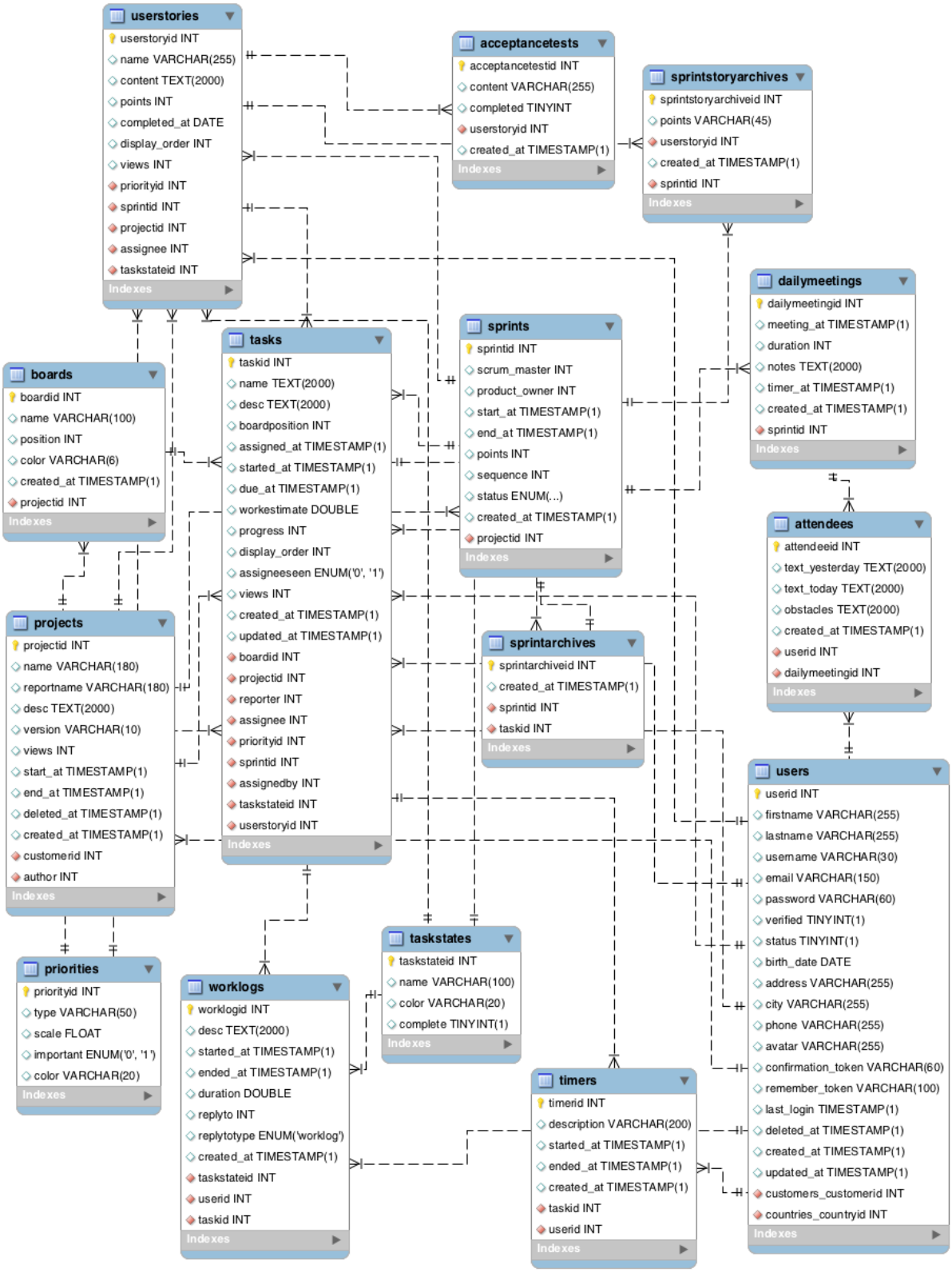
Da lahko določimo pravice uporabnikom samo na enem projektu, smo pravice razširili še na dodatno povezovalno tabelo **projectaccesses** (dostop na projektu), ki služi določanju dostopa uporabnika na posameznem projektu. S poljem *access* (dostop) določimo kakšne pravice ima uporabnik na projektu. Možne vrednosti so: branje, komentiranje, poročanje, spreminjanje, ustvarjanje, administrator projekta. Vsak nadaljnji dostop ima zajete pravice vseh prejšnjih. Dostop »poročanje« je namenjen za naročnika. Omogoča mu, da ustvari na projektu novo nalogo, ki se lahko upošteva v kasnejših sprintih. Pravice na projektu lahko določata zgolj sistemski administrator in administrator projekta. Na projektu je lahko naenkrat več administratorjev projekta.

Tabela **customers** vsebuje splošne informacije o naročniku za lažje komuniciranje. Posamezen projekt je vezan na naročnika, zato je šifra naročnika prisotna v tabeli projektov (projects).

Tabela **activities** (aktivnosti) hrani zgodovino aktivnosti uporabnikov na projektu. Nekaj od aktivnosti, ki se beležijo: ustvarjen nov projekt, nova naloga, zaključek naloge, izbris naloge, dodelitev naloge drugemu uporabniku, zabeleženo delo na nalogi, ustvarjen nov sprint, dodana naloga na sprint, ustvarjen nov dnevni sestanek, dodan udeleženec na dnevnem sestanku, itd.

V tabeli **sessions** (seje) hranimo seznam uporabniških sej, ki so veljavne za omejeno časovno obdobje. Čas je odvisen od izbora možnosti »zapomni si me« ob prijavi. Če uporabnik ne izbere možnosti, bo seja veljavna zgolj dokler je odprt brskalnik.

Drug sklop entitetnih tipov zajema jedro sistema in je prikazan na sliki 4.4. To so projekti, naloge, zabeleženo opravljeno delo, sprinti, dnevni sestanki in Scrum tabla.



Slika 4.5. Logični model podat. baze – projekti, zgodbe, naloge, sprinti in dnevni sestanki

Tabela **boards** zajema šifrant stolpcev v Scrum tabli. En stolpec pomeni npr. »v izdelavi« ali »zaključeno«. Vsakemu stolpcu lahko določimo lastno barvo naslovne vrstice za lažjo prepoznavnost.

Tabela **userstories** hrani seznam uporabniških zgodb. Vsebuje polja *name* (kratek naslov zgodbe), *content* (stavek zgodbe), *sprintid* (na katerem sprintu se trenutno nahaja), *points* (točke zgodbe za uporabo na sprintu), *display_order* (zaporedje prikaza na sprintu in seznamu zahtev), *assignee* (zadolžen za zgodbo).

Tabela **sprintstoryarchives** služi za arhiviranje podatkov, zato da imamo zgodovinsko informacijo, na katerih zgodbah se je opravljalo delo v določenih sprintih. Če zgodba ni bila dokončana v enem sprintu, se jo lahko prenese v nov sprint. V primeru, da ne bi imeli arhiva povezav med zgodbami in sprinti, bi se zgubila informacija, ko se zgodba dodeli novemu sprintu.

Tabela **acceptancetests** vsebuje seznam sprejemnih testov na uporabniški zgodbi. Vsebuje polja *content* (vsebina testa) in *completed* (ali je bil test že upoštevan).

V tabeli **tasks** shranjujemo seznam nalog. Tabela vsebuje polja *boardposition* (zaporedje prikaza v stolpcu Scrum table), *assigned_at* (datum določitve opravljalca), *assignee* (kdo bo opravljal nalogo), *assignedby* (kdo je nalogo dodelil), *reporter* (kdo je ustvaril nalogo), *started_at* (datum začetka naloge), *due_at* (rok zaključka naloge), *workestimate* (predviden čas dela za primerjavo z vnešenim časom; uporabnik lahko vnese čas v urah in minutah, zabeleži pa se samo v minutah zaradi preprostejšega preračunavanja), *progress* (ročno vnešena ocena, koliko dela je bilo na nalogi že opravljenega glede na celoto), *display_order* (zaporedje prikaza na tekstovnih seznamih), *boardid* (v katerem stolpcu v Scrum tabli se nahaja, vsebuje prazno vrednost, če ni v nobenem stolpcu), *priorityid* (šifra prioritete naloge), *taskstateid* (šifra opravljenosti naloge, npr. če je naloga zaključena).

V tabeli **sprints** vodimo šifrant sprintov. Tu določimo, kdo je skrbnik procesa in skrbnik izdelka. Shranimo datum začetka in konec sprinta (*start_at, end_at*). Število točk na sprintu določimo s poljem *points*. Polje *sequence* vsebuje zaporedno številko sprinta, *status* pa trenutno stanje sprinta (aktiven ali zaključen).

Tabela **dailymeeetings** vsebuje seznam dnevnih sestankov sprinta. V polju *meeting_at* je zapisan datum sestanka, *notes* služi za dodatno beležko, kamor lahko skrbnik procesa zabeleži opombe s sestanka, *duration* (čas) pa vsebuje število minut izvajanja sestanka. V primeru, da je skrbnik procesa pognal števec, se v polje *timer_at* zapiše datum začetka števca, ob koncu pa se preračuna časovna razlika med časom ustavitve in začetka, ter se prišteje število

zaokroženo na cele minute k polju *duration*. Formula je: duration + zaokrožene_minute(trenutni čas - čas začetka timer_at). Formula deluje tudi v primeru, da skrbnik procesa zapre in ponovno odpre brskalnik.

Tabela **sprintarchives** služi za arhiviranje podatkov, zato da imamo zgodovinsko informacijo, na katerih nalogah se je opravljalo delo v določenih sprintih. Če naloga ni bila dokončana v enem sprintu, se jo lahko prenese v nov sprint. V primeru, da ne bi imeli arhiva povezav med nalogami in sprinti, bi se zgubila informacija, ko se naloga dodeli novemu sprintu.

Tabela **attendees** zajema seznam prisotnih članov ekip na dnevnem sestanku. V polje *text_yesterday* se shrani uporabnikov odgovor na vprašanje »Kaj ste naredili od zadnjega sestanka«, v polje *text_today* shranimo odgovor na vprašanje »Kaj boste storili do prihodnjega sestanka« in v polje *obstacles* zapišemo ovire, na katere je naletel uporabnik.

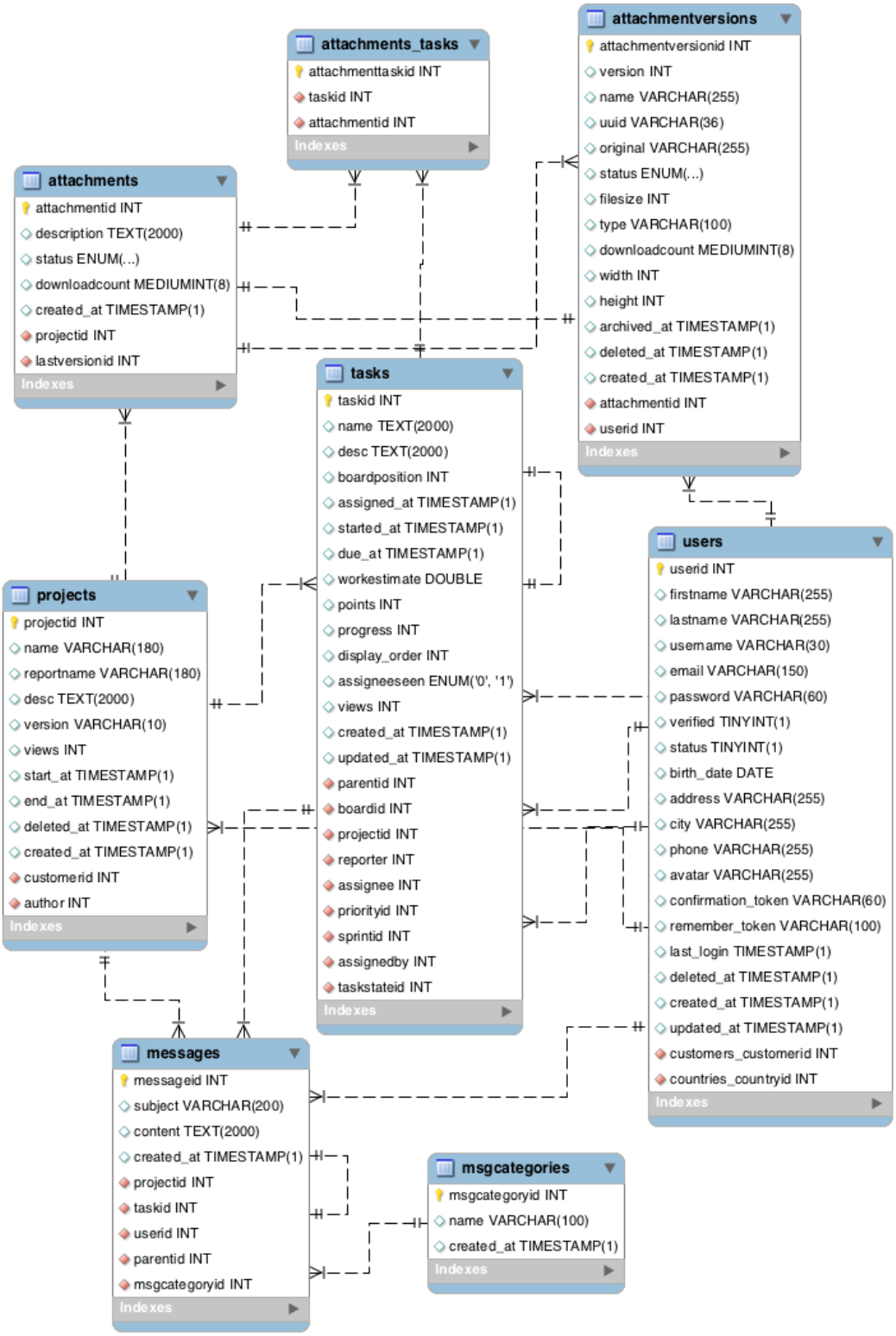
V tabeli **priorities** hranimo fiksen šifrant prioritet. Polje *scale* (lestvica) vsebuje zaporedno številko moči prioritete, s poljem *important* (pomembno) pa ločimo manj pomembne prioritete od bolj pomembnih. To se uporablja pri označevanju pomembnejših nalog z značko.

Tabela **worklogs** vsebuje zabeleženo delo člana ekipe na posamezni nalogi. V polje *desc* zabeležimo kratek opis dela, s polji *started_at* (začetek) in *ended_at* (konec) pa obdobje opravljanja dela. Zaradi lažjega preračunavanja porabljenega časa čez vse naloge shranimo razliko med začetkom in koncem še kot število minut v polje *duration* (čas). S posameznim delom lahko spremenimo tudi stanje naloge (zaključeno), to kot informativno označimo s poljem *taskstateid* (šifra stanja naloge). Delo se lahko navezuje na drug zapis dela v isti nalogi, kar označimo s poljem *replyto* (odgovor na delo). Seznam del na nalogi se v tem primeru prikaže gnezdeno.

Tabela **taskstates** vsebuje šifrant stanj nalog. Trenutno se uporablja zgolj za označevanje ali je naloga dokončana ali nedokončana.

Na koncu imamo še tabelo **timers**, ki hrani seznam trenutno pognanih števcev izvajanja dela. Ko uporabnik požene števec na nalogi, se mu prikaže v levem spodnjem kotu oznaka, koliko časa se delo na nalogi izvaja. Če te tabele ne bi imeli, bi se števec ponastavil, ko zapremo okno brskalnika ali gremo na drugo napravo. Ko uporabnik konča z delom oz želi zabeležiti interval dela na nalogi, enostavno zaključi števec, vnese opis narejenega dela in potrdi. Števec se izbriše, v tabelo worklogs pa se zabeleži opravljeno delo s samodejno preračunanim časom dela.

Tretji sklop vsebuje entitetne tipe, prikazane na sliki 4.5. Ti entitetni tipi služijo preostali funkcionalnosti, kot so priponke in sistem za izmenjavo sporočil.



Slika 4.6. Logični model podatkovne baze – priponke in sporočila

Tabela **attachments** (priponke) vsebuje seznam priponk na projektih ali nalogah. Za priponke, ki so vezane samo na projekt in ne na konkretno nalogo, je dovolj polje *projectid* (šifra projekta). Če pa želimo priponko vezati še na vsaj eno nalogo, potrebujemo povezovalno tabelo **attachments_tasks**. S pomočjo te pomožne tabele shranimo informacijo na katere naloge se priponka navezuje. Priponka, ki je vezana samo na celoten projekt, ne bo imela zapisa v tej povezovalni tabeli.

Na zapisu posamezne priponke lahko zamenjamo samo vsebino, tako da naložimo novo priponko čez obstoječo. V ta namen uporabimo tabelo **attachmentversions** (različice priponk), kamor zapišemo originalne lastnosti priponke (originalno ime priponke, velikost v zlogih, vrsta datoteke, ter v primeru da je vsebina priponke slika, se shrani še njena širina in višina). Zapis se ustvari tudi v primeru, če naložimo samo prvo različico priponke. V tabeli attachments nam polje *lastversionid* (šifra zadnje različice) služi kot kazalec na zadnjo različico.

V tabeli **msgcategories** beležimo seznam kategorij sporočil. Kategorije uporabljamo za boljšo organiziranost.

Zadnja tabela je **messages**. Ta vsebuje seznam sporočil. V polje *subject* (zadeva) shranimo naslov sporočila, v polje *content* (vsebina) pa obogateno daljše besedilo samega sporočila. Da lahko naredimo funkcionalnost gnezdenih sporočil, smo vključili še polje *parentid*, v katerega zabeležimo šifro sporočila, na katerega se nanaša. S poljem *msgcategoryid* razporedimo sporočilo v eno od kategorij.

4.3 Izvedba

Orodje je zasnovano v obliki spletne strani in je dostopno preko interneta. Zaradi omejevanja dostopa nepooblaščenim osebam do tajnih podatkov mora imeti sistem ob vstopu preverjanje identitete. Ko uporabnik odpre vstopno stran, se mu prikaže obrazec za prijavo. Prijavi se z elektronskim naslovom in geslom. Ponavadi se lahko na spletnih portalih namesto elektronskega naslova uporablja uporabniško ime, zato smo omogočili tudi prijavo z uporabniškim imenom. V primeru, da je uporabnik pozabil svoje geslo, ga lahko ponastavi s funkcionalnostjo »pozabljeno geslo«. Uporabnik vnese svoj elektronski naslov, na katerega dobi navodila za ponastavitev gesla. Logika za sistem prijave in pozabljenega gesla se nahaja v razredih **Auth\AuthController.php** in **Auth\PasswordController.php.**

4.3.1 Osnova aplikacije

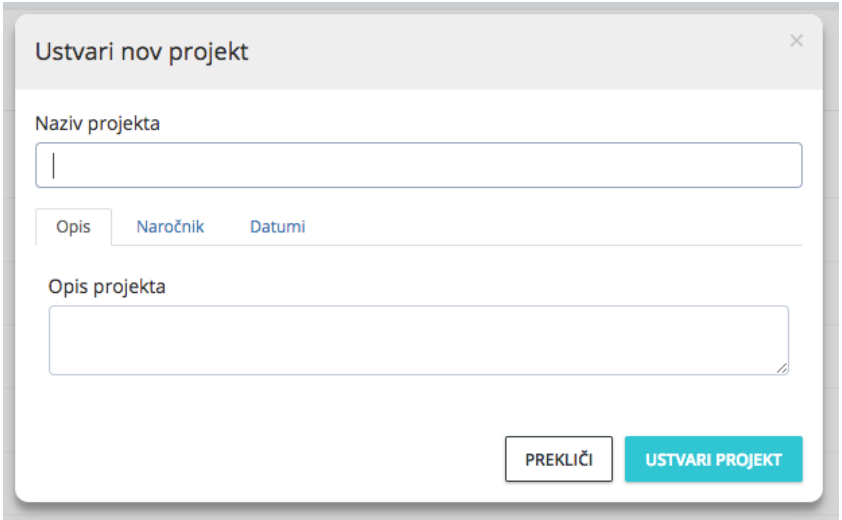
Razred **ProjectController.php** vsebuje programsko logiko, ki se navezuje na izvajanje akcij nad projekti:

- **__construct**: konstruktor razreda zajema preverjanje dostopa do funkcionalnosti celotnega razreda; če uporabnik ni prijavljen, se izvajanje ustavi, uporabnika pa preusmeri na prijavni obrazec.
- **projects**: sistem preveri uporabnikove pravice in izpiše samo seznam projektov, do katerih ima dostop.
- **overview**(šifra projekta): izpiše zaslon s pregledom nad projektom.
- **create**(obrazec): ustvari nov projekt na podlagi posredovanih podatkov z obrazca. Če je določena tudi stranka, ustvari še stranko.
- **remove**(šifra projekta): v podatkovni bazi označi projekt kot izbrisan.
- **edit**(šifra projekta): vrne obrazec za urejanje obstoječega projekta.
- **save**(obrazec, šifra projekta): shrani spremembe podatkov o projektu.
- **tasks**(šifra projekta): izpiše zaslon s seznamom nalog na projektu.
- **userstories**(šifra projekta): izpiše zaslon s seznamom uporabniških zgodb na projektu.
- **users**(šifra projekta): izpiše seznam uporabnikov, ki imajo dostop do projekta.
- **addUser**(obrazec, šifra projekta): obstoječemu uporabniku doda pravice za dostop na izbran projekt.
- **inviteUser**(obrazec, šifra projekta): ustvari novega uporabnika z nepotrjenim stanjem, pošlje povabilo na njegov elektronski naslov in mu doda pravice za dostop na izbran projekt.
- **removeUser**(šifra projekta, šifra uporabnika): odstrani uporabnika s projekta, uporabniške zgodbe in naloge dodeljene uporabniku pa označi kot nedodeljene.

Aplikacija je zasnovana na tak način, da čim manj moti uporabnika s ponovnim nalaganjem celotne strani. To dosežemo z uporabo tehnologije AJAX. AJAX nam omogoča, da preko strežnika naložimo in zamenjamo samo določen del spletne strani. Spletni brskalniki imajo drugačne načine za asinhrono povezovanje na strežnik. Pri tem smo si pomagali s knjižnico jQuery, ki vsebuje ovoj okoli različnih implementacij, in nam s tem bistveno olajša delo.

AJAX smo uporabili na mestih, kjer od uporabnika pričakujemo vnos ali spremembo podatkov. Na sliki 4.6 je prikazan primer dialoga za vnos podatkov o novem projektu. S klikom na gumb »Nov projekt« se preko klica AJAX naloži vsebina dialoga. Bistvena prednost prikaza lebdečih dialogov pred zamenjavo celotne strani je v tem, da lahko

uporabnik primerja podatke z obstoječe strani, lahko pa tudi prekliče akcijo in ostane na istem mestu kot pred začetkom akcije.

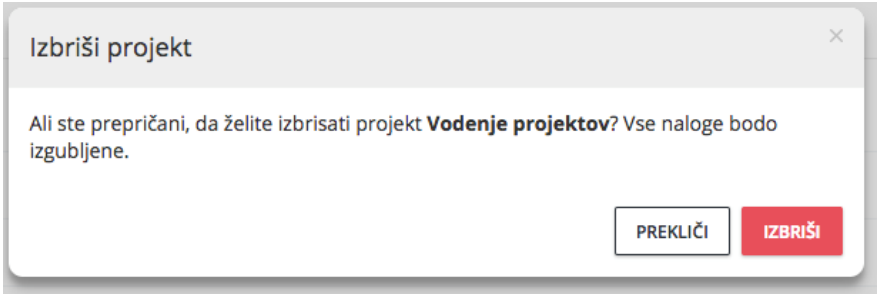


Slika 4.7. Dialog za ustvarjanje novega projekta

Preko AJAX pa lahko dobimo povratno informacijo s strežnika, na katero se odzovemo in spremenimo izgled vsebine strani na uporabniški strani. To tehniko smo uporabili na primer pri izbrisu projekta. Namesto da bi ponovno naložili celotno stran, se s seznama projektov odstrani samo vrstica s pravkar izbrisanim projektom.

Vse spremembe preko AJAX klicev moramo izvajati preko HTTP metode POST (pošlji). Če bi izvajali klice preko metode GET (pridobi), bi lahko pri tem prišlo do varnostnih zlorab s ponarejanjem spletnih zahtev (CSRF, angl. *Cross-Site Request Forgery)*. Pri tem druga spletna stran v ozadju sproži zahtevek GET AJAX na naš strežnik, ne da bi uporabnik za to vedel. Ker je uporabnik avtenticiran na naši strani, se GET zahtevek izvede, pri tem pa bi lahko pobrisal kritične podatke. Podobno bi lahko izvedel napad tudi s POST, tako da bi poneveril obrazec na drugem strežniku, ki kaže na naš strežnik. Zaradi tega moramo v vse zahtevke POST vstaviti še žeton za enkratno uporabo, katerega pa nobena druga stran ne pozna. Ob prejemu zahtevka POST se preveri pravilnost žetona in se akcija izvede samo v primeru, da se žeton ujema.

Za vse kritične akcije se pred proženjem prikaže uporabniku še dialog za potrditev, s katerim mora potrditi, da se strinja z njihovo izvedbo (slika 4.7). Pri tem se izognemo težavam, ki bi nastale zaradi nenamernih klikov.



Slika 4.8. Dialog za potrditev kritične akcije

Funkcionalnosti, ki vplivajo na podatke o uporabnikih, smo zapisali v razredu **UserController.php**:

- **__construct**: podobno kot pri razredu ProjectController zajema konstruktor razreda preverjanje dostopa do funkcionalnosti celotnega razreda; če uporabnik ni prijavljen, se izvajanje ustavi, uporabnika pa preusmeri na prijavni obrazec.
- **users**: izpiše seznam vseh uporabnikov v sistemu, zaradi tega je izpis na voljo samo sistemskim administratorjem.
- **profile**(šifra uporabnika): prikaže zaslon s podrobnostmi o uporabniku.
- **projects**(šifra uporabnika): izpiše seznam projektov, do katerih ima izbran uporabnik dostop.
- **create**(obrazec): ustvari novega uporabnika in ga dodeli na izbran projekt in stranko.
- **remove**(šifra uporabnika): označi uporabnika kot izbrisanega in mu onemogoči dostop, uporabnik pa se ne prikazuje več na seznamu uporabnikov.
- **edit**(šifra uporabnika): prikaže zaslon z obrazcem za urejanje uporabnika.
- **editPopup**(šifra uporabnika): prikaže vmesnik za dialog, v katerem lahko uredimo uporabnika.
- **save**(podatki, šifra uporabnika): shrani spremembe podatkov za uporabnika.
- **uploadAvatar**(obrazec s sliko): shrani novo profilno sliko uporabnika.

Programska logika za upravljanje s sporočili se obdeluje s krmilnikom **MessageController.php**:

- **__construct**: tudi v tem razredu konstruktor skrbi za dostop do funkcij.
- **create**(obrazec, šifra projekta): ustvari novo sporočilo na določenem projektu.
- **reply**(šifra sporočila, obrazec): shrani novo sporočilo, ki je odgovor na obstoječe sporočilo.

- **saveChanges**(šifra sporočila, obrazec): posodobi zadevo in vsebino obstoječega sporočila.
- **remove**(šifra sporočila): izbriše sporočilo iz podatkovne baze. Če je sporočilo imelo kak odgovor, označi prvega kot glavno sporočilo in nastavi ostala sporočila v hierarhiji kot odgovore na prvo sporočilo.
- **listProject**(šifra projekta, šifra kategorije sporočil): izpiše seznam vseh sporočil v določeni kategoriji sporočil.
- **view**(šifra sporočila, šifra kategorije sporočil): prikaže podrobnosti o sporočilu in možne odgovore.

Na spletu lahko vnos obogatenega besedila predstavlja stranem veliko varnostno luknjo. Obogateno besedilo se v največ primerih oblikuje z oznakami HTML (angl. *tags*), ki se shranijo v podatkovni bazi. Tako besedilo se kasneje prikaže kot obogaten izpis pri npr. ogledu sporočila. Nikakor ne smemo pustiti vnosa vseh oznak HTML, saj bi lahko z oznako »script« nepridipravu omogočili vnos JavaScript kode. JavaScript koda bi se nato izvedla v brskalniku s pravicami prijavljenega uporabnika, ki si ogleduje sporočilo. Ta vrsta napada se imenuje shranjen napad XSS. Ko je bil načrtovan standard HTML, varnost ni bila prioriteta. Zaradi tega je nešteto možnih oblik napadov XSS. Vse vrste napadov je zelo težko zajeziti, zato smo se poslužili odprtokodne rešitve. Laravel že sam po sebi sicer omogoča čiščenje besedila, vendar bi v primeru čiščenja obogatenega besedila dobili izpisane znake HTML. Za namen čiščenja obogatenega besedila smo zato posegli po modulu Purifier [10]. Z enim enostavnim ukazom »clean« (počisti) lahko pretvorimo zlonamerno kodo v neškodljivo besedilno sporočilo.

Nalaganje in prenos datotek je zajeto v krmilniku **FileController.php**. Pripenjanje datotek smo omogočili zato, da lahko uporabniki hranijo material in dokumentacijo projekta skupaj z nalogami:

- **__construct**: konstruktor skrbi za dostop do funkcij znotraj krmilnika.
- **thumbnail**(šifra datoteke): izvede prenos/prikaz pomanjšane slike priponke, v primeru da je vrsta priponke slika.
- **download**(šifra različice datoteke): izvede prenos datoteke k uporabniku.
- **taskUpload**(šifra naloge, podatki): shrani datoteko na strežnik in jo dodeli na nalogo; v primeru da je naložena slika, ustvari še pomanjšano sliko za predogled.
- **projectUpload**(šifra projekta, podatki): shrani datoteko na strežnik in jo dodeli na projekt; podobno kot pri taskUpload ustvari pomanjšano sliko za predogled.
- **projectFiles**(šifra projekta): izpiše seznam vseh priponk na projektu in prikaže obrazec za nalaganje priponk na projekt.

- **remove**(šifra datoeke): odstrani priponko s projekta.

Uporabnikom so bolj prijazni interaktivni vmesniki kot pa klikanje po hierarhijah map na disku. Datoteko lahko naložimo v sistem na način povleci-in-spusti, tako da uporabnik povleče datoteko z drugega okna v brskalnik in s tem sproži prenos. Uporabniku še zmeraj omogočamo nalaganje datotek na klasičen način s klikom na gumb, ki nam pokaže dialog za izbor datoteke.

4.3.2 Beleženje dnevnih sestankov

Z beleženjem dnevnih sestankov se ukvarja krmilnik **MeetingController.php**:

- **__construct**: konstruktor ščiti dostop do funkcij krmilnika pred neprijavljenimi uporabniki.
- **list**(šifra projekta): prikaže seznam dnevnih sestankov za projekt.
- **createAttendeeForm**(šifra sestanka): prikaže obrazec za dodajanje novega udeleženca dnevnega sestanka. Če so dodani že vsi možni člani, se prikaže opozorilo, da so vsi člani že prisotni na sestanku.
- **create**(šifra sprinta, obrazec): shrani nov dnevni sestanek v podatkovno bazo. Če za izbran datum že obstaja dnevni sestanek, sistem ne dovoli vnosa.
- **remove**(šifra sestanka): izbriše dnevni sestanek iz podatkovne baze. Zaradi povezave s tablo udeležencev se izvede tudi veriženo brisanje podatkov o udeležencih, ki so bili del dnevnega sestanka.
- **start**(šifra sestanka): požene števec izvajanja dnevnega sestanka, ki je v pomoč skrbniku procesa pri vodenju sestanka.
- **stop**(šifra sestanka): ustavi števec izvajanja dnevnega sestanka in posodobi polje s časom dnevnega sestanka z novim časom.
- **edit**(šifra sestanka): prikaže obrazec za urejanje osnovnih podatkov o dnevnem sestanku.
- **save**(šifra sestanka, obrazec): shrani spremembe osnovnih podatkov o dnevnem sestanku.
- **createAttendee**(šifra sestanka, obrazec): ustvari zapis v tabeli seznama udeležencev in s tem doda člana na dnevni sestanek.
- **editAttendee**(šifra udeleženca): prikaže obrazec za urejanje besedilnih polj za udeleženca.
- **saveAttendee**(šifra udeleženca, obrazec): shrani podatke za udeleženca na dnevnem sestanku.
- **removeAttendee**(šifra udeleženca): odstrani udeleženca z dnevnega sestanka.

Čas izvajanja sestanka lahko skrbnik procesa nastavi ročno z urejanjem podatkov o dnevnem sestanku, ali pa se dinamično posodablja s poganjanjem in ustavljanjem števca. Števec mora biti pognan vsaj 30 sekund, da se zabeleži sprememba, saj se čas zaokroži na najbližjo minuto.

4.3.3 Uporabniške zgodbe

V metodologiji Scrum pišemo opravila v obliki uporabniških zgodb. Z uporabniškimi zgodbami in sprejemnimi testi upravlja krmilnik **UserStoryController.php**:

- **__construct**: konstruktor omejuje dostop do funkcij v krmilniku za samo prijavljene uporabnike.
- **userstory**(šifra zgodbe): izpiše zaslon s podrobnostmi zgodbe.
- **update**(obrazec, šifra zgodbe): shrani spremembe podatkov na zgodbi.
- **create**(obrazec, šifra projekta): ustvari novo uporabniško zgodbo na projektu.
- **removeUserStory**(šifra zgodbe): odstrani zgodbo s projekta.
- **addTaskList**(šifra zgodbe): prikaže obrazec za dodajanje naloge na zgodbo.
- **saveTask**(obrazec, šifra zgodbe): na zgodbo dodeli nalogo.
- **removeTask**(šifra zgodbe, šifra naloge): z zgodbe odstrani nalogo.
- **createAcceptanceTest**(obrazec, šifra zgodbe): shrani nov sprejemni test na uporabniško zgodbo.
- **editAcceptanceTest**(šifra testa): prikaže obrazec za urejanje sprejemnega testa.
- **saveAcceptanceTest**(šifra testa, obrazec): shrani spremembe na obstoječem sprejemnem testu.
- **removeAcceptanceTest**(šifra testa): izbriše sprejemni test.

4.3.4 Naloge izdelka

Naloge predstavljajo manjšo enoto dela na projektu. Naloge lahko uporabljamo individualno, ali pa kot del uporabniških zgodb. Z nalogami upravlja krmilnik **TaskController.php**:

- **__construct**: konstruktor omejuje dostop do funkcij v krmilniku za samo prijavljene uporabnike.
- **task**(šifra naloge): izpiše zaslon s podrobnostmi o nalogi.
- **update**(obrazec, šifra naloge): shrani spremembe podatkov na nalogi v podatkovno bazo.
- **create**(obrazec, šifra projekta): ustvari novo nalogo na projektu.
- **removeTask**(šifra naloge): odstrani nalogo s projekta.

- **createWorkLog**(obrazec, šifra naloge): shrani zabeležen čas dela uporabnika na nalogi.
- **saveWorkLog**(obrazec, šifra dela): posodobi obstoječ zabeležen čas dela.
- **logTimer**(obrazec): ustvari zabeležen čas dela uporabnika na nalogi na podlagi pognanega števca, čas dela je shranjen v minutah zaokrožen navzgor.
- **deleteTimer**(šifra števca): izbriše obstoječ števec dela za uporabnika.
- **startTimer**(šifra naloge): zabeleži začetek števca dela za uporabnika, pri tem preveri, da v podatkovni bazi še ni števca zanj.
- **user**(šifra uporabnika): prikaže seznam nalog, ki so dodeljene uporabniku. Če izpis zahteva uporabnik, ki nima administrativnih pravic, izpiše samo naloge na projektih, kjer imata oba dostop.
- **editWorkLog**(šifra dela): prikaže obrazec za urejanje zabeleženega časa dela.
- **removeWorkLog**(šifra dela): odstrani zabeležen čas dela.
- **addUserStoryList**(šifra naloge): prikaže seznam možnih uporabniških zgodb, na katere lahko dodelimo nalogo.
- **saveUserStory**(šifra naloge, obrazec): dodeli nalogo na uporabniško zgodbo.
- **removeUserStory**(šifra naloge): odstrani nalogo z uporabniške zgodbe.

4.3.5 Naloge sprinta

Upravljanje sprintov se izvaja preko krmilnika **SprintController.php**. Krmilnik je preprost in ima predvsem krmili funkcionalnost ustavarjanja, urejanja in brisanja sprintov, ter urejanja seznama nalog na sprintu:

- **__construct**: kot pri drugih krmilnikih, tudi ta konstruktor skrbi za dostop do funkcij krmilnika samo prijavljenim.
- **list**(šifra projekta): izpiše seznam sprintov za določen projekt.
- **view**(šifra sprinta): prikaže podrobnosti za določen sprint, kar zajema seznam nalog sprinta in možnosti za dodajanje in odstranjevanje nalog.
- **create**(šifra projekta, obrazec): shrani podatke o novem sprintu v podatkovno bazo, pri tem pa preveri, da ne obstaja sprint v izvajanju, saj v sistemu ni dovoljeno imeti sočasno dveh sprintov v izvajanju. Če se je skrbnik procesa odločil za prenos nedokončanih nalog z zadnjega sprinta, se prenesejo te naloge na novo ustvarjen sprint.
- **remove**(šifra sprinta): izbriše izbran sprint.
- **edit**(šifra sprinta): prikaže obrazec za urejanje podatkov o sprintu.

- **save**(šifra sprinta, obrazec): shrani spremembe podatkov o sprintu v podatkovno bazo. Če je bil sprint zaključen, prestavi seznam nalog v arhiv sprinta, če pa je bil sprint ponovno odprt, povrne naloge iz arhiva.
- **addUserStoryList**(šifra sprinta): prikaže obrazec s seznamom nedokončanih zgodb, ki še niso del sprinta, tako da lahko dodamo zgodbe na sprint.
- **saveUserStory**(šifra sprinta, obrazec): doda zgodbo na sprint. V primeru, da je sprint že zaključen, doda zgodbo v arhiv sprinta.
- **removeUserStory**(šifra sprinta, šifra zgodbe): odstrani zgodbo s sprinta. Če je bila zgodba v arhivu sprinta, jo odstrani tudi od tam.
- **addTaskList**(šifra sprinta): prikaže obrazec s seznamom nedokončanih nalog, ki še niso del sprinta, tako da lahko dodamo naloge na sprint.
- **saveTask**(šifra sprinta, obrazec): doda nalogo na sprint. V primeru, da je sprint že zaključen, doda nalogo v arhiv sprinta.
- **removeTask**(šifra sprinta, šifra naloge): odstrani nalogo s sprinta. Če je bila naloga v arhivu sprinta, jo odstrani tudi od tam.
- **moveUserStory**(obrazec): zamenja vrstni red zgodbe na seznamu zgodb sprinta.
- **moveTask**(obrazec): zamenja vrstni red naloge na seznamu nalog sprinta.

4.3.6 Scrum tabla

Razred **BoardController.php** vsebuje programsko logiko, ki se navezuje na interakcijo s Scrum tablo:

- **__construct**: konstruktor skrbi za dostop do funkcij v krmnilniku.
- **board**(šifra projekta): izpiše Scrum tablo za izbran projekt. Če uporabljamo sprinte, se na tabli prikažejo naloge s trenutnega sprinta.
- **boardMoveTask**(podatki): na podlagi podatkov premakne nalogo znotraj enega stolpca Scrum table, ali pa celo med različnimi stolpci. Nalogo lahko premakne tudi v seznam nalog izven Scrum table.
- **boardRemove**(stolpec): odstrani stolpec s Scrum table, pri tem pa prestavi naloge s stolpca nazaj v seznam nalog.
- **boardSort**(tabla, polje): razvrsti naloge v izbranem stolpcu Scrum table po enem od možnih polj: imenu naloge, prioriteti, rokom izvedbe, datum ustvarjanja naloge.
- **boardEdit**(podatki): shrani spremembe za osnovne podatke stolpca na Scrum tabli.
- **boardNew**(šifra projekta, obrazec): ustvari nov stolpec v Scrum tabli na izbranem projektu. Novo ustvarjeni stolpec se postavi na desno stran obstoječih stolpcev.
- **boardMoveColumn**(podatki): prestavi pozicijo določenega stolpca v Scrum tabli.

4.3.7 Iskanje po projektu

Razred **SearchController.php** se uporablja za prikaz rezultatov iskanja po projektu. Iskanje izvaja po projektih, uporabniških zgodbah, nalogah, priponkah in sporočilih:

- **__construct**: konstruktor skrbi za dostop do funkcij v krmnilniku.
- **search**(obrazec): izpiše rezultate iskanja s povezavami do posameznih vsebin.

4.3.8 Diagrami preostalega in opravljenega dela

Največjo preglednost nad stanjem projekta pridobimo z diagrami preostalega dela. V okviru orodja smo razvili dve vrsti diagramov – diagram preostalega dela (angl. *burn-down diagram*) in diagram opravljenega dela (angl. *burn-up diagram)*. Diagram preostalega dela nam omogoča pregled nad količino preostalega dela. Diagram opravljenega dela je obrnjen ravno nasprotno od diagrama preosalega dela, prikazuje pa koliko dela je že bilo narejenega ter koliko je dela v celoti in kako se je količina dela spreminjala tekom časa.

4.3.8.1 Diagram preostalega dela

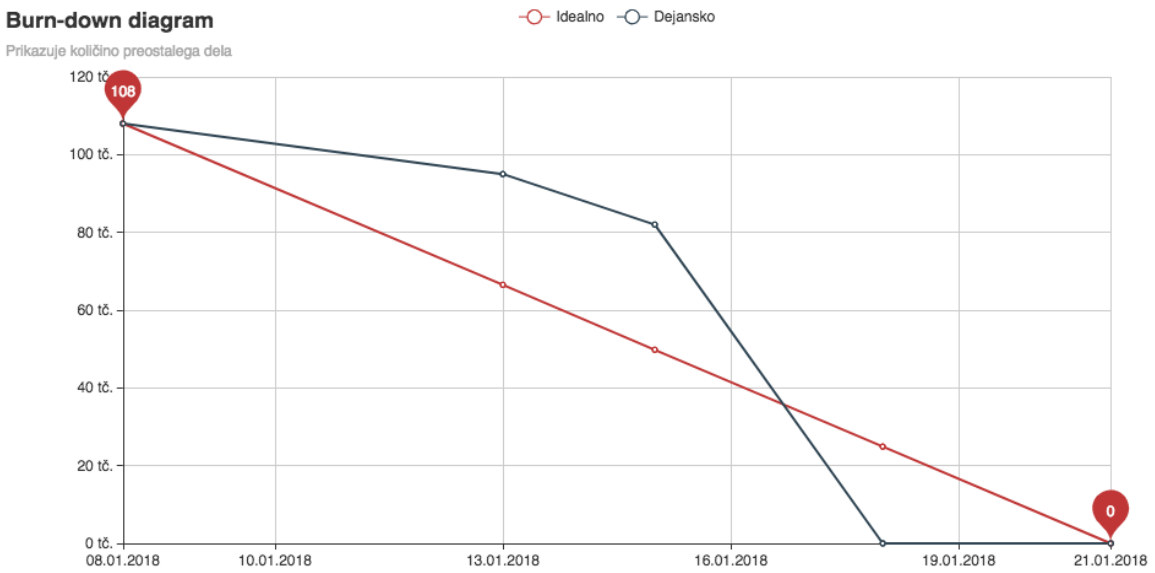
Pripravili smo diagram preostalega dela na sprintu. Za izgradnjo diagrama smo uporabili odprtokodno knjižnico ECharts [11].

Za ECharts smo se odločili, ker knjižnica omogoča gradnjo veliko različnih diagramov, ni odvisna od drugih knjižnic, je enostavna za vgradnjo, omogoča prenos slike diagrama in ima solidno dokumentacijo.

V krmnilniku **DiagramController.php** smo nato spisali funkcije, ki vračajo podatke za izris diagrama:

- **__construct**: konstuktor skrbi za dostop do funkcij znotraj krmilnika.
- **view**(šifra sprinta): pripravi podatke za izris diagramov, jih posreduje pogledu in izriše diagram.

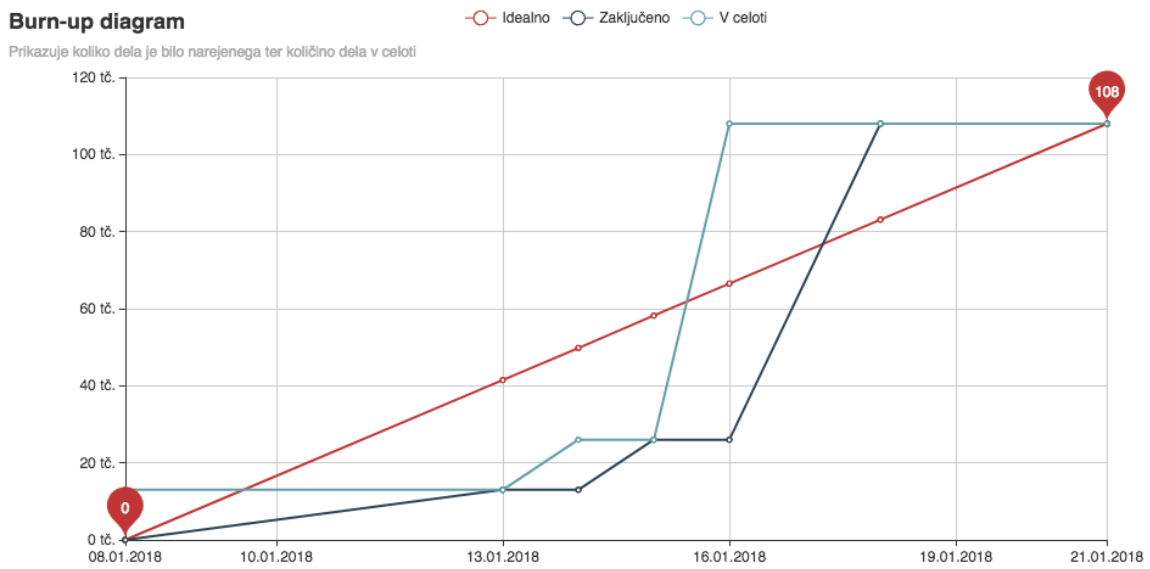
Ko smo imeli pripravljene podatke in vključili JavaScript kodo za izris, se je izrisal diagram na sliki 4.9.



Slika 4.9. Diagram preostalega dela z idealnim in dejanskim stanjem

Črna črta predstavlja količino točk z zaključenih zgodb, rdeča črta pa idealno izvajanje projekta od začetka do konca. Z diagrama je razvidno, da smo nekatere dni naredili več dela (črna črta hitro skoči navzdol), nekatere pa manj (črna črta je bolj horizontalna). Na nekaterih mestih črna črta seka rdečo črto. Tam kjer črna črta pade pod rdečo pomeni, da smo naredili več dela kot pričakovano. Kjer pa črna črta zraste nad rdečo, smo v zaostanku.

4.3.8.2 Diagram opravljenega dela



Slika 4.10. Diagram opravljenega dela

S pomočjo diagrama opravljenega dela lahko ugotovimo, če se je v času izvajanja iteracije dodajalo delo. Na sprintu se to lahko zgodi samo, če je ekipa predčasno dokončala vse zgodbe na sprintu in zaprosila skrbnika izdelka za dodatne zgodbe. Na sliki 4.10 je razvidno, da je ekipa zaprosila za dodatno delo dvakrat.

Podobno kot pri diagramu preostalega dela predstavlja rdeča črta idealno izvajanje dela od začetka do konca. Črna črta predstavlja točke zaključenih zgodb na sprintu. Modra črta predstavlja skupno število točk na sprintu. Iz te je razvidno, kako se je delo dodajalo med izvajanjem sprinta.

Poglavje 5 Analiza aplikacije

5.1 Primerjava z obstoječimi rešitvami

Za metodologijo Scrum sem se odločil, saj jo uporablja 54% podjetij, ki se ukvarja s profesionalnim razvojem programske opreme [12]. Analiza je pokazala, da si anketiranci pogosto želijo od orodja tudi možnost izrisa diagrama opravljenega dela, navidezne table in sporočanje. V našem orodju smo vključili vse tri funkcionalnosti.

Ko smo se odločili za izdelavo lastnega orodja smo se zavedali, da je na trgu že veliko obstoječih aplikacij za vodenje projektov. V okviru izvajanja dela v podjetju smo v zadnjih letih preskusili že veliko orodij, vendar nam nobeno do sedaj ni popolnoma ustrezalo. Orodja so bila preobširna, niso zajemala želene funkcionalnosti, ali pa so bila predraga. V sklopu diplomske naloge smo preskusili nekaj orodij za vodenje projektov, ki jih omenja analiza [12]. Vsako orodje bomo na kratko predstavili in zapisali, katere funkcionalnosti so mu manjkale in katere so še posebej izstopale zaradi dobre izvedbe.

5.1.1 VersionOne

VersionOne je eno najbolj znanih orodij za vodenje projektov po metodologiji Scrum. Omogoča vodenje projektov tudi po drugih popularnih metodologijah, kot sta Kanban in vitek razvoj programske opreme. Orodje je precej obsežno, zaradi česar je uporaba samega orodja kompleksnejša. Navigacija se nahaja na levem in zgornjem robu zaslona, zgornja navigacija pa se še dodatno odpre v obširen seznam dodatnih funkcionalnosti. Od vseh preskušenih orodij je imel VersionOne še najboljšo izvedbo vizualne table. Na voljo je zastonjska različica z omejenimi funkcionalnostmi in enim projektom. Na njihovi spletni strani smo sicer našli predlogo za vodenje dnevnih sestankov, samo orodje pa ne omogoča vnosa podatkov o dnevnih sestankih ali ovirah. Za beleženje časa lahko vnesemo samo količino ur za določen dan, pri tem pa ne moremo zabeležiti kdo je delo opravljal, v katerem času in kaj je bilo narejenega.

5.1.2 CA Agile Central

Orodje Rally se je pred tremi leti preimenovalo v CA Agile Central. Orodje je nekoliko manj obširno kot VersionOne, navigacija pa je bistveno bolj pregledna. Nahaja se samo na levi strani, nam najbolj uporabljane dele funkcionalnosti pa lahko dodamo ali odstranimo s hitrega seznama z zaznamki. Še posebej nam je bil všeč vmesnik za nadziranje iteracij (sprintov), ki je preprost in intuitiven. Orodje vključuje precej različnih diagramov (med drugimi tudi burn-down in burn-up), ki opravljajo nalogo poročil. Presenetilo pa nas je, da je veliko delov orodja v beta stanju, med njimi tudi vizualna tabla. Orodju manjkajo možnosti sporočanja med uporabniki, vodenje in beleženje dnevnih sestankov ter beleženje časa.

5.1.3 ScrumWorks

Orodje je na voljo zgolj kot aplikacija za operacijska sistema Windows in Linux. Navigacija je preprosta in takoj razvidna. Omogoča nam vodenje sprintov in izvajanja sestankov za retrospektivo sprinta. Orodje ne zajema izmenjave sporočil, ne omogoča vodenja in beleženja dnevnih sestankov. Spletna stran sicer omenja burn-up diagram, vendar ga v orodju nismo zasledili. Za dostopanje do navodil za uporabo potrebujemo aplikacijo Adobe Flash Player. ScrumWorks je v lasti podjetja Collab.net, ki se je v lanskem letu združilo s podjetjem VersionOne, zato je tudi prihodnost obeh orodij (ScrumWorks in VersionOne) neznana.

5.1.4 Trac

Trac je preprosto orodje, ki nam pomaga pri vodenju nalog na projektu. Napisan je v programskem jeziku Python, namestiti pa si ga moramo na lasten strežnik. Orodje je zastonj, na voljo pa je izvorna koda. Sistem bazira predvsem na pisanju spletnih strani na sistemu Wiki. To nam omogoča, da napišemo celotno dokumentacijo projekta. Vsebuje tudi določanje mejnikov in vnos nalog oz. poročanje hroščev. Manjka vodenje sprintov, dnevnih sestankov, nima nobenih grafikonov ter ne omogoča beleženja časa.

5.1.5 Mingle

Orodje Mingle se grafično precej razlikuje od ostalih. Uporabniški vmesnik je preprost, z navigacijo na vrhu strani. Skoraj vse vrste podatkov so predstavljene kot vizualne table. Na voljo imamo tablo za uporabniške zgodbe, sprinte in celo retrospektivo sprinta. Zanimiva nam je bila implementacija drevesa celotne vsebine na enem samem diagramu. Tak drevesni prikaz pa ne more biti pregleden za večje projekte, saj že pri testnih podatkih s 4 sprinti in 13 uporabniškimi zgodbami ni mogoče prikazati celotnega drevesa na enem zaslonu. Orodje

omogoča izmenjavo sporočil na uporabniških zgodbah. Manjka pa mu splošno sporočanje na okviru projekta, beleženje časa in vodenje dnevnih sestankov.

5.1.6 Xplanner

Za Xplanner je na voljo izvorna koda v programskem jeziku Java, orodje pa je zastonj za uporabo. Določamo lahko sprinte in dodajamo uporabniške zgodbe. Vnesemo lahko konkreten začetek in konec izvajanja dela, kdo je izvajal delo ter kratek opis. Orodje smo lahko namestili s strežnikom Jetty, zaradi manjkajoče dokumentacije pa smo morali poseči po izvorni kodi in privzetih konfiguracijskih datotekah, da smo se lahko prijavili. Na njihovi uradni strani obstaja povezava do demonstracijskega okolja, vendar povezava ne deluje. Orodju manjka seznam zahtev, nazadnje pa je bilo posodobljeno pred več kot 6 leti.

5.1.7 JIRA

Podjetje Atlassian ima na voljo več različnih izdelkov, med njimi je orodje za agilno vodenje projektov JIRA. Orodje je za manjša podjetja cenovno ugodno, ponujajo pa rešitev v oblaku in gostovanje orodja na lastnem strežniku. Mi smo si ogledali rešitev v oblaku. Vodenje projektov lahko izvajamo po metodologiji Scrum in Kanban. Vsebuje seznam zgodb, vodenje sprintov, vizualno tablo, burn-down diagram in osnovno beleženje časa. Nismo pa zasledili burn-up diagrama, vodenje dnevnih sestankov in komunikacije med člani. Za komunikacijo podjetje omogoča drug izdelek Hipchat za dodatno plačilo. Namesto burn-up diagrama ima na voljo kumulativni diagram poteka (angl. *Cumulative Flow Diagram*). Diagram se od burn-up diagrama razlikuje v tem, da poleg ostalih podatkov prikazuje še delo v izvajanju.

5.1.8 Trello

Trello je preprosto orodje, glavni vmesnik pa ima zasnovan na podlagi vizualne table. Na voljo imamo dodajanje kartic, ki predstavljajo uporabniške zgodbe oz. naloge. Na naloge lahko dodajamo člane, oznake, priponke in rok izdelave. Pri tem pa manjkajo beleženje časa, vodenje sprintov, dnevnih sestankov in diagrami.

5.1.9 Primerjava funkcionalnosti med orodji

Za lažjo predstavo smo pripravili tabelo 5.1. V tabeli so zajete želene funkcionalnosti.

Funkcionalnost	Version One	CA Agile Central	Scrum Works	Trac	Mingle	Xplanner	JIRA	Trello
Vizualna tabla	✔	✔	✔	✘	✔	✘	✔	✔
Sprinti	✔	✔	✔	✘	✔	✔	✔	✘
Dnevni sestanki	✘	✘	✘	✘	✘	✘	✘	✘
Beleženje časa	✘	✘	✔	✘	✘	✔	✔	✘
Beleženje časa s priročnim števcem	✘	✘	✘	✘	✘	✘	✘	✘
Sporočanje	✔	✘	✘	✘	✘	✘	✘	✔
Diagram preostalega dela	✔	✔	✔	✘	✔	✔	✔	✘
Diagram opravljenega dela	✔	✔	✘	✘	✔	✘	✘	✘
Seznam zahtev	✔	✔	✔	✔	✔	✘	✔	✘
Rešitev v oblaku ali spletni strežnik	✔	✔	✘	✔	✔	✔	✔	✔

Tabela 5.1. Primerjava orodij za vodenje projektov

Kot je razvidno iz tabele, nobeno od orodij ne podpira vseh želenih funkcionalnosti. Nobeno od orodij ne zajema vodenja dnevnih sestankov ali beleženja časa s števcem. V sklopu diplomske naloge smo v razvitem orodju podprli vse točke in s tem pridobili večjo vrednost.

5.2 Težave pri izvedbi rešitve

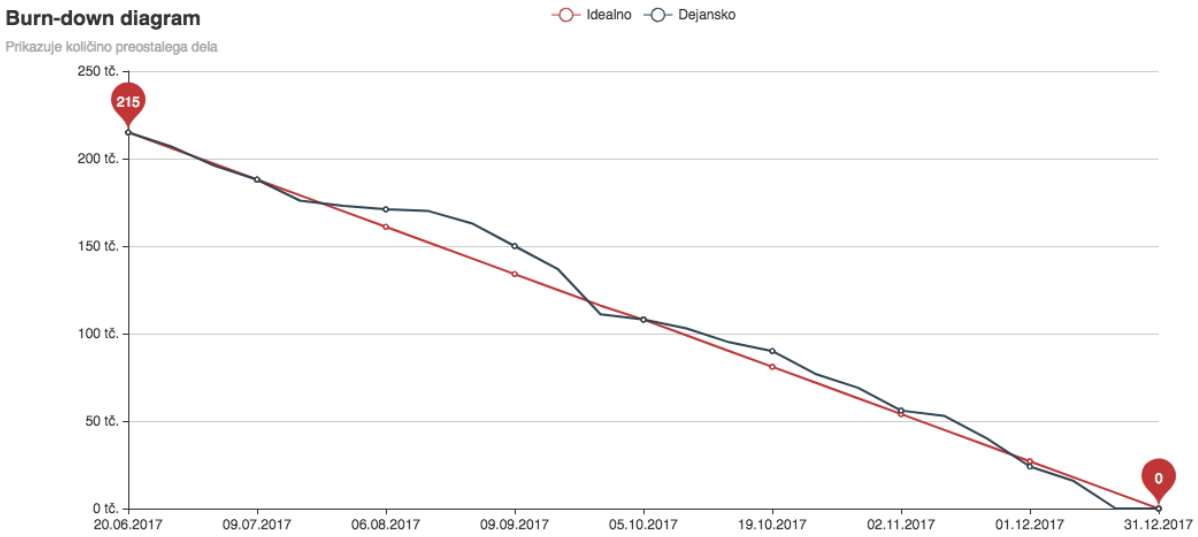
Med razvojem orodja smo naleteli na več težav. V temu poglavju bomo opisali nekaj bolj zanimivih.

5.2.1 Knjižnica za izris diagramov

Ko smo prenesli knjižnico ECharts z uradne strani in jo poskusili vgraditi v stran smo ugotovili, da je zadnja različica na voljo zgolj v kitajskem jeziku. Datoteko smo morali na roko prevesti v angleščino. Pri tem smo si pomagali s pomočjo spletnega orodja Google Translate [13], ki je solidno opravilo svoje delo. Na srečo smo morali zamenjati samo okoli 10 besedil.

5.2.2 Časovna skala na diagramu

V prvem poskusu izrisa diagrama preostalega dela se nam je izrisal diagram na sliki 5.1.



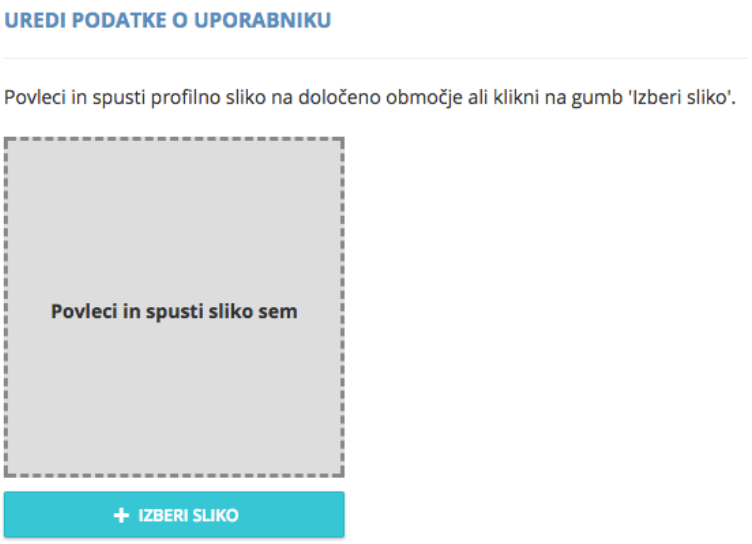
Slika 5.1. Prikaz diagrama preostalega dela z narobe izrisano časovno skalo

Na prvi pogled izgleda diagram pravilen. Rdeča črta predstavlja idealno izvajanje dela od začetka do konca z uspešnim zaključkom vsega dela. S horizontalne osi je razvidno, da skala ni linearna in tako z diagrama nimamo dobre predstave, v katerih obdobjih smo opravili več dela. Težava je v tem, da smo knjižnici podali seznam datumov s količino točk dokončanih zgodb, knjižnica pa je posamezne datume izrisala med seboj enako oddaljene. Na pravilno izrisanem diagramu pride do večjega izraza tudi razlika med idealno in dejansko črto.

Vrsto diagrama smo morali spremeniti na časovni izris ter pripraviti vrednosti idealne črte sorazmerno z datumom glede na dan opravljanja dela.

5.2.3 Vgradnja knjižnice za nalaganje datotek

Za nalaganje datotek smo se odločili, da ga izvedemo na način povleci-in-spusti, saj je zelo intuitivna. Pri tem smo pazili, da je vmesnik dokaj preprost za uporabo. Ko smo poskusili vgraditi nalaganje datotek, je vmesnik, ki ga daje na razpolago knjižnica, omogočal preveč funkcionalnosti. Precej časa smo porabili, da smo vmesnik oklestili, da izgleda kot je prikazano na sliki 5.2. Prenos na strežnik se izvede samodejno po izboru priponke.



Slika 5.2. Nalaganje profilne slike

5.3 Možnosti za izboljšave

Izdelava orodja za vodenje projektov predstavlja velik zalogaj, zato v okviru diplomske naloge nismo mogli implementirati več funkcionalnosti, ki bi nam še dodatno olajšale delo. V času izdelave diplomske naloge smo naleteli na nekaj možnih izboljšav, ki jih bomo opisali v tem poglavju.

5.3.1 Načrtovanje izdaj

Metodologija Scrum zajema tudi planiranje izdaj (angl. *release planning*). Z njimi lahko razčlenimo projekt na več manjših segmentov. Naročnik npr. želi imeti osnovno delujočo aplikacijo, da jo lahko zgodaj objavi in čimprej začne dobivati dohodek. Kasneje se aplikacijo dograjuje v želenih intervalih s predpisano funkcionalnostjo. Prvo različico z objavo lahko določimo s prvo izdajo, podobno pa tudi vsako nadaljnjo različico.

Naše orodje je precej enostavno razširiti, saj lahko dodamo dodatno tabelo z izdajami in povezovalno tabelo med izdajami in sprinti.

5.3.2 Vodenje stroškov

Vodenje stroškov (angl. *budget tracking*) sicer ni del metodologije Scrum, vendar je nadzor nad stroški ključnega pomena pri vodenju podjetja. Ker z orodjem že omogočamo beleženje časa in vnos ocenjenega časa, lahko orodje dokaj preprosto razširimo z dodatnimi izpisi, ki agregirajo porabljen čas na nalogah po izbranih obdobjih in jih primerjajo z ocenjenim časom. Posameznemu članu ekipe lahko dodelimo strošek delovne ure in njihovo zabeleženo delo pomnožimo z urno postavko.

5.3.3 Integracija z drugimi sistemi

Ena bolj želenih funkcionalnosti pri orodjih za vodenje projektov je integracija z drugimi sistemi [12], kot npr. sistem za upravljanje z različicami izvorne kode (angl. *version control system*).

Sistem bi lahko dopolnili tako, da avtomatsko povzema šifre nalog iz besedil vnosa spremembe programske kode v sistem Git. Nalogi se lahko doda referenca na spremembo v kodi ali avtomatsko zaključi nalogo.

5.3.4 Poročila

Sistem lahko bistveno izboljšamo z raznimi tekstovnimi in grafičnimi poročili. Ker v sistemu že vodimo veliko informacij, lahko poročila enostavno dodamo v orodje kot dodatne izpise. Primeri poročil:

- opravljeno delo po članu ekipe v določenem obdobju,
- vsota razlik med ocenjenim časom in dejanskim časom na projektu,
- izpis nalog, na katerih se je porabilo več časa, kot je bilo načrtovano,
- izpis seznama opravljenih nalog na sprintu primeren za tiskanje,
- grafični prikaz opravljenega dela v izbranem obdobju med različnimi projekti

Poglavje 6 Sklepne ugotovitve

V sklopu diplomske naloge smo izdelali orodje za vodenje projektov po agilni metodologiji Scrum. Orodje smo izdelali zaradi potrebe v podjetju. Glavni namen orodja je bila preprosta uporaba, možnost beleženja časa na nalogah in preglednost. V času izdelave smo orodje sami preskušali z vnosom podatkov na praktičnem primeru. Najprej smo zabeležili zgodbe, sprinte in porabljen čas v preglednico Excel. Nekaj podatkov smo vnesli z orodjem phpMyAdmin, nato pa postopoma testirali nove funkcionalnosti našega orodja z vnosom preostalih podatkov. V času, ki nam je bil na voljo, nam je uspelo implementirati vse želene zahteve. Pri tem nam je bistveno pomagala izbira programskega ogrodja Laravel, saj smo bili z njim že predhodno seznanjeni in smo lahko hitro opravili delo.

Med samim razvojem orodja smo naleteli na nekaj težav, ki pa smo jih vse dokaj hitro rešili, saj imamo že veliko delovnih izkušenj na tem področju. Še največ težav nam je predstavljalo ravno nalaganje priponk, saj smo se z uporabljeno knjižnico srečali prvič. Kljub večletnim delovnim izkušnjam sem vseeno pridobil nova znanja, predvsem pri pravilni rabi metodologije Scrum.

Orodje smo dobro načrtovali in se držali kodirnih standardov, zato z možnimi nadgradnjami sistema ne bi imeli težav. V sistem bi dodali dodatne krmilnike in izpise, ki ne bi vplivali na delovanje preostalih funkcionalnosti. Vsa programska koda je napisana čitljivo in bogato opremljena s komentarji.

Kazalo slik in tabel

Literatura

[1] SitePoint: *The State of PHP MVC Frameworks in 2017*, 2017, Dosegljivo na: https://www.sitepoint.com/the-state-of-php-mvc-frameworks-in-2017/. [Dostopano: 23. 12. 2017].

[2] Jeff Sutherland: *V gruču do uspeha – umetnost vodenja projektov z metodo SCRUM*, Ljubljana: Pasadena, 2016.

[3] Igor Rožanc. (2008). *Študijsko gradivo za interno uporabo pri predmetu Razvoj programskih sistemov 2* [Online]. Fakulteta za računalništvo in informatiko. Dosegljivo: http://studentski.net/gradivo/ulj_fri_ri3_tpo_sno_predavanja_05

[4] Andrew Stellman, Jennifer Greene: *Learning Agile – Understanding Scrum, XP, Lean, and Kanban*, O'Reilly, 2015

[5] Mike Cohn: *Scrum & XP – Better Together*, Scrum Alliance, 2014. [Online]. Dosegljivo: https://www.scrumalliance.org/community/spotlight/mike-cohn/april-2014/scrum-xp-better-together. [Dostopano: 23. 12. 2017].

[6] Ala'a Elbeheri: *Burn up vs. burn down chart*, LinkedIn, 2016. [Online]. Dosegljivo: https://www.linkedin.com/pulse/burn-up-vs-down-chart-alaa-el-beheri-cisa-rmp-pmp-bcp-itil/. [Dostopano: 24. 12. 2017].

[7] Agile Business Consortium. *What is DSDM, 2016.* [Online]. Dosegljivo: https://www.agilebusiness.org/what-is-dsdm. [Dostopano: 25. 12. 2017].

[8] *Bootstrap* [Online]. Dosegljivo: https://getbootstrap.com/. [Dostopano: 28. 12. 2017].

[9] *Laravel* [Online]. Dosegljivo: https://laravel.com/. [Dostopano: 11. 1. 2018].

[10] *HTMLPurifier for Laravel 5* [Online]. Dosegljivo: https://github.com/mewebstudio/Purifier. [Dostopano: 2. 1. 2018]

[11] ECharts dokumentacija [Online]. Dosegljivo: https://ecomfe.github.io/echarts-doc/public/en/tutorial.html. [Dostopano: 2. 1. 2018]

[12] Gayane Azizyan, Miganoush Katrin Magarian in Mira Kajko-Mattson. Survey of Agile Tool Usage and Needs. V Agile Conference, 2011, [Online]. Dosegljivo: IEEE Xplore, http://ieeexplore.ieee.org/document/6005503/. [Dostopano: 28. 3. 2017].

[13] Google Translate [Online]. Dosegljivo: https://translate.google.com/. [Dostopano: 2. 1. 2018]

[14] Mike Cohn: *User Stories Applied – For Agile Software Development*, Adison-Wesley, 2004

[15] Henrik Kniberg, Mattias Skarin: *Kanban and Scrum – making the most of both*, InfoQ, 2010

[16] Matthias Marschall: *Kanban vs Scrum vs Agile*, Agile Web Development and Operations, 2015. [Online]. Dosegljivo: https://www.agileweboperations.com/scrum-vs-kanban. [Dostopano: 18. 12. 2017]

[17] *jQuery* [Online]. Dosegljivo: https://jquery.com/. [Dostopano: 2. 1. 2018]

[18] *Dragula* [Online]. Dosegljivo: https://github.com/bevacqua/dragula. [Dostopano: 2. 1. 2018]

[19] *jQuery File Upload* [Online]. Dosegljivo: https://github.com/blueimp/jQuery-File-Upload. [Dostopano: 2. 1. 2018]

[20] Kent Beck, Cynthia Andres: *Extreme programming explained*, Addison Wesley Professional, 2004

[21] Franc Solina: *Projektno vodenje razvoja programske opreme*, Založba FE in FRI, 1997