Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

David Bašelj

Razvoj spletne aplikacije Sledenje

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Viljan Mahnič

Ljubljana, 2018

Copyright. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Proučite delovni proces v podjetju, ki se ukvarja z digitalizacijo in elektronsko hrambo papirne dokumentacije. Analizirajte pomanjkljivosti v ročnem spremljanju tega procesa in izdelajte aplikacijo, ki bo omogočala sledenje vseh faz v postopku od sprejema gradiva v obdelavo do njegovega arhiviranja v elektronski obliki oziroma fizičnega uničenja. V nalogi predstavite specifikacijo zahtev, načrt podatkovne baze, postopek razvoja in uporabljena orodja ter funkcionalnosti, ki so na voljo uporabniku.

Največja zahvala gre mentorju prof. dr. Viljanu Mahniču, ki me je vodil skozi proces pisanja diplomske naloge in mi je vseskozi stal ob strani. Vedno si je vzel čas in v celoti pregledal diplomsko nalogo ter predlagal popravke. Zahvaljujem se tudi Marku Zupančiču ter Matjažu Zagorcu iz podjetja Mikrografija d.o.o., ki sta me usmerjala pri izdelavi naloge. Zahvaljujem se tudi vsem svojim bližnjim, ki so mi stali ob strani tekom celotenga študija. Zahvala gre tudi punci, ki mi je nudila podporo tekom pisanja diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
MVC	model-view-controller	model-pogled-krmilnik
DAO	Database Access Objects	objekti za dostop do baze
AJAX	asynchronous JavaScript and XML	asinhroni JavaScript in XML
HTML	Hyper Text Markup Language	jezik za označevanje nadbesedila
CSS	Cascading Style Sheets	kaskadne stilske podloge
SQL	Structured Query Language	strukturirani povpraševalni jezik
IDE	Integrated development environment	integrirano razvojno okolje
PHP	Hypertext Preprocessor	skriptni programski jezik za izdelavo spletnih strani
API	Application Programming Interface	aplikacijski programski vmesnik
DRY	Don't Repat Yourself	ne ponavljaj kode
OCR	Optical Character Recognition	optično prepoznavanje znakov
REST	Representational State Transfer	arhitektura za izmenjavo podatkov med spletnimi storitvami

Povzetek

Naslov: Razvoj spletne aplikacije Sledenje

Avtor: David Bašelj

V tem diplomskem delu je opisan razvoj spletne aplikacije Sledenje, ki omogoča lažje vodenje delovnega procesa arhiviranja dokumentov v podjetju Mikrografija d.o.o. Pred uvedbo aplikacije je delovni proces arhiviranja potekal ročno, brez informacijskega sistema, zato je bilo delo oteženo in zamudno. Vsa dokumentacija za optično prebrane dokumente se je vodila ročno na listih. Na podlagi opisanega delovnega procesa in v sodelovnju z zaposlenimi, smo definirali zahtevane funkcionalnosti nove aplikacije. V nadaljevanju diplomskega dela so opisane uporabljene tehnologije in orodja, ki so nam bili v pomoč pri razvoju. Predstavljen je tudi podatkovni model, ki je bil uporabljen v aplikaciji. Rezultat diplomskega dela je razvita aplikacija, ki rešuje opisane težave in zaposlenim v podjetju omogoča lažje delo. Proti koncu diplomske naloge je najprej opisanih nekaj izmed najpomembnejših metod, ki smo jih uporabili za ravoj aplikacije, nato pa sledi še opis delujoče aplikacije. V opisu aplikacije so predstavljeni pogledi uporabniškega vmesnika, ki jih zaposleni vsakodnevno uporabljajo tekom izvajanja delovnega procesa. Na koncu diplomske naloge so opisane sklepne ugotovitve, do katerih smo prišli tekom razvoja aplikacije.

Ključne besede: spletna aplikacija, php, yii, sledenje.

Abstract

Title: Development of web application Sledenje

Author: David Bašelj

This graduation thesis describes the development of the web application Tracking, which makes it easier to manage the document archiving process in the company Mikrografija d.o.o. Prior to the launch of the application, the archival workflow was carried out manually, without an information system, and the work was therefore difficult and time consuming. All documentation for scanned documents was handled manually on sheets. Based on the described work process and in collaboration with employees, we defined the required functionality of the new application. The diploma thesis describes the technologies and tools used to help us develop application. A data model that was used in the application is also presented. The result of the diploma work is a developed application that solves the problems described and makes it easier for employees in the company to work. Towards the end of the thesis, there are first described some of the most important methods that we used for application development, followed by a description of the working application. The application description describes the user interface that the employees use on a daily basis during the work process. At the end of the thesis, the conclusions reached during the development of the application are described.

Keywords: web application, php, yii, tracking.

Poglavje 1

Uvod

Ideja za diplomsko nalogo izhaja iz podjetja Mikrografija d.o.o. Podjetje se ukvarja z optičnim branjem, obdelovanjem, hrambo in arhiviranjem papirnate dokumentacije. Podjetje nudi tudi storitve v oblaku, kjer lahko stranke, ki so naročile arhiviranje dokumentacije, dostopajo do arhiviranih dokumentov preko oblačne storitve. Podjetje je zelo prilagodljivo in se trudi ugoditi vsem željam stranke. Skozi leta poslovanja in dela z mnogo strankami se je v podjetju nakopičilo veliko dokumentov, ki so potrebni za vodenje projekta oziroma naročila stranke. Zaradi širjenja podjetja na tuje trge in posledično zaradi vse večjega števila naročil, katere je potrebno uspešno izpeljati do konca, nastaja zmeda pri sledenju delovnega procesa arhiviranja dokumentov vseh teh naročil. Prišla je potreba po centraliziranemu sistemu, ki bo poenotil naročila in tako omogočal lažje vodenje naročil, kot tudi sam proces arhiviranja dokumentov v podjetju. Kot zaposlen v podjetju strmim k temu, da bi s svojim znanjem kar se da prispeval k razvoju podjetja, ob enem bi pa tudi rad utrdil svoje tehnične veščine, ki sem jih pridobil na fakulteti. Tako je znotraj podjetja nastala ideja, da je potrebno vpeljati neko informacijsko rešitev, ki bo omogočala lažje sledenje povečanega delovnega procesa. Nadrejeni so podali predlog, da bi se znotraj podjetja razvila aplikacija, ki bi poenostavila delo in reševala zgoraj opisane težave. Na trgu obstajajo različne programske rešitve, ki bi lahko poenostavile delovni proces v Mikro-

grafiji, vendar bi bila vpeljava takega sistema v podjetje zelo draga, in ker ima podjetje svoje razvijalce, se je vodstvo podjetja odločilo, da se razvije lastno programsko rešitev z imenom Sledenje.

V nadaljevanju se bomo v poglavju 2 srečali z opisom problema, ki ga diplomska naloga rešuje. Za lažje razumevanje na začetku poglavja najprej predstavimo osnovne izraze, nato pa nadaljujemo z opisom poteka dela v produkciji Mikrografije pred upeljavo aplikacije ter opisom, kako poteka delovni proces v podjetju. Na koncu poglavja pa opišemo zahtevane funkcionalnosti, ki smo jih zastavili pri načrtovanju.

V poglavju 3 predstavimo tehnološki del aplikacije. Na kratko predstavimo uporabljene tehnologije ter opišemo podatkovni model. Za zaključek poglavja pa predstavimo razvojno okolje, ki smo ga uporabili za razvoj aplikacije.

V poglavju 4 predstavimo najpomembnejše oziroma najzanimivejše metode, ki smo jih uporabili za razvoj aplikacije. Na koncu poglavja pa predstavimo delovanje razvite aplikacije.

V poglavju 5 opišemo sklepne ugotovitve, do katerih smo prišli tekom izdelave aplikacije in diplomske naloge.

Poglavje 2

Opis problema

Pred razvojem nove aplikacije je dobro razumeti problem oziroma določen proces, pri katerem prihaja do težav. Če hočemo težave odpraviti, je zelo pomembno, da problem oziroma proces dobro razumemo. Zato v sledečem poglavju opišemo, kako poteka delovni proces v produkciji Mikrografije. Na začetku najprej definiramo pojme, ki bodo pripomogli k lažjemu razumevanju diplomske naloge. V nadaljevanju opišemo, kako je delo potekalo pred uvedbo aplikacije in kako poteka delovni proces v produkciji Mikrografije. Na koncu pa še opišemo zahtevane funkcionalnosti, ki naj bi jih nova aplikacija podprla.

2.1 Definicija izrazov in pojmov

Zaradi lažjega razumevanja diplomske naloge, v nadaljevanju opišemo izraze ter pojme, ki bodo pripomogli k razumevanju besedila.

- barkoda oz. črtna koda (ang. barcode) - Način zapisa številk in črk s črtami in presledki različnih širin, ki so optično berljivi in predstavljajo podatke. V nadaljevanju bomo uporabljali pravilen izraz črtna koda, na slikah pa bo pogosto viden izraz barkoda, saj se ta uporablja znotraj podjetja in aplikacije.

- gradivo - Papirnata dokumentacija, ki jo je potrebno optično prebrati.

V redkih primerih so lahko to tudi mikrofilmi, ki se jih prav tako pretvori v digitalno obliko s pomočjo posebne naprave.

- tip enote gradiva - Oblika, v kateri se lahko pojavi gradivo. To so lahko škatla, knjiga, mapa, načrt, registrator, fascikel, mikrofilm.

- Enota gradiva - V diplomski nalogi se pogosto pojavi omenjen izraz. Enota gradiva označuje nek fizičen predmet, ki vsebuje papirnato dokumentacijo. Najpogosteje enota gradiva predstavjla kartonsko škatlo velikosti 40 x 30 x 30 cm *(D x Š x V)*. Lahko pa je tudi paleta ali paket. Vsaka enota gradiva ima določeno unikatno 14 mestno črtno kodo.

- podenota gradiva - Označuje gradivo, ki se nahaja v enoti gradiva. Tip podenote gradiva je lahko različen. Vsaka podenota gradiva ima določeno unikatno 14 ali 15 mestno črtno kodo.

- metapodatki - Podatki, ki označujejo, opisujejo, identificirajo druge podatke, dokumente, npr. dodatni metapodatki pri podenoti gradiva opisujejo podenoto gradiva.

- status enote - Vsaka enota gradiva ima tekom delovnega procesa določen status. Status določa, v kateri fazi (*prevzeto, popisano, pripravljanje, priprava zaključena, skeniranje,...*) delovnega procesa se nahaja enota gradiva.

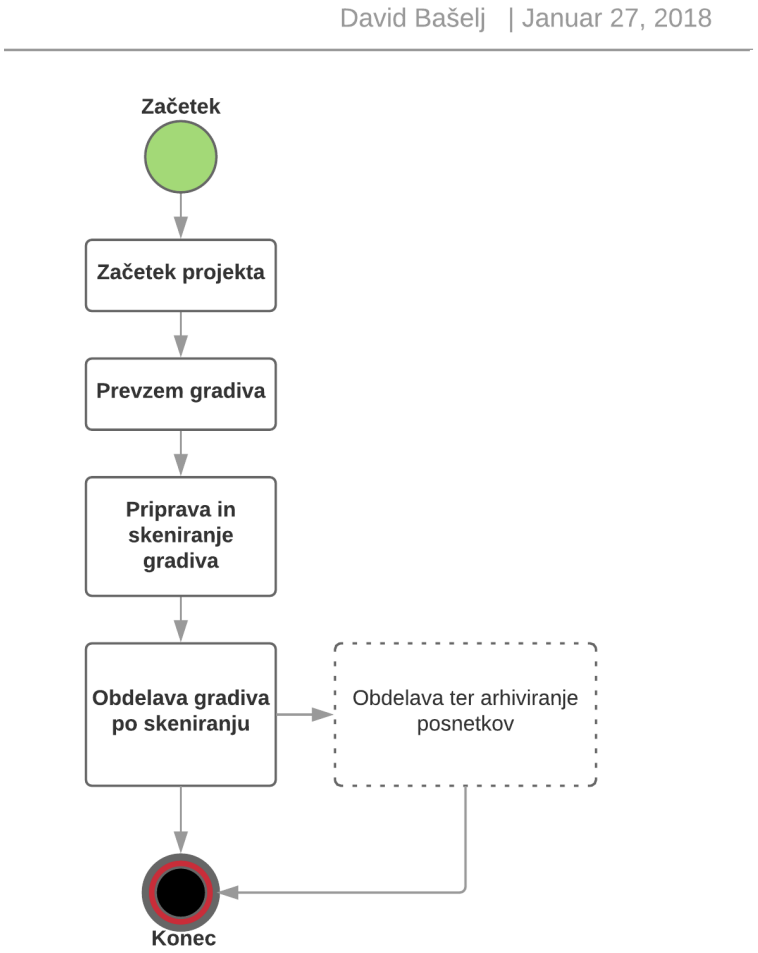
- produkcija - Fizičen prostor v Mikrografiji, kjer se opravlja obdelava dokumentov.

- časovna sled - Vsaka operacija, ki spremeni status enote gradiva, se zabeleži v aplikaciji.

2.2 Potek dela pred uvedbo aplikacije

Ene izmed mnogih storitev Mikrografije so optično branje papirnate dokumentacije, pretvorba papirnih dokumentov v digitalno obliko, hranjenje fizičnega gradiva in pa hranjenje digitalnih posnetkov. Za obdelavo dokumentacije je potrebno veliko spremnih dokumentov, ki spremljajo gradivo od prevzema, pa vse do hranjenja oziroma uničenja gradiva. Pred aplikacijo Sledenje so vodje oddelkov v produkciji, kjer se optično bere gradivo, vodile različne evidence v obliki word, excel, pdf. Te datoteke so se nahajale na osebnih računalnikih ali pa na lokalnem omrežju in zaradi vse več projektov je postajalo delo s temi datotekami vse težje. Cilj je bil, da se ta proces centralizira in s tem prepreči razdrobljenost podatkov, ki so ključni za uspešno vodenje delovnega procesa.

Na sliki 2.1 lahko vidimo poenostavljen diagram delovnega procesa arhiviranja dokumentacije. Na začetku projekta oziroma naročila, se ustvari dokument *specifikacija zahtev storitve za naročnika*, ki je pomemben za nadaljni delovni proces. Po ustvarjeni specifikaciji se začne z prevzemanjem gradiva, kjer se ročno izpolnjujejo zapisniki o prevzemu gradiva. Tekom priprave in optičnega branja dokumentov se poleg obdelovanja gradiva izpolnjuejo dodatni dokumenti (*spremni list, evidence*), ki služijo kot sledljivostni listi gradiva. V nadalnjih točkah na kratko opišemo vsakega od teh dokumentov (specifikacija, zapisnik o prevzemu, spremni list, evidence), ki so se pred uvedbo aplikacije ustvarjali ročno.



Slika 2.1: Poenostavljen delovni proces

2.2.1 Specifikacija zahtev storitve za naročnika

Vhod dokumentacije v proces prevzema gradiva ter začetek obdelovanja gradiva je pogojen z izpolnjenim obrazcem *specifikacija zahtev storitve za naročnika*. Dokumenti, prisotni tekom priprave specifikacije so:

1. seznam oddanega gradiva, ki ga pripravi naročnik

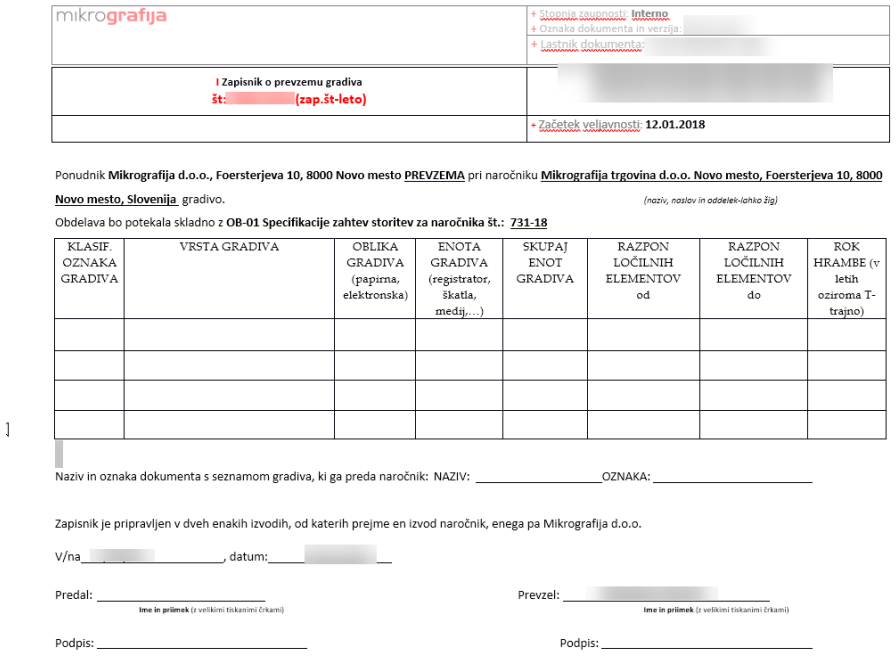
2. zapisnik o prevzemu dokumentacije, ki ga podpiše naročnik ob predaji gradiva, na njem je zabeležena stvar predaje oz. prevzema, količina, kraj in datum predaje

3. klasifikacijski načrt, ki je lahko del seznama, ki ga odda naročnik, ali del zapisnika o prevzemu, ali pa bo še nastal v prostorih izvajalca

4. specifikacija zahtev storitve za naročnika

2.2.2 Zapisnik o prevzemu gradiva

Na sliki 2.2 je viden zapisnik o prevzemu gradiva, ki se je pred uvedbo aplikacije ustvarjal ročno. Prevzemni zapisnik služi kot dokument med naročnikom in izvajalcem, kjer so navedene enote gradiva. Tako imata oba tako naročnik kot izvajalec pravno podlago o predani/prevzeti dokumentaciji.



mikrografija

+ Stopnja zaupnosti: **interno**
+ Oznaka dokumenta in verzija:
+ Lastnik dokumenta:

I Zapisnik o prevzemu gradiva
št: (zap.št-leto)

+ Začetek veljavnosti: **12.01.2018**

Ponudnik **Mikrografija d.o.o., Foersterjeva 10, 8000 Novo mesto** PREVZEMA pri naročniku **Mikrografija trgovina d.o.o. Novo mesto, Foersterjeva 10, 8000 Novo mesto, Slovenija** gradivo.
(naziv, naslov in oddelek-lahko žig)

Obdelava bo potekala skladno z **OB-01 Specifikacije zahtev storitev za naročnika št.: 731-18**

KLASIF. OZNAKA GRADIVA	VRSTA GRADIVA	OBLIKA GRADIVA (papirna, elektronska)	ENOTA GRADIVA (registrator, škatla, medij,...)	SKUPAJ ENOT GRADIVA	RAZPON LOČILNIH ELEMENTOV od	RAZPON LOČILNIH ELEMENTOV do	ROK HRAMBE (v letih oziroma T-trajno)

Naziv in oznaka dokumenta s seznamom gradiva, ki ga preda naročnik: NAZIV: _____ OZNAKA: _____

Zapisnik je pripravljen v dveh enakih izvodih, od katerih prejme en izvod naročnik, enega pa Mikrografija d.o.o.

V/na _____, datum: _____

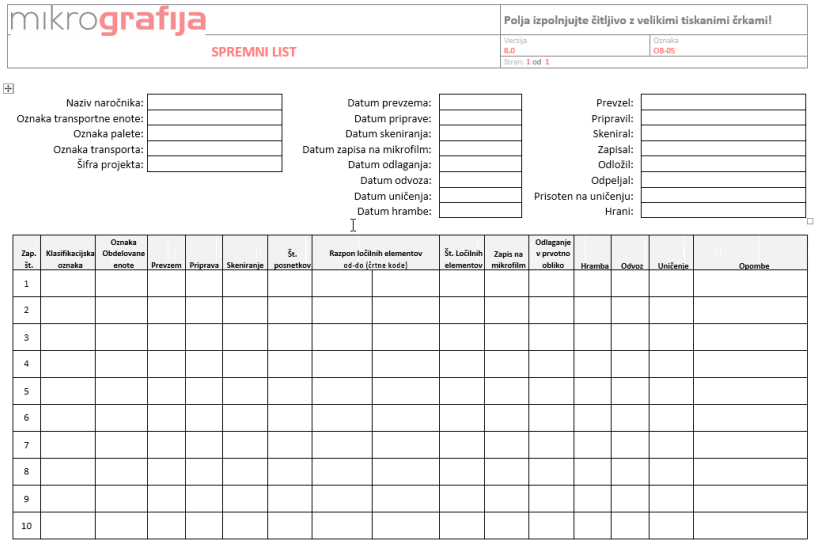
Predal: _____ Prevzel: _____
Ime in priimek (z velikimi tiskanimi črkami) Ime in priimek (z velikimi tiskanimi črkami)

Podpis: _____ Podpis: _____

Slika 2.2: Zapisnik o prevzemu gradiva

2.2.3 Spremni list

Spremni list je **sledljivostni list** od prevzema dokumentacije do optičnega branja dokumentacije. Vsaka enota gradiva ima svoj spremni list kot na sliki 2.3, ki je shranjen v plastificirani vrečki. Vse skupaj je prilepljeno na enoto gradiva. Na spremnem listu so pomembni podatki, s katerih se razberejo podatki, kdaj in kdo je opravil določen proces v produkciji. Ti podatki se dopisujejo na spremni list tekom delovnega procesa.



mikrografija

SPREMNI LIST

Polja izpolnjujte čitljivo z velikimi tiskanimi črkami!

Verzija 8.0 | Oznaka OB-05 | Stran 1 od 1

Naziv naročnika:
Oznaka transportne enote:
Oznaka palete:
Oznaka transporta:
Šifra projekta:

Datum prevzema:
Datum priprave:
Datum skeniranja:
Datum zapisa na mikrofilm:
Datum odlaganja:
Datum odvoza:
Datum uničenja:
Datum hrambe:

Prevzel:
Pripravil:
Skeniral:
Zapisal:
Odložil:
Odpeljal:
Prisoten na uničenju:
Hrani:

Zap. št.	Klasifikacijska oznaka	Oznaka Obdelovane enote	Prevzem	Priprava	Skeniranje	Št. posnetkov	Razpon ločilnih elementov od-do (črtne kode)		Št. Ločilnih elementov	Zapis na mikrofilm	Odlaganje v prvotno obliko	Hramba	Odvoz	Uničenje	Opombe
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															

Slika 2.3: Primer spremnega lista

2.2.4 Evidence priprave, optičnega branja, optične prepoznave

Evidence (excel dokumenti) služijo sledljivosti procesa od optičnega branja pa do konca procesa, oziroma do predaje gradiva naročniku. V evidencah se nahajajo vsi pomembni podatki glede vsebine škatle, kdo je pripravljal, kdo je opravljal optično branje, čas optičnega branja itd. Iz teh evidenc se je nato delala statistika.

Odločilni razlogi oziroma potrebe po razvoju nove aplikacije so bile:

1. porabljalo se je vse več listov za spremne ter evidenčne liste
2. potreba po hitrejšem izpolnjevanju spremnih oz. evidenčnih listov.
3. zaradi vse več projektov in dokumentov potrebnih za projekte je bilo vse težje slediti projektom
4. potrebno poenostaviti proces, vsak v podjetju mora bit zamenljiv

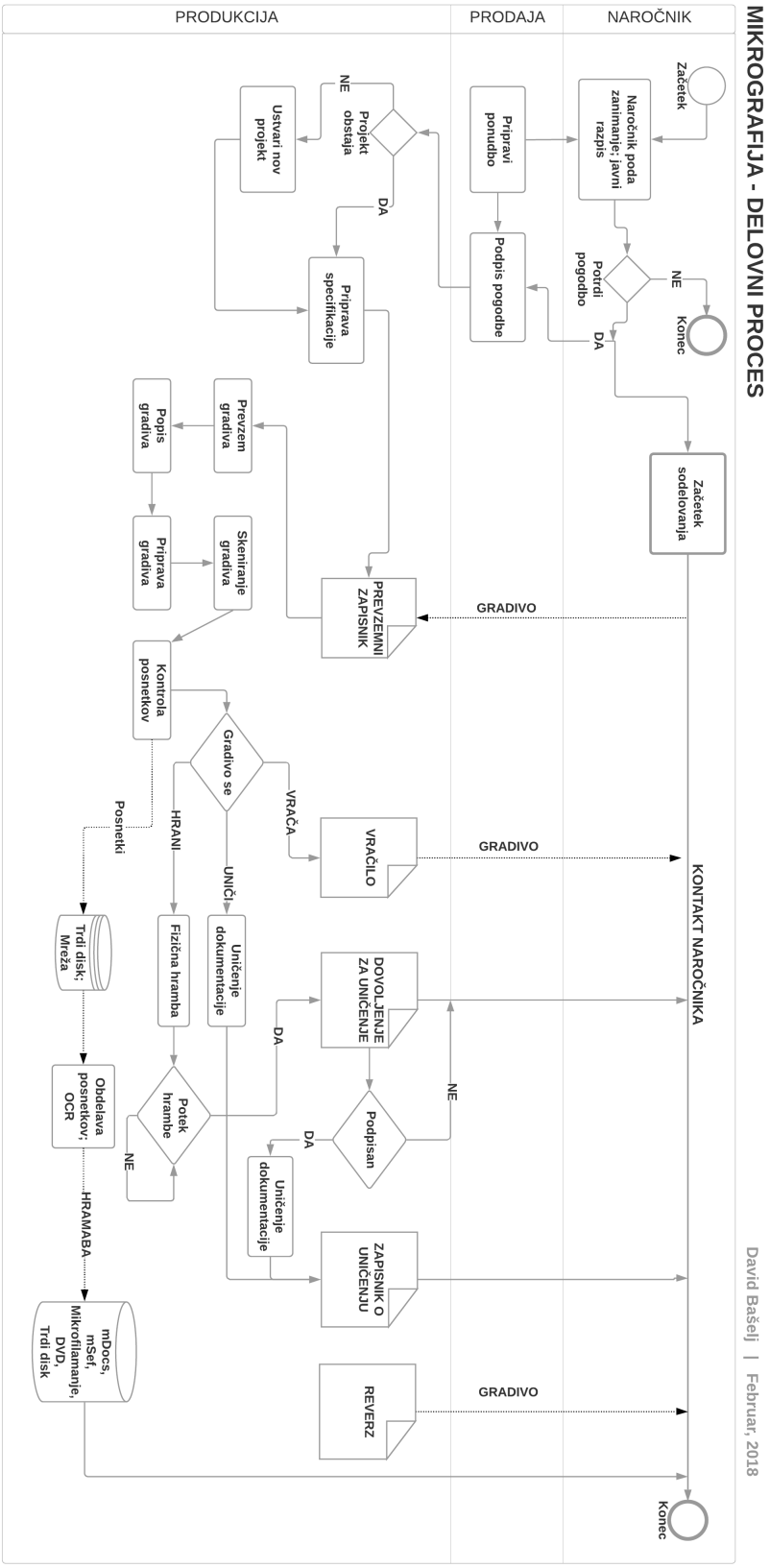
2.3 Opis delovnega procesa

Na sliki 2.4 je viden podrobnejši diagram, ki prikazuje, kako poteka delovni proces arhiviranja gradiva. Predpostavljajmo, da ima neko podjetje, ki je šlo v stečaj, in ima veliko količino dokumentacije, katero je potrebno hraniti. Da se bi izognili dragim stroškom hranjenja te dokumentacije, se odločijo, da jo bodo hranili v elektronski obliki. Zahtevajo ponudbo od Mikrografije ali pa razpišejo razpis, na katerega se lahko podjetja prijavijo. V kolikor se podjetje odloči za sodelovanje z Mikrografijo, se sklene pogodba. Nato se začne delo v produkciji, kot je prikazano na sliki 2.4. Ker je to nov naročnik, se ustvari nov projekt, ki ga je potrebno voditi. Nato se na podlagi naročnikovih želja ustvari specifikacijo, ki natanko določa, kako in na kakšen način se bo dokumentacija obdelovala in tudi kaj se z njo stori po obdelavi. V primeru različnih vrst dokumentacije oziroma, če naročnik zahteva, da se del dokumentacije optično bere in nato uniči, del dokumentacija pa optično bere in vrne, se ustvarita dve različni specifikaciji. Tako lahko za nek projekt ustvarimo eno ali več specifikacij. Ko vodje ustvarijo specifikacijo, se lahko začne s prevzemanjem gradiva. Za vsak prevzem se mora obvezno kreirati prevzemni zapisnik. Na kakšen način in kako pogosto se gradivo prevzema, je zapisano v specifikaciji. Ob kreiranju prevzemnega zapisnika, ko gradivo fizično prispe v skladišča Mikrografije, se naredi prevzem gradiva, kjer se popiše vse enote gradiva (ponavadi so to škatle na paletah s črtnimi kodami). Po prevzemu se

naredi popis gradiva, kjer se popiše vse podenote gradiva, šele nato se lahko začne gradivo optično brati. Po potrebi je potrebno pred optičnem branjem gradiva vso dokumentacijo pripraviti (odstranjevanje sponk). Po optičnem branju se posnetke prekontrolira in ob odkritju napak se dokumentacijo še enkrat optično prebere. Po zaključenem delovnem procesu se z gradivom naredi, kot je zapisano v specifikaciji. Če se gradivo:

- **vrača**, se naredi *zapisnik o vračilu gradiva* in se gradivo vrne naročniku
- **uniči**, se gradivo pelje na uničenje in se ustvari *zapisnik o uničenju*, ki se ga pošlje naročniku
- **hrani**, se gradivo fizično shrani v skladiščih Mikrografije. Gradivo se lahko hrani trajno ali pa ima določen čas hranjenja. Po poteku roka hranjenja se od naročnika pridobi *dovoljenje za uničenje gradiva*, nato se gradivo uniči in se naročniku preda *zapisnik o uničenju* gradiva

Na sliki 2.4 lahko na koncu delovnega procesa vidimo, da se posnetki izvažajo na trdi disk. To pomeni, da se posnetki (.tif,.png) izvozijo iz optičnega bralnika in se odložijo nekam na strežnik posebej namenjen za to. Informatiki v oddelku za optično prepoznavanje znakov (ang. optical character recognition - OCR) nato te posnetke obdelajo tako, kot je zapisano v specifikaciji oziroma kakor je dogovorjeno s stranko. Posnetke se lahko zapisuje na mikrofilme, digitalne pomnilniške medije (ang. digital versatile disc - DVD), naloži na trdi disk ali pa se jih vnese v dokumentni sistem. Po koncu obdelave se posnetke preda naročniku v dogovorjeni obliki. Na sliki 2.4 je prikazan tudi reverz. Lahko se zgodi, da naročnik potrebuje gradivo, ki se fizično hrani v Mikrografiji. Takrat se naredi zapisnik reverz, in se željeno gradivo preda naročniku.



Slika 2.4: Delovni proces arhiviranja gradiva v Mikrografiji

2.4 Zahtevane funkcionalnosti

Pri izdelavi spletne aplikacije Sledenje smo se osredotočili predvsem na uporabnike, saj so oni tisti, katerim smo želeli olajšati delo v proukciji Mikrografije. Kljub raznim predlogom, ki so jih imeli zaposleni, smo se zavedali, da vseh ne bomo mogli izvesti in, da bomo morali delovni proces poenostaviti do te mere, da bo aplikacija funkcionalna in enostavna za uporabo. Na podlagi zgoraj opisanega delovnega procesa, smo se lotili določanja funkcionalnosti. Po preučevanju diagrama 2.4 smo napisali zahtevane funkcionalnosti, ki so ključne za uspešen doseg željenega cilja. Funkcionalnosti smo določili v grobem in se nismo preveč spuščali v podrobnosti. Napisali smo samo ključne točke, ki so služile kot smernice pri razvoju, nismo pa opisovali vsakega pogleda posebej in kaj naj ta pogled omogoča.

Pred začetkom pisanja funkcionalnosti smo najprej določili uporabniške vloge, ki bodo možne v aplikaciji:

- osnovni uporabniki - so zaposleni v podjetju, ki delajo na delovnem procesu arhiviranja gradiva. Delajo na pripravi gradiva, optičnem branju posnetkov in končni kontroli posnetkov. Aplikacija mora poskrbeti, da lahko osnovni uporabniki urejajo samo tiste poglede v aplikaciji, ki so vezani na pripravo gradiva, optično branje in kontrolo posnetkov

- napredni uporabniki - podedujejo vse pravice osnovnega uporabnika. So tisti zaposleni, ki so vodje ali pa tisti, ki jim je bila znotraj podjetja dodeljena pravica za prevzemanje in popisovanje gradiva

- skrbnik aplikacije - podeduje vse pravice naprednega uporabnika. So tisti zaposleni, ki skrbijo za pravilno delovanje aplikacije. Skrbniku je omogčen dostop do vseh delov aplikacije

Funkcionalnost	**Opis**
Prijava uporabnika	Uporabnik se lahko prijavi v aplikacijo. Če uporabnik ni prijavljen, lahko dostopa samo do prijavnega okna. Vse ostale funkcije in pogledi so zaklenjeni. Na prijavnem oknu lahko obkljuka *Zapomni si me*, da mu ni potrebno vsakič znova vpisovati gesla. Podpreti je treba dve vrsti vpisa: • prijava preko internetnega protokola za dostopanje do imenika (ang. lightweight directory access protocol - LDAP) • prijava za uporabnike, ki niso v internem imeniku LDAP
Nastavitve uporabnika	Prijavljen uporabnik si lahko nastavi samo jezik uporabniškega vmesnika. Ostale nastavitve uporabnika lahko spreminja samo skrbnik sistema.
Nadzorna plošča	Ob prijavi se uporabniku prikažejo osnovne informacije na nadzorni plošči. Vsebuje naj preprost iskalnik po črtnih kodah, možnost zaznamkov ter zadnje izvedene zahtevke. Podpirati mora tudi delovni proces *priprave* in *optičnega branja*.

Šifranti *enote, oddelki, način prevzema, optično bralne naprave, lokacije, lokacije uničenja*	Napredni uporabnik ima vse ustvari-beri-posodobi-izbriši (ang. create read update delete - CRUD) pravice nad naštetimi šifranti. Uporabnik lahko šifro izbriše samo v primeru, če ta še ni bila uporabljena.
Šifranta *projekti* in *podjetja*	Osnovnemu uporabniku je na voljo samo pogled projektov in podjetij, katere lahko filtrira in poljubno išče. Napredni uporabnik ima vse CRUD pravice. V kolikor je šifra že nekje uporabljena je aplikacija ne sme dovoliti izbrisati.
Aplikacijski programski vmesnik (ang. application programming interface - API)	Aplikacija mora nuditi API, preko katerega lahko neodvisna aplikacija za OCR prepoznavo TIS [21] izvaja klice preko arhitekture za izmenjavo podatkov med spletnimi storitvami (ang. representational state transfer - REST) in zapisuje podatke v podatkovno bazo. Do vmesnika API lahko dostopa samo skrbnik sistema.

	Specifikacija služi kot pogodba med naročnikom in izvajalcem del in je neke vrste vrhovni dokument v aplikaciji. Aplikacija mora omogočati: • CRUD operacije nad specifikacijami • možnost tiskanja • nalaganje dokumentov kot priponke
Delo s specifikacijami	Osnovni uporabnik ima možnost vpogleda v vse specifikacije v aplikaciji. Prav tako lahko specifikacijo odpre in jo natisne na podlagi podane predloge. Napredni uporabnik pa lahko specifikacijo ureja ali pa ustvari novo. Specifikacijo se lahko briše samo v primeru, če nanjo ni bil vezan še noben prevzemni zapisnik. Specifikacijo naj bo možno tudi uvoziti iz dokumentnega sistema, ki ga uporablja podjetje Mikrografija.

Zapisniki	Aplikacija naj nudi možnost dela z sledečimi zapisniki: • zapisnik o prevzemu gradiva • zapisnik o vračilu gradiva • zapisnik o uničenju • zapisnik o dovoljenju za uničenje gradiva Osnovni uporabnik ima možnost pregleda zapisnikov in tiskanja. Napredni uporabnik pa jih lahko tudi ustvarja ali briše. Za uspešno ustvarjen zapisnik je nujno potrebno navesti *tip zapisnika* ter navesti *specifikacijo*, na katero se zapisnik navezuje. Prav tako lahko vsakemu zapisniku dodamo priponke ali pa ga natisnemo. Zapisnik naj omogoča tudi izbiro nadrejenega zapisnika.

Zapisnik o prevzemu gradiva	Pogled zapisnika o prevzemu gradiva, naj poleg prikaza podatkov zapisnika in priponk omogoča tudi prikaz procesov: • prevzema gradiva • popisa gradiva • priprave gradiva • optičnega branja • končne kontrole Aplikacija naj na ustrezen način prikaže vse enote gradiva, ki pripadajo določenemu procesu.
Zapisnik o vračilu gradiva	Poleg pregleda podatkov zapisnika naj pogled vsebuje enote gradiva, ki se bodo vračale naročniku ter referenco na *zapisnik o prevzemu*, kjer so se enote obdelovale.
Zapisnik o uničenju	Poleg pregleda podatkov zapisnika naj pogled vsebuje enote gradiva, ki se bodo uničile, dodatne metapodatke o gradivu ter referenco na *zapisnik o dovoljenju za uničenje*.
Zapisnik o dovoljenju za uničenje	Poleg pregleda podatkov zapisnika naj pogled vsebuje enote gradiva, ki potrebujejo dovoljenje za uničenje ter referenco na *zapisnik o prevzemu*, kjer so se enote obdelovale.

Prevzem gradiva (vnašanje enot)	Osnovni uporabnik ima samo možnost pregleda prevzemov gradiva. Napredni uporabnik lahko izvaja vse operacije nad prevzemi gradiva. Prevzem gradiva se generira na podlagi zapisnika o prevzemu gradiva in le ta je obvezen. Velja pravilo, da ima eden zapisnik o prevzemu natanko en prevzem gradiva. Enote, ki so lahko različnega tipa se lahko vnašajo ročno ali pa se uvozijo iz *excel* datoteke. Vsaka enota mora imeti unikatno črtno kodo ne glede na tip enote. Pri vnašanju enot gradiva se vnese *tip enote* ter *črtno kodo*, aplikacija pa mora poskrbeti, da uporabnik ne more vnesti črtne kode, ki že obstaja v sistemu.
Popis gradiva (vnašanje podenot)	Osnovni uporabnik ima možnost pregleda popisov gradiva. Prav tako lahko popis odpre in išče podatke po vseh enotah gradiva, ki se navezujejo na ta popis. Napredni uporabnik lahko izvaja vse operacije nad popisom. Popis se lahko kreira samo na podlagi prevzema gradiva. Torej, če prevzem gradiva ni bil kreiran, tudi popisa ne moremo narediti. Napredni uporabnik lahko vnaša podenote gradiva, ki vsebujejo *črtno kodo, črtno kodo podenote* ter ostale atribute. Vnese lahko poljubno število *črtnih kod podenot* (vsaka mora biti unikatna). Če vnešena črtna koda enote ne obstaja v prevzemu, mora aplikacija o tem obvestiti uporabnika in mu preprečiti vnos take črtne kode enote.

Končna kontrola (kontroliranje optičnega branja)	Pri končni kontroli naj imajo uporabniki možnost pregleda vseh ustvarjenih končnih kontrol in urejanja le teh. Uporabnik lahko končno kontrolo ustvari na pogledu zapisnika ali pa na prevzemu, kjer lahko izbere enoto ali več enot, ki jih bo prekontroliral. Uporabnik naj ima pri vnosu končne kontrole možnost izbrati ali so izbrane škatle brez napak in če je kontrola končana. V primeru napake, lahko uporabnik vnese tudi napake ter število napak poleg vsake enote.
Fizična hramba gradiva	Uporabnik naj ima možnost pregleda vseh specifikacij, ki so v aplikaciji in imajo izbrano fizično hrambo. Prikazano naj bo tudi število vseh enot in število enot za uničenje, ki spadajo pod specifikacijo. Ob kliku na specifikacijo naj se uporabniku odpre pogled z vsemi enotami ter podenotami te specifikacije. Na jasen način naj bodo prikazane enote, ki so za uničenje. Na podlagi enot, ki so za uničenje naj aplikacija uporabniku omogoča izbiro enot ter kreiranje zapisnika o dovoljenju za uničenje z izbranimi enotami.
Reverzi	Do reverzov lahko dostopajo samo napredni uporabniki. Uporabnik ima možnost pregleda in urejanja vseh reverzov. Lahko ustvari prazen reverz brez enot gradiva (enote se ročno dopišejo) ali pa ustvari reverz neposredno iz prevzema gradiva, kjer lahko izbere enote, ki naj bodo vključene v reverz.

Pregled podatkov iz programa za OCR prepoznavo	Aplikacija naj omogoča sledenje enotam, ki so se obdelale v programu za optično prepozavanje znakov. Osnovni uporabnik naj ima možnost pregleda ter filtriranja enot.
Časovne sledi	Aplikacija, naj v ozadju izvaja shranjevanje dogodkov, ki spremenijo status enote gradiva. Beleži naj tudi čas, ki je potekel med spremembo statusa enote gradiva.
Beleženje časa (Delovni proces)	Aplikacija mora uporabnikom na enostaven način omogočiti upravljanje z delovnim procesom. Uporabniki, ki pripravljajo gradivo, lahko v aplikaciji vnesejo črtno kodo enote in pritisnejo gumb za začetek. Tekom priprave lahko dajo enoto na pavzo ali pa jo zaključijo. Sistem mora samodejno shraniti časovne sledi. Uporabniki optičnega branja lahko prav tako vnesejo črtno kodo in pred začetkom optičnega branja v aplikaciji pritisnejo gumb za začetek. Tekom optičnega branja lahko škatlo pavzirajo ali pa jo zaključijo. Če enota še ni bila zaključena na pripravi, aplikacija ne sme dovoliti optično bralnega procesa. Pri ročnem optičnem branju lahko uporabnik vnese *porabljen čas, št. posnetkov* ter *tip optičnega bralnika*.

Poročila in statistika	Do pogledov o poročilih in statistiki lahko dostopajo samo napredni uporabniki. Aplikacija naj na podlagi zapisov v podatkovni bazi kreira statistiko in nudi pregled poročil: • prevzemi gradiva - število enot gradiva po letih • projekti - statistika optičnega branja glede na projekt • fizična hramba - statistika gradiva, ki se hrani v fizični hrambi • časovne sledi - pregled časovnih sledi
Administracija aplikacije	Do administrativnega dela aplikacije lahko dostopa samo skrbnik aplikacije. Skrbniški del aplikacije mora omogočati vse CRUD operacije sledečih funkcij: • poslovne enote - možnost urejanja poslovnih enot, ki pripadajo podjetju • uporabniki - možnost spreminjanja nastavitev in podatkov uporabnikov • pravice dostopa - možnost dodajanja različnih pravic uporabnikom za dostop do aplikacije

Poglavje 3

Tehnična zasnova spletne aplikacije

3.1 Uporabljene tehnologije ter orodja

Na podlagi definiranih zahtevanih funkcionalnosti v podpoglavju 2.4 smo lahko izbrali tehnologije in orodja, v katerih smo razvijali programsko rešitev. Spletne tehnologije ter okolje, v katerem smo razvijali rešitev, bi lahko izbrali že na začetku, vendar smo se odločili, da najprej definiramo funkcionalnosti in na podlagi le teh začnemo izbirati tehnologije, s katerimi si bomo pomagali pri razvoju aplikacije. S tem smo se poskušali izogniti, da nas tekom razvoja kaj ne preseneti in, da se posledično česa ne bi dalo razviti, zaradi pomanjkanja funkcij izbrane tehnologije. V prvem delu tega podpoglavja naštejemo ter na kratko opišemo vse tehnologije, ki smo jih uporabili v aplikaciji tekom razvoja. V drugem delu tega podpoglavja pa opišemo orodja, katera so nam bila v pomoč tekom razvoja aplikacije.

3.1.1 Tehnologije

Za razvoj aplikacije smo uporabili razne tehnologije, ki omogočajo prikaz podatkov preko spleta kot tudi samo običajno hranjenje podatkov v podatkovnih bazah. V samem začetku načrtovanja še pred razvojem smo določili

oziroma izbrali glavne tehonogije, ki jih bomo uporabljali tekom razovja aplikacije. Odločili smo se, da bomo našo aplikacijo razvijali z eno izmed spletnih tehnologij. Taka odločitev je bila najbolj smiselna, saj bo aplikacija dostopna na vsakem računalniku, ki ima dostop do interneta in ima nameščen spletni brskalnik, ob tem pa ni potrebno skrbeti, katero verzijo aplikacije ima uporabnik, saj bo le ta vedno dostopal do najnovejše. Pri izbiri tehnologij smo izbrali tiste, ki so najbolj ustrezale našim kriterijem:

- koliko zaposlenih pozna tehnologijo
- krivulja učenja oziroma zahtevnost
- kako velika skupnost podpira tehnologijo
- težavnost vzdrževanja in nadgradenj
- pretekle izkušnje
- ali je kos vsem zahtevam/funkcionalnostim aplikacije

PHP

PHP [17] je skriptni jezik, ki se izvaja oziroma prevede na strežniku, nato pa je vsebina servirana odjemalcu. Akronim PHP je okrajšava za skriptni programski jezik za izdelavo spletnih strani (ang. Hypertext Preprocessor). PHP koda je lahko vgrajena v HTML ali pa je lahko uporabljena v kombinaciji z različnimi spletnimi sistemi za predloge. Koda je običajno procesirana z PHP interpreterjem, ki je implementiran kot modul na strežniku. Torej strežnik s pomočjo interpeterja sprocesira php kodo, ki lahko vsebuje tudi slike in jih na to servira na spletu. PHP je zelo popularen in deluje praktično na vsaki platformi in to zastonj. Pri izdelavi aplikacije smo uporabili PHP verzijo 7.1, saj prinaša veliko prednosti pred starejšimi verzijami. Glavna prednost najnovejše verzije je hitrost procesiranja skript in boljše upravljanje s pomnilnikom.

MySQL

MySQL [15] je odprtokodna implementacija relacijske podatkovne baze, ki za delovanje oziroma za delo s podatki uporablja strukturirani povpraševalni jezik SQL (ang. structured query language). Za dostop do podatkovne baze MySQL se uporabljajo razni odjemalci, ki se povežejo na strežnik. Torej MySQL deluje kot odjemalec-strežnik, pri čemer je lahko strežnik nameščen kot sistem, porazdeljen na več strežnikov, kar omogča boljšo odzivnost in hitrost izvedbe ukazov. Napisan je v C in C++ in deluje na različnih operacijskih sistemih, zaradi česar je MySQL eden najbol razširjenih sistemov za upravljanje podatkovnih baz. Obstaja več verzij podatkovnih baz MySQL. Odločili smo se, da uporabimo MariaDB [14], ki je ločena veja (ang. fork) razvoja podatkovne baze MySQL. Razvija jo skupnost ter nekaj prvotnih razvijalcev MySQL, ki so odšli zaradi skrbi, ko je Oracle prevzel MySQL. MariaDB je povsem odprtokodna in zastonjska implementacija baze MySQL, prav tako pa pride že v paketu z XAMPP.

Yii 2

Yii 2 [25] ali na kratko yii je moderno visoko preformančno odprtokodno PHP ogrodje. Namenjeno je za razvoj spletnih aplikacij in aplikacijskih programskih vmesnikov API (ang. application programming interface). Ogrodje yii omogoča programerjem razvoj kompleksnih aplikacij. Promovira DRY (ang. don't repeat yourself) vzorec in spodbuja k hitremu razvoju, saj omogča izjemno učinkovito, razširljivo in trajnostno razvijanje spletnih aplikacij. Yii slovi po tem, da je zelo hitro ogrodje, saj je optimizirano za hitrost in je namenjeno za vse tipe aplikacij. Ogrodje je bilo prav tako zgrajeno za poslovne aplikacije. V paketu pride ogordje z ogromno orodji, ki omogočajo hiter razvoj. Lastnosti, ki jih podpira, MVC (ang. model-view-controller), DAO (ang. data access object), validacija vnosnih polj, asinhroni JavaScript in xml AJAX (ang. asynchronous JavaScript and XMLHttpRequest), avtentikacija in avtorizacija, različne teme, spletne storitve, internacionalizacija, varnost, poročanje napak, avtomatsko generiranje kode in še mnogo več.

Glavna prenost tega ogrodja je, da ima velik poudarek na počasnem nalaganju (ang. lazy loading). Na primer datoteka z razredom se naloži šele takrat, ko je ta razred uporabljen; objekt je kreiran šele ob prvem klicu objekta. Večina drugih kompetentnih ogrodij ne omogoča te funkcionalnosti (npr. razred za dostop do podatkovne baze je kreiran v vsakem primeru, če je le ta uporabljen v zahtevi ali ne), zaradi česar niso tako hitra. Ker tudi večina razvijalcev v podjetju pozna ogrodje Yii oziroma se je z njim že srečala, smo se odločili za uporabo le tega.

HTML — CSS — JavaScript

Jezik za označevanje nadbesedila HTML (ang. hyper text markup language) je označevalni jezik za izdelavo spletnih strani [9]. Predstavlja osnovo spletnega dokumenta. Poleg prikaza dokumenta v spletnem brskalniku se z njim hkrati določi tudi zgradba in semantični pomen delov dokumenta. HTML dokument je datoteka, katero zna spletni brskalnik odpreti in prevesti v berljivo obliko. Za urejanje HTML dokumenta lahko uporabljamo čisto preprost urejevalnik teksta. Osnovni elementi pisanja so značke, ki so lahko enojne (<tag> značke, ki ne rabijo zaključka) ali dvojne (<tag>< /tag> značke, ki potrebujejo zaključek). HTML dokument vsebuje štiri osnovne značke: HTML, HEAD, TITLE ter BODY.

Kaskadne stilske podloge CSS (ang. cascading style sheets) so preprost slogovni jezik, s katerim definiramo stil HTML elementov. S tem jezikom opišemo, kako naj brskalnik prikaže neko značko. Določamo lahko barve, poravnave, odmike, pozicijo, obrobe, senčenje in še ogromno drugih atributov. Prav tako lahko določamo animacije elementov, ki se zgodijo, ko uporabnik izvaja različne akcije nad elementi [7].

JavaScript je visoko nivojski šibko tipitiziran programski jezik. JavaScript je poleg HTML in CSS jezik, ki ga zna brskalnik interpretirati. Na spletnih straneh omogoča interaktivnost in dinamične spremembe, kot so nalaganje novih vsebin, posodobitve, animacije. Vse te spremembe se izvedejo, ne da bi se stran ponovno naložila. Vsaka moderna stran uporablja JavaScript,

uporabljamo ga tudi mi. S pomočjo dodatnih JavaScript knjižnic jQuery [12], Select2 [20], Bootstrap [5] smo poskušali ustvariti čimbolj prijetno uporabniško izkušnjo.

3.1.2 Orodja

Tekom razvoja spletne aplikacije smo uporabljali razna orodja, ki so nam pomagala pri realizaciji rešitve oziroma pri pisanju programske kode.

IntelliJ IDEA

IntelliJ IDEA je integrirano razvojno okolje IDE (ang. integrated development environment) razvito s strani podjetja JetBrains [10]. IDE je okolje, ki maksimizira programerjevo produktivnost, saj vsebuje veliko komponent, ki olajšajo programiranje. S pomočjo raznih vtičnikov, ki jih nasnamemo, lahko omogočimo programiranje v jeziku PHP. Nudi avtomatsko dokončevanje kode, prestrukturiranje kode, zaznavanje napak v realnem času, prevajalnik, razhroščevalnik, orodje za potisk kode na produkcijo in še veliko več. Prav tako nudi podporo in orodja za razvoj spletnih vsebin kot so HTML, CSS in JavaScript.

MySQL Workbench

MySQL Workbench je enotno vizualno orodje za načrtovalce in skrbnike podatkovnih baz [16]. Omogoča načrtovanje podatkovne baze, načrtovanje SQL stavkov in vsa administrativna orodja za delo in konfiguracijo baze. Prav tako je na voljo na vseh platformah in je na voljo zastonj. Orodje je zelo močno, predvsem pa je uporabno pri izdelavi podatkovnega modela in administraciji uporabnikov in konfiguraciji baze.

XAMPP

XAMPP je najbolj popularno odprtokodno PHP razvijalsko okolje [22]. Kratica je setavljena iz Apache + MariaDB + PHP + Perl. Paket XAMPP

omogoča zelo enostavno namestitev spletnega strežnika, ki v nekaj minutah nudi razvojno okolje. Poleg paketa pride tudi prednameščena razširitev phpMyAdmin [18], ki omogča enostavno upravljanje podatkovne baze kar v spletnem brskalniku.

Apache je robustna ter odprtokodna implementacija HTTP stržnika ali z drugimi besedami spletnega strežnika, ki servira http/https zahteve odjemalcem [2]. Apache je delo ogromno ljudi po celem svetu in je danes najbolj razširjen spletni strežnik. Na voljo je v različnih platformah.

Composer

Composer je orodje za urejanje PHP knjižnic, ki se jih uporablja v projektu [6]. Napisan je v jeziku PHP in se samostojno namesti na sistemu. Omogoča, da zelo enostavno dodajamo, urejamo ali zbrišemo knjižnice, od katerih je naš projekt odvisen.

Gii

Gii je razširitev za ogrodje Yii 2, ki omogoče generiranje kode modelov, pogledov, modulov, razširitev preko spletnega vmesnika [8]. Orodje smo uporabljali predvsem za generiranje modelov iz podatkovne baze in generiranje pogledov spletne aplikacije.

Xdebug

Razširitev za PHP, ki nam omogoča razhroščevanje in nudi pomoč pri razvijanju programske kode [23]. Lahko ga vključimo v IDE in tako korak po korak razhroščujemo kodo, ki teče na strežniku.

3.2 Podatkovni model

Pred vsakim razvojem aplikacije, ki vsebuje podatkovno bazo, je ključnega pomena narediti dober podatkovni model. Kakovost načrtovanega podatkov-

nega modela se nato odraža tekom razvoja celotne aplikacije. V nadaljevanju predstavljam podatkovni model, ki smo ga uporabili v spletni aplikaciji. Na začetku pred razvojem smo definirali tabele in razmerja med njimi, ker pa je razvoj aplikacije potekal po agilnem postopku razvoja programske opreme in se je rešitev rezvijala v iteracijah, je tekom razvoja prihajalo do sprememb v podatkovnem modelu. Največkrat je bilo potrebno dodati dodatne atribute ali pa dodatne tabele, ki so omogočale razvoj dodatnih funkcionalnosti. Zaradi tega predstavljam podatkovni model, ki se uporablja v trenutni verziji spletne aplikacije, ob pisanju diplomske naloge. Na sliki 3.1 ali 3.2 je viden podatkovni model z vsemi tabelami:

- *user* - tabela, ki predstavlja uporabnike v aplikaciji. Vsebuje vse potrebne atribute, ki so potrebni za uporabniški profil ter vpis v aplikacijo. Vsak zapis v tabeli predstavja natanko enega uporabnika v aplikaciji. V tabeli so shranjena gesla v *hash* obliki, za kar poskrbi orgordje Yii 2. Pomemben je tudi atribut *ldap*, ki aplikaciji pove, ali se uporabnik vpisuje preko protokola ldap ali pa preko gesla, ki je zapisano v tabeli *user*

- *account* - tabela, ki predstavlja naslove in kontakte naročnikov ali partnerjev. Vsebuje atribute, s katerimi so predstavljeni naročniki. Vsak zapis v tabeli predstavlja enega naročnika.

- *project* - tabela, kjer vsak zapis predstavlja natanko en projekt. Za nekega naročnika *account* lahko obstaja več projektov

- *specification* - tabela, ki predstavlja specifikacijo in vsebuje preko 300 atributov. Vsak atribut predstavlja določen paramater specifikacije. Vsak zapis v tabeli predstavlja natanko eno specifikacijo v realnem svetu. Če projekt *project* zahteva različne načine ravnanja z gradivom, lahko za vsak način ravnanja z gradivom za ta projekt ustvarimo posebno specifikacijo. Količina gradiva ne vpliva na število specifikacij. Torej ena specifikacija zadostuje za vso gradivo, kjer se gradivo obdeluje na enak način

- *collectionrecord* - tabela, ki predstavlja zapisnike. Vsak zapis v tabeli predstavlja natanko en zapisnik. Vrsta zapisnika je določena z atributom *type*. Za določeno specifikacijo *specification* lahko ustvarjamo poljubno število zapisnikov

- *attachment* - tabela, ki je povezana na *collectionrecord, specification* predstavlja priponke v aplikaciji, katere lahko naložimo oziroma pripnemo poleg zapisnikov ali specifikacije. Tabela vsebuje le atribute, ki opisujejo priponko. Tako je najpomembnejši atribut *name*, preko katerega lahko razberemo ime priponke, ki je dejansko shranjena na trdem disku in ne v podatkovni bazi. Zaradi tega, lahko aplikacija tudi ob velikem številu priponk še vedno omogoča hiter prenos. Če bi priponke hranili v podatkovni bazi v podatkovnem polju *blob*, bi s časoma podatkovna baza postajala vse večja, naša aplikacija pa vse počasnejša in počasnejša.

- *acquisition, aquisitiondetail* - tabeli, ki sta v medsebojnem razmerju 1:n, predstavljata prevzem gradiva. Vsak zapis v tabeli *acquisition* predstavlja natanko en prevzem gradiva, medtem ko vsak zapis v tabeli *aquisitiondetail* predstavlja natanko eno enoto gradiva. Torej en prevzem gradiva ima lahko več enot gradiva. Tabeli *collectionrecord* in *acquisition* sta v razmerju 1:1. Torej en zapisnik o prevzemu *collectionrecord* ima lahko natanko en prevzem gradiva *acquisition*

- *inventory, inventorydetail* - tabeli, ki sta v medsebojnem razmerju 1:n, predstavljata popis gradiva. Vsak zapis v tabeli *inventory* predstavlja natanko en popis in je v razmerju z *acquisition* 1:1. Torej en prevzem gradiva ima lahko natanko en popis gradiva. Eden zapis v *inventorydetail* predstavlja natanko eno podenoto gradiva. Za vsako enoto gradiva *aquisitiondetail* obstaja 1:n podenot gradiva *inventorydetail*

- *checking, checkingdetail* - tabeli v razmerju 1:n, ki sta namenjeni končni kontroli posnetkov. Vsak zapis v tabeli *checking* predstavlja natanko

eno kontrolo, medtem ko vsak zapis v *checkingdetail* predstavlja enoto gradiva, ki se je kontrolirala. Torej ob kontroli se lahko kontrolira eno ali več enot gradiva

- *storage, storagedetail* - tabeli, ki sta v medsebojnenm razmerju 1:n, predstavljata fizično hrambo gradiva. Vsak zapis v tabeli *storage* predstavlja enoto gradiva, medtem ko vsak zapis v tabeli *storagedetail* predstavlja podenoto gradiva. Pomembni atributi so predvsem *storagelocation_id, line, regal, shelf* v tabeli *storage*, ki se uporabljajo za shranjevanje lokacije hrambe enote gradiva

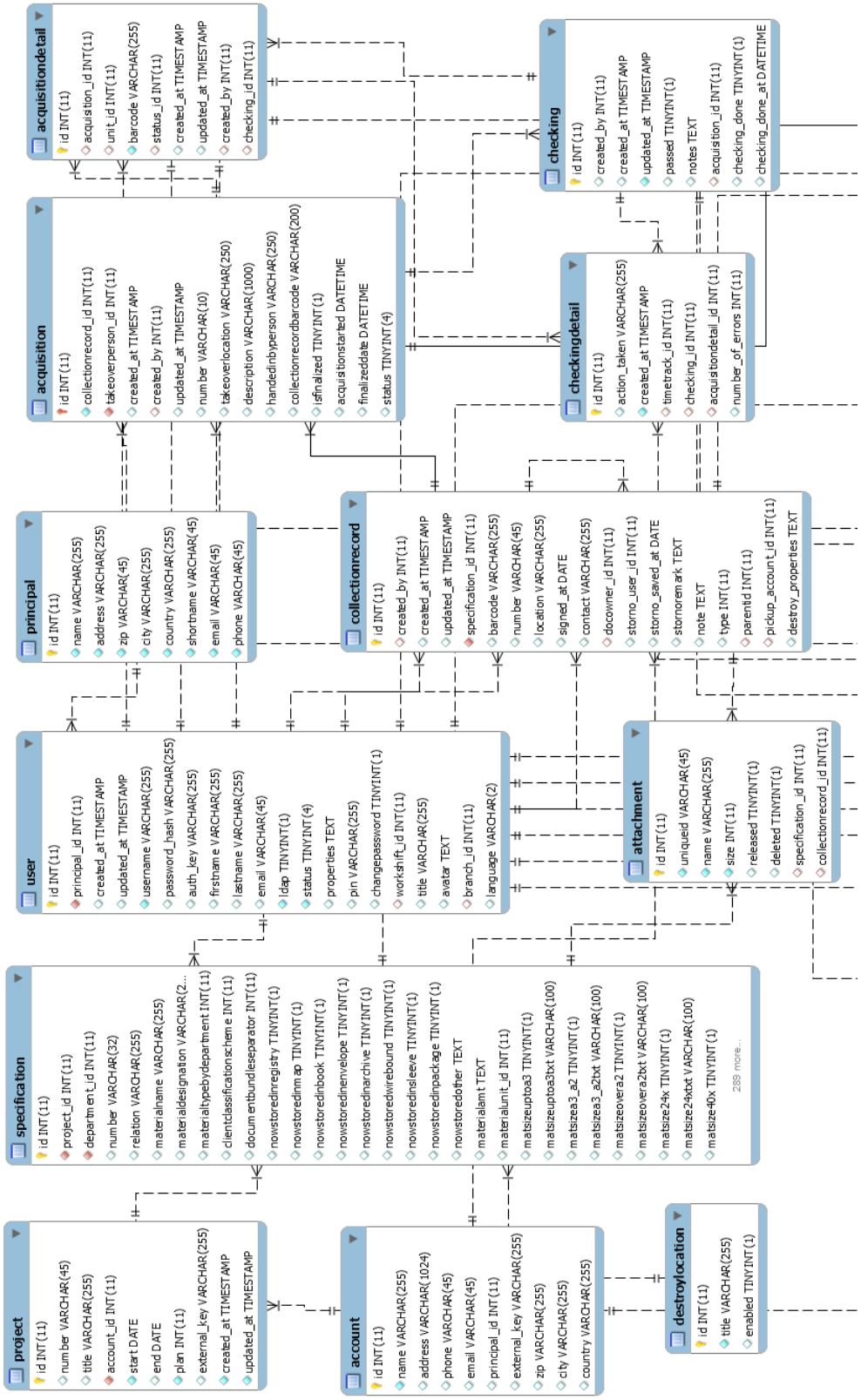
- *reversal, reversaldeatil* - tabeli, ki sta v medsebojnenm razmerju 1:n, predstavljata reverze. Vsak zapis v tabeli *reversal* predstavlja en reverz, medtem ko vsak zapis v *reversaldetail* predstavlja enote prevzema, ki so bile vrnjene

- *timetrack* - tabela, ki predstavlja zapise časovnih sledi. Vsaka vrstica predstavlja natanko en zapis časovne sledi, ki se zgodi vsakič, ko uporabnik začne oziroma konča neko delovno operacijo (pripravo, optično branje, končno kontrolo). Najpomembnejši atribut je *ticks*, v katerem je zapisan pretečeni čas v sekundah od začetka pa do konca neke delovne operacije

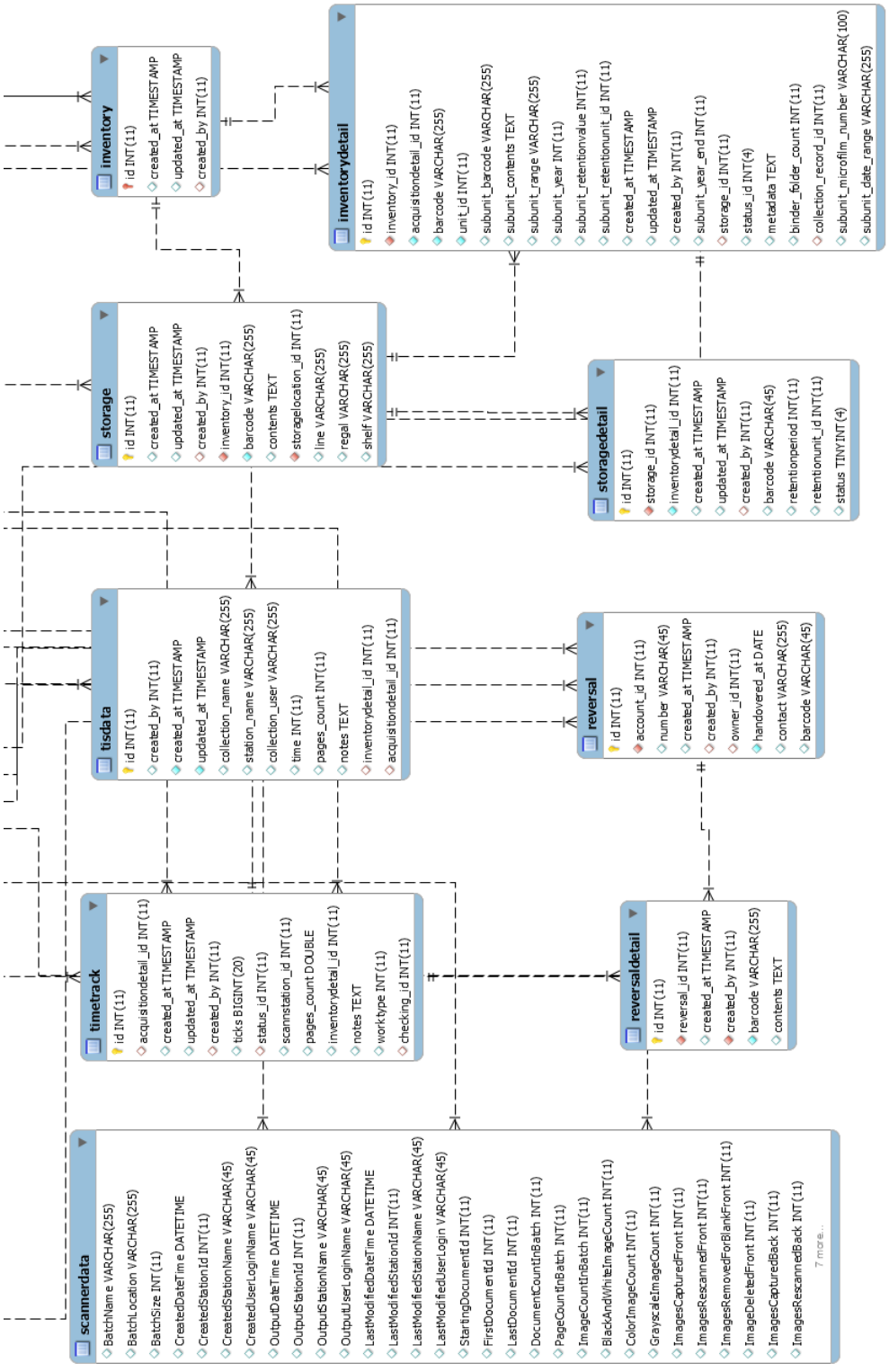
- *tisdata* - tabela, ki predstavlja zapise podatkov iz programa za OCR prepoznavo posnetkov. Vsak zapis v tabeli predstavlja podatke ene podenote gradiva, ki jih program zapiše tekom OCR obdelave gradiva

- *scannerdata* - tabela predstavlja zapise iz optičnih bralnikov. Vsak zapis v tabeli, predstavlja podatke za eno podenoto gradiva, ki so se generirali ob zaključku optičnega branja oziroma ob izvozu posnetkov iz aplikacije za optično branje posnetkov. Atributi predstavljajo vsa polja, ki jih generira aplikacija za optično branje posnetkov ob izvozu optično prebranih posnetkov. Najpomembnešji atribut je *ImageCountInBatch* iz katerega razberemo, koliko posnetkov vsebuje podenota gradiva.

- *destroylocation* - tabela, kjer vsak zapis v tabeli predstavlja lokacijo, kjer se uničuje gradivo
- *principal* - tabela, kjer vsak zapis v tabeli predstavlja osnovne atribute lastnika aplikacije. Torej v našem primeru je v tabeli samo en zapis, kjer so v atributih zapisani podatki o Mikrografiji



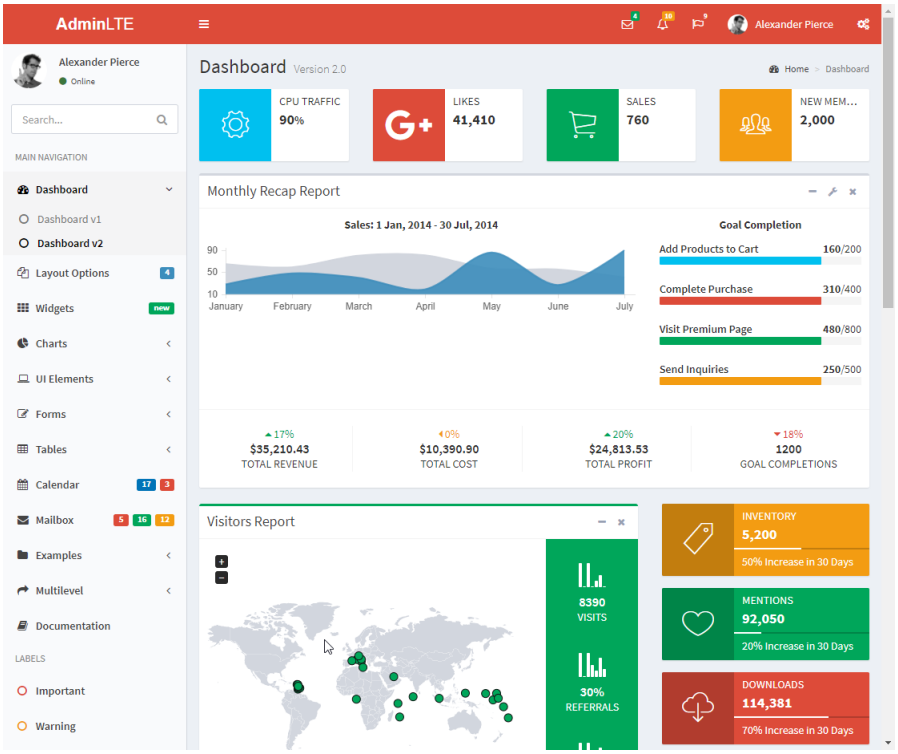
Slika 3.1: Prvi del podatkovnega modela uporabljenega v aplikaciji



Slika 3.2: Drugi del podatkovnega modela uporabljenega v aplikaciji

3.3 Predloga spletnega vmesnika

Velika večina sodobnih spletnih aplikacij, ki so dosegljive na spletu, uporablja jezike HTML, CSS, JavaScript. Čeprav se bo aplikacija uporabljala na namiznih računalnikih, je bila želja vodij, da se aplikacijo lahko prilagaja velikosti zaslona. Tako lahko uporabnik aplikacijo uporablja na namiznem računalniku ali pa na mobilni napravi. Zaradi tega smo se odločili, da uporabimo knjižnjico Bootstrap [4]. V času začetka razvoja aplikacije je bila na voljo verzija v4 knjižnice Bootsrap, vendar je bila ta še v stopnji *alpha*, zato smo se raje izognili nevšečnostim, ki jih lahko prinese uporaba alpha ali beta knjižnic in smo zato izbrali verzijo v3. Odločili smo se, da uporabimo temo AdminLTE 2 [1]. To je odprtokodna tema, ki za osnovo uporablja Bootstrap, določene stilske predloge pa so spremenjene, s čimer je dosežen izgled kot je na sliki 3.3. Poleg knjižnice Bootstrap ima tema integriranih še kar nekaj ostalih knjižnic, katere so nam služile kot zgled pri izdelavi naše spletne aplikacije. Ponuja pa tudi možnost izbire barvne teme. Za aplikacijo Sledenje smo uporabili barvno temo svetla rdeča (ang. red light), ki se ujema z barvami podjetja Mikrografija.



Slika 3.3: Primer spletne predloge AdminLTE 2

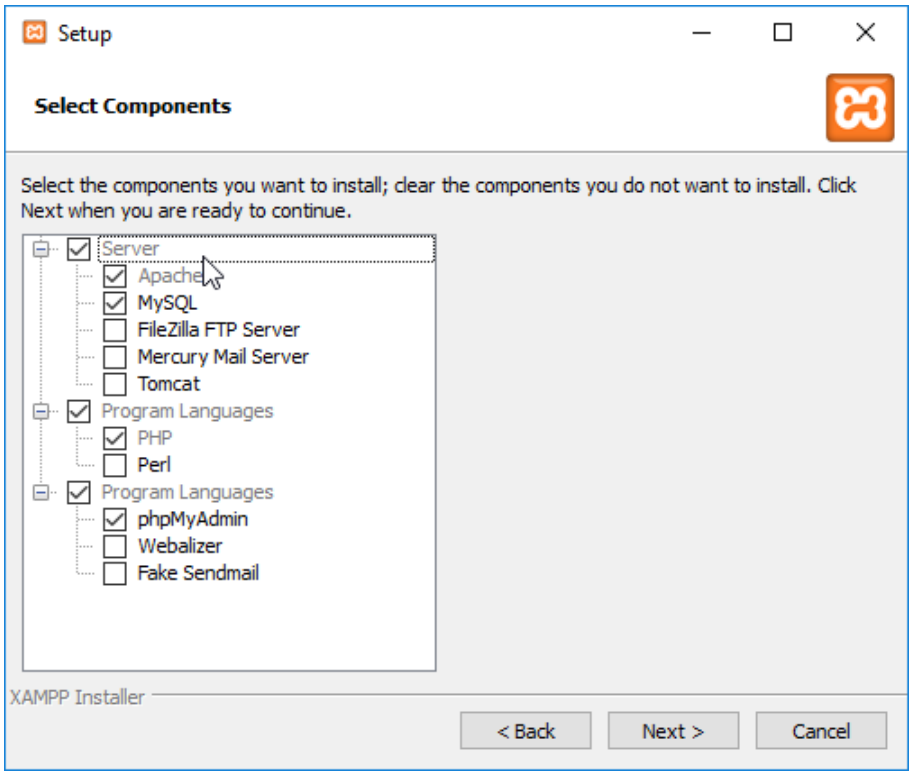
3.4 Postavitev razvojnega okolja

Pred začetkom razvoja spletne aplikacije smo najprej postavili razvojno okolje. Za pisanje kode smo uporabili IntelljiJ IDEA, ki smo ga za potrebe našega projekta nadgradili z sledečimi vtičniki:

- PHP - omogoča urejanje in razhroščevanje php kode
- PHP Annotations - nudi podporo anotacijam nad funkcijami in razredi
- PHP Toolbox - boljša navigacija ina samodokončevanje kode
- Yii2 Support - podpora za Yii2 [25]
- Yii::t - podpora za i18n

- Bootstrap 3 - predloge ter samodokončevanje kode za Bootstrap

Vsi našteti vtičniki so dosegljivi na uradni strani IntelliJ IDEA Plugins [11]. Za potrebe vključevanja PHP knjižnic v projekt in tudi za inicializacijo projekta smo namestili Composer. Za poganjanje PHP kode smo uporabili XAMPP, ob namestitvi pa smo za naše potrebe namestili samo Apache, MySQL, PHP ter phpMyAdmin kot je razvidno na spodnji sliki 3.4



Slika 3.4: Namestitev XAMMP

Po končani namestitvi, smo v nadzorni plošči XAMPP namestili Apache ter MySQL kot storitev (ang. service). Tako je vsakič, ko znova zaženemo računalnik, aplikacija že dosegljiva na *localhost* in nam ni potrebno zaganjati strežnika Apache ali podatkovne baze saj že tečeta v ozadju. Za potrebe razvoja smo namestili razhroščevalnik Xdebug. V konzoli *cmd* zaženemo ukaz ***php -i***, da dobimo izpis php verzije in nastavitev na našem računalniku. Ta

izpis prilepimo na spletno stran [24] orodja Xdebug in dobimo natančna navodila, kako konfigurirati *php.ini* datoteko. Nastavitve, ki smo jih uporabili:

```
[XDebug]
zend_extension = C:\xampp\php\ext\php_xdebug-2.5.4-7.1-vc14.dll
;xdebug.remote_autostart = 1
;xdebug.profiler_append = 0
xdebug.profiler_enable = 0
xdebug.profiler_enable_trigger = 1
xdebug.profiler_output_dir = "c:\xampp\tmp"
xdebug.profiler_output_name = "cachegrind.out.%t-%s"
xdebug.remote_enable = 1
xdebug.remote_handler = "dbgp"
xdebug.remote_host = "localhost"
;xdebug.remote_log="c:\xampp\tmp\xdebug.txt"
xdebug.remote_port = 9000
;xdebug.trace_output_dir = "c:\xampp\tmp"
; 3600 (1 hour), 36000 = 10h
;xdebug.remote_cookie_expire_time = 36000
```

Za potrebe projekta smo v *php.ini* omogočili *short_open_tag=1*, ki omogča zapis PHP kode v formatu *<?= //koda ?>*. V konzoli *cmd* smo pognali ukaz *composer create-project –prefer-dist yiisoft/yii2-app-advanced sledenje*. Ta ukaz sporoči orodju composer, naj kreira projekt *sledenje* na podlagi predloge *yii2-app-advanced* [3]. V nastavitvah strežnika Apache *httpd.conf* smo dodali sledeče nastavitve, ki povejo strežniku, kje se nahaja naš projekt in na kakšen način naj ga servira:

```
<VirtualHost localhost.sledenje:80>
ServerName localhost.sledenje

DocumentRoot "C:/path_to_folder/sledenje/"
<Directory "C:/path_to_folder/sledenje/">
```

```
# use mod_rewrite for pretty URL support
RewriteEngine on
# If a directory or a file exists, use the request directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# Otherwise forward the request to index.php
RewriteRule . index.php

# use index.php as index file
DirectoryIndex index.php

# ...other settings...
# Apache 2.4
Require all granted
AllowOverride All

</Directory>
</VirtualHost>
```

Ustvarili smo novo podatkovno bazo *mikro_sledenje* in v splošnih nastavitvah projekta *sledenje/common/config/main-local.php* nastavili komponento

```
'db' => [
'class' => 'yii\db\Connection',
'dsn' => 'mysql:host=localhost;dbname=mikro_sledenje',
'username' => 'root',
'password' => '',
'charset' => 'utf8',
]
```

da aplikacija lahko dostopa do podatkovne baze. Pognali smo še ukaz *yii migrate*, s čimer smo pognali dve migraciji, ki ustvarita tabeli *migrations* ter *user*. V konzoli smo pognali še ukaz *composer require dmstr/yii2-adminlte-asset 2.3.4*, s čimer smo v projekt dodali temo AdminLTE 2, ki smo jo

izbrali tekom načrtovanja. Po končanem osnovnem konfiguriranju smo imeli pripravljeno predlogo, na podlagi katere smo lahko začeli razvijati spletno aplikacijo, ki je dosegljiva na naslovu *localhost.sledenje*.

Poglavje 4

Razvoj in predstavitev aplikacije

V prvem delu tega poglavja predstavimo ključne metode in načine, ki smo jih uporabili za razvoj spletne aplikacije. V drugem podpoglavju 4.2 tega poglavja pa predstavimo osnovno delovanje aplikacije, kjer si implementirane funkcionalnosti sledijo kronološko po času, kot poteka delo v produkciji.

4.1 Razvijanje funkcionalnosti aplikacije

Razvoj spletne aplikacije Sledenje je potekal tesno v sodelovnju z zaposlenimi v produkciji Mikrografije. Tako smo bili ves čas v stiku in smo jih lahko sproti spraševali za nasvete/ideje, hkrati pa prikazovali rezultate razvijanja. Kljub temu, da smo izbrali temo za lepši izgled aplikacije, je bil ves čas poudarek na funkcionalnostih in ne na izgledu.

Ko smo postavili razvojno okolje, je aplikacija že delovala vključno s prijavo in registracijo. Registracijo smo takoj odstranili, saj je aplikacija Sledenje zaprtega tipa in se uporabnik ne more registrirat samostojno. Poverilnice mu morajo biti dodeljene s strani nadrejenih. Na začetku smo s pomočjo zaposlenih določili polja za podatkovne tabele in jih nato generirali s pomočjo orodja MySQL Workbench [16]. Tekom razvoja aplikacije, predvsem v ka-

snejših fazah, podatkovnega modela nismo več spreminjali ročno, ampak smo zato uporabili migracijske skripte ogrodja Yii 2. Prednost popravljanja podatkovne baze z migracijsko skripto je predvsem ta, da ko prenesemo kodo v produkcijsko okolje, tam samo poženemo skripte, podatkovna baza pa se posodobi in postane ekvivalentna bazi v razvojnem okolju. Migracijo izvedemo s konzolnim ukazom *yii migrate/up*, če pa gre kaj narobe, lahko poženemo ukaz *yii migrate/down*, ki vrne podatkovno bazo v prvotno stanje, a le v primeru če smo pravilno spisali *function down()*. Primer kode, kjer smo z migracijo dodali v vsako tabelo po en atribut.

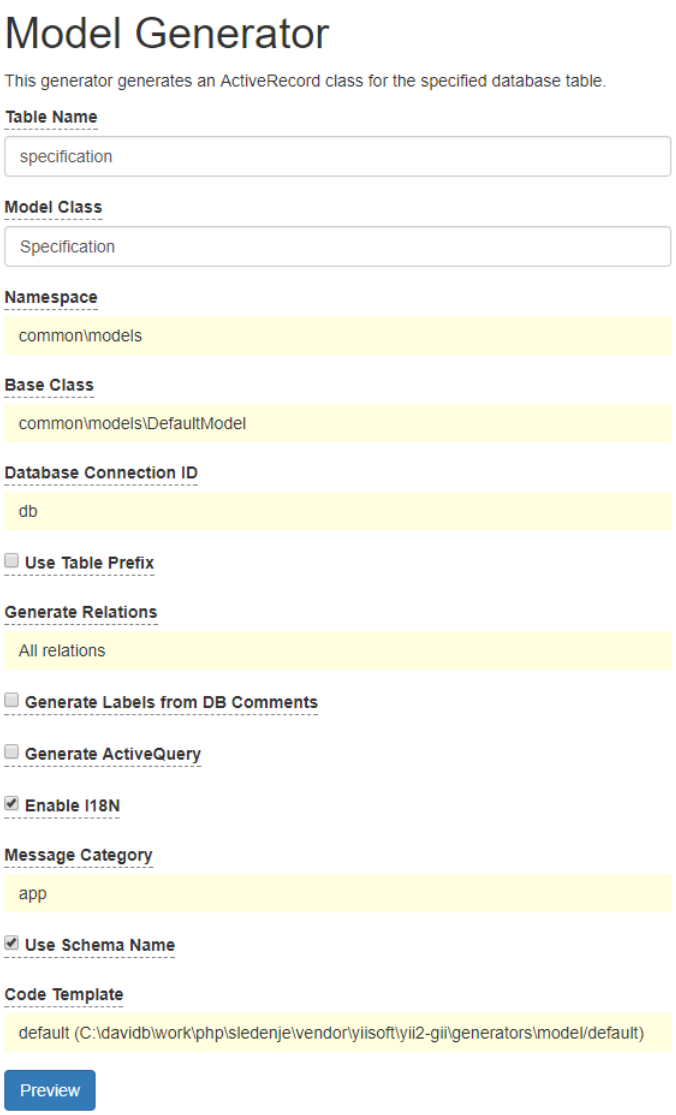
```
class m160504_135733_20160504_0 extends Migration
{
    public function up()
    {
        $this->addColumn(
            \common\models\User::tableName(),
            'title', $this->string(255));
        $this->addColumn(
            \common\models\Collectionrecord::tableName(),
            'destroy_properties', $this->text());
    }

    public function down()
    {
        $this->dropColumn(
            \common\models\User::tableName(), 'title');
        $this->dropColumn(
            \common\models\Collectionrecord::tableName(),
            'destroy_properties');
    }

}
```

Projekt, ki smo ga tekom postavitve razvojnega okolja generirali s pomočjo orodja composer, ima stukturo, prikazano na sliki 4.1. Prikazana struktura promovira princip DRY (ang. don't repeat yourself), saj nas spodbuja k temu, da že napisane kode ne ponavljamo. Pri razvoju spletne aplikacije smo uporabili vzorec MVC, ki je viden v mapi *frontend* na sliki 4.1.

```
common                  frontend
    config/                 assets/
    mail/                   config/
    models/                 controllers/
    tests/                  models/
console                     runtime/
    config/                 tests/
    controllers/            views/
    migrations/             web/
    models/                 widgets/
    runtime/            vendor/
                        environments/
```

Slika 4.1: Projektna struktura

Modele, ki smo jih generirali z orodjem Gii, smo shranili v *common/models*. To so modeli, ki so generirani avtomatično in jih ob spremembi podatkovnega modela lahko enostavno ponovno generiramo. Kot je prikazano na sliki 4.2, samo vnesemo ime tabele v bazi in na podlagi tega zna orodje generirati celoten podatkovni model. Prav tako nas vpraša, če želimo obstoječi model povoziti v primeru, če ta že obstaja. Težava pa nastane, če omenjene modele spreminjamo, saj jih ob ponovnem generiranju povozimo in ročno napisano kodo izgubimo. Zato smo modele razširili v *frontend/models*, v katere smo lahko pisali poljubno logiko in je nismo izgubili ob ponovnem generiranju modelov.

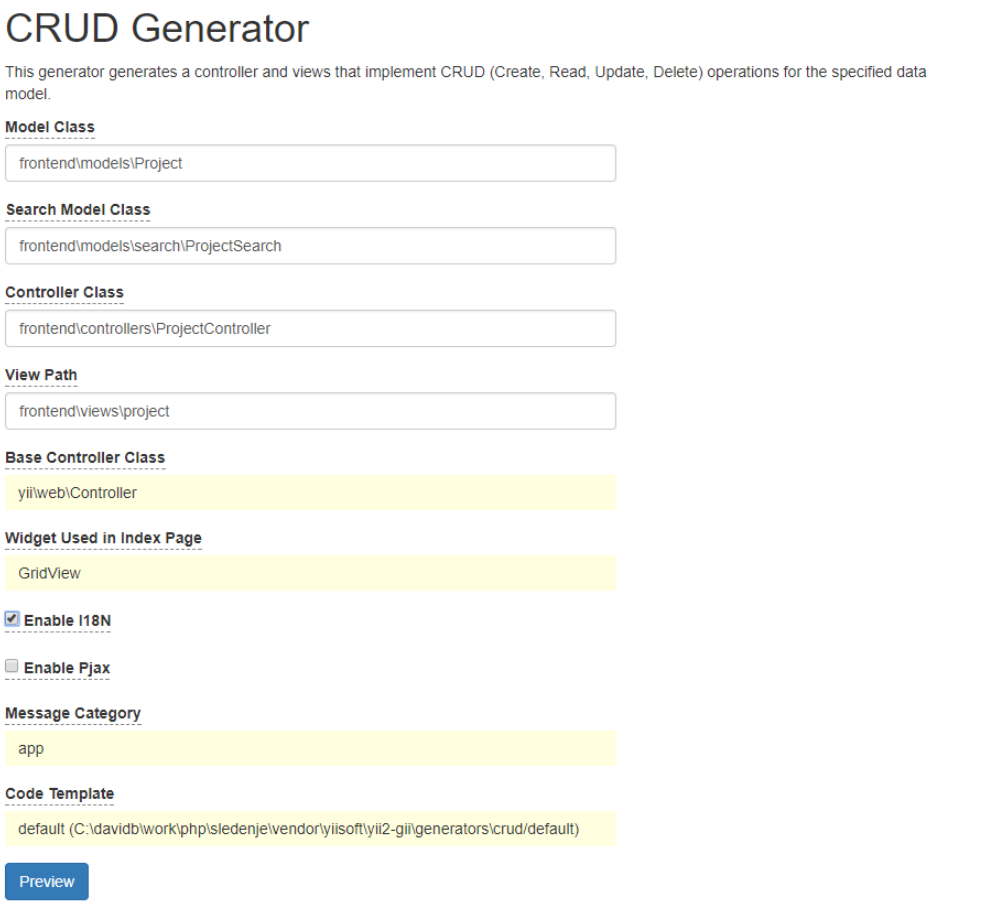


Slika 4.2: Generiranje modelov s pomočjo orodja gii

Po končanem generiranju modelov smo začeli razvijati funkcionalnosti. Ena izmed zahtev je bila, da do aplikacije lahko dostopajo samo prijavljeni uporabniki. To smo rešili tako, da na vstopni točki aplikacije preverjamo, ali je bila identiteta uporabnika preverjena s pomočjo vgrajenih filtrov ogrodja Yii 2. Spodaj je primer, kjer uporabljamo filter. Uporabniki, ki so avtenticirani (znak @), imajo dostop do vseh strani. Neprijavljeni uporabniki pa lahko dostopajo samo do strani *login, error*.

```
'as access' => [
    'class' => yii2mod\rbac\filters\AccessControl::className(),
   'rules' => [
        [
            'actions' => ['login', 'error'],
            'allow' => true,
        ],
        [
            'allow' => true,
            'roles' => ['@'],
        ]
    ],
],
```

Ponovno smo si pomagali z orodjem Gii. Generirali smo vse CRUD poglede za upravljanje s šifranti. Na sliki 4.3 je primer, kako smo za *projekte* generirali vse CRUD operacije.



Slika 4.3: Generiranje CRUD pogledov s pomočjo orodja gii

Zahteve, da lahko do določene vsebine dostopajo samo določeni uporabniki, smo rešili z uporabo knjižnice *yii2mod/yii2-rbac* [19]. Po končanju konfiguracije knjižnice lahko vsakemu uporabniku dodelimo avtorizacijske pravice. Za potrebe naše aplikacije smo definirali tri uporabniške vloge:

- osnovni uporabnik - osnovna uporabniška dovoljenja
- napredni uporabnik - napredna uporabniška dovoljenja, prav tako podeduje vsa dovoljenja osnovnega uporabnika
- skrbnik aplikacije - administratorska uporabniška dovoljenja, prav tako podeduje vsa dovoljenja naprednega uporabnika

Po kreiranju uporabniških vlog, smo lahko na CRUD pogledih oziroma v kontrolerjih definirali, katere uporabniške vloge lahko dostopajo do vsebine za določen/o kontroler/akcijo. Spodaj je primer, kjer smo v kontrolerju *ProjectController* nastavili pravila, da lahko do pogleda dostopajo osnovni uporabniki, do pogledov za urejanje vsebine pa imajo dostop samo napredni uporabniki. Prav tako smo morali na pogledu *view* ustrezno preverjati uporabniške poverilnice in na podlagi njih prikazati ustrezne gumbe in povezave. Na primer uporabniku, ki nima pravic za urejanje, ni potrebno prikazati gumba *uredi*.

```
'access' => [
    'class' => AccessControl::className(),
    'rules' => [
        [
            'allow' => true,
            'actions' => ['update', 'create', 'delete'],
            'roles' => ['manager'],
        ],
        [
            'allow' => true,
            'actions' => ['view', 'index'],
            'roles' => ['@'],
        ],
    ],
],
```

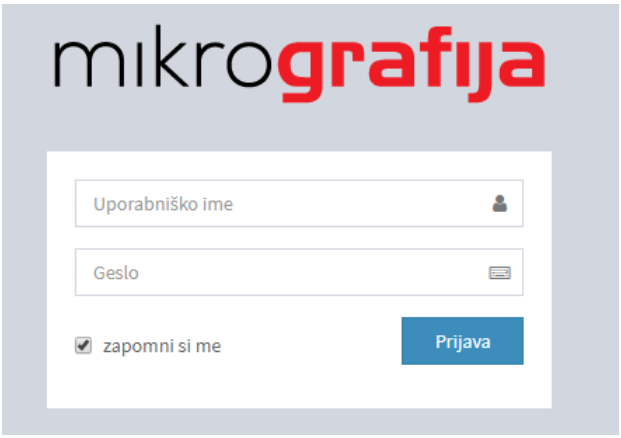
Ena izmed funkcionalnosti je bila, da se lahko uporabniki prijavijo preko protokola ldap. To smo implementirali s pomočjo knjižnice *adldap/adldap* [13], ki z malo konfiguracije omogoča enostavno delo z protokolom ldap. Spodaj je primer kode, kako smo implementirali prijavo uporabnika preko protokola ldap. Če je uporabnikova identiteta potrjena, posodobimo uporabniške atribute, ki jih pridobimo iz imenika ldap ter uporabnika vpišemo v aplikacijo.

```
if ($ldap)
{
    $status = \Yii::$app->ldap->
    authenticate($this->username, $this->password);
    if ($status) {

        $userProperties = \Yii::$app->ldap->
            getUserInfo($this->username);

        // update user properties from LDAP
        $usr->title = $userProperties['title'];
        $usr->firstname = $userProperties['givenname'];
        $usr->lastname = $userProperties['sn'];
        $usr->email = $userProperties['mail'];
        $usr->avatar = base64_encode(
            $userProperties['thumbnailphoto']);

        Yii::$app->user->login($this->getUser(),
            $this->rememberMe ? 3600 * 24 * 30 : 0);

        $usr->password_hash = Yii::$app->security->
            generatePasswordHash($this->password);

        $usr->save();
    } else {
        $this->addError('password', Yii::t
            ('app', 'Incorrect username or password.'));
    }
    return $status;
}
```

4.2 Predstavitev delovanja aplikacije

Skozi to podpoglavje, bomo poskušali prikazati uporabo aplikacije ter njenih funkcionalnostih na primeru, tako kot to poteka v realnem okolju. Zaradi kompleksnosti aplikacije bomo predstavili le bistvene poglede, ki so pomembni za razumevanje. Prav tako smo na določenih slikah zatemnili podatke, ki so občutljive narave, vendar zaradi tega razumevanje delovanja aplikacije ne bo nič težje.

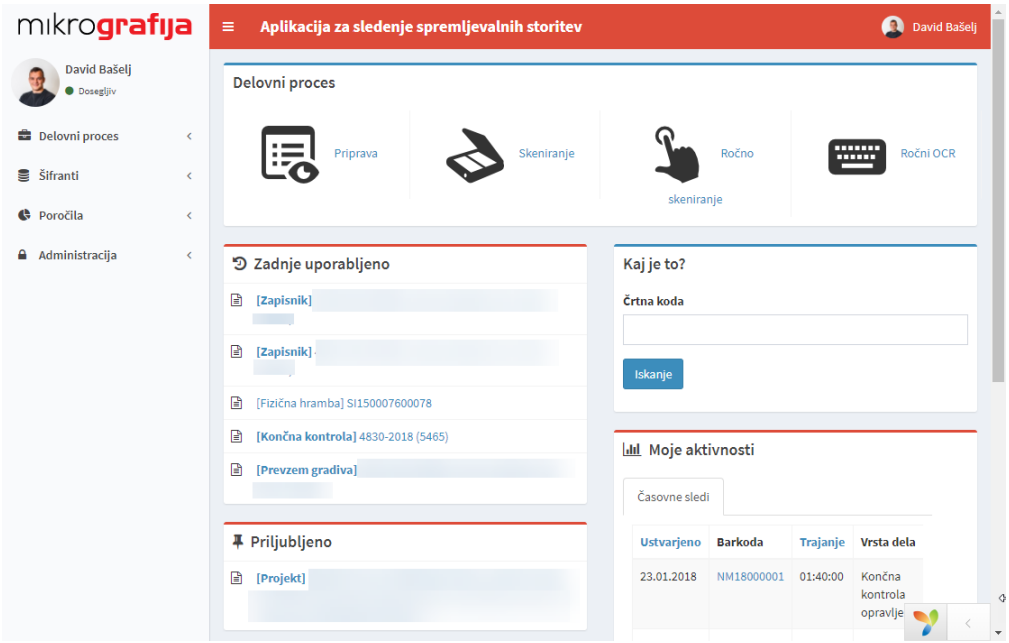
Na sliki 4.4 lahko vidmo pogled, ki se prikaže uporabniku ob vstopu v spletno aplikacijo. Uporabnik se lahko vpiše z uporabniškim imenom in geslom. V kolikor mu nadrejeni omogočijo, se lahko vpiše z domenskim uporabniškim imenom ter geslom, sicer pa s tistim, ki mu je bilo dodeljeno.



Slika 4.4: Vpis v spletno aplikacijo

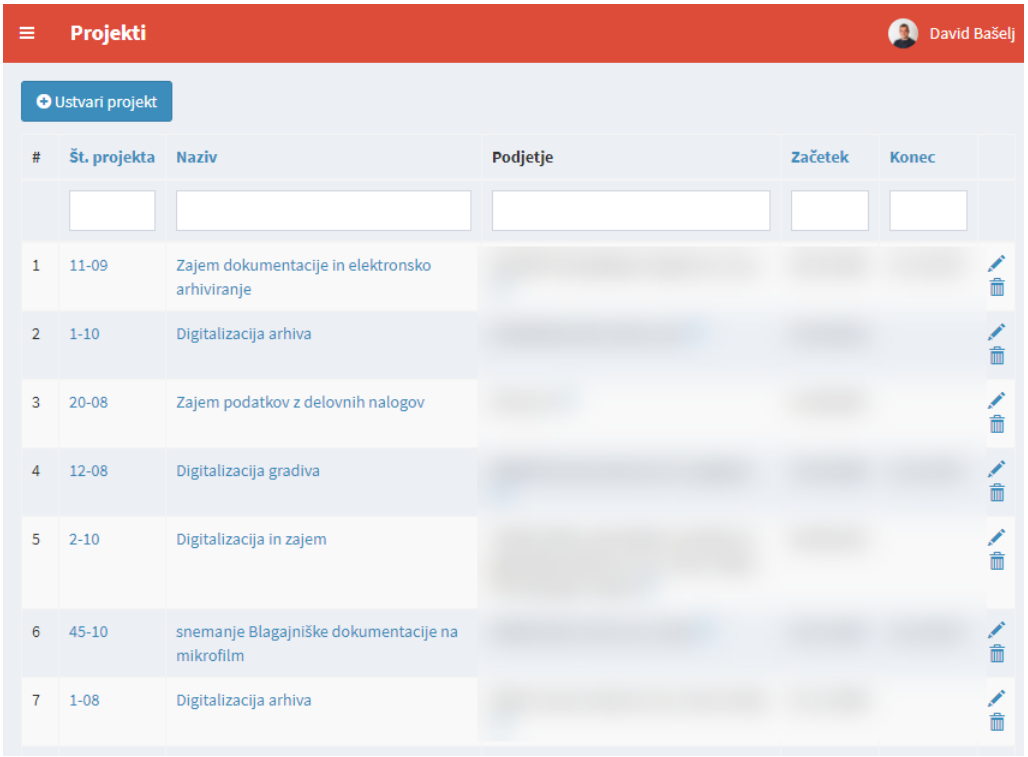
Po uspešnem vpisu v spletno aplikacijo se uporabniku prikaže nadzorna plošča, kot je vidna na sliki 4.5. Na levi strani slike je viden navigacijski meni, s pomočjo katerega lahko uporabnik navigira po spletni strani. Glede na uporabniške pravice, se različnim uporabnikom prikazuje različen meni. V primeru na sliki je viden meni, kot ga vidu admin spletne strani. V nadaljevanju bomo na preostalih slikah ta meni izpustili, zaradi boljše preglednosti slik.

Na sliki 4.5 lahko na vrhu nadzorne plošče vidimo *delovni proces*, kjer lahko uporabnik izbere proces, na katerem trenutno dela in vnaša podatke za izbrano enoto. Preostanek nadzorne plošče je razdeljen na 5 delov. *Zadnje uporabljeno*, kjer lahko uporabnik vidi vire, do katerih je nazadnje dostopal; *Kaj je to?*, ki uporabniku omogoča iskanje po aplikaciji s pomočjo črtnih kod; *Moje aktivnosti*, kjer so prikazane zadnje časovne sledi vpisanega uporabnika; *Priljubljeno*, kjer so prikazani zaznamki, katere si lahko uporabnik naredi v ostalih pogledih aplikacije.



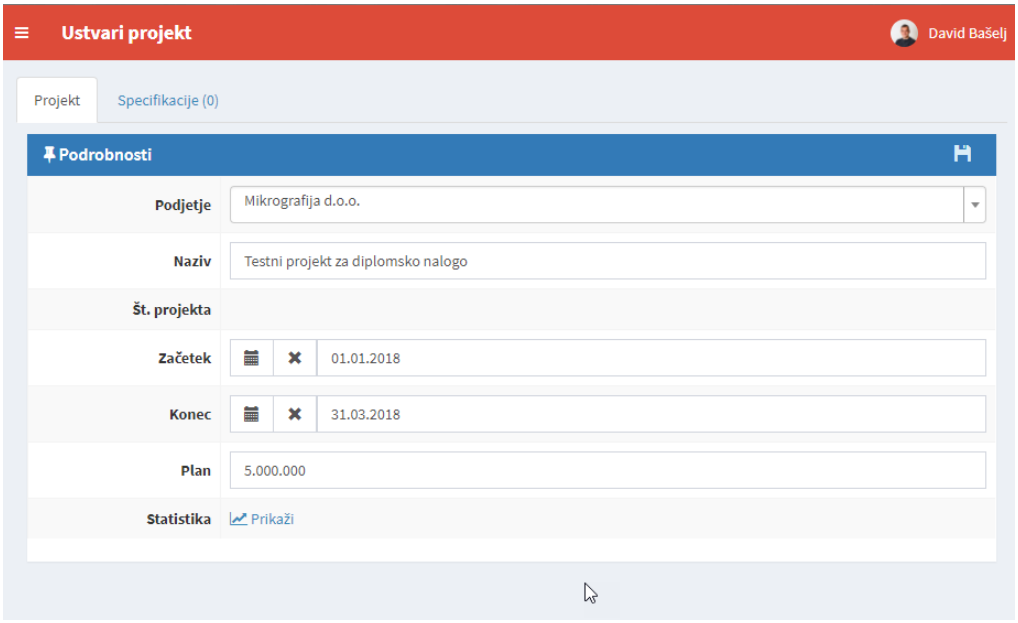
Slika 4.5: Pogled nadzorne plošče

Na sliki 4.6 lahko vidimo pogled, kjer so vidni vsi projekti. Do tega pogleda uporabnik dostopa tako, da v meniju klikne na *Šifranti->Projekti*. Tak pogled smo uporabili tudi pri ostalih šifrantih podatkovnega modela, zato jih v nadaljevanju ne bomo posebej predstavljali.

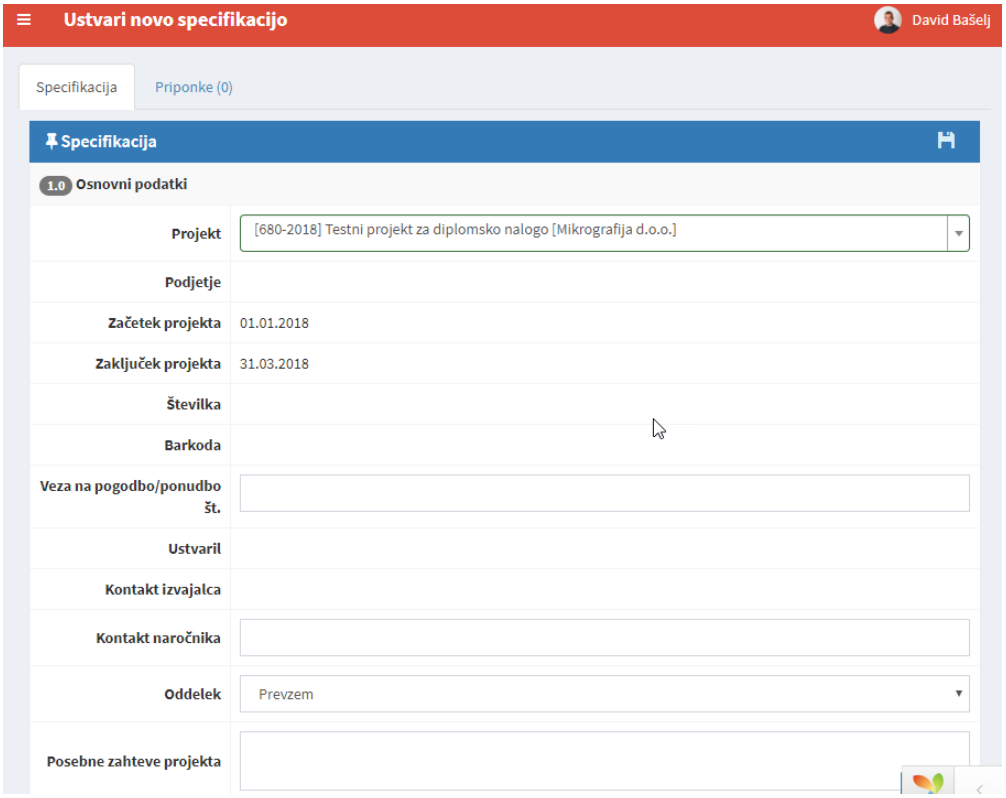


Slika 4.6: Pregled projektov

Na sliki 4.7 je viden pogled, preko katerega lahko napredni uporabnik ustvari nov projekt. Projekt se lahko ustvari na podlagi podjetja, ki smo ga že predhodno vnesli v šifrant podjetij. Nastavimo lahko tudi začetek in konec projekta in pa tudi planirano število posnetkov.



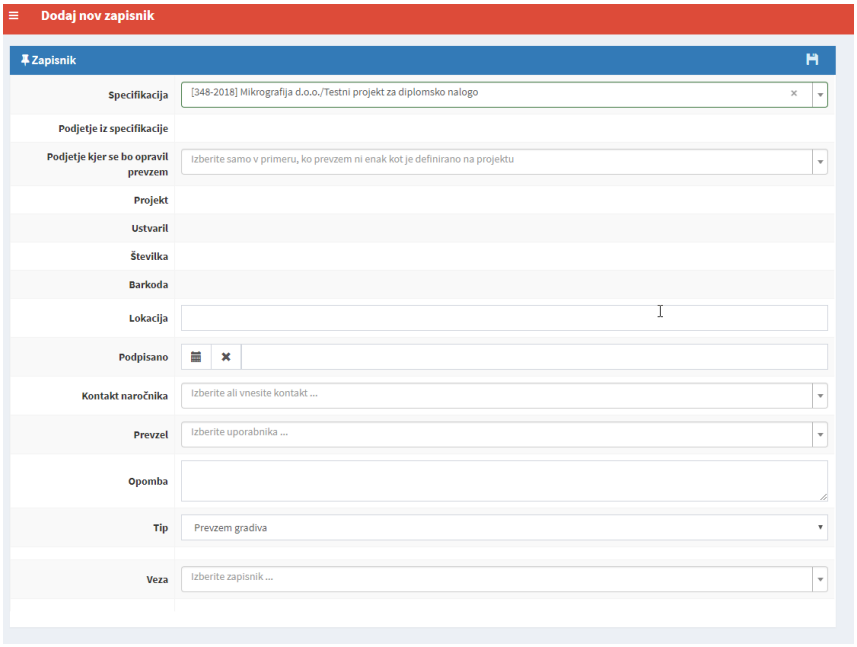
Slika 4.7: Ustvarjanje novega projekta



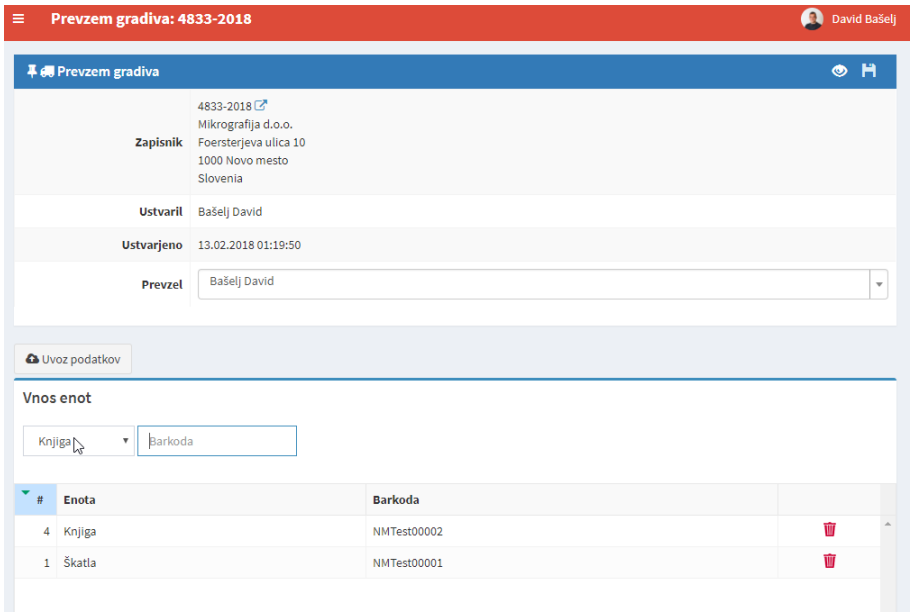
Slika 4.8: Ustvarjanje nove specifikacije

Na podlagi projekta, ki smo ga ustvarili na sliki 4.7, lahko sedaj ustvarimo novo specifikacijo, na podlagi katere bo potekal delovni proces. Na sliki 4.8 je viden pogled, za kreiranje nove specifikacije na podlagi projekta, ki smo ga že ustvarili. Zaradi velikosti obrazca, ki vsebuje zelo veliko polj (preko 300, kot smo že navedli v podatkovnem modelu), je prikazan samo del obrazca za ustvarjanje nove specifikacije.

Na podlagi ustvarjene specifikacije lahko ustvarimo nov zapisnik o prevzemu gradiva. Na sliki 4.9 je viden pogled, s pomočjo katerega ustvarimo nov zapisnik o prevzemu gradiva. Ob kreiranju prevzemnega zapisnika izberemo specifikacijo, ki smo jo že kreirali in izpolnimo preostale atribute. Atribut *veza* predstavlja povezavo na predhodni zapisnik.

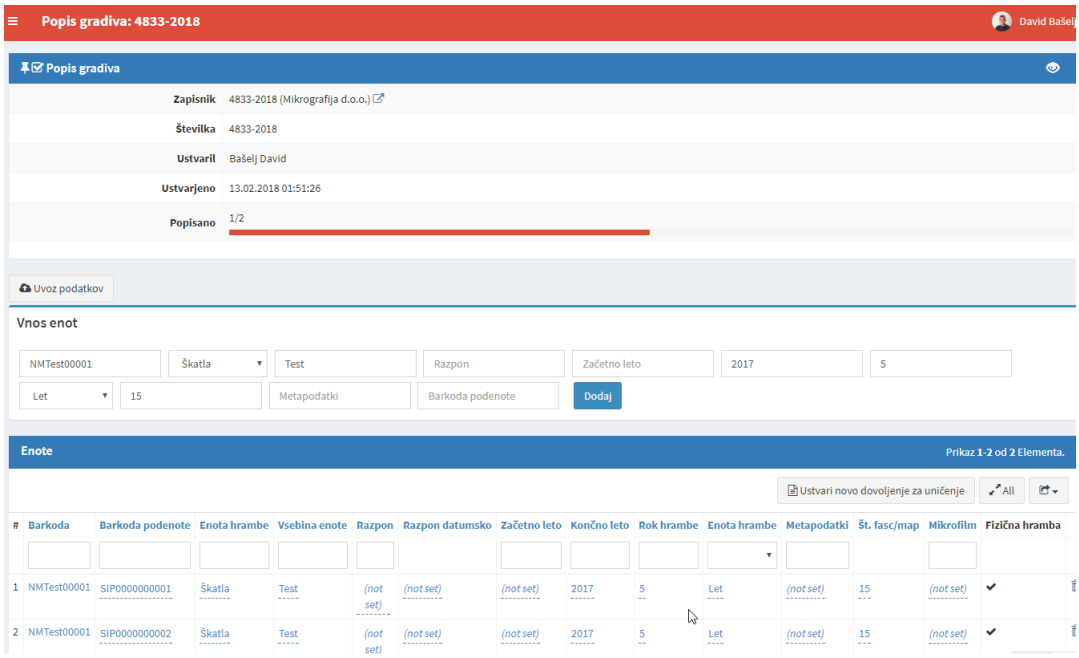


Slika 4.9: Ustvarjanje novega zapisnika o prevzemu gradiva



Slika 4.10: Prevzemanje gradiva

Po ustvarjenem zapisniku o prevzemu gradiva, se lahko prične s prevzemanjem gradiva. Na sliki 4.10 je viden postopek vnosa enot gradiva. Uporabnik lahko izbere tip enote gradiva, ter vnese črtno kodo enote. V primeru, da vnešena črtna koda že obstaja v sistemu, aplikacija uporabnika o tem obvesti in mu prepreči vnos take črtne kode. Po končanem vnosu lahko uporabnik shrani vnešene enote ali pa jih izbriše. Uporabnik lahko enote gradiva uvozi v aplikacijo tudi iz *excel* datoteke s pomočjo gumba *Uvoz podatkov*.

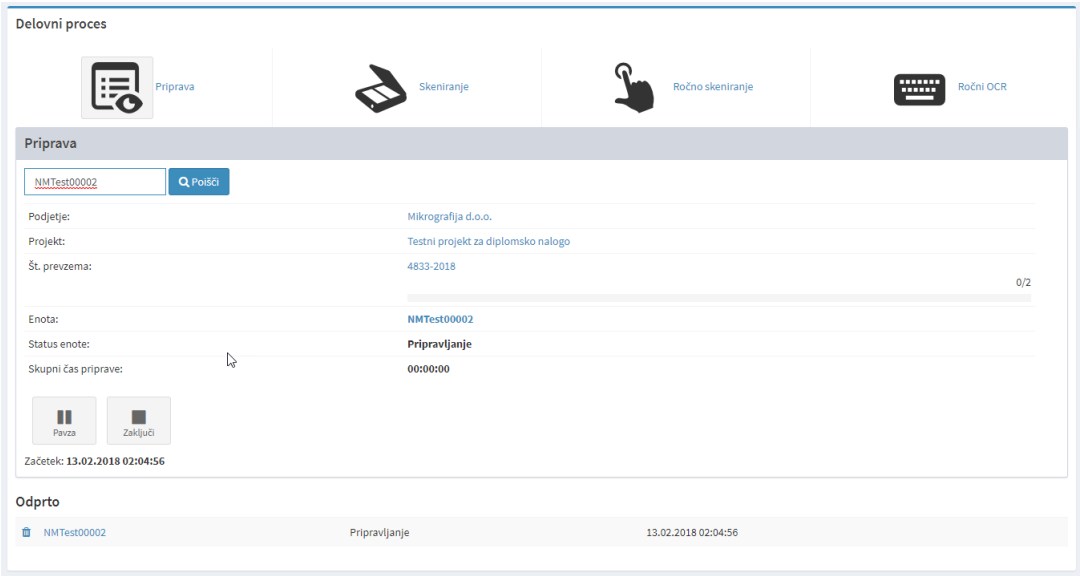


Slika 4.11: Popisovanje gradiva

Po končanem prevzemu gradiva se prične popis gradiva. Na sliki 4.11 je viden pogled, kjer uporabnik popisuje gradivo. Uporabnik lahko vnaša podenote gradiva ali pa jih uvozi iz *excel* datoteke. Aplikacija preverja, koliko enot gradiva je bilo že popisanih, in ne dovoli popisa enote gradiva, ki ni del prevzema. Za poljubno enoto gradiva lahko uporabnik vnese poljubno število podenot gradiva, vendar tudi črtne kode podenote gradiva morajo biti unikatne.

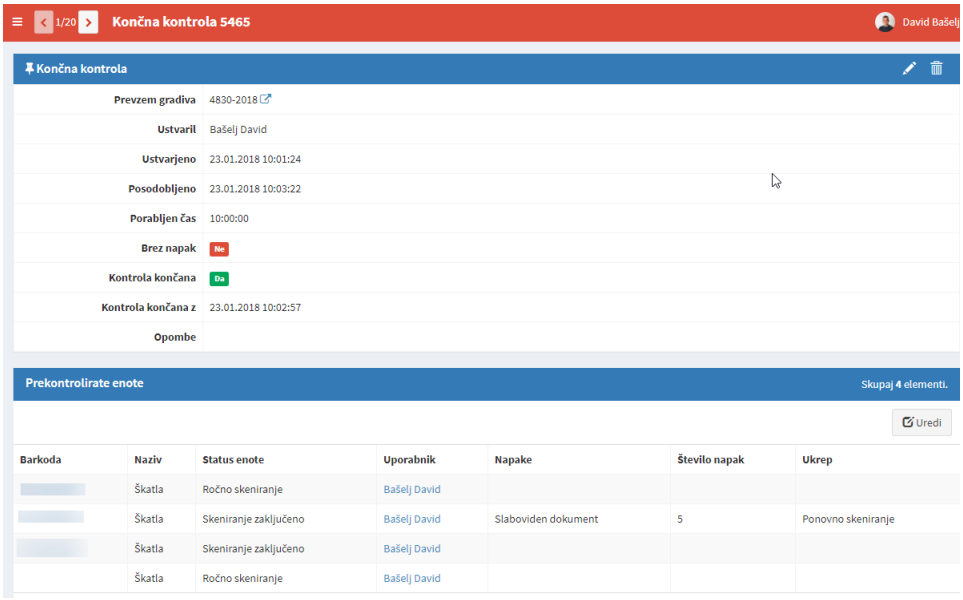
Po končanem prevzemu ter popisu gradiva se prične delovni proces. Na

sliki 4.12 je viden delovni proces, kjer se pripravlja gradivo. Uporabnik, ki dela na pripravi gradiva, v aplikaciji poišče enoto gradiva, ki jo bo pripravljal, in pritisne gumb *start*. Ob koncu priprave stisne gumb *zaključi* in v aplikacijo se zapiše časovna sled. Uporabnik, lahko v primeru prekinitve dela, malice ali česar koli drugega, klikne gumb *Pavza* in z delom nadaljuje kasneje. S tem se v časovnih sledeh upošteva samo čas dela in ne časa, ko uporabnik ni delal na procesu. Po enakem principu deluje tudi optično branje ter ročno optično branje. Ročno optično prepozavanje znakov še ni implementirano, saj so potrebne uskladitve z OCR oddelkom.

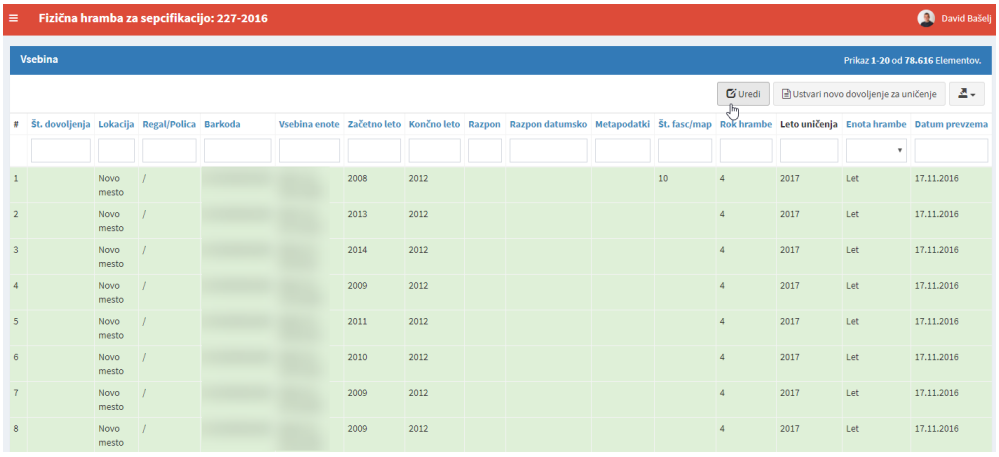


Slika 4.12: Priprava gradiva

Na sliki 4.13 je viden pogled, kjer lahko uporabnik doda in ureja končno kontrolo za eno ali več enot gradiva. V primeru najdenih napak uporabnik to ustrezno zavede v aplikacijo. Če so bile najdene napake, lahko uporabnik izbere vrsto napake in vnese število napak ter ukrep, ki je bil izveden zaradi napak tako, kot je razvidno na sliki.



Slika 4.13: Končna kontrola

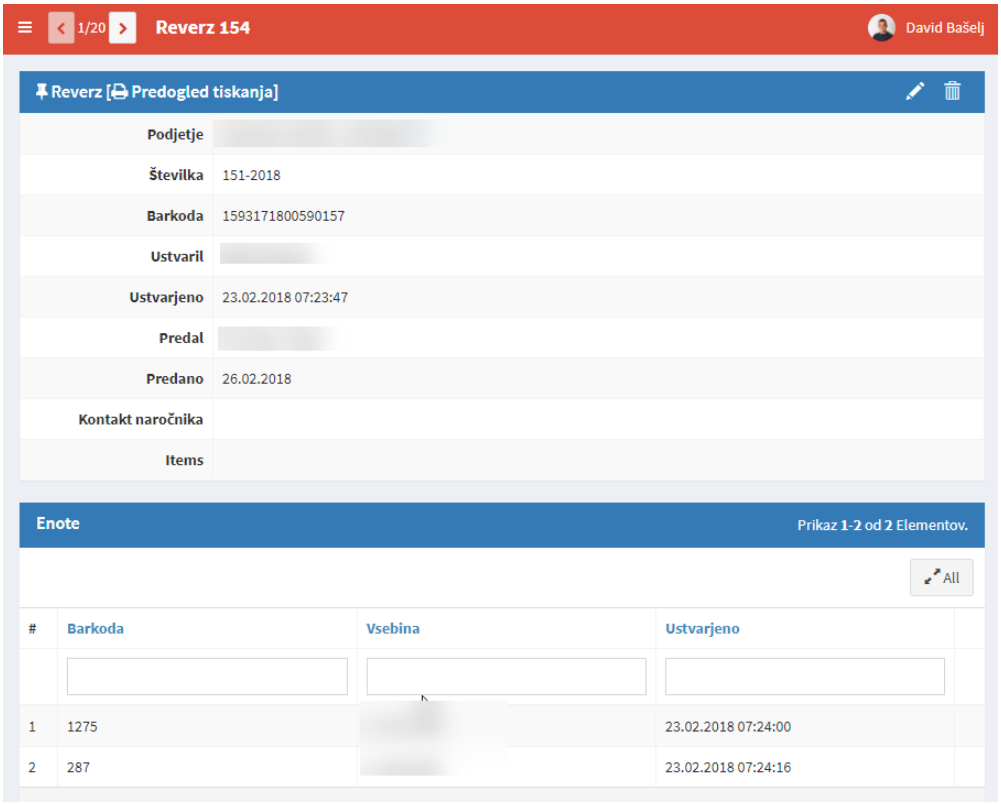


Slika 4.14: Fizična hramba

Na sliki 4.14 je viden pogled fizične hrambe za določeno specifikacijo, kjer je bila omogočena fizična hramba. Vidne so vse podenote gradiva, ki so vezane na to specifikacijo. Po pretečenem roku hrambe se v aplikaciji podenota gradiva obarva z zeleno, kot indikator, da je podenoti gradiva po-

tekel rok hranjenja in se jo lahko uniči. Uporabnik lahko klikne na gumb *Ustvari novo dovoljenje za uničenje*, kjer lahko izbere podenote gradiva, za katere želi narediti zapisnik o dovoljenju za uničenje (dokument, ki dovoljuje uničenje dokumentacije). Zapisnik o dovoljenju za uničenje naredi po podobnem postopku, kot smo ga opisali na sliki 4.9. Po pridobljenem podpisanem zapisniku o dovoljenju za uničenje se prične postopek uničenja dokumentacije. Kreira se nov zapisnik o uničenju, gradivo pa se uniči. Zapisniku o uničenju se v vezi doda zapisnik o dovoljenju za uničenje, kot je opisano na sliki 4.9.

Na sliki 4.15 je viden pogled že ustvarjenega reverza za neko podjetje. Pogled je razdeljen na dva dela. Na vrhu so vidni podatki o reverzu, spodaj pa so vidne enote gradiva, ki so se predale naročniku ob tem reverzu.



Slika 4.15: Reverz

Poglavje 5

Sklepne ugotovitve

V okviru diplomske naloge smo izdelali spletno aplikacijo Sledenje za spremljanje in sledenje delovnega procesa arhiviranja dokumentacije v podjetju Mikrografija. Aplikacija pokriva celoten delovni proces in s tem zaposlenim v podjetju omogoča lažje delo. V uvodnem poglavju smo na kratko predstavili podjetje in opisali, od kje izhaja ideja za diplomsko nalogo. V naslednjem poglavju smo okvirno predstavili delovni proces in kako je sledenje delovnega procesa potekalo pred uvedbo aplikacije. Na podlagi tega smo nato podrobneje predstavili delovni proces arhiviranja dokumentacije. Po opisanem delovnem procesu smo določili uporabniške vloge, ki lahko nastopajo v aplikaciji, in funkcionalnosti, ki jih mora aplikacija podpirati. Pri pisanju funkcionalnosti smo se osredotočali predvsem na to, da jih opišemo čim bolj abstraktno. V nadaljevanju smo na podlagi opisanih funkcionalnostih predstavili podatkovni model, ki smo ga razvili za potrebe aplikacije. Proti koncu smo opisali ključne metode, ki smo jih uporabili pri razvoju aplikacije. Na koncu smo naredili še predstavitev delovanja aplikacije v takem vrstnem redu, kot poteka obdelava gradiva.

Tekom razvoja aplikacije smo naleteli na težave, ki smo jih v sodelovanju z zaposlenimi uspešno odpravili. Ker se je aplikacija razvijala postopoma, funkcionalnosti pa so se nadgrajevale, je bil tesen stik in dobra komunikacija z zaposlenimi ključnega pomena. Ena izmed težav je bila, kako prenesti de-

lovni proces v aplikacijo, da se funkcionalnosti delovnega procesa ohranijo, uporabniški vmesnik pa bo pregleden in enostaven za uporabo. To smo dosegli tako, da smo določili enotno terminologijo, ki se bo uporabljala v aplikaciji in, da smo natančno določili kaj se lahko zgodi v vsaki fazi delovnega procesa. Pred vpeljavo aplikacije v produkcijo so imeli uporabniki možnost testiranja aplikacije. Tekom tega testiranja so bile ugotovljene nekatere napake, ki smo jih uspešno odpravili pred vpeljavo aplikacije v produkcijo.

Cilj diplomske naloge je bil dosežen. Razvili smo spletno aplikacijo, ki zaposlenim v podjetju omogoča boljše sledenje delovnega procesa. Aplikacijo aktivno uporablja od 30 do 50 uporabnikov. Število uporabnikov narašča in pada, glede na velikost projekta. Aplikacija sedaj uporabnikom omogoča hiter pregled prevzete in obdelane dokumentacije s podpisanimi zapisniki o prevzemu, vračilu, uničenju in dovoljenju za uničenje gradiva. Pred aplikacijo so uporabniki uporabljali excel tabele, kar je lahko privedlo do podvojenih šifer ali celo izgubljenih. Aplikacija sedaj omogoča unikatne šifre in do teh napak ne prihaja več. Uporabniki prav tako vidijo, kateri zapisniki še niso bili podpisani s strani stranke. Aplikacija sedaj pri popisu dokumentacije na podlagi tipa dokumentacije sama dopolnjuje zakonski rok hrambe, pred aplikacijo pa se je ta rok določal ročno, kar je lahko privedlo do napake in dokumentacija se je hranila predolgo ali pa se je predčasno uničila. Aplikacija na podlagi preteka roka hrambe dokumentacije, ki se ga je vneslo v popisu gradiva, omogoča pregled dokumentacije ki se lahko uniči. Na podlagi podatkov v podatkovni bazi je možno pridobiti informacije kolikšne so proste kapacitete v arhivu in koliko kapacitet je še potrebnih za bodoče projekte. Tega podatka uporabniki pred aplikacijo niso imeli. Pred aplikacijo so zaposleni na optičnih bralnikih ročno pisali črtno kodo za vsako obdelano enoto gradiva, zato je prihajalo do pogostih napak. Na podlagi zgoraj opisanih prednosti so izkušnje uporabnikov z aplikacijo pozitivne, saj aplikacija zmanjšuje napake tekom delovnega procesa in omogoča hitrejše delo. Uvedba aplikacije Sledenje v podjetju je doprinesla kar nekaj prednosti. To so lažja sledljivost projektom, vsi podatki so dostopni na enem mestu, aplikacija je

dostopna povsod (doma, na poti, v tujini), enotnost podatkov, samodejno preračunavanje statistike in ustvarjanje poročil, zaradi boljše sledljivosti je lažja organizacija in vodenje projektov.

V aplikaciji je še veliko prostora za izboljšave in nadaljnji razvoj. Aplikacijo bi bilo potrebno v naslednji fazi povezati z OCR oddelkom. Trenutno aplikacija omogoča sledenje samo do končne kontrole, v povezavi z OCR oddelkom pa bi bila vidna celotna sled delovnega procesa od prevzema gradiva do predaje optično prebranih posnetkov naročniku. S tem bi še dodatno zmanjšali napake in eliminirali možnost, da se dokumentacija znotraj delovnega procesa izgubi. Ena izmed funkcionalnosti bi bila tudi možnost elektronskega podpisovanja zapisnikov s prenosno tablico ali mobilnim telefonom.

Literatura

[1] AdminLTE. Dosegljivo: https://adminlte.io/themes/AdminLTE/index.html, 2018. [Dostopano: 13. 01. 2018].

[2] Apache. Dosegljivo: https://httpd.apache.org/ABOUT_APACHE.html, 2018. [Dostopano: 02. 01. 2018].

[3] Yii 2 Advanced Project Template. Dosegljivo: https://github.com/yiisoft/yii2-app-advanced, 2018. [Dostopano: 13. 01. 2018].

[4] Bootstrap. Dosegljivo: https://getbootstrap.com/docs/3.3/, 2018. [Dostopano: 13. 01. 2018].

[5] Botstrap javascript. Dosegljivo: https://getbootstrap.com/docs/3.3/javascript/, 2018. [Dostopano: 13. 01. 2018].

[6] Composer. Dosegljivo: https://getcomposer.org/, 2018. [Dostopano: 13. 01. 2018].

[7] CSS. Dosegljivo: https://sl.wikipedia.org/wiki/CSS, 2018. [Dostopano: 12. 01. 2018].

[8] Gii. Dosegljivo: http://www.yiiframework.com/doc-2.0/ext-gii-index.html, 2018. [Dostopano: 13. 01. 2018].

[9] HTML. Dosegljivo: https://sl.wikipedia.org/wiki/HTML, 2018. [Dostopano: 12. 01. 2018].

[10] IntelliJ IDEA. Dosegljivo: https://www.jetbrains.com/idea/, 2018. [Dostopano: 13. 01. 2018].

[11] JetBrains Plugins Repository. Dosegljivo: https://plugins.jetbrains.com/, 2018. [Dostopano: 13. 01. 2018].

[12] jQuery. Dosegljivo: https://jquery.com/, 2018. [Dostopano: 13. 01. 2018].

[13] A PHP LDAP Package for humans. Dosegljivo: https://github.com/Adldap2/Adldap2, 2018. [Dostopano: 13. 01. 2018].

[14] MariaDB. Dosegljivo: https://mariadb.org/, 2018. [Dostopano: 15. 02. 2018].

[15] MySQL. Dosegljivo: https://en.wikipedia.org/wiki/MySQL, 2017. [Dostopano: 1. 12. 2017].

[16] MySQL Workbench. Dosegljivo: https://www.mysql.com/products/workbench/, 2018. [Dostopano: 13. 01. 2018].

[17] PHP. Dosegljivo: https://en.wikipedia.org/wiki/PHP, 2017. [Dostopano: 30. 11. 2017].

[18] phpMyAdmin. Dosegljivo: https://www.phpmyadmin.net/, 2018. [Dostopano: 14. 01. 2018].

[19] RBAC Manager for Yii 2. Dosegljivo: https://github.com/yii2mod/yii2-rbac, 2018. [Dostopano: 13. 01. 2018].

[20] Select2. Dosegljivo: https://select2.org/, 2018. [Dostopano: 13. 01. 2018].

[21] TIS. Dosegljivo: https://www.topimagesystems.com/, 2018. [Dostopano: 7. 02. 2018].

[22] XAMMP. Dosegljivo: https://www.apachefriends.org/index.html, 2018. [Dostopano: 10. 1. 2018].

[23] Xdebug. Dosegljivo: https://xdebug.org/index.php, 2018. [Dostopano: 13. 01. 2018].

[24] Xdebug Wizard. Dosegljivo: https://xdebug.org/wizard.php, 2018. [Dostopano: 13. 01. 2018].

[25] Yii2. Dosegljivo: http://www.yiiframework.com/about/, 2017. [Dostopano: 2. 12. 2017].