Mrežni okvir za poslovnu inteligenciju u oblaku

Strusa, Janko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Janko Strusa

Zagreb, 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Prof. dr. sc. Mario Essert, dipl. ing.

Student:

Janko Strusa

Zagreb, 2016.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se prof. dr. sc. Mariu Essertu na iskustvu koje sam stekao tijekom izrade ovog rada, na njegovim savjetima, vodstvu i strpljenju.

Također se želim zahvaliti mojoj djevojci i obitelji na potpori koju su mi pružili tijekom studija.

Janko Strusa

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **Janko Strusa** Mat. br.:0248002217

Naslov rada na hrvatskom jeziku: **Mrežni okvir za poslovnu inteligenciju u oblaku**

Naslov rada na engleskom jeziku: **The framework for business intelligence in the cloud**

Opis zadatka:

Mrežni okviri za poslovnu inteligenciju u oblaku se prema MIT-ovom obrazloženju pojma poslovne inteligencije (*http://www.mit-software.hr/usluge/bi/bi1/*) počinju intenzivno razvijati od kada su poduzeća/tvrtke automatizirale svoje poslovne procese. To se dogodilo ugradbom različitih transakcijskih sustava u poslovanje, koji su se vrlo brzo pokazali kao generatori velikih količina podataka. S čisto tehničke strane, poslovna inteligencija je proces unutar kojeg se sirovi podaci oblikuju u strukturno i vizualno prihvatljive informacije. Cilj ovog rada je načiniti jedan takav alat za opći poslovni sustav temeljen na WIKI-okviru, koji će osim stvaranja i održavanja klasičnih WIKI-članaka omogućiti korisničku interakciju s Excel tablicama, on-line grafovima i mrežno označivanje slika i PDF dokumenata u oblaku.

U radu treba projektrati i programski izvesti sljedeće:

1. U *Web2py* alatu načiniti obrasce i tablice za prikupljanje i prezentaciju podataka.
2. Ostvariti alat koji u kontekst WIKI članaka može ugraditi on-line grafove s različitim oblicima prikaza podataka prošlog ili trenutačnog stanja poslovnog procesa.
3. Omogućiti povezivanje članaka s datotekama svih ekstenzija (nacrti, modeli, popratna dokumentacija i sl.) i njihovu dostupnost u oblaku putem mrežnog preglednika (eng. browser-a).
4. Označivanje slika (.jpg i .png formata) i pdf datoteka s vizualnim oznakama. Oznake se mogu smještati u slojevima nad slikom i pratiti prikladnim opisima (legendama).

Usporediti dobiveno rješenje s profesionalnim alatima na Internetu.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
5. svibnja 2016.	7. srpnja 2016.	13., 14. i 15. srpnja 2016.

Zadatak zadao:

M. Essert

Prof.dr.sc. Mario Essert

Predsjednik Povjerenstva:

Prof. dr. sc. Franjo Cajner

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

U ovom diplomskom radu dat je kratak pregled o tome što čini sustave poslovne inteligencije koja je bila osnova za izradu aplikacije, nakon čega slijedi opis tehnologija koje su se koristile prilikom izrade. Na kraju se nalazi detaljan pregled aplikacije s uputama za korištenje. Osnova na kojoj je rađena aplikacija je mogućnost prikupljanja podataka preko mrežnog preglednika, sve u svrhu spremanja na udaljenom serveru i kasnijem dohvaćanju tih istih podataka. Organizacija strukture aplikacije je poput wiki sustava koji su se pokazali uspješni za suradnju više pojedinaca koji međusobno nisu u osobnom kontaktu. Stoga je bilo potrebno koristiti moderne mrežne tehnologije za izradu obrazaca, sučelja i alata koji korisnicima mogu pružiti rad unutar mrežnog preglednika. Osim pružanja mogućnosti rada unutar mrežnog preglednika pažnja je bila posvećena izradi alata koji svojim korištenjem daju osjećaj interaktivnosti na razini lokalnog računalnog programa.

Ključne riječi:

- poslovna inteligencija
- mrežne tehnologije
- prikupljanje podataka
- vizualizacija podataka
- anotacija slika
- anotacija pdf

SUMMARY

This thesis summarizes the accounting systems of business intelligence that was the basis for the development of application, followed by a description of the technologies that were used in the making. Finally, there are detailed examination of the application with instructions for use. The basis on which the application is made is the opportunity to collect data via web browser all in order to save it on a remote server and later retrieving those same data. The organization structure of the application is like a wiki system, which proved successful for cooperation of individuals who are not in personal contact. Therefore, it was necessary to use modern network technologies to produce forms, interface and tools that can offer users capabilities of working in the web browser. In addition to providing work opportunities within web browser, attention was paid to the development of tools that its use gives a sense of interactivity at the level of the local computer program.

Key words:

- business intelligence
- network technologies
- data collection
- data visualization
- image annotation
- pdf annotation

1. UVOD

U ovom diplomskom radu istražena je mogućnost stvaranja i upotrebe online sustava sličnog Wiki tehnologiji, a u svrhu poslovne inteligencije u proizvodnji. Vođen idejom Wikipedije, koja je u kratkom roku stekla veliku popularnost, ovaj diplomski rad želi iskoristiti njezine značajke u oblikovanju poslovnog vođenja sustava. Wikipedija se pokazala kao dobro prihvaćen model lakog stvaranja i održavanja sadržaja mrežnih stranica. Stvaranje i održavanje wiki sadržaja pritom nije zadaća pojedinca, već je suradnja više ljudi koji mogu biti stručnjaci u tom polju, ali i ne moraju. Mrežne stranice koje nastaju i čuvaju se u Wikipediji zvat ćemo člancima. Članci su preko mreže dostupni svima pa svojim sadržajem djeluju na kolektivnu svijest svih koji ih čitaju.

Kao što je neki proizvod rezultat rada više ljudi koji međusobno surađuju, tako se i informacijski sustav vezan uz taj proizvod može objediniti i dati podatkovnu sliku proizvodnog procesa svim suradnicima. Povezivanjem ideje Wiki sustava i poslovne inteligencije svi suradnici i korisnici mogu dobiti željenu informaciju na uvid. Svijest o proizvodu tako je povećana i obuhvaća druge djelatnosti koje ne moraju biti temeljno područje interesa pojedinca. Na taj se način potiče suradnja i bolji proces planiranja proizvoda.

Sam Wiki sustav nije dovoljan da bi se postigla potpuna funkcionalnost poslovne inteligencije, pa se zato ovim radom stvorilo više dodatnih programskih alata koji su nužni pri obradi, vizualizaciji i spremanju podataka. Načinjeni alati služe kao dopuna Wiki sustavu i kao cjelina daju bolje rješenje jer omogućuje stvaranje kvalitetnijeg informacijskog sadržaja relevantnog za proizvodne sustave.

2. POSLOVNA INTELIGENCIJA

Razvojem industrije količina generiranih podataka naglo raste. Podatci se skupljaju kako bi se stvorila objektivna slika poslovnog sustava na temelju koje se kasnije može proaktivno djelovati u svrhu povećanja produktivnosti, profita i slično. Stoga se razvijaju tehnike i alati za skupljanje i transformaciju nestrukturiranih podataka u informativne podatke, koji su pogodni statističkoj analizi. Takvi alati i tehnike objedinjeni su u sustavu koji se zove *poslovna inteligencija*. Sustavi poslovne inteligencije u mogućnosti su obraditi veliku količinu podataka kako bi olakšali, razvili i načinili nove strateške poslovne prilike.

Poslovnom inteligencijom korisnik ima uvid u povijesno, trenutno i predviđeno stanje poslovnog sustava. Na temelju takva uvida u stanje sustava korisnik je u mogućnosti donijeti zaključke i predvidjeti akcije kojima će pozitivno djelovati.

2.1. Nastanak ideje poslovne inteligencije

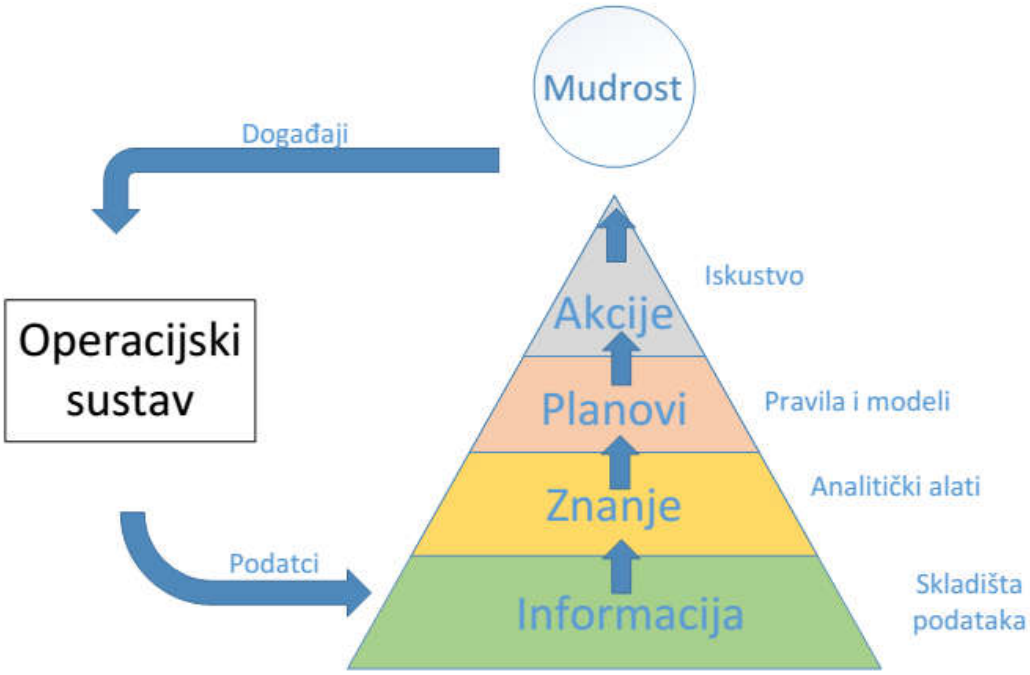
Poslovna inteligencija kao zasebna disciplina započinje ranih 1990-ih kao način da se osigura krajnjim korisnicima bolji pristup informacijama potrebnih za donošenja odluka. Početni cilj bio je pružiti korisnicima brz i lak način informiranja kako oni ne ni morali ovisiti o informatičkim uredima za stvaranje prilagođenih izvještaja. Do ranih 1990-ih poslovna inteligencija sastojala se od skladišta podataka, upita i alata za izvještaj.

Tvrtke su započele s gradnjom skladišta podataka kako bi se osobna računala rasteretila od slanja upita. Skladišta podataka pružala su korisnicima slanje velikih količina zahtjeva bez utjecaja na performanse osobnih računala. U to vrijeme korisnici su nužno trebali poznavati SQL (*engl. Structured Query Language*), strukturni jezik za upite, kako bi uspješno mogli slati upite prema skladištima podataka. Proizvođači softvera, predviđajući svijetlu budućnost poslovne inteligencije, počinju isporučivati alate za upite i izvještaje koji skrivaju SQL jezik iza grafičkih sučelja na osobnim računalima. Također se u kasnim 1990-im počinju nuditi alati dostupni preko mrežnih preglednika zajedno s analitičkim alatima za stvaranje potpunog sustava poslovne inteligencije.

2.2. Koncept poslovne inteligencije

Često se smatra da poslovna inteligencija obuhvaća slanje upita, izvješćivanje i analitičke alate. Međutim, pojam poslovne inteligencije širi je od skupa tih alata. To su sustavi koji su u stanju učiti o stanju organizacije i na taj način donositi strateške odluke.

Poslovna inteligencija obuhvaća sve radnje: od prikupljanja i pročišćivanja podataka, pa do odluka koje utječu na rad sustava. Odluke se donose prema prethodnom iskustvu i modelu sustava.



Slika 1 Tijek informacija

Podatci

Ciklus započinje u trenutku kada operateri koji vode sustav snime događaje i spreme ih u računalnu bazu. Ti događaji mogu biti skladištenje, održavanje, naplata, proizvodnja i slično. To su detaljni podatci koji se kodiraju i spremaju u digitalnom obliku u temelj tijeka informacije, kako prikazuje slika 1.

Informacija

Iz nestrukturiranih podataka stvaraju se baze koje objedinjuju više podataka i sistematiziraju podatke. Tako dostupni podatci imaju jasnu strukturu, smisao, atribute i hijerarhiju. Nizom spremljenih informacija definira se poslovni sustav.

Znanje

Analitičkim alatima korisnik može doći do informacije i analizirati je. Analizom informacije mogu se uočiti uzorci i trendovi podataka. Stečeno znanje o podatcima može se primijeniti kako bi se predvidjelo djelovanje sustava u budućnosti.

Planovi

Poznavanjem ponašanja sustava korisnik može donijeti odluke i stvarati pravila kako bi pokušao utjecati na trendove i uzorke koje je uočio. Planovi mogu biti jednostavni, kao na primjer naručivanje dodatnih dijelova, jer je trend stanja na skladištu prikazao vrijeme eksploatacije manje od vremena potrebnog za nabavu dijelova. Planovi dakako mogu biti i složeni, dobiveni kao rezultat testnog algoritma ili računalom simuliranog modela.

Akcije

Provođenjem planova pokreću se poslovni postupci koji se tijekom vremena preko prikupljenih podataka reflektiraju natrag u sustav.

Mudrost

Svakim ponavljanjem ciklusa, osoblje stječe nova saznanja o procesu i kako određeni poslovni postupci (akcije) djeluju na poslovanje.

2.3. Primjena poslovne inteligencije

Automatizacija procesa koja se provodi kako bi se povećala profitabilnost uključuje mnoge parametre koji mogu pozitivno ili negativno djelovati na profit. Male promjene parametara mogu kao posljedicu imati veliku razliku u profitu. Rastom kompleksnosti sustava neophodno je i povećanje broja tih parametara. Takvi procesi redovito nadilaze ljudske mogućnosti za praćenjem. Sistematizacijom i organizacijom podataka u računalu moguće je pratiti takve

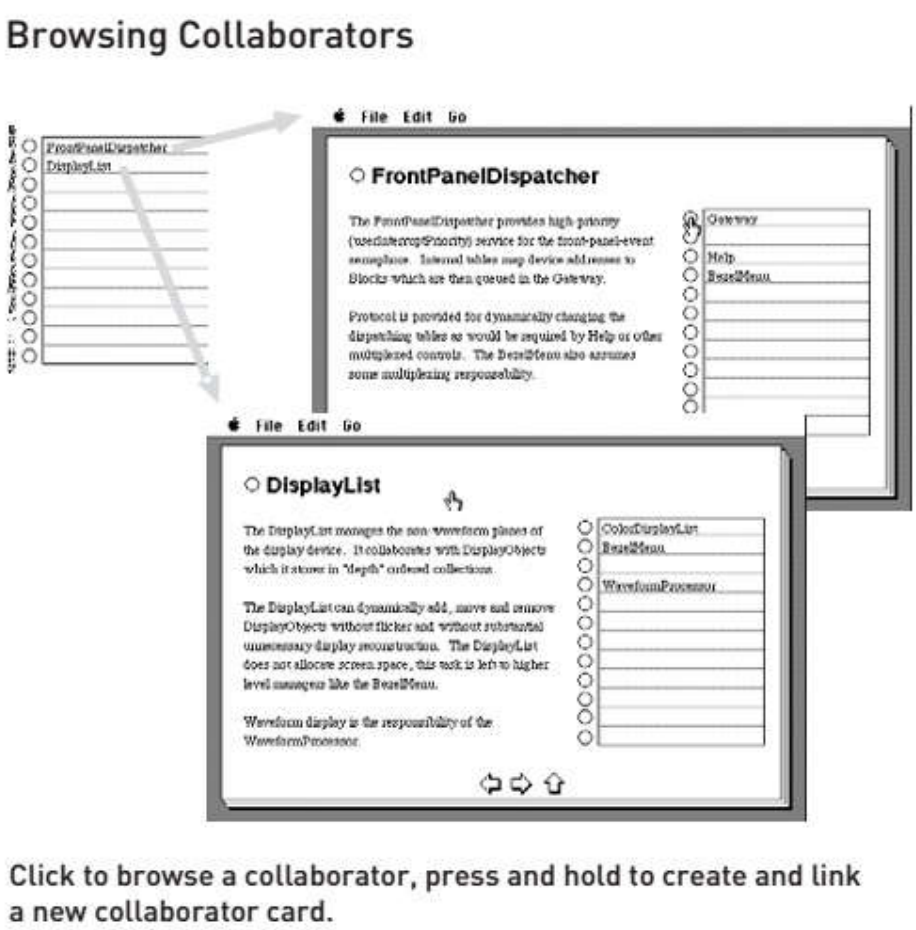
sustave. Stoga se stvaraju, unaprjeđuju i primjenjuju brojni računalni alati koji automatiziraju sistematizaciju i organizaciju poslovnih ili proizvodnih podataka.

3. USTROJ WIKI SUSTAVA

Od svojih je početaka wiki prošao dalek put. Od same ideje i jednostavnog razmišljanja Howarda Cunninghama pa do mrežne stranice Wikipedija, koja je danas postala sinonim za enciklopediju. S više od 4 milijuna članaka Wikipedija je postala vodeći izvor znanja na internetu.

3.1. Rana primjena wiki filozofije

Nakon završetka školovanja, Howard Cunningham zaposlio se u Tek Labs, gdje je imao zadatak istražiti mogućnosti organiziranja njihovih softverskih projekata. Tako je započeo istraživati načine kako dokumentirati ideje i projekte kako bi zaposlenici unutar različitih odjela mogli dijeliti znanje. Prilikom proučavanja kako to postići, Cunningham je sustav zamislio kao metafore koje vode na pravo mjesto unutar sustava. Nakon dugo vremena Cunningham je napokon otkrio alat koji će mu pomoći realizirati njegovu ideju. Apple Computer je svojim novim softverom zvanim *HyperCard* omogućio brzo i jednostavno povezivanje sadržaja. Otkriće *hyperlinka* ili jednostavno poveznice, ljudima je bilo omogućeno klikom na ekranu doći do sljedećeg teksta ili multimedijskog sadržaja. Iako je *HyperCard* bilo dobro rješenje, bilo je i nedostataka. Povezivanje kartica međusobno bilo je komplicirano. Povezivanje dviju kartica odvijalo se tako da se prvo moralo doći na karticu koju želimo povezati na prijašnju, a bez poveznice pristup toj kartici je otežan. Stoga je Cunningham doradio program tako da je stvaranje poveznice postalo jednostavno poput pisanja riječi. Pisanje riječi unutar liste jednostavno je označavalo novu poveznicu kojom je odmah bio povezan s karticom istoga naziva. U slučaju da kartica ne postoji ona se mogla stvoriti jednostavno pritiskom na poveznicu. Tako je stvaranje novoga sadržaja postalo višestruko jednostavnije i zaposlenici su to prepoznali.



Slika 2 Rekonstrukcija *HyperText* kartica

3.2. Moderan ustroj Wiki sustava

Wiki je kolaborativna baza podataka i zamišljena je kao najjednostavnija moguća baza podataka. Osnovni način funkcioniranja svakog wiki sustava veoma je jednostavan. Sastoji se od stvaranja i održavanja sadržaja, a uređivanje sadržaja obavljaju sami korisnici. Tako imamo više korisnika koji rade na jednom članku. Oni su zaduženi za ispravnost, količinu i kvalitetu sadržaja. U današnje je vrijeme takva praksa veoma zastupljena. Korisnici diljem svijeta imaju mogućnost međusobnom suradnjom doprinositi sadržaju, a samim time omogućuje se i brži rast baze podataka.

4. TEHNOLOGIJE I ALATI

4.1. HTML

HTML (*engl. HyperText Markup Language*) standardni je markup jezik koji se koristi za stvaranje mrežnih stranica, odnosno za stvaranje korisničkog sučelja za mobilne i mrežne aplikacije. Mrežni preglednici (*engl. web browser*) mogu čitati HTML dokumente i prevesti ih u vizualne i čujne značajke koje čine mrežnu stranicu. Struktura stranice semantički je opisana i sadrži znakove za prezentaciju unutar preglednika.

Elementi poput naslova, paragrafa, listi, obrazaca i slika opisani su pomoću etiketa koristeći izlomljene zagrade. [Slika 3]

```
<!DOCTYPE html>
<html>
<head>
<title>Naziv stranice</title>
</head>
<body>
<h1>Naslov</h1>
<p>Paragraf</p>
</body>
</html>
```

Slika 3 Primjer HTML markup jezika

Razlikuju se dva tipa etiketa:

- **Sadržajne** – izravno ugrađuju sadržaj unutar stranice (poput slika, videa, formi itd.)
- **Opisne –** oblikuju sadržaj/tekst koji obuhvaćaju unutar sebe (poput naslova, paragrafa, poveznica itd.)

Etikete također mogu sadržavati i atribute koji izmjenjuju standardno ponašanje etikete.

```
<etiketa atribute="vrijednost">Sadržaj unutar etikete</etiketa>
```

Slika 4 Osnovna struktura etikete HTML jezika

Većina elemenata mogu sadržavati neke od najčešćih atributa:

- **id** – unikatni identifikator za element, na stranici se vrijednost tog atributa ne smije ponavljati unutar ostalih elemenata
- **class –** služi kao klasifikacija elemenata
- **style –** uređivanje stila elementa (visine, boje, obruba)
- **title –** atribut koji dodaje opisni tekst elementu, vidljiv kod prelaska miša preko elementa

4.2. CSS

Korištenjem CSS (*engl. Cascading Style Sheets*) upotpunjuje se vizualni dojam stranice stvorene pomoću HTML-a. Budući da HTML pruža strukturni prikaz s malim mogućnostima stiliziranja stvoren je kao jezik koji omogućuje dodatnu manipulaciju s HTML elementima.

Dizajniran je da bi se koristio kao zaseban dokument kako bi se odvojio sadržaj stranice od vizualne prezentacije, te se zato uglavnom poziva preko poveznice koja upućuje na smještaj tog dokumenta. Također je moguća i izravna ugradnja unutar HTML dokumenta, ali zbog bolje preglednosti preporučuje se poziv preko poveznice.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" type="text/css" href="stil.css">
    <style>
    #xyz { color: red; }
    </style>
  </head>
  <body>
    <p id="xyz" style="color: blue;"> Stilizirani paragraf </p>
  </body>
</html>
```

Slika 5 Primjer ugradnje stila unutar HTML dokumenta

Poziv preko poveznice definira se pomoću etikete *link* čija je vrijednost atributa *href* jednaka poveznici na datoteku. Korištenjem etikete *style*, stil se može izravno ugraditi. Posljednja je mogućnost korištenja atributa *style* unutar etikete koju se želi stilizirati. [Slika 5]

```
p {
      color: blue;
}

.klasa {
      border: 1px solid red;
}

#identifikator {
      background-color: lightblue;
}
```

Slika 6 Primjer CSS jezika za stiliziranje

Osnovna ideja jezika temelji se na selektorima koji pokazuju na HTML elemente nad kojima se primjenjuju izmjene koje odgovaraju opisu stila. Iznimno u slučaju korištenja atributa *style* unutar etikete, selektor nije potreban već se stil primjenjuje na tom elementu.

Selektor koji nosi naziv etikete označava sve etikete tog tipa unutar HTML dokumenta, nakon čega primjenjuje stil definiran u vitičastim zagradama. Ukoliko se ispred naziva selektora nalazi točka, utoliko ona označava sve elemente čiji atribut *class* sadrži vrijednost tog naziva selektora. Slično vrijedi i za korištenje ljestvi ispred naziva selektora, ali se u tom slučaju naziv selektora odnosi na elemente koji sadrže atribut *id* s tom vrijednošću.

4.3. Web2py

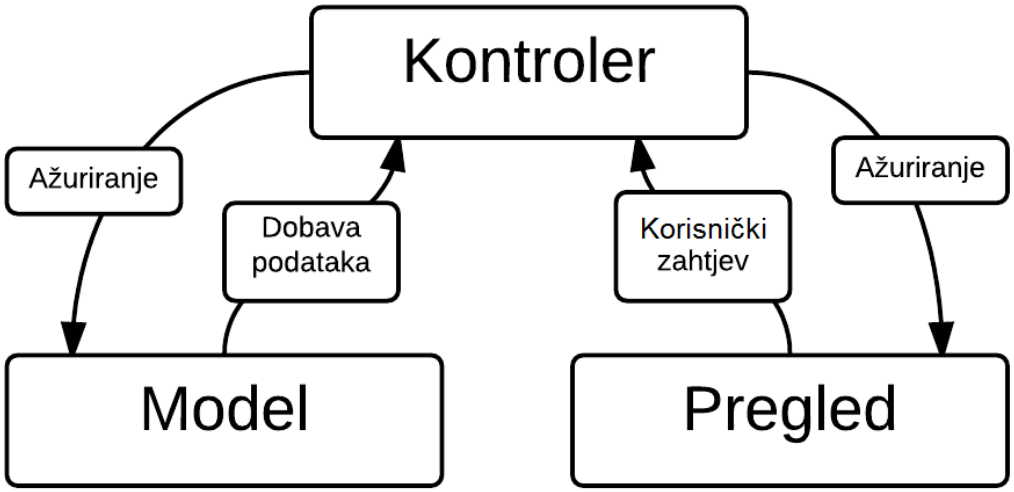
Projekt Web2Py započeo je 2007. godine prof. Massimo Di Pierro kao edukativni alat s ciljem web razvoja na brži, jednostavniji i sigurniji način. Razvijan je kao mrežni okvir (*engl. web framework*) otvorenog koda. Dobitnik je nagrade Bossie Award 2010. godine kao najbolji softver otvorenog koda, dok 2012. godine dobiva nagradu *Technology of the Year* od InfoWorld izdavača.

Kod izrade neke mrežne aplikacije web2py mrežni okvir koristi se na serverskoj strani. On omogućuje obradbu SQL baznih upita na serveru, povezivanje s bazom i generiranje HTML stranica koje se prikazuju korisniku aplikacije.

Web2py je izgrađen na MVC (model-view-controller) arhitekturi koja se temelji na načelu odvajanja programskih komponenti aplikacije ovisno o njihovoj namjeni. Takav dizajn olakšava razvoj kompleksnijih aplikacija, lako se održava, ispravlja i mijenja.

MVC se sastoji od:

- **Modela** (*engl. model)* – to su podatci aplikacije, u ovom to je baza podataka
- **Pregleda** *(engl. view)* **–** prikaz dohvaćenih podataka na mrežnoj stranici
- **Kontrolera** *(engl. controller***) –** upravljanje podatcima i interakcija s korisničkim zahtjevima



Slika 7 Tok informacije unutar MVC strukture

4.4. Javascript

Korištenje mrežnih stranica svodi se na slanje zahtjeva prema serveru i dohvaćanje odgovora na taj zahtjev. Tako za svaku promjenu prikaza trebamo proći taj proces koji se u cijelosti provodi na strani servera. Prilikom primanja takvog zahtjeva server počinje s obradom zahtjeva na način da vidi koja se informacija očekuje kao odgovor. Nakon toga obradom podataka, dobavljenih od strane korisnika ili baze, dolazi do konačnog rješenja koje se prezentira davaocu zahtjeva kao odgovor. Takav je proces obrade diskretiziran u vremenu i kao takav potražuje mnogo zahtjeva prema serveru. Za svaku promjenu podatka unutar baze ili unutar sučelja korisnika potrebno je stvoriti novu HTML stranicu koja unutar sebe sadrži željeni podatak. Zbog toga imamo mnogo redundantnih dohvaćanja stranica koje su mogle biti samo uređene na strani korisnika. Korištenje takvog sustava za korisnika bi bilo intuitivno i ne bi iziskivalo vrijeme čekanja učitavanja stranice.

Aplikacija koja će se opisati u ovom radu bit će u mogućnosti spremati statističke podatke i prikazati ih. Statistički podatci obično dolaze kao velike baze podložne promjenama i redovitim novim unosima. Stoga će se za ovu aplikaciju ugraditi mogućnost dinamičkog rada na tim podatcima, a sve u svrhu lakšeg korištenja aplikacije prilikom manipulacije podatcima.

Za izradu takve mogućnosti potrebno je koristiti neki od programskih jezika koji će se odvijati na strani klijenta. U ovom slučaju koristit će se JavaScript. JavaScript se izvodi u pregledniku korisnika i potpuno je neovisan od servera, a također je u mogućnosti slati zahtjeve prema serveru koji se u potpunosti odvijaju u pozadini bez znanja korisnika. S takvom mogućnošću ne prekida se korisnika pri upotrebi aplikacije, a opet postoji komunikaciju prema serveru.

JavaScript kao skriptni jezik omogućuje mogućnost izvođenja programa na strani klijenta i može se koristiti u njegovom izvornom obliku, no to bi nepotrebno otežalo razvoj i preglednost aplikacije sa stajališta njezine izrade. Koristit će se neki od dostupnih zbirki funkcija ili okvira koji će izradu aplikacije znatno olakšati. U ovom će se slučaju koristiti JavaScript framework zvan AngularJS.

4.4.1. AngularJS

AngularJS je strukturni framework za izradu dinamičnih mrežnih aplikacija. Pisan je pomoću JavaScripta i kao takav potpuno se izvodi na klijentskoj strani. Dizajniran je s idejom CRUD (*engl. Create Read Update and Delete*), što bi predstavljalo jednostavne aplikacije dizajnirane za izradu, čitanje, ažuriranje i brisanje podataka, bez obrade podataka koja bi se odvijala na strani klijenta.

Kako je HTML dizajniran za izradu statičnih stranica, pomoću njega je stvaranje dinamičkih aplikacija nemoguće. Koristeći AngularJS može se proširiti funkcionalnost HTML elementa na takav način da ih se može dinamički obrađivati i dodavati nove mogućnosti koje inače ne postoje koristeći HTML.

Neke od značajki AngularJS su tzv. povezivanje podataka, vlastiti sustav za stvaranje i upravljanje komponentama, podrška za upravljanje i validaciju obrazaca, korištenje direktiva itd.

4.4.2. D3JS

D3JS je JavaScript skup funkcija za manipulaciju dokumentima nad prikupljenim podatcima. D3JS omogućuje jednostavno vizualiziranje podataka pomoću HTML, SVG i CSS tehnologija. Optimiziran je kako bi što bolje iskoristio mogućnosti modernih mrežnih preglednika.

D3JS povezuje proizvoljan skup podataka s objektima dokumenta nakon čega primjenjuje transformacije dokumenta prema tim podatcima. Kao primjer može se navesti generiranje vektorske grafike (SVG) za stvaranje stupčastih grafikona gdje je visina elementa izravno vezana uz vrijednost podatka. Alat omogućuje glatke prijelaze između podataka, a također i interaktivnost.

Zbog korištenja mrežnog standarda za povezivanje podataka s elementom unutar dokumenta pokazao se bržim i pouzdaniji s obzirom na mrežni okvir koji se oslanjaju na DOM (*engl. Document Object Model*) pretragu za ažuriranje atributa. Zbog toga je povoljan za korištenje s velikim skupovima podataka i za njihovo dinamičko ažuriranje i animiranje.

Selektiranje elemenata koristeći W3C DOM API temelji se na sistematičnoj iteraciji kroz DOM stablo. Iziskuje programsko praćenje trenutnog stanja, dok su metode sintaksno zamorne. D3JS način rada deklarativan je kada se operacije odvijaju na proizvoljnim točkama unutar DOM stabla.

Kao primjer može se usporediti ažuriranje boje svih <p> elemenata unutar HTML dokumenta.

Primjer za W3C DOM API [13Slika 8]

```
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
  var paragraph = paragraphs.item(i);
  paragraph.style.setProperty("color", "white", null);
}
```

Slika 8 Primjer JavaScript jezika prema W3C DOM API

Primjer za D3JS API [Slika 9]

```
d3.selectAll("p").style("color", "white");
```

Slika 9 Primjer JavaScript jezika s D3 API

Iz ovoga je vidljivo da je D3JS API jednostavniji i ne zahtijeva od programera pisanje dodatnog algoritma za pretraživanje.

Za iscrtavanje grafikona unutar mrežnog preglednika koristi se D3JS zajedno s C3JS skupom funkcija. Kako je D3JS prilagođen za općenito iscrtavanje vektorske grafike, uz njega se koristi još i C3JS skup funkcija koje su prilagođene za izradu elemenata koji karakteriziraju upravo grafikone.

5. APLIKACIJA – projektirani i izvedeni programski alati

Aplikacija zamišljena i stvorena u ovom diplomskom radu objedinjava Wiki paradigmu s alatima poslovne inteligencije. U njoj su korištene mnogobrojne mrežne tehnologije, pa je dohvatljiva preko mreže pomoću mrežnih preglednika (*engl. browser-a*), dok se izvođenje aplikacije događa na jednom ili više udaljenih servera.

Na serverskoj strani koristi se web2py mrežni okvir koji poslužuje korisnika, a u kojem je čitava aplikacija razvijena. Korisničko sučelje dijelom je složeno od statičkih stranica generiranih na serveru, a dijelom su to kao zasebne aplikacije pisane pomoću JavaScript programskog jezika. Prednost takvih aplikacija u tomu je što se izvršavaju unutar preglednika na korisničkoj strani i ne opterećuju server, čime se postiže veća brzina izvođenja. Istodobno se dopunjuje korisničko sučelje s dinamičkim, promjenljivim elementima aplikacije, što za posljedicu daje znatno povećanu produktivnost korisnika.

Za izradu aplikacija koje se izvode na korisničkoj strani (u pregledniku) koristi se AngularJS, Javascript mrežni okvir (*engl. framework*) za interakciju s korisnikom i D3JS za iscrtavanje vektorske grafike unutar preglednika.

5.1. Pregled strukture aplikacije

Osnova je Wiki stranice članak koji sadrži informacije o temi i reference na druge članke. Za stvaranje i uređivanje članaka koristi se Web2py Wiki modul. Unutar tog modula sadržani su obrasci za unos teksta i autorizaciju korisnika, rad s bazom i sl. Osnovna struktura članka proširena je s alatima karakterističnim za poslovnu inteligenciju.

Članak se sastoji od naslova, sadržaja i ključnih riječi. Uz članak mogu biti vezane meta stranice koje upotpunjuju informacije unutar članka.

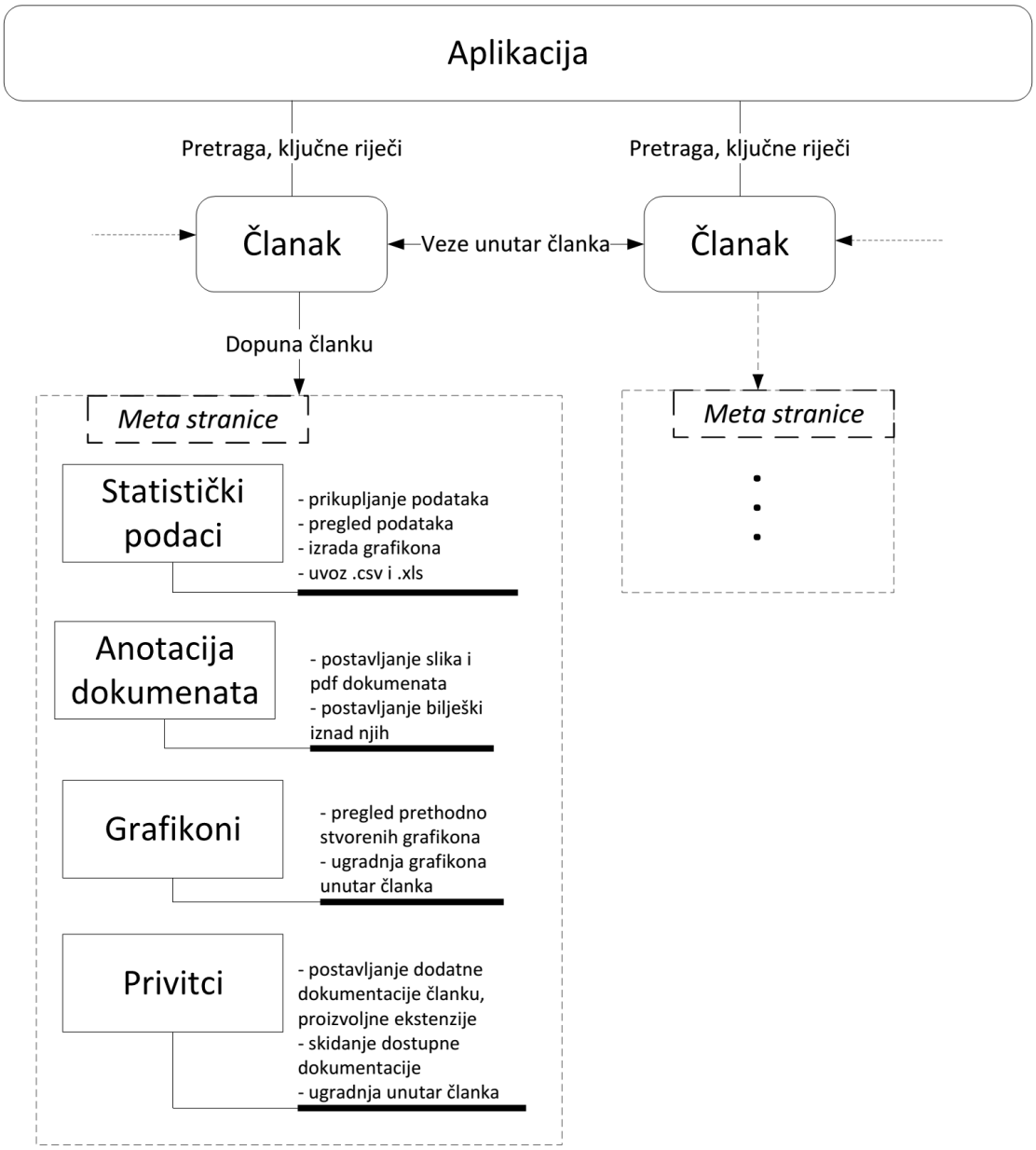
Meta stranice koje mogu biti povezane s člankom su:

- **Privitci** – dodavanje dokumenata i ostalih datoteka koje su vezane uz članak
- **Statistički podatci** – dodavanje i uređivanje statističkih podataka koji su vezani uz tematiku članka
- **Grafovi** – pregled stvorenih grafikona

- **Označene slike i označeni PDF dokumenti** – stvaranje i spremanje po slojevima označenih digitaliziranih slika ili PDF dokumenata s pripadajućim vizualnim oznakama.

Korisnici su podijeljeni u dvije skupine:

- **urednik** – ima sve ovlasti nad načinjenim sadržajem drugih korisnika, može uređivati njihove članke i privitke
- **autor** – običan korisnik koji ima dopuštenje stvoriti i uređivati vlastiti sadržaj
- **posjetitelji** – korisnici koji mogu gledati mrežne stranice aplikacije, ali ih ne mogu mijenjati.



Slika 10 Struktura aplikacije

5.2. Sučelje

Sučelje Wiki aplikacije na vrhu zaslona ima navigacijsku traku. S njezine lijeve strane nalazi se poveznica za navigaciju kroz članke i upravljanje člancima, dok se s desne strane nalazi padajući meni za autorizaciju korisnika. Meni za autorizaciju korisnika sadrži poveznice na

stranice za prijavljivanje već postojećeg korisnika, registraciju novog korisnika i mogućnost traženja nove lozinke za već postojećeg korisnika.

Navigacijski dio trake sastoji se od poveznice *Početna* koja vraća korisnika na početnu stranicu, koja je inicijalno postavljena na članak s imenom *Index*. Drugi dio navigacijske poveznice prikazuje padajući meni pod imenom *Meta stranice*, unutar kojeg se nalaze poveznice na *Podatci*, *Grafovi* i *Označivanje*. Tu je za administratora i prijavljene korisnike niz poveznica koje se mogu urediti iz posebnog članka unutar Wiki baze koji se zove *Wiki Menu*. Posebnost je *Wiki Menu* članka u tome što je članak integriran unutar navigacijske trake. Zbog toga se njegov sadržaj može oblikovati kao lista s poveznicama, što će kao rezultat dati poveznice u navigacijskoj traci. Na koncu u istoj traci postoji padajući meni [Wiki] koji sadrži opcije vezane uz upravljanje s Wiki alatom kojim se stvara bilo koji članak.

Slika 11 Sučelje aplikacije

Unutar menija [Wiki] postoje ove poveznice:

- **Uredi Stranicu** – otvara sučelje za uređivanje članka na kojem se korisnik trenutno nalazi
- **Uredi Datoteke Stranice –** poveznica na stranicu s datotekama koje su pridodane članku, a gdje se također mogu postaviti i nove datoteke

- **Stvori Novu Stranicu** – preusmjerava korisnika na obrazac za stvaranje novog članka
- **Pregled Stranica** – prikazuje se u slučaju da je prijavljeni korisnik član grupe urednik i vodi na stranicu za administriranje članaka
- **Uredi Meni** – otvara članak *Wiki Menu* koji služi za definiranje dodatnih poveznica unutar navigacijske trake
- **Pretraži Stranice –** prikazuje stranicu s tražilicom za pretraživanje članaka.

Ispod navigacijske trake nalaze se lijeva bočna traka i tijelo stranice. U lijevoj bočnoj traci nalazi se obrazac za pretraživanje članaka prema ključnim riječima. Ispod tog obrasca nalaze se najzastupljenije ključne riječi. Pritiskom na neku od tih ključnih riječi pokreće se pretraga s tim upitom. Ključne riječi pojavljuju se u ovom skupu ovisno o njihovoj zastupljenosti kroz članke. Njihova veličina (font s kojim su napisane) naglašava zastupljenost (frekvenciju) pojavnosti riječi u odnosu na ostale. Ako se ključna riječ pojavljuje u više članaka, bit će prikazana u fontu s više točaka (piksela).

Tijelo stranice sadrži članak iznad kojeg se nalazi sekundarna navigacijska traka.

Sekundarna navigacijska traka grafički je upravljački element koji korisniku omogućuje praćenje pozicije ispisa sadržaja tijela stranice. Ona prikazuje putanju ispisane informacije s obzirom na njeno mjesto unutar aplikacije.

Slika 12 Sekundarna navigacijska traka

Na slici 3. prikazana je sekundarna navigacijska traka s više dubina. U ovom slučaju korisnik vidi radni list pod imenom *Registrirana vozila 2013* na meta stranici *Podatci* koja je vezana uz članak *Hrvatska*. Osim uvida u položaj gdje se korisnik nalazi, ona također služi za brzu navigaciju unutar te strukture, što znači da se korisnik može pritiskom lijeve tipke miša na neku od poveznica unutar te trake automatski preusmjeriti na mjesto koje pokazuje poveznica.

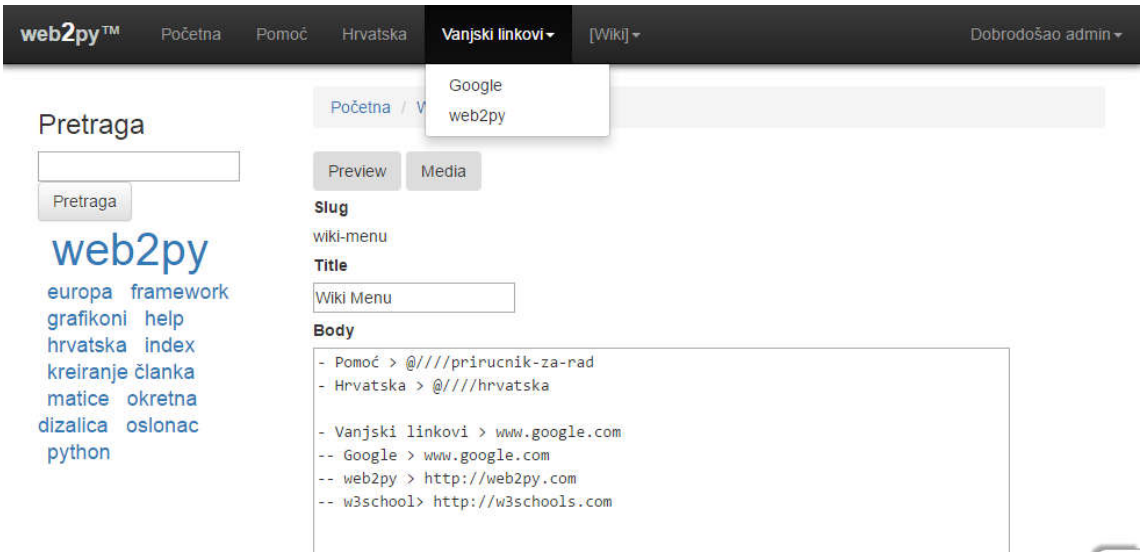
5.2.1. Uređivanje menija

Navigacijsku traku u vrhu aplikacije moguće je dodatno uređivati na način da joj se dodaju poveznice ili padajući meni. Stvaranje dodatnih elemenata u navigacijskoj traci omogućeno je

samo korisnicima koji su dio grupe korisnika „urednik“. Ako korisnik zadovoljava taj uvjet, nakon autorizacije njegov padajući meni *[Wiki]* sadržavat će dodatnu poveznicu *Uredi meni*. Odabirom te poveznice otvara se stranica za uređivanje članka *Wiki Menu*. Karakteristika tog članka u tomu je što je on integriran unutar navigacijske trake.

Da bi se uspješno dodala poveznica potrebno je pravilno formatirati taj članak. Ukoliko korisnik želi dodati padajući meni u navigacijskoj traci, ukoliko u članku on mora koristit markmin sintaksu za stvaranje lista. Svaki unos unutar liste prepoznat će se kao element menija koji će se ugraditi u navigacijsku traku. U slučaju druge razine lista ona će biti prepoznata kao padajući meni i tako će se ugraditi unutar navigacijske trake. [Slika 13]

Prilikom stvaranja poveznica potrebno je koristiti sintaksu *Naziv > poveznica* gdje je *Naziv* željeno ime poveznice koje će se prikazati korisniku, a *poveznica* je mrežna adresa na koju će korisnik biti preusmjeren. Moguće je koristiti poveznice na druge stranice i markmin sintaksu za stvaranje poveznica na članke stvorene unutar aplikacije.

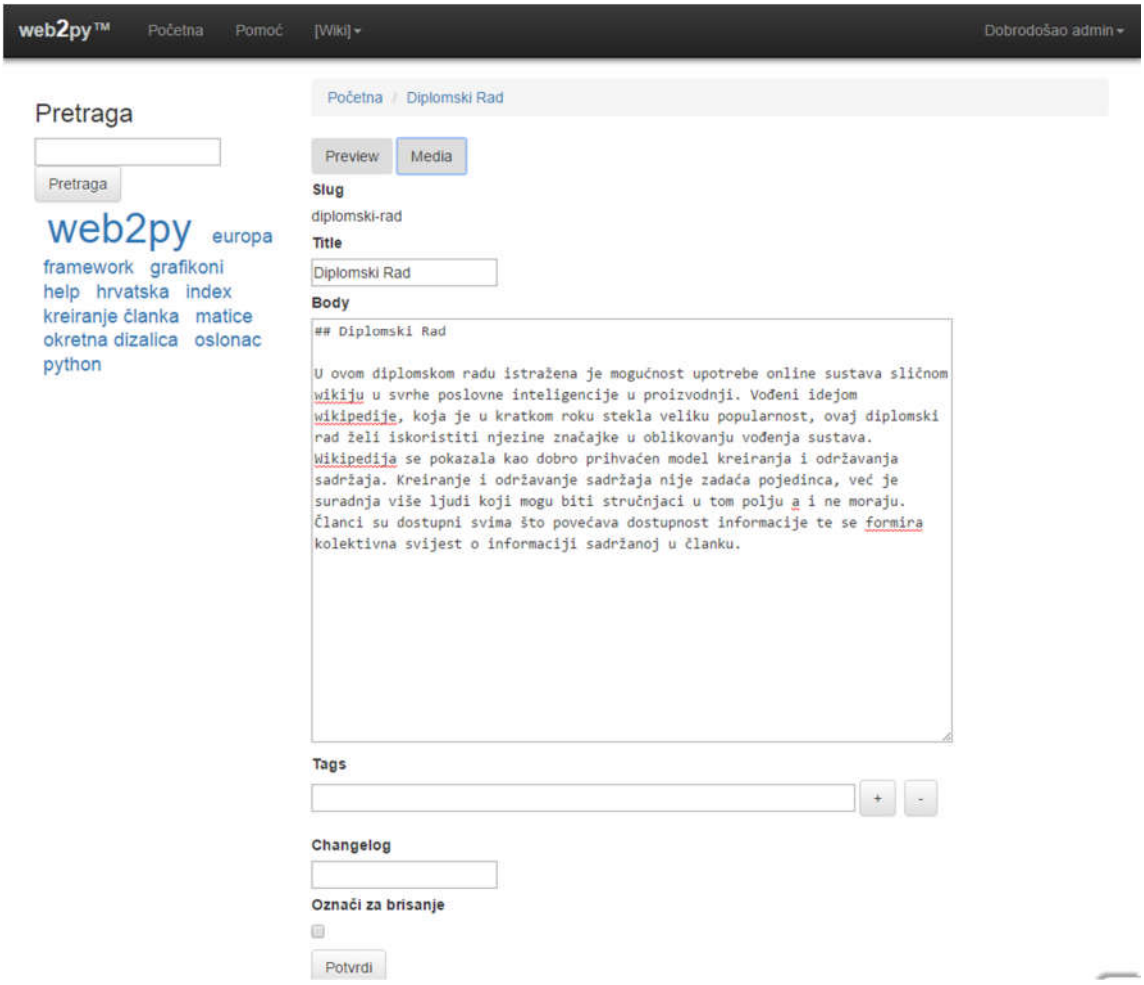


Slika 13 Uređivanje *Wiki Menu* članka za dodatnu navigaciju

5.3. Članci

5.3.1. Stvaranje članka

Stvaranje novog članka moguće je iz padajućeg menija *[Wiki]* pritiskom na poveznicu *Stvori novu stranicu*. Korisnik odabire naziv stranice i potvrđuje obrazac, nakon čega se preusmjerava na uređivanje novog članka.



Slika 14 Prikaz uređivanja članka

5.3.2. Markmin

Kod uređivanja članka koristi se markmin sintaksa koja omogućuje promjenu izgleda članka, stvaranje poveznica, integriranje slika i slično.

Tablica 1. Prikaz Markmin sintakse

Markmin sintaksa	Prikaz u pregledniku
# Naslov Tekst članka	Naslov Tekst članka
## Odjeljak Tekst članka	Odjeljak Tekst članka
### Pododjeljak Tekst članka	Pododjeljak Tekst članka
podebljani tekst	**podebljani tekst**
"nakošeni tekst"	*nakošeni tekst*
~~precrtani tekst~~	~~precrtani tekst~~
``crvena boja teksta s **podebljanim** dijelom``:red	crvena boja teksta s **podebljanim** dijelom
``plava boja teksta sa žutom pozadinom``:color[blue:#ffff00]	plava boja teksta sa žutom pozadinom

Za postavljanje poveznica unutar članka koristi se sintaksa s dvije uglate zagrade [[naziv poveznica]]. Poveznica može biti vanjska ili između članaka. U slučaju da se koristi poveznica između članaka, dovoljno je koristiti skraćenu poveznicu u obliku *@////slug* gdje je *slug* jedinstveni identifikator za svaki članak. Unutar članka također je moguće postaviti reference prema drugom dijelu članka u dva koraka:

1. Načini se *anchor* [[naziv]] na odredištu, mjestu gdje se želi poveznicom doći
2. U ishodištu, odakle se želi stići na odredište, postavlja se poveznica [[veza-na-drugi-dio-članka #naziv]].

Datoteke koje sadrže ekstenzije mp3, mp4 ili neku od ekstenzija za slike bit će automatski prepoznate i integrirat će se ovisno o tipu. Dokumenti koji imaju neku od ekstenzija koje su podržane od strane *Google Doc Viewer* bit će integrirane unutar članka tako da je moguć njihov pregled. Dovoljno je na mjesto gdje se želi postaviti preglednik staviti izravnu poveznicu do dokumenta, bez Markmin sintakse. Formule je moguće integrirati s LaTeX sintaksom $$formula$$. Stvaranje QR koda moguće je pomoću sintakse qr:poveznica, nakon čega će se poveznica prikazati kao QR kod.

5.4. Statistički podatci

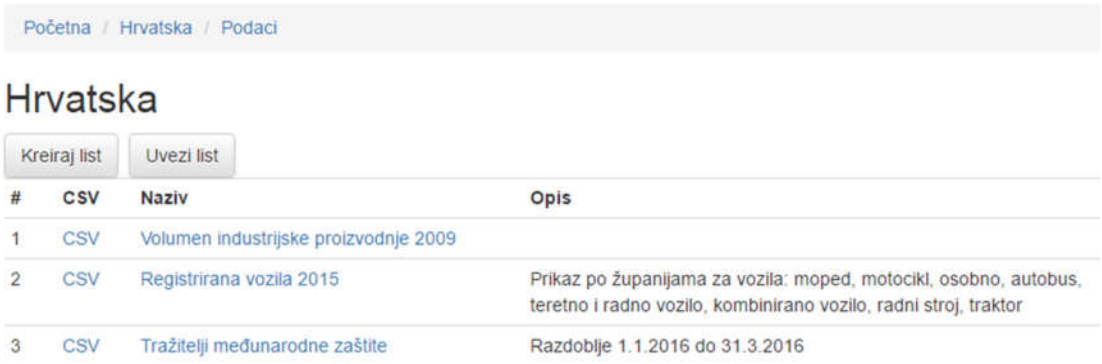
Skupljanje podataka jedna je od važnijih aktivnosti poslovne inteligencije, pa je zato potrebno osigurati alate koji omogućuju korisniku spremanje podataka unutar baze i njihovo dohvaćanje. Takvi alati trebaju biti što jednostavniji i intuitivniji u svome korištenju kako bi korisnik brzo svladao njihovu upotrebu.

Alat stvoren za prikupljanje podataka načinjen je prema zamisli sličnoj programu Excel, gdje se podatci prikupljanju, prikazuju i obrađuju u prikladnim radnim listovima koji se sastoje od tablica s više stupaca i redaka. Ispunjavanje radnog lista odvija se u potpunosti unutar Internet preglednika.

Osim ručnog unosa moguće je također uvesti radni list iz datoteka koje podržavaju tablični zapis. Podržani formati za uvoz su csv i xls.

Skup podataka zapisanih u jednoj tablici u daljnjem će se tekstu nazivati radni list. Radni list definiran je svojim nazivom, opisom i tablicom koja sadrži podatke. Veže se uz članak, što znači da je radni list dostupan kao meta stranica tog članka i posjeduje informacije koji najčešće imaju korelaciju sa sadržajem članka.

Za pregled podataka nekog članka prvo je potrebno otvoriti taj članak, a zatim se može iz padajućeg menija *Meta stranice* odabrati poveznica *Podatci*. Nakon toga korisnik će biti odveden na stranicu unutar koje će moći načiniti novi prazni list ili uvesti radni list iz datoteke, te dobiti pregled prethodno stvorenih radnih listova.



Slika 15. Meta stranica podatci

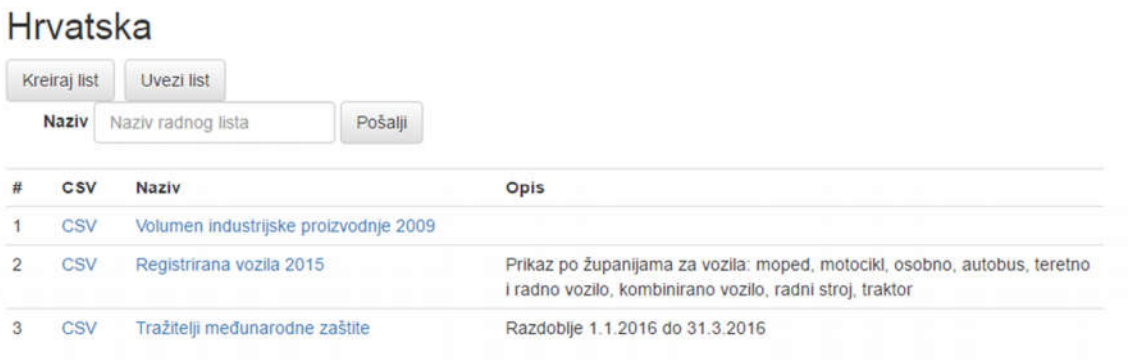
Prethodno stvoreni radni listovi prikazani su kao numerirana lista, a svaki je označen svojim nazivom i opisom. Također, svaki radni list podržava izvoz u datoteku *csv* formata. Pritiskom na poveznicu *CSV* uz naziv radnog lista, kako se vidi na slici 5., unutar preglednika će se otvoriti

datoteka koja će sadržavati svu informaciju (unesene tablične podatke) tog radnog lista. Ta datoteka može se naknadno pregledavati ili uređivati bilo kojim programom koji podržava csv format.

5.4.1. Stvaranje novog radnog lista

Moguća su dva načina stvaranja novog radnog lista. Prvi je pokretanje praznog lista koji će korisnik popuniti, a drugi je način uvoz podataka iz datoteke s računala.

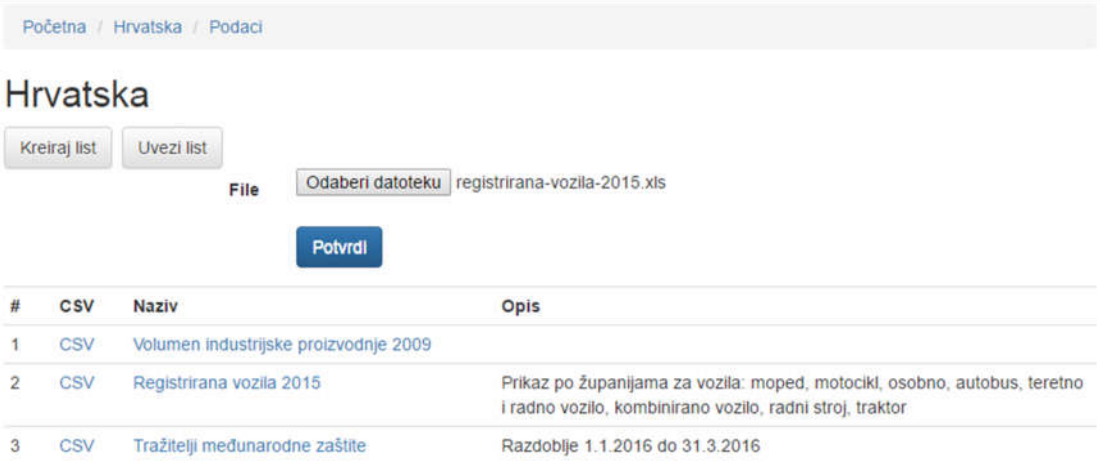
U slučaju da korisnik želi stvoriti prazan list dovoljno je kliknuti na gumb *Kreiraj list*, nakon čega će se prikazati polje za unos naziva. Nakon što je odabran naziv radnog lista, potvrđuje se odabir pritiskom na gumb *Pošalji.*



Slika 16. Meta stranica podatci s otvorenim obrascem za stvaranje radnog lista

5.4.2. Uvoz radnog lista

Ako korisnik posjeduje podatke u jednoj od podržanih datoteka csv ili xls formata, moguće je klikom na *Uvezi list* načiniti upload (učitavanje) te datoteke u aplikaciju na server. Pritisak na gumb *Uvezi list* otvara obrazac s mogućnošću odabira datoteke koja se nalazi na računalu. Nakon što je korisnik odabrao datoteku, potvrđuje svoj odabir na gumbu *Potvrdi.*



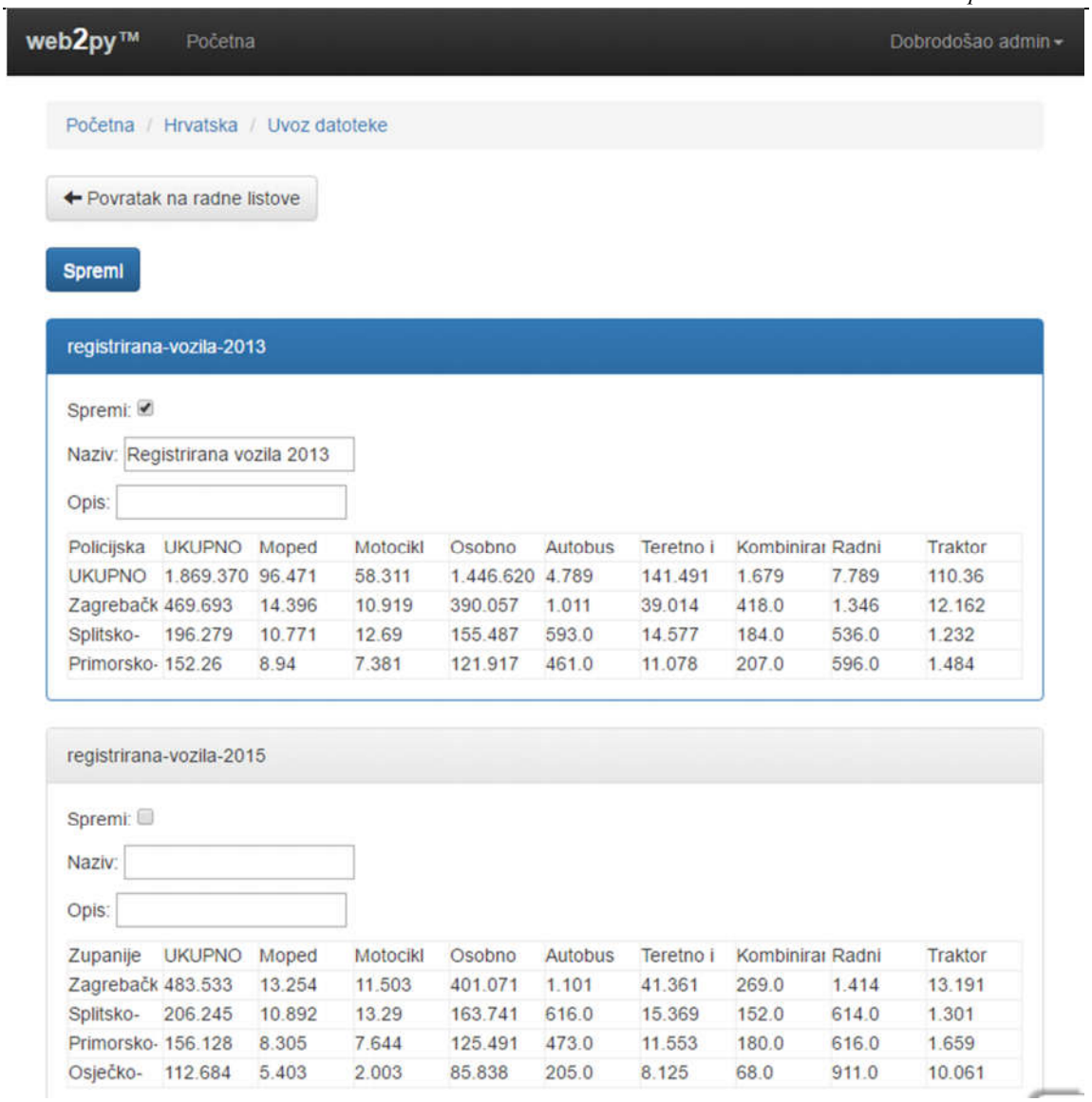
Slika 17. Meta stranica podatci s otvorenim obrascem za uvoz datoteke

Nakon što korisnik potvrdi učitavanje datoteke preusmjeren je na stranicu koja pruža kratak pregled te datoteke. Prilikom pregleda datoteke korisnik ima uvid u stanje i sadržaj podataka datoteke, na temelju čega može donijeti odluku je li datoteka pogodna za uvoz. U slučaju da se želi odustati od uvoza dovoljno je kliknuti na gumb *Povratak na radne listove,* te će biti preusmjeren natrag na stranicu *Podatci.*

Kod xls formata karakteristično je da mogu sadržavati više tablica koje su neovisne jedna o drugoj. U tom se slučaju korisniku daje mogućnost odabira tablica koje će se spremiti u bazu. [Slika 18]

Odabir tablice vrši se označivanjem (*engl. checking*), tj. označaavnjem kvačice u okvir *Spremi*. Označena tablica dodatno je naglašena plavim okvirom i zaglavljem.

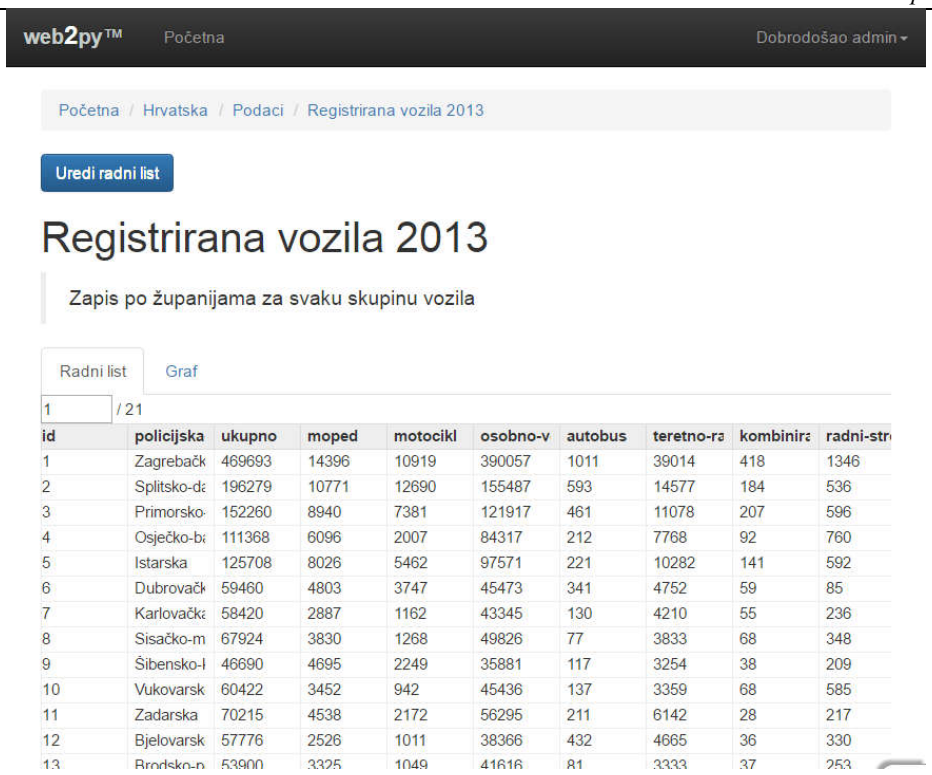
Nakon što su odabrani nazivi i označeni radni listovi, pritiskom na gumb *Spremi* stvorit će se novi radni listovi s popunjenim vrijednostima ćelija i korisnik će biti preusmjeren na pregled radnog lista.



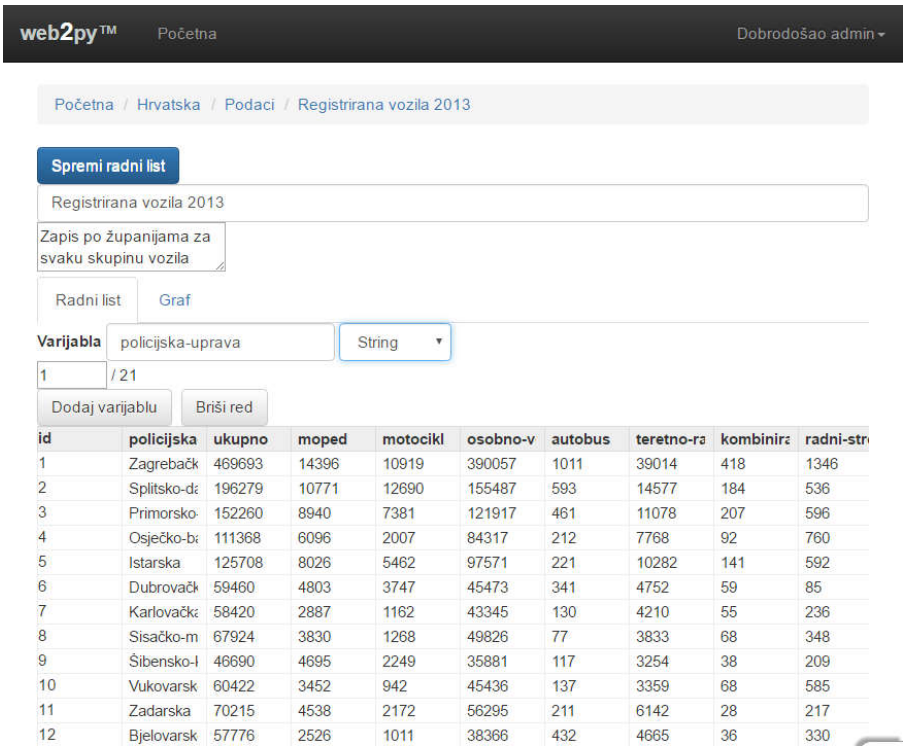
Slika 18. Pregled radnog lista prije uvoza

5.4.3. Uređivanje radnog lista

Prilikom pregledavanja radnih listova korisnicima s određenim privilegijama omogućeno je uređivanje tih listova. Uređivanje se odvija unutar istog prozora gdje je moguć i pregled te se korisnicima koji uređuju, prikazuju dodatne mogućnosti uz osnovnu tablicu koja služi za pregled. [Slika 20]



Slika 19 Pregled radnog lista unutar mrežnog preglednika



Slika 20 Uređivanje radnog lista

Korisnici koji su član grupe urednik ili autor radnog lista posjeduju tu mogućnost. U slučaju da korisnik zadovoljava jedan od tih uvjeta, prezentirat će mu se gumb *Uredi radni list* čijim pritiskom pokreće uređivanje. [Slika 19]

Kod uređivanja je moguć ručni unos podataka, definiranje varijabli, brisanje redova i konačno spremanje tih promjena.

Unos podataka odvija se ručno na način da se u zadnji red unutar ćelija unose podatci. Ovisno o stupcu u kojem se nalazi ćelija podataka unutar ćelije će biti vezan uz varijablu stupca.

5.4.3.1. Definiranje varijabli

Naziv varijable vezane uz stupac nalazi se u samom vrhu tablice i dodatno je naglašena debljim slovima. Moguće ju je mijenjati, ali nije poželjno jer postoji mogućnost da je vezana uz neki prethodno stvoreni grafikon. Nova varijabla definira se dodavanjem stupca pritiskom na gumb *Dodaj stupac*, nakon čega se upiše željeni naziv. Kod izbora naziva varijable važno je slijediti neke smjernice kako bi se osiguralo da varijabla bude spremljena unutar baze u pravilnom formatu.

Varijable ne smiju sadržavati dijakritička slova niti razmake. Također nisu dozvoljeni posebni znakovi, izuzev crtice i donje crte. Varijabla može biti definirana s velikim i malim slovima u čijem se slučaju raspoznaje razlika između dvije varijable jednakog naziva s različitom veličinom slova. Poželjno je koristiti neki standard formatiranja koji će se slijediti preko cijelog radnog lista.

Primjer nekih formata su:

- **novaVarijabla** – format zvan *camel case*, gdje se razmak briše i prvo se slovo nakon razmaka piše velikim slovom
- **nova-varijabla** – crtica umjesto razmaka
- **nova_varijabla –** donja crta umjesto razmaka

Važno je naziv varijable izabrati na način da bude jedinstven za radni list u kojem se nalazi. Ako dođe do ponavljanja varijabli, korisnik neće moći raspoznati razliku između njih dvije.

Osim naziva, varijabla je također definirana svojim tipom. Tip varijable odnosi se na vrstu podatka koji sadrži.

Razlikuju se četiri tipa varijable:

- **Broj** – podatci vezani uz tu varijablu su tipa broja i mogu biti cijeli ili racionalni brojevi; sustav ne prepoznaje razliku između njih
- **Tekst** – stupac sadrži podatke u obliku teksta, poput naziva. U slučaju da se koriste brojevi unutar ćelija tog stupca oni će isto biti smatrani tekstom
- **Vrijeme** – predviđa da će se koristiti neki format zapisa koji će se odrediti kod definiranja varijable
- ***Boolean*** – podatak poprima samo jednu od dvije vrijednosti *True* ili *False.*

Dodatnim definiranjem tipa varijable nastoji se postići bolja sistematizacija podataka koja se kasnije može upotrijebiti prilikom obrade podataka.

5.5. Grafikoni

Prikaz podataka unutar tablica strukturiran je i detaljan. Korisnik koji želi izvući neku informaciju treba proučiti svaki podatak i stvoriti veću sliku o cijelom skupu. Ljudi su sposobni za apstraktno razmišljanje i zbog toga mogu mnogo lakše izvući informaciju kada im se predstave podatci u vidu grafikona. Takvom vizualizacijom statističkih podataka mogu se uočiti trendovi i uzorci koji se inače ne bi mogli razabrati bez detaljne statističke analize.

Grafikoni prikazuju međuodnos dviju ili više promjenljivih veličina.

Budući da u poslovnoj inteligenciji postoji potreba za upotrebom grafikona, bilo je potrebno načiniti prikladne programske module i u ovom radu.

Grafikoni stvoreni unutar ove aplikacije mogu se podijeliti u dvije osnovne kategorije **–** na dinamičke i na statičke grafove. Ovisno o potrebi korisnika mogu se u trenutku stvaranja grafikona definirati hoće li grafikon biti dinamički ili statički. Svaka vrsta ima svoju funkciju i prednost, a na korisniku je da donese odluku koja kategorija više odgovara njegovim zahtjevima.

Dinamički su grafikoni takvi grafikoni koji će se prilikom svakog učitavanja stranice ažurirati, ovisno o stanju radnog lista uz koji su vezani. S prikazom uvijek ažurnih podataka grafikoni vjerno prikazuju trenutno stanje nekog sustava ili elementa koji se prati unutar radnog lista. Uređivanjem radnog lista, tj. podataka u njemu, neizravno se uređuje i grafikon koji će se dinamički prikazati.

Statički grafikoni spremaju se sa stanjem u trenutku njihova stvaranja, što znači da prikazuju neko prošlo stanje radnog lista. U Wiki se članku mogu prepoznati po tomu što je uz opis grafikona ispisano i vrijeme kada je grafikon spremljen.

5.5.1. Izrada grafikona

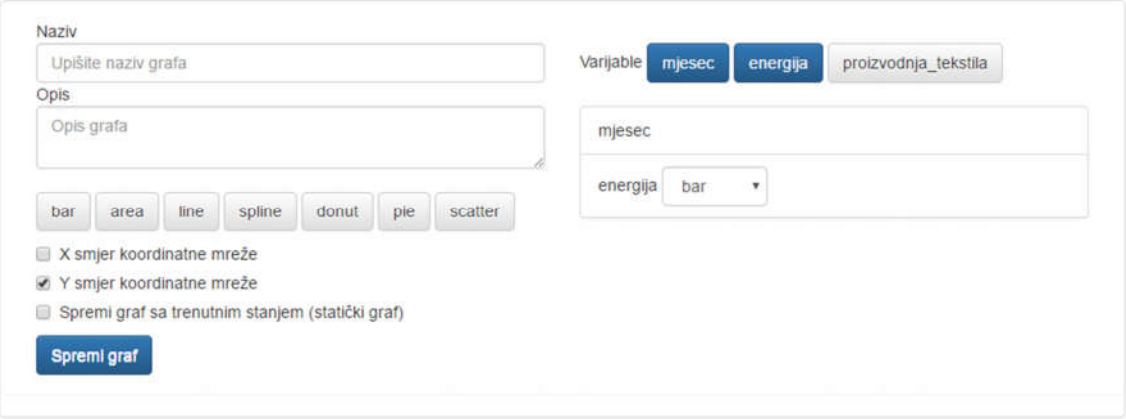
Cilj izrade grafikona je da se iz sakupljenih statističkih podataka vizualizira neki proces. Stoga se i stvaranje grafikona veže uz prethodno sakupljene podatke koji se nalaze spremljeni u obliku radnih listova. Ukoliko se želi stvoriti novi grafikon, potrebno je otvoriti radni list i učitati ili unijeti podatke koji se žele vizualizirati.

Odabirom kartice *Graf* prikazat će se alat čija je svrha definiranje opcija za stvaranje grafikona. Kartica se sastoji od dva dijela. Na vrhu kartice nalazi se prvi dio s upravljačkom pločom za odabir opcija. Drugi dio nalazi se ispod upravljačke ploče i unutar njega se iscrtava grafikon prema odabranim opcijama. Prilikom svake promjene neke od opcija, iscrtani grafikon će se ažurirati i u svakom će trenutku nacrtani graf biti rezultat odabranih opcija i stanja radnog lista.

U slučaju da se grafikon ne prikazuje, razlog može biti da je radni list prazan ili se kombinacijom odabranih opcija ne može iscrtati grafikon. Kao primjer može se navesti slučaj kada se za ordinatu odaberu vrijednosti koje se ne mogu numerički prikazati.

5.5.1.1. Upravljačka ploča

Unutar upravljačke ploče stvaranje grafikona započinje odabirom naziva i opisa, nakon čega se definiraju podatci koji će se koristiti za stvaranje grafikona. Podatci unutar tablice vežu se uz varijable **–** imena stupaca tablice u radnom listu. Pa se zato u prikazu grafa koriste iste te varijable preko kojih su definirani podatci koji se žele vizualizirati. Varijable koje su definirane unutar radnog lista bit će vidljive i u upravljačkoj ploči za izradu grafikona. Prikazane su kao izbornik s gumbima koje korisnik može označiti. Klikom na gumb željene varijable, on će poprimiti aktivno (uključeno) stanje. To je stanje istaknuto plavom bojom i označava varijablu **–** ime stupca tablice iz kojeg će se dohvaćati željeni podatci. Ista ta varijabla prikazuje se i unutar liste ispod izbornika varijabli.



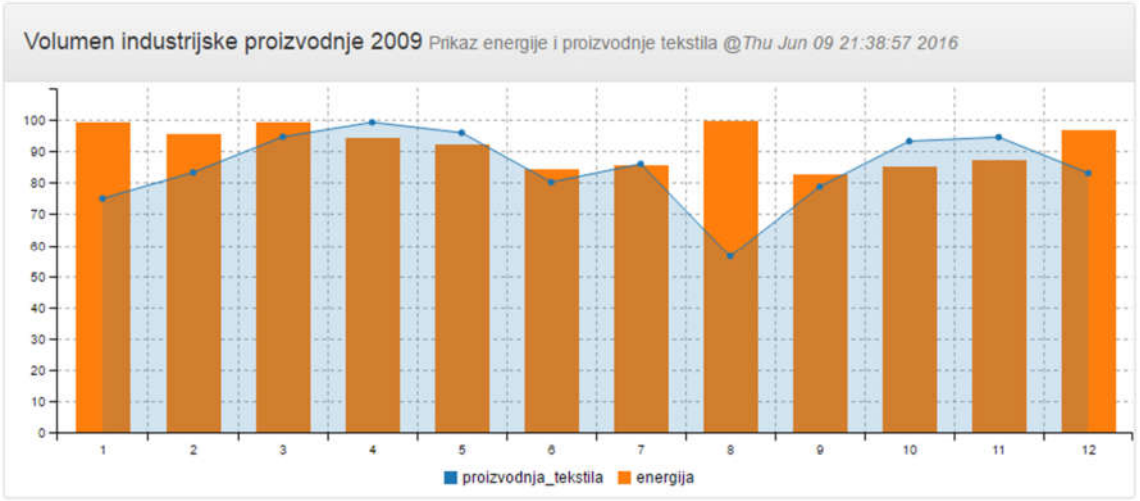
Slika 21. Upravljačka ploča za izradu grafikona

Raspored elemenata unutar liste varijabli moguće je mijenjati metodom „povuci i pusti“. Ovo je standardna metoda rada u Windows okruženju, a sastoji se od pritiska i držanja lijeve tipke miša iznad elementa liste, nakon čega se povlačenjem miša unutar liste mijenja i položaj tog elementa. Na taj se način mijenja raspored varijabli i shodno tomu se mijenja izgled grafikona.

Prvi element unutar liste označenih varijabli vezan je uz apscisu grafikona, te je dovoljno varijablu koja se želi prikazati na apscisi povući na prvo mjesto. Ostale varijable će se prikazivati na ordinati u međuodnosu s varijablom na apscisi. Kako je varijabla na apscisi ujedno i prvi element na listi, međuodnos ostalih varijabli s liste bit će u sprezi s prvom varijablom liste.

Nakon određivanja koje se varijable žele koristiti prilikom iscrtavanja i njihovog redoslijeda, može se pristupiti odabiru tipa grafikona. Odabirom tipa grafikona utječe se na način prikaza podataka.

Tip grafikona koji će se iscrtati vezan je uz varijablu, što znači da je za svaku varijablu moguće definirati poseban izgled. Kao primjer, može se navesti mogućnost kombiniranja stupčastih i linijskih grafikona koji se prikazuju unutar istog koordinatnog sustava. Zbog takve fleksibilnosti vizualizacije, odabir tipa grafikona provodi se preko padajućeg menija koji se nalazi uz naziv varijable.

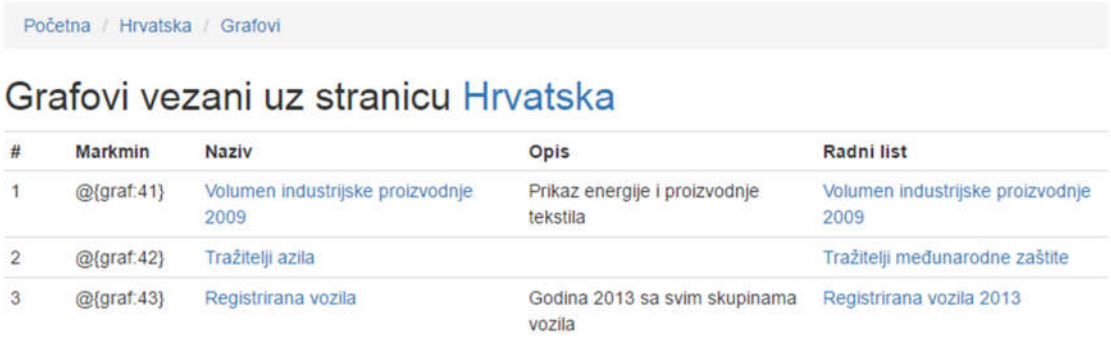


Slika 22. Primjer grafikona

5.5.2. Ugradnja grafikona unutar članka

Prilikom uređivanja članaka postoji mogućnost upotpunjavanja sadržaja s pratećim grafikonima. Kako se prethodno navelo, svaki članak sadrži svoje grafikone koje su stvorili korisnici. Te iste grafikone moguće je ugraditi unutar članka. Za ugradnju se koristi markmin sintaksa koja je stvorena posebno za slučaj ugradnje grafikona. Osnovni oblik sintakse za ugradnju grafova je *@{graf:id}* gdje je *id* identifikacijski broj grafikona.

Postojeće grafikone za ugradnju moguće je vidjeti na meta stranici članka koja se zove *Grafovi.* Dostupna je iz navigacijskog menija *Meta stranice* > *Grafovi* i na njoj se nalazi popis grafikona koji su prethodno stvoreni.



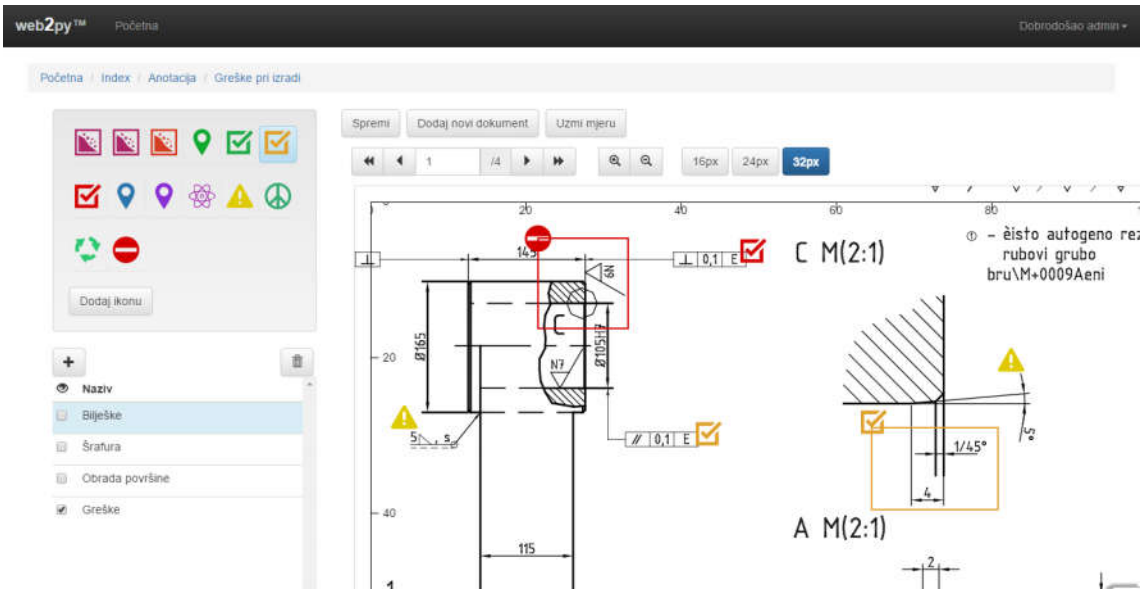
Slika 23 Popis grafikona stvorenij za članak *Hrvatska*

Stupci popisa su:

- # – redni broj stvorenog grafikona
- **Markmin** – sintaksa koja se koristi kako bi se unutar članka ugradio grafikon
- **Naziv** – naziv grafikona koji je dodijeljen prilikom stvaranja; također i poveznica na stranicu za pregled grafikona
- **Opis** – opis grafikona koji je definiran prilikom stvaranja
- **Radni list** – poveznica na radni list s podatcima koji se koriste za iscrtavanje grafikona

5.6. Označivanje slika i PDF dokumenata

Prilikom praćenja sustava može doći do potrebe slikovne dokumentacije događaja ili značajki. Prostor za označivanje omogućuje uvoz slika i PDF datoteka nad kojima će korisnik moći postavljati bilješke. Bilješke koje će korisnik koristiti proizvoljne su i ovise o potrebi korisnika da naglasi nešto na slici. Kao primjer mogu se navesti označivanje mjesta kvara, položaj elemenata sustava, ispravci na nacrtima, mjesto neke radnje i sl.



Slika 24. Primjer radnog prostora za anotaciju

Takva vrsta dokumentiranja postignuta je korištenjem slojeva koji se nalaze iznad slike.

Učitane slike mogu biti PDF datoteke s više stranica i slikovne datoteke. Prilikom postavljanja slike, modul označivanja u ovoj aplikaciji prepoznat će tip datoteke i ovisno o tome pravilno dodati sliku kao zadnju stranicu unutar radnog prostora. Moguće je istodobno u jednom radnom listu koristiti i PDF datoteke i slikovne datoteke. Jedina razlika u tome što će se u slučaju PDF datoteke s više stranica, radnom listu dodati svaka stranica posebno. Kao primjer, PDF s petnaestak stranica će prilikom učitavanja u radni list dodati svih petnaest stranica na kraj radnog lista.

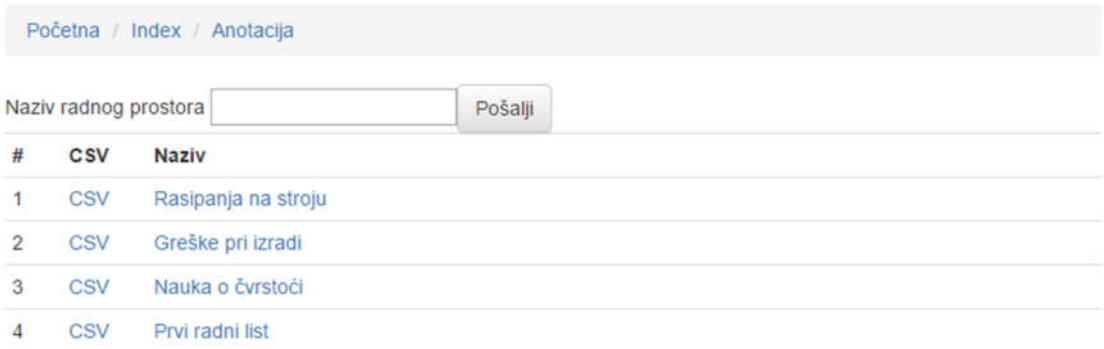
Korisnik može definirati više slojeva i u svakom od njih postavljati željene vizualne oznake, čiji će položaj biti zapamćen kao (x,y) koordinata s pripadnim nazivom u dvodimenzionalnom prostoru iznad slike. Karakteristika sloja je da obuhvaća jednu ili više vizualnih oznaka.

Oznaka ili bilješka, koja se može postaviti iznad slike, u obliku je ikone u boji kojoj je korisnik pridružio neko značenje. Stoga se o njoj može razmišljati kao o oznaci koja upućuje ili bilješci koja obavješćuje o nekom obilježju, karakteristici na nekoj slici. Za ovu bilješku ili vizualni marker, u nastavku će se koristiti ime oznaka.

Dakle, svaki radni prostor sadrži svoje oznake koje korisnik mora definirati prije nego ih može postaviti u neki od slojeva. Prethodno definirane oznake bit će vidljive i dohvatljive unutar legende radnog prostora.

5.6.1. Pregled postojećih radnih prostora

Prostor za označivanje dokumenata dostupan je na padajućem meniju *Meta stranice* s poveznicom *Označivanje*. Klikom na poveznicu otvara se sučelje na čijem vrhu postoji opcija za stvaranje novog radnog prostora. Ispod opcije stvaranja prikazani su nazivi postojećih radnih prostora.

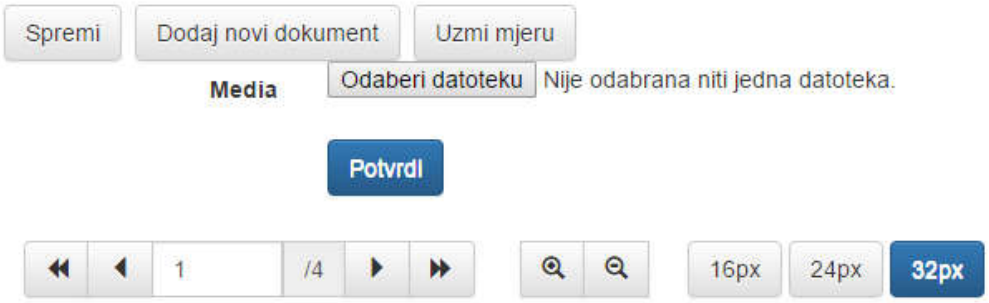


Slika 25. Lista radnih listova za anotaciju

5.6.2. Stvaranje prostora za označivanje

Za stvaranje novog radnog prostora dovoljno je upisati željeni naziv u polje *Naziv radnog lista*, nakon čega se naziv potvrdi pritiskom na gumb *Pošalji*. U pozadini će se stvoriti početni radni prostor s tim nazivom i korisnik će biti preusmjeren na stranicu za uređivanje, tj. označivanje slika ili PDF dokumenata.

Radni prostor početno ne sadrži ni dokumente ni oznake (bilješke). Dodavanje dokumenta moguće je klikom na gumb *Dodaj novi dokument*, nakon čega se prikazuje obrazac za odabir datoteke s računala.



Slika 26. Obrazac za učitavanje datoteke s računala

Kao novi dokument moguće je koristiti slike i PDF datoteke. PDF dokument može sadržavati više stranica i prilikom dodavanja svaka stranica će se tretirati kao dodatni dokument za označivanje.

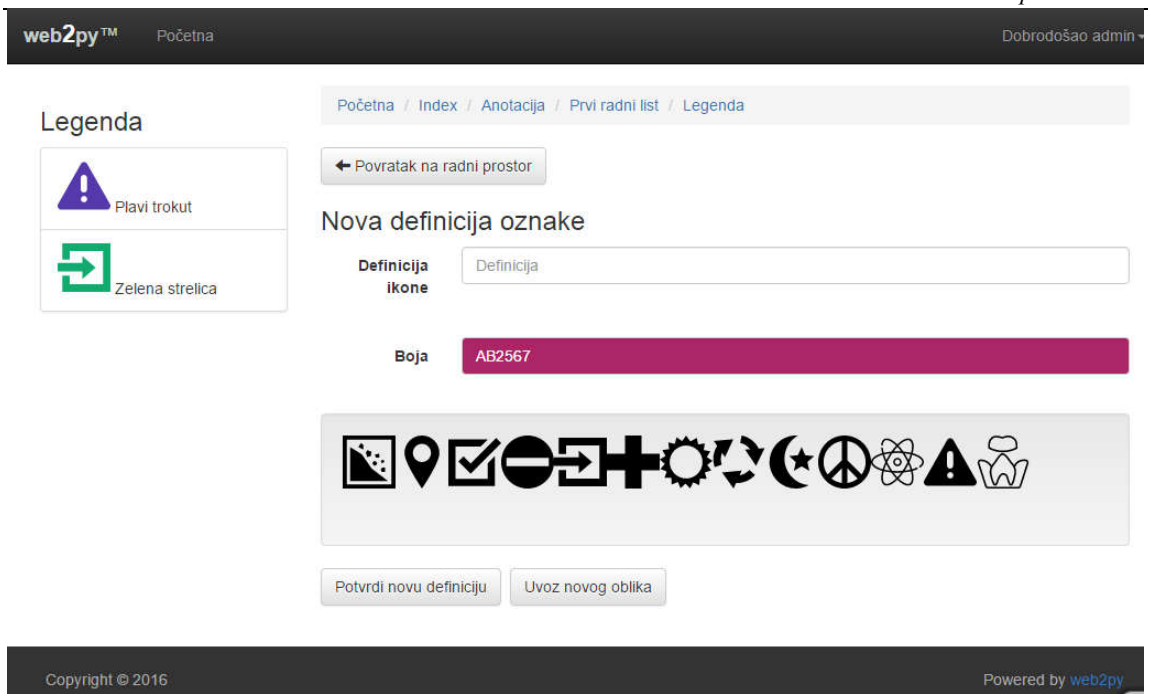
5.6.2.1. Definiranje oznaka/bilješki

Nakon dodanog dokumenta mogu se definirati bilješke koje će se u radu nad učitanim dokumentima koristiti. Klikom na gumb *Dodaj ikonu* korisnik se preusmjerava na stranicu u kojoj postoji mogućnost umetanja nove definicije bilješke u legendu. [Slika 27]

Svaku oznaku (bilješku) karakterizira ikona, boja i definicija.

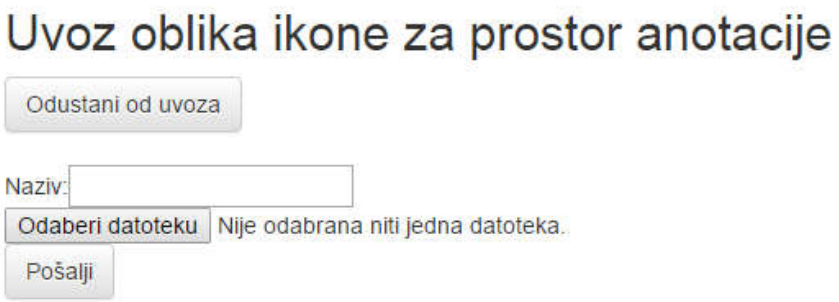
Na lijevoj strani radnog prostora nalazi se legenda s prethodno definiranim ikonama, dok se u sredini nalazi obrazac za stvaranje nove definicije u legendi.

Unutar obrasca za stvaranje potrebno je odrediti definiciju ikone, odabrati boju i na kraju označiti ikonu koja se želi koristiti. Nakon odabira ikone ona će poprimiti odabranu boju što korisniku pruža pregled ikone prije potvrde. Nakon popunjavanja obrasca nova se definicija stvara klikom lijeve tipke miša na gumb *Potvrdi novu definiciju*. Stranica će se osvježiti i korisnik će unutar legende vidjeti novi unos koji odgovara stvorenoj oznaci.



Slika 27. Obrazac za definiranje oznaka radnog prostora

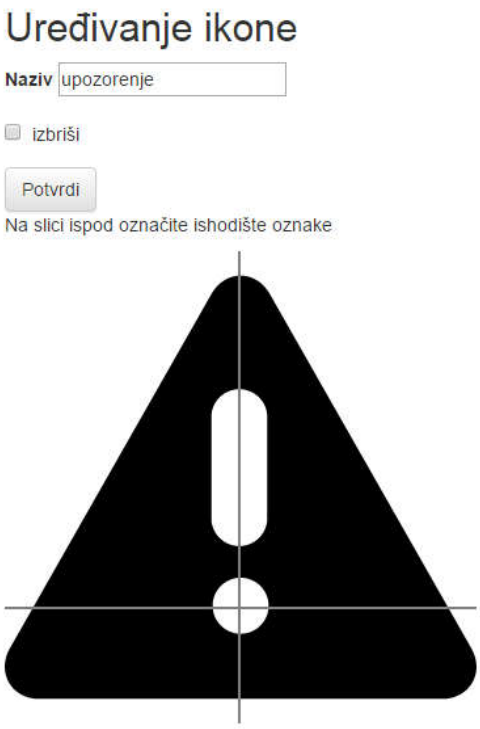
U slučaju da lista dostupnih ikona ne sadrži željeni oblik, moguće ga je uvesti s računala nakon čega će ikona biti dostupna i za stvaranje definicija oznaka ostalih radnih listova. Njezin uvoz će biti globalan, a ne samo vezan uz radni list iz kojeg je uvezena. Moguće je koristiti samo svg datoteke za uvoz novog oblika. Uvoz se pokreće klikom lijeve tipke miša na gumb *Uvoz novog oblika*, nakon čega će korisnik biti preusmjeren na stranicu s obrascem.



Slika 28 Obrazac za uvoz novog oblika ikone s računala

Nakon što se popuni obrazac s nazivom ikone i odabere svg datoteka s računala, korisnik potvrđuje unos s gumbom *Pošalji*. Nakon toga korisnik je preusmjeren na dodatni obrazac koji sadrži učitanu datoteku unutar stranice. Taj obrazac služi za postavljanje ishodišta ikone. Kako bi se postavilo ishodište dovoljno je kliknut lijevom tipkom miša na željenu poziciju unutar slike. Nakon klika pojavit će se dvije sive linije čije sjecište označava trenutno označeno

ishodište. Ishodište ikone koristi se za pravilno pozicioniranje oznake unutar radnog prostora. U slučaju kada datoteka nije pravilno učitana i ikona se ne prikazuje, brisanje je moguće označavanjem okvira *izbriši*. Željene izmjene korisnik potvrđuje lijevom pritiskom miša na gumb *Potvrdi* i biva preusmjeren natrag na stranicu za definiranje oznaka.



Slika 29 Postavljanje ishodišta učitane ikone

Kad su definicije oznaka/bilješki uređene, radni prostor spreman je za rad.

5.6.3. Uređivanje prostora za označivanje

S lijeve strane radnog prostora vidljiva je legenda sa svojim definicijama. Ispod legende nalazi se radna ploča sa slojevima na kojoj se može upravljati s vidljivošću sloja. Desno se nalazi prostor gdje se prikazuju dokumenti preko kojih se polažu slojevi u koje će se smještati oznake prilikom označivanja dokumenta. Postavljene oznake vezanu su uz slojeve i ovisno o vidljivosti sloja iscrtavaju se ili ne iscrtavaju iznad dokumenta. Ako se sloj označi, nad dokumentom će se prikazati postavljene oznake u tom sloju, a ako se isključi oznaka sloja onda nad

dokumentom nestaju i sve postavljene oznake u tom sloju, a prikazuju se samo oznake označenih slojeva.

Svaki prostor označivanja može sadržavati više dokumenata koji se tretiraju kao dodatne stranice.

Iznad prostora za iscrtavanje nalaze se opcije za spremanje promjena, dodavanje novog dokumenta, navigaciju kroz stranice, promjenu veličine iscrtanog dokumenta i promjenu veličine ikone za iscrtavanje.

U slučaju da je korisnik član grupe korisnika "urednik" ili je sam stvorio radni prostor, omogućeno mu je uređivanje.

5.6.3.1. Legenda

Unutar legende nalaze se prethodno definirane bilješke. Definirane su ikonom, bojom i definicijom – značenjem koje je iskazano riječima. Uz svaki radni prostor za označivanje definirane su pridružene bilješke koje se moraju definirati prilikom stvaranja radnog prostora ili naknadno, prilikom uređivanja.

Prijelazom oznake miša preko ikone bilješke u legendi, iznad ikone se pojavljuje oblačić s njezinom definicijom. Bilješku koju želimo dodati na dokument označimo lijevom tipkom miša unutar legende. Jednom označena bilješka naglašena je vizualno: svijetlo-plavim pravokutnikom.

5.6.3.2. Slojevi

Sloj u kojem se želi raditi, označuje se (*engl. checking*) lijevom tipkom miša. Označeni sloj poprima svijetlo plavu boju što znači da je trenutno aktivan. Svako pridruživanje oznaka nad dokumentom spremat će se u taj sloj. Skrivanje i prikazivanje sloja s postavljenim oznakama odvija se preko kvačica koje se nalaze lijevo od naziva sloja. Sloj koji se prikazuje ima aktiviranu kvačicu, dok skriveni sloj nema kvačicu. Da bi se dodao novi sloj, dovoljno je kliknuti na gumb s oznakom plus iznad radne ploče slojeva. Rezultat će biti novi sloj koji se prikazuje ispod posljednjeg sloja. Brisanje slojeva moguće je nakon pritiska na gumb s ikonom kante za smeće koja se nalazi iznad radne ploče slojeva s desne strane. Nakon što se aktivira mogućnost brisanja, uz svaki sloj s desne strane pojavit će se ikona križića. Klikom na ikonu križića sloj se briše. Povratak izbrisanog sloja (sa svim njegovim postavljenim oznakama) više nije moguć, pa je potreban oprez kod brisanja slojeva.

Slika 30. Upravljačka ploča slojeva

5.6.3.3. Prozor za pregled dokumenta

Unutar prozora za pregled iscrtavaju se dokumenti koje smo prethodno dodali radnom prostoru za anotaciju. Iscrtani dokument ovisi o broju stranice na kojoj se nalazi. Trenutna stranica može se vidjeti u navigacijskom dijelu iznad dokumenta.

Prozor je interaktivan, što znači da reagira na korisnikove unose mišem. Povlačenje dokumenta unutar prozora moguće je držanjem lijeve tipke miša bilo gdje unutar prozora. Dokument će pratiti pokrete miša i shodno tome se pomicati. Zumiranje dokumenata moguće je odraditi i s pomakom kotačića miša.

Postoje dva tipa oznake koje se mogu vezati uz dokument. Prva je oznaka u obliku ikone i predstavlja točku na dokumentu. Drugi je tip oznake u obliku ikone s pravokutnikom i označuje površinu na dokumentu.

Dodavanje oznake u obliku ikone odrađuje se kombinacijom tipke „ctrl“ na tipkovnici i klikom miša na mjesto gdje se želi dodati bilješka. Na mjestu klika mišem iscrtat će se ikona trenutno aktivne bilješke.

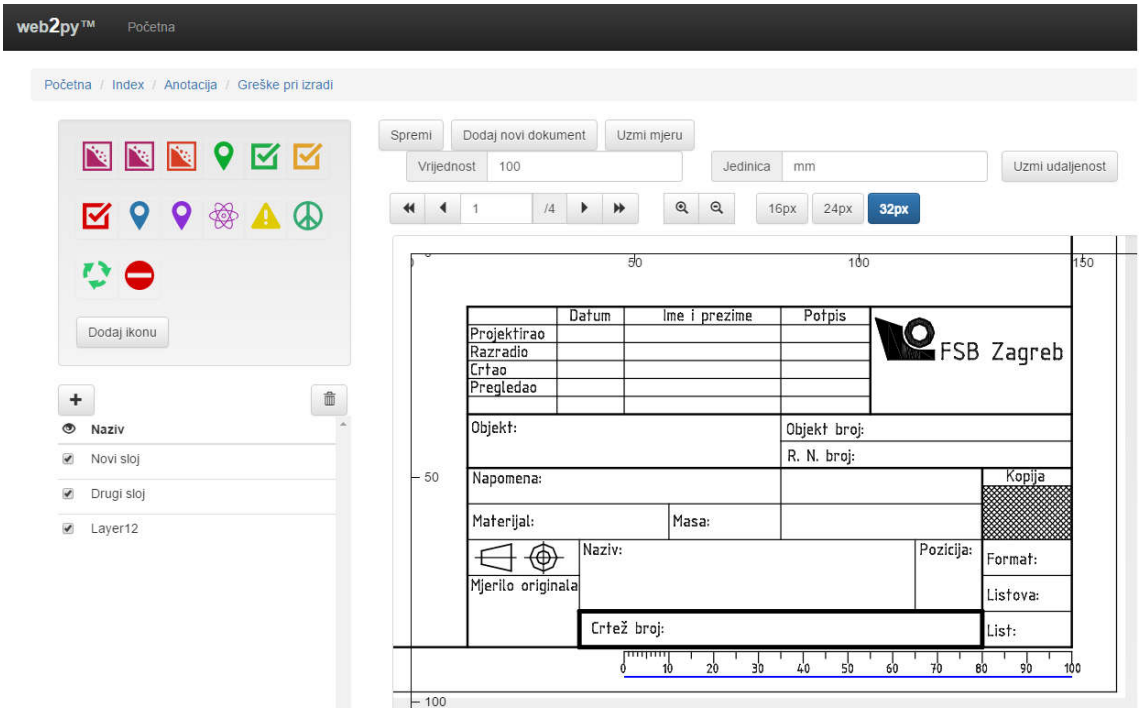
Oznaka u obliku površine stvara se istodobnim pritiskom tipke „shift“ i povlačenja pritisnute lijeve tipke miša preko dokumenta. Indikator površine koja će se označiti u obliku je sivog kvadrata. Nakon otpuštanja lijeve tipke miša na željenoj se poziciji iscrtava kvadrat koji poprima boju trenutno aktivne bilješke, a u gornjem lijevom kutu kvadrata iscrtava se ikona oznake.

Prije odlaska sa stranice potrebno je spremiti izmjene.

5.6.3.4. Definiranje mjerila

U slučajevima kada dokumenti sadrže vlastito mjerilo definirano sa skalom, numeričkom vrijednošću i mjernom jedinicom, postoji mogućnost da se to isto mjerilo preslika unutar radnog prostora. Preslikavanjem mjerila dobiva se dodana vrijednost sakupljenih podataka jer će se udaljenosti na dokumentu moći mjeriti. Kao primjer mogu se navesti karte i tehnički crteži koji sadrže mjerilo.

Za postavljanje mjerila potrebno je unutar obrasca koji se dobiva pritiskom na gumb *Uzmi mjeru*, definirati udaljenost i mjernu jedinicu za mjeru koja će se preslikati s dokumenta. Mogućnost definicije dužine na dokumentu pokreće se klikom na gumb *Uzmi udaljenost*, prilikom čega korisnik povlači dužinu unutar prostora za pregled dokumenta. Dužina je naglašena plavom bojom i potrebno ju je uzeti na način da odgovora dužini mjerila s dokumenta. [Slika 31]



Slika 31 Uzimanje mjerila s dokumenta

Jednom definirano mjerilo bit će vezano uz stranicu na kojoj je stvoreno te će se pomoću njega oko dokumenta crtati skala koja vjerno prikazuje dimenzije unutar dokumenta.

5.6.4. Izvoz podataka iz prostora za označivanje

Kako bi se mogla odraditi dodatna analiza sakupljenih oznaka nad dokumentima omogućen je izvoz podataka kao csv datoteka. Izvoz je moguć s meta stranice *Označivanje* pritiskom na poveznicu CSV koja se nalazi uz naziv radnog lista za kojeg se želi skinuti podatke. [Slika 25]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	sloj-naziv	sloj-id	stranica-r	legenda-c	legenda-i	mjera-x	mjera-y	jedinica	poz-x	poz-y	sirina	visina	datoteka	stranica-datoteke	
2	Layer12	7	1	Zabranjen	69	220.468	155.9026	mm	0.292302	0.121156			URL	1	
3	Layer12	7	1	Zabranjen	69	220.468	155.9026	mm	0.390576	0.137785			URL	1	
4	Drugi sloj	6	1	dobro je	49	220.468	155.9026	mm	0.286422	0.253002			URL	1	
5	Drugi sloj	6	1	Mir	67	220.468	155.9026	mm	0.721515	0.40029			URL	1	
6	Novi sloj	3	1	Mir	67	220.468	155.9026	mm	0.721515	0.40029			URL	1	
7	Novi sloj	3	1	Zabranjen	69	220.468	155.9026	mm	0.292302	0.121156			URL	1	
8	Novi sloj	3	1	Zabranjen	69	220.468	155.9026	mm	0.390576	0.137785			URL	1	
9	Layer12	7	1	Zabranjen	69	220.468	155.9026	mm	0.253664	0.382473			URL	1	
10	Novi sloj	3	1	Recikliraj	68	220.468	155.9026	mm	0.560245	0.422858			URL	1	
11	Drugi sloj	6	2	Mir	67				0.456409	0.464939			URL	1	
12	Drugi sloj	6	3	Atom	60				0.475966	0.531522			URL	1	
13	Novi sloj	3	1	Upozoren	61	220.468	155.9026	mm	0.130192	0.325458			URL	1	
14	Novi sloj	3	1	Upozoren	61	220.468	155.9026	mm	0.152031	0.457304			URL	1	
15	Layer12	7	1	Zabranjen	69	220.468	155.9026	mm	0.572005	0.074832	0.16211	0.236373	URL	1	
16	Layer12	7	1	Zabranjen	69	220.468	155.9026	mm	0.48045	0.475121	0.178069	0.148475	URL	1	
17	Novi sloj	3	1	Upozoren	61	220.468	155.9026	mm	0.572005	0.074832	0.16211	0.236373	URL	1	
18	Novi sloj	3	1	Upozoren	61	220.468	155.9026	mm	0.48045	0.475121	0.178069	0.148475	URL	1	

Slika 32. Csv datoteka izvezena iz prostora za anotaciju

Izvezena datoteka sadrži sve oznake koje su postavljene unutar radnog prostora u svim slojevima. Zbog toga svaki red predstavlja jednu oznaku, dok su stupci ostale značajke te oznake.

Prvi stupac prikazuje sloj za koji je oznaka vezana, dok je u drugom stupcu jedinstveni identifikator tog sloja. Dalje se nalazi redni broj stranica unutar radnog lista, definicija ikone i jedinstveni identifikator ikone. U šestom i sedmom stupcu se nalazi mjera za x i y koordinatu koju je korisnik definirao na radnom listu, a u slučaju da mjera nije definirana polje je prazno. Za *mjera-x* i *mjera-y* karakteristično je da su također i visina i širina dokumenta. Osmi stupac je jedinica mjere.

Nakon toga slijedi položaj ikone s koordinatama koje su vezane uz dimenzije dokumenta. U slučaju da je oznaka koja predstavlja točku koordinate, *poz-x* i *poz-y* označavaju položaj oznake na radnom listu. Ako je oznaka u obliku površine, onda postoje još dvije dodatne varijable – *sirina* i *visina*, koje predstavljaju širinu i visinu površine. Koordinate su iskazane kao relativne u odnosu na širinu u slučaju koordinate *poz-x* i visinu u slučaju koordinate *poz-y*. Množenjem s *mjera-x* i *mjera-y* respektivno, dobivaju se apsolutne koordinate.

Potom se nalazi poveznica do datoteke, nakon koje se nalazi broj stranice unutar datoteke za koju je oznaka vezana.

Zbog načina na koji se u radnom prostoru objedinjuju PDF i slikovne datoteke, broj stranice je važan u slučajevima kada je uvezena PDF datoteka s više stranica. Kod slika će taj broj stranice uvijek biti jedan, dok se kod PDF datoteka može dogoditi da se oznaka nalazi na nekoj stranici unutar PDF-a.

6. ZAKLJUČAK

U ovom diplomskom radu proučeni su koncepti koji čine sustav poslovne inteligencije. Jedan je od osnovnih koncepata poslovne inteligencije prikupljanje velikih količina podataka u svrhu njihove analize, a izradom ovog diplomskog rada stvorilo se više načina za prikupljanje statističkih podataka.

Da bi se ti podatci stavili unutar nekog konteksta koristi se wiki sustav s člancima, a svaki od njih sadrži svoje skupove podataka.

Jedan od načina prikupljanja podataka je unutar radnih listova unutar tablica koje korisniku pružaju lak i pregledan način za ažuriranje i unos podataka. Korištenjem modernih internet tehnologija korisniku se pruža mogućnost rada unutar preglednika na udaljenom serveru na način sličan kakav se može očekivati od lokalnih računalnih programa. Iz sakupljenih podataka također je moguće stvarati grafikone koje je kasnije moguće ugraditi unutar članka i tako vizualizirati taj skup podataka.

Također su se stvorili alati koji pomažu prilikom izdvajanja podataka sa slika koje mogu doći u vidu PDF dokumenata ili nekih standardnih formata digitalnih slika. S tim se alatima unutar mrežnog preglednika prikazuju slike i PDF dokumenti iznad kojih se unutar slojeva mogu dodati oznake. Prilikom objedinjavanja PDF dokumenata i digitalnih slika unutar jednog radnog prostora pojavio se problem koji je onemogućavao da se unutar mrežnog preglednika oni tretiraju jednako. Kombinacijom HTML standarda za prikaz slika i Javascript skupa funkcija za iscrtavanje PDF dokumenata, unutar mrežnog preglednika objedinjuju se na način da korisnik ne raspoznaje razliku i omogućuje mu da se koriste zajedno unutar jednog radnog prostora za anotaciju.

Sakupljene podatke moguće je izvesti i koristiti na lokalnom računalu za daljnju analizu ili za prijenos između različitih sustava.

Na kraju se može zaključiti da danas postoje tehnologije koje se mogu iskoristiti za stvaranje mrežnih aplikacija koje bi na siguran i efikasan način zamijenile lokalne računalne programe. Iako sustav poslovne inteligencije nije u potpunosti razvijen, uspješno je prikazana mogućnost spremanja podataka u oblaku koji je dostupan svim korisnicima bez potrebe za međusobnim prijenosom podataka.

LITERATURA

[1] Eckerson, W. W.: Performance dashboards, John Wiley & Sons Inc., New Jersey, 2011.

[2] Sherman, R.: Busines Intelligence Guidebook: From Dana Integration To Analytics, Elsevier Inc., Waltham, 2015.

[3] Lih, A.: The Wikipedia Revolution, Hyperion, New York, 2009

[4] Miller, M.: Sams Teach Yourself Wikipedia In 10 Minutes, Pearson Education Inc., New York, 2010

[5] W3schools, <http://www.w3schools.com>. Pristupljeno 24. lipnja 2016.

[6] HTML, <https://en.wikipedia.org/wiki/HTML>. Pristupljeno 24. lipnja 2016.

[7] HTML attribute, <https://en.wikipedia.org/wiki/HTML_attribute>. Pristupljeno 27. lipnja 2016.

[8] Cascading Style Sheets, <https://en.wikipedia.org/wiki/Cascading_Style_Sheets>. Pristupljeno 27. lipnja 2016.

[9] Web2py, <http://web2py.com/>. Pristupljeno 10. lipnja 2016.

[10] Di Pierro, M.: Web2py Complete Reference Manual, 15th edition, 2013

[11] Di Pierro, M.: Web2py Application Development Cookbook, Pack Publishing, 2012

[12] Green B., Seshadri S.: AngularJS, O'Reilly Media, 2013

[13] Wahlin D.: AngularJS in 60 Minutes, Wahlin Consulting, 2013./2014.

[14] D3JS, <https://d3js.org>. Pristupljeno 18. lipnja 2016.

[15] C3JS, <http://c3js.org>. Pristupljeno 18. lipnja 2016.

[16] Murray S.: Interactive Dana Visualization, O'Reilly Media, Sebastopol 2013

PRILOZI

I. CD-R disc

II. Programski kod

PROGRAMSKI KOD

Web2py: *controllers/annotate.py* – Python programski jezik

```
@auth.requires(auth.has_membership(group_id='wiki_editor'))
def index():
    slug = request.args(0, default='index')
    article = db(db.wiki_page.slug == slug).select().first().id

    bc_article = db(db.wiki_page.slug == slug).select().first()

    workspaces = db(db.annotation_workspace.article_ref == article).select()

    response.bredcrumbs = [
        dict(url=URL(c='default'), title='Početna'),
        dict(url=URL(request.application, c='default', f='index', args=[slug]),
title=bc_article.title),
        dict(url='#', title='Anotacija')
    ]

    form = FORM('Naziv radnog prostora ', INPUT(_name='name'),INPUT(_type='submit'))
    if form.accepts(request, session):
        id = db.annotation_workspace.validate_and_insert(name=form.vars.name,article_ref =
article)['id']
        db.annotation_layers.insert(name="Sloj1", layer_id = 1, workspace_ref = id,
visible=True)
        response.flash = id
        redirect(URL('edit', args=[id]))
    elif form.errors:
        response.flash = 'form has errors'

    return dict(workspaces=workspaces, form=form)

def export():
    from gluon.dal import Rows, Row

    import os
    workspace_id = request.args(0)
    name = db(db.annotation_workspace.id ==
workspace_id).select(db.annotation_workspace.name).first().name

    icons = db(db.annotation_layers.workspace_ref == workspace_id)\
            (db.icon_reference.global_layer_ref == db.annotation_layers.id)\
            (db.icon_definition.id == db.icon_reference.icon_definition)\
        .select()

    areas = db(db.annotation_layers.workspace_ref == workspace_id)\
            (db.area_definition.global_layer_ref == db.annotation_layers.id)\
            (db.icon_definition.id == db.area_definition.icon_definition)\
            (db.area_definition.global_layer_ref == db.annotation_layers.id)\
        .select()

    field_names = [
        'sloj-naziv', 'sloj-id', 'stranica-radnog-prostora', 'legenda-opis', 'legenda-id',
'mjera', 'jedinica', 'poz-x', 'poz-y', 'sirina', 'visina', 'datoteka', 'stranica-datoteke'
```

```
    ]
    data = []

    for icon in icons:

        file, page, measure, unit = get_file(icon.icon_reference.page_ref, workspace_id)
        data.append(
            [icon.annotation_layers.name, icon.annotation_layers.id,
             icon.icon_reference.page_ref, icon.icon_definition.description,
             icon.icon_definition.id,
             measure, unit,
             icon.icon_reference.coordinate_x,
             icon.icon_reference.coordinate_y, None,
             None, '=HYPERLINK("' + URL(c='default', f='download',
args=os.path.join('upload', file), host=True) + '", "URL")',
             page]
        )

    for area in areas:

        file, page, measure, unit = get_file(area.area_definition.page_ref, workspace_id)
        data.append(
            [area.annotation_layers.name, area.annotation_layers.id,
             area.area_definition.page_ref, area.icon_definition.description,
             area.icon_definition.id, measure, unit,
             area.area_definition.coordinate_x,
             area.area_definition.coordinate_y, area.area_definition.width,
             area.area_definition.height,
             '=HYPERLINK("' + URL(c='default', f='download', args=os.path.join('upload',
file), host=True) + '", "URL")',
             page]
        )


    response.view='generic.csv'
    return dict(filename= name + '.csv', csvdata=data, field_names=field_names)

def get_file(page, workspace):
    media = db(db.annotation_pages.workspace_ref ==
workspace)(db.annotation_pages.workspace_page == page).select().first()
    if media['measure_value'] is not None:
        measure = media['measure_value']/media['measure']

    else:
        measure = None
    unit = media['unit']

    file_id = media['file_ref']

    page = media['file_page']
    file = db(db.annotation_media.id == file_id).select().first()['media']
    return file, page, measure, unit

def workspace_setup():
    workspace_id = request.args(0)
```

```
    row = db(db.annotation_workspace.id == workspace_id)(db.wiki_page.id ==
db.annotation_workspace.article_ref).select().first()

    article = row.wiki_page
    workspace = row.annotation_workspace


    response.bredcrumbs = [
        dict(url=URL(request.application), title='Početna'),
        dict(url=URL(request.application, 'default', 'index', args=[article.slug]),
title=article.title),
        dict(url=URL(request.application, 'annotate', 'index', args=[article.slug]),
title='Anotacija'),
        dict(url=URL(request.application, 'annotate', 'edit', args=[workspace_id]),
title=workspace.name),
        dict(url='#', title='Legenda')
    ]

    new_icon = request.vars
    if 'icon' in new_icon:
        if new_icon.icon == '':
            response.flash = 'Odaberite ikonu i probajte opet'
        else:
            db.icon_definition.validate_and_insert(icon_reference=new_icon.icon,
                                                   workspace_reference=workspace_id,
                                                   description=new_icon.definition,
                                                   color=new_icon.color)
            redirect(URL('workspace_setup', args=[workspace_id]))

    legend = db((db.icon_definition.workspace_reference == workspace_id) &
(db.icon_definition.icon_reference == db.icon.id)).select()

    #legend = db(db.icon_definition.workspace_reference == workspace_id).select()
    #legend = db(db.icon_definition.workspace_reference ==
workspace_id).select(left=db.icon_definition.on(db.icon_definition.icon_reference ==
db.icon.id))
    #legend = db(db.icon_definition.icon_reference == db.icon.id).select()
    #legend = db(db.icon.id == db.icon_definition.icon_reference).select()

    icons = db(db.icon).select()
    return dict(icons=icons, legend=legend)

def edit():
    workspace_id = request.args(0)
    #row = db(db.pdf_workspace.id == workspace_id).select().first()

    pages = db(db.annotation_pages.workspace_ref ==
workspace_id).select(orderby=db.annotation_pages.workspace_page)

    article = db(db.annotation_workspace.id == workspace_id)(db.wiki_page.id ==
db.annotation_workspace.article_ref).select().first()

    bc_article = article.wiki_page

    workspace = db(db.annotation_workspace.id == workspace_id).select().first()

    response.bredcrumbs = [
```

```
        dict(url=URL(request.application), title='Početna'),
        dict(url=URL(request.application, 'default', 'index', args=[bc_article.slug]),
title=bc_article.title),
        dict(url=URL(request.application, 'annotate', 'index', args=[bc_article.slug]),
title='Anotacija'),
        dict(url=URL(request.application, 'annotate', 'edit', args=[workspace_id]),
title=workspace.name)
    ]

    db.annotation_media.annotation_ref.readable = False
    db.annotation_media.annotation_ref.writable = False

    form = SQLFORM(db.annotation_media)
    form.vars.annotation_ref=workspace_id
    if form.accepts(request.vars):
        _create_pages(workspace_id, form.vars.id)
        response.flash='Uspješno učitano'
    return dict(file=file, form=form, pages=pages)

def view():
    workspace_id = request.args(0)

    pages = db(db.annotation_pages.workspace_ref ==
workspace_id).select(orderby=db.annotation_pages.workspace_page)

    article = db(db.annotation_workspace.id == workspace_id)(db.wiki_page.id ==
db.annotation_workspace.article_ref).select().first()

    bc_article = article.wiki_page

    workspace = db(db.annotation_workspace.id == workspace_id).select().first()

    response.bredcrumbs = [
        dict(url=URL(request.application), title='Početna'),
        dict(url=URL(request.application, 'default', 'index', args=[bc_article.slug]),
title=bc_article.title),
        dict(url=URL(request.application, 'annotate', 'index', args=[bc_article.slug]),
title='Anotacija'),
        dict(url=URL(request.application, 'annotate', 'edit', args=[workspace_id]),
title=workspace.name)
    ]

    db.annotation_media.annotation_ref.readable = False
    db.annotation_media.annotation_ref.writable = False

    form = SQLFORM(db.annotation_media)
    form.vars.annotation_ref=workspace_id
    if form.accepts(request.vars):
        _create_pages(workspace_id, form.vars.id)
        response.flash='Uspješno učitano'

    return dict(file=file, form=form, pages=pages)

def get_pages():
    import gluon.contrib.simplejson
    workspace_id = gluon.contrib.simplejson.loads(request.body.read())['workspace_id']
```

```
    pages = db(db.annotation_pages.workspace_ref == workspace_id)(db.annotation_media.id ==
db.annotation_pages.file_ref).select(
                orderby=db.annotation_pages.workspace_page
            )
    json = []

    for page in pages:
        objekat = dict(type=page.annotation_pages.file_type,
                       source=page.annotation_media.media,
                       source_num=page.annotation_pages.file_page,
                       workspace_page=page.annotation_pages.workspace_page,
                       measure=page.annotation_pages.measure,
                       unit=page.annotation_pages.unit,
                       measureValue=page.annotation_pages.measure_value)
        json.append(objekat)

    return gluon.contrib.simplejson.dumps(json)

def get_layers():
    import gluon.contrib.simplejson
    workspace_id = gluon.contrib.simplejson.loads(request.body.read())['workspace_id']
    layers = db(db.annotation_layers.workspace_ref == workspace_id).select()

    json = dict(layers=[], icons=[], areas=[])

    for layer in layers:
        object_layers = dict(name=layer.name, visible=layer.visible, id=layer.layer_id)

        icons = db((db.icon_reference.global_layer_ref == layer.id) &
(db.icon_definition.id == db.icon_reference.icon_definition)).select()

        areas = db(db.area_definition.global_layer_ref == layer.id).select()

        for icon in icons:
            object_icons = dict(color=icon.icon_definition.color,

coordinates=[icon.icon_reference.coordinate_x,icon.icon_reference.coordinate_y],
                                iconRef=icon.icon_reference.icon_definition,
                                id=icon.icon_reference.icon_id,
                                layerRef=icon.icon_reference.layer_ref,
                                pageRef=icon.icon_reference.page_ref)
            json['icons'].append(object_icons)

        json['layers'].append(object_layers)

        for area in areas:
            object_areas = dict(id=area.area_id,
                                pageRef = area.page_ref,
                                layerRef = area.layer_ref,
                                coordinates = [area.coordinate_x,
                                               area.coordinate_y,
                                               area.width,
                                               area.height],
                                iconRef = area.icon_definition)

            json['areas'].append(object_areas)
```

```
    return gluon.contrib.simplejson.dumps(json)

def get_legend():
    import gluon.contrib.simplejson
    workspace_id = gluon.contrib.simplejson.loads(request.body.read())['workspace_id']

    json = []
    legend = db((db.icon_definition.workspace_reference == workspace_id) &
(db.icon_definition.icon_reference == db.icon.id)).select()

    for icon in legend:
        svg_container = '<svg width="32" height="32" viewBox="' + icon.icon.viewbox + '"
fill="#' \
                        + icon.icon_definition.color + '">' + XML(icon.icon.filevalue) +
'</svg>'

        objekat = dict(svg=svg_container,
                       definition=icon.icon_definition.description,
                       color=icon.icon_definition.color,
                       id=icon.icon_definition.id,
                       offset=[icon.icon.x, icon.icon.y])

        json.append(objekat)
    return gluon.contrib.simplejson.dumps(json)

def _create_pages(workspace_id, file_id):
    file = db(db.annotation_media.id == file_id).select().first().media
    extension = file.split(".")[-1]

    pages = db(db.annotation_pages.workspace_ref ==
workspace_id).select(db.annotation_pages.workspace_page,
orderby=db.annotation_pages.workspace_page).last()

    if pages == None:
        last_page_of_workspace = 0
    else:
        last_page_of_workspace = pages.workspace_page

    if (extension == 'pdf'):
        import PyPDF2
        import os
        url = os.path.join(request.folder,'uploads', file)
        num_of_pages = PyPDF2.PdfFileReader(url).getNumPages()

        for page in range(1, num_of_pages + 1):
            db.annotation_pages.insert(workspace_ref = workspace_id,
                                                    file_type = True,
                                                    file_ref = file_id,
                                                    workspace_page = last_page_of_workspace
+ page,
                                                    file_page = page)

    elif (extension in ['png', 'jpg', 'jpeg', 'gif', 'svg'] ):
        db.annotation_pages.insert(workspace_ref = workspace_id,
                                       file_type = False,
                                       file_ref = file_id,
                                       workspace_page = last_page_of_workspace + 1,
```

```
                                        file_page = 1)

def save_workspace():
    import gluon.contrib.simplejson

    json = gluon.contrib.simplejson.loads(request.body.read())
    pages = json['pages']
    workspace_id = json['workspace_id']
    icon_definition = json['iconDefinition']
    area_definition = json['areaDefinition']
    layers = json['layers']

    for page in pages:

        if 'measure' in page:
            db((db.annotation_pages.workspace_ref == workspace_id) &
(db.annotation_pages.workspace_page ==
page['workspace_page'])).update(measure=page['measure'])

        if 'unit' in page:
            db((db.annotation_pages.workspace_ref == workspace_id) &
(db.annotation_pages.workspace_page == page['workspace_page'])).update(unit = page['unit'])

        if 'measureValue' in page:
            db((db.annotation_pages.workspace_ref == workspace_id) &
(db.annotation_pages.workspace_page == page['workspace_page'])).update(measure_value =
page['measureValue'])

    for layer in layers:
        db.annotation_layers.update_or_insert(
            (db.annotation_layers.workspace_ref == workspace_id) &
            (db.annotation_layers.layer_id == layer['id']),
            name = layer['name'],
            visible = layer['visible'],
            layer_id = layer['id'],
            workspace_ref = workspace_id)

    for icon in icon_definition:
        global_layer_id = db(db.annotation_layers.workspace_ref == workspace_id)\
                          (db.annotation_layers.layer_id ==
icon['layerRef']).select(db.annotation_layers.id).first().id

        db.icon_reference.update_or_insert(
            (db.icon_reference.global_layer_ref == global_layer_id) &
            (db.icon_reference.layer_ref == icon['layerRef']) &
            (db.icon_reference.icon_id == icon['id']),
            global_layer_ref = global_layer_id,
            layer_ref = icon['layerRef'],
            page_ref = icon['pageRef'],
            coordinate_x = icon['coordinates'][0],
            coordinate_y = icon['coordinates'][1],
            icon_id = icon['id'],
            icon_definition = icon['iconRef'])

    for area in area_definition:
        global_layer_id = db(db.annotation_layers.workspace_ref == workspace_id)\
```

```
                            (db.annotation_layers.layer_id ==
icon['layerRef']).select(db.annotation_layers.id).first().id

        db.area_definition.update_or_insert(
            (db.area_definition.global_layer_ref == global_layer_id) &
            (db.area_definition.area_id == area['id']),
            area_id = area['id'],
            layer_ref = area['layerRef'],
            global_layer_ref = global_layer_id,
            page_ref = area['pageRef'],
            coordinate_x = area['coordinates'][0],
            coordinate_y = area['coordinates'][1],
            width = area['coordinates'][2],
            height = area['coordinates'][3],
            icon_definition = area['iconRef']
        )

    return gluon.contrib.simplejson.dumps(json)

def view_load():
    response.view = 'annotate/view.load'
```

AngularJS, D3JS: *static/js/d3.layers.js* – JavaScript programski jezik

```
angular.module('d3', [])
    .factory('d3Service', [function () {
        var d3;
        // insert d3 code here
        return d3;
    }]);

var app = angular.module('app', ['ngSanitize']);

app.directive("layerWindow", function ($window) {
    return {
        restrict: "EA",
        template: '<svg width="100%" height="100%" style="position:fixed">' +
        '<g id="g-canvas" class="hidden">' +
        '<foreignObject y="0px" x="0px">' +
        '<div>' +
        '<canvas id="the-canvas" style="position:fixed" ></canvas>' +
        '</div>' +
        '</foreignObject>' +
        '</g>' +
        '<g class="hidden" id="image-canvas">' +
        '<image></image>' +
        '</g>' +
        '</svg>',
        templateNamespace: 'svg',
        replace: true,
        scope: {
            pages: '=',
            iconDefinition: '=',
            appendSelection: "=",
            selectionArea: "=",
            picture: "=",
```

```
            layerControl: "=layercontrol",
            selected: "=",
            area: "=",
            pdf: "=",
            icons: "=",
            takeMeasure: "="
        },
        link: function (scope, elem, attrs, ngModel) {
            scope.area = [];
            scope.appendSelection = function (areaCoordinates) {

                var pos = scope.area.length;
                scope.area[pos] = [{id: "area" + pos, coordinates: areaCoordinates, opis:
"opis"}];

            };

            var d3 = $window.d3;

            var PDFJS = $window.PDFJS;

            var url = d3.select("#pdf-url").node().value;
            //var rawSvg = elem.find("svg")[0];
            //var canvas = d3.select(rawSvg);

            //d3.select("svg").remove();

            var canvas = d3.select('svg')
            roditelj = canvas.select(function () {
                return this.parentNode;
            })

            var svgCanvas = roditelj.append('svg').style('position', 'relative');

            var viewport = svgCanvas.append("g").attr('class', 'viewport');

            var layerStack = viewport.append("g").attr('class', 'layerstack');

            var scaleX = d3.scale.linear()

            var xAxis = d3.svg.axis()
                .tickSize(10)
                .tickPadding(-5)

            var measureX = viewport.append("g")
                    .attr('class', 'x axis')
                    .attr("transform", "translate(0,0)")

            var scaleY = d3.scale.linear()


            var yAxis = d3.svg.axis()
                .tickSize(10)
                .tickPadding(5)
                .orient("right")

            var measureY = viewport.append("g")
```

```
                    .attr('class', 'y axis')
                    .attr("transform", "translate(0,0)")

            var foreignobject = canvas.select('#g-canvas');

            var imagecanvas = canvas.select('#image-canvas');

            var dragPad = layerStack.append("rect")
                .attr("x", "0")
                .attr("y", "0")
                .attr("width", scope.pdf.width)
                .attr("height", scope.pdf.height)
                .attr("fill", "red")
                .style('opacity', 0)

            var zoom = d3.behavior.zoom()
                .scaleExtent([0.1, 10])
                .on("zoom", function () {
                    if (d3.event.scale != scope.pdf.zoom) {
                        foreignobject.attr("class", function() {return
(scope.pages[scope.pdf.page - 1].type) ? "hide hidden" : ""});
                        imagecanvas.attr("class", function() {return
(scope.pages[scope.pdf.page - 1].type) ? "" : "hide hidden"})
                        layerStack.attr("class", "layerstack hide hidden")
                    }

                    layerStack.attr("transform", "translate(" + d3.event.translate + ")")
                    imagecanvas.attr("transform", "translate(" + d3.event.translate + ")")
                    foreignobject.attr("transform", "translate(" + d3.event.translate +
")")

                    scope.$apply(function () {
                        scope.pdf.zoom = d3.event.scale;
                    })
                });

            var drag = d3.behavior.drag()
                .on("dragstart", function () {
                    d3.event.sourceEvent.stopPropagation();
                    pinpoint = d3.mouse(this);
                    selection = d3.select("#layer" + scope.selected.layerId).append('rect')
                        .attr("x", pinpoint[0])
                        .attr("y", pinpoint[1])
                        .style('opacity', 0.2)
                })
                .on("dragend", function () {
                    selection.remove();

                    // area defintion [x, y, width, height]
                    areaCoordinates = [selection.attr("x") / scope.pdf.width,
selection.attr("y") / scope.pdf.height, selection.attr("width") / scope.pdf.width,
selection.attr("height") / scope.pdf.height];
                    var pos = scope.selectionArea.length;

                    scope.$apply(function () {
                        scope.selectionArea[pos] = {
                            id: pos + 1,
```

```
                            layerRef: scope.selected.layerId,
                            pageRef: scope.pdf.page,
                            iconRef: scope.selected.icon['id'],
                            coordinates: areaCoordinates
                        };
                    });

                    console.log(scope.selectionArea)
                })
                .on("drag", function () {
                    if (((d3.mouse(this)[0] - pinpoint[0]) > 0 ) && ((d3.mouse(this)[1] -
pinpoint[1]) > 0)) { // Slučaj za četvrti kvadrant, obje vrijednosti pozitivne
                        selection.attr("width", function () {
                            return (d3.mouse(this)[0] - pinpoint[0])
                        })
                            .attr("height", function () {
                                return (d3.mouse(this)[1] - pinpoint[1])
                            })
                    }
                    else if (((d3.mouse(this)[0] - pinpoint[0]) < 0 ) &&
((d3.mouse(this)[1] - pinpoint[1]) > 0)) { // Slučaj za treći kvadrant, vrijednost x
negativna
                        selection.attr("width", function () {
                            return (pinpoint[0] - d3.mouse(this)[0] )
                        })
                            .attr("height", function () {
                                return (d3.mouse(this)[1] - pinpoint[1])
                            })
                            .attr("x", d3.mouse(this)[0])
                    }
                    else if (((d3.mouse(this)[0] - pinpoint[0]) > 0 ) &&
((d3.mouse(this)[1] - pinpoint[1]) < 0)) { // Slučaj za prvi kvadrant, vrijednost y
negativna
                        selection.attr("width", function () {
                            return (d3.mouse(this)[0] - pinpoint[0] )
                        })
                            .attr("height", function () {
                                return (pinpoint[1] - d3.mouse(this)[1])
                            })
                            .attr("y", d3.mouse(this)[1])
                    }
                    else if (((d3.mouse(this)[0] - pinpoint[0]) < 0 ) &&
((d3.mouse(this)[1] - pinpoint[1]) < 0)) { // Slučaj za drugi kvadrant, vrijednost x, y
negativna
                        selection.attr("width", function () {
                            return (pinpoint[0] - d3.mouse(this)[0])
                        })
                            .attr("height", function () {
                                return (pinpoint[1] - d3.mouse(this)[1])
                            })
                            .attr("x", d3.mouse(this)[0])
                            .attr("y", d3.mouse(this)[1])
                    }
                    else {
                    }
                });
```

```
            svgCanvas.call(zoom);

            function show_pdf(page, scaleZoom) {
                console.log(url)
                PDFJS.getDocument(url + '/' + scope.pages[scope.pdf.page -
1].source).then(function (pdf) {
                    pdf.getPage(page).then(function (page) {
                        var scale = scaleZoom;
                        var viewport = page.getViewport(scale);

                        var canvas = document.getElementById('the-canvas');
                        var context = canvas.getContext('2d');

                        canvas.height = viewport.height;
                        canvas.width = viewport.width;

                        foreignobject.select("foreignObject").attr("width",
viewport.width).attr("height", viewport.height);

                        var renderContext = {
                            canvasContext: context,
                            viewport: viewport
                        };
                        page.render(renderContext).promise.then(function () {
                            foreignobject.attr("class", function() {return
(scope.pages[scope.pdf.page - 1].type) ? "" : "hide hidden"})
                            layerStack.attr("class", "layerstack")
                            scope.$apply(function () {
                                scope.pdf.height = viewport.height;
                                scope.pdf.width = viewport.width;
                                console.log(scope.pages)
                                scope.pages[scope.pdf.page - 1].height =
viewport.height/scaleZoom
                                scope.pages[scope.pdf.page - 1].width =
viewport.width/scaleZoom
                                redraw();
                            });
                            foreignobject.attr("width", scope.pdf.width).attr("height",
scope.pdf.height)
                        });
                    });
                });
            }

            scope.$watchGroup(['pdf.zoom', 'pdf.page', 'pages'], function () {
                if (scope.pages[scope.pdf.page - 1].type == true)
                {
                    show_pdf(scope.pages[scope.pdf.page - 1].source_num, scope.pdf.zoom,
scope.pages[scope.pdf.page - 1].source);
                    foreignobject.attr("class", "");
                    imagecanvas.attr("class",  "hide hidden");
                }
                else if (scope.pages[scope.pdf.page - 1].type == false) {
                    foreignobject.attr("class", "hide hidden");
                    imagecanvas.attr("class",  "")

                    image = new Image();
```

```
                    image.src = url + '/' + scope.pages[scope.pdf.page - 1].source;
                    image.onload = function() {
                        layerStack.attr("class", "layerstack")

                        var width = this.width
                        var height = this.height
                        scope.$apply(function () {
                            scope.pdf.height = height * scope.pdf.zoom;
                            scope.pdf.width = width * scope.pdf.zoom;
                        })
                        imagecanvas.select('image')
                        .attr('image-rendering','optimizeQuality')
                        .attr("xlink:href", function () {return url + '/' +
scope.pages[scope.pdf.page - 1].source})
                        .attr("height", this.height * scope.pdf.zoom)
                        .attr("width", this.width * scope.pdf.zoom);
                    }

                }
                else {console.log("neznam koji je format")}

                scope.selected.page = scope.pages[scope.pdf.page - 1]
            }, true);


            // d3.select(".the-canvas").on('resize', console.log('resize'));

            //  var width = d3.select(".layer-window").node().parentNode.clientWidth;
            //  var height = d3.select(".layer-window").node().parentNode.clientHeight;

            var layer = layerStack.selectAll("g.layer")
                .data(scope.layerControl, function (d) {
                    return d.id;
                })
                .enter()
                .append("g")
                .attr("id", function (l) {
                    return "layer" + l.id;
                })
                .attr("class", function (l) {
                    var css = l.name;
                    if (!l.visible) {
                        css = css + ' hidden hide'
                    }

                    return css + ' layer';
                })


            scope.$watch('selectionArea', function (newValue, oldValue) {

                redraw();

            }, true);

            scope.$watch('[pdf.height, pdf.page]', function () {
                canvas.attr("width", 1500).attr("height", scope.pdf.height);
```

```
                svgCanvas.attr("width", 1500).attr("height", scope.pdf.height);
                dragPad.attr("width", scope.pdf.width).attr("height", scope.pdf.height);
                layerStack.attr("width", scope.pdf.width).attr("height", scope.pdf.height)

                redraw();

            }, true);

            function resize() {
                d3.selectAll(".icon").selectAll("svg")
                        .attr("width", scope.selected.iconDimension).attr("height",
scope.selected.iconDimension)
                d3.selectAll(".area-icon").selectAll("svg")
                        .attr("width", scope.selected.iconDimension).attr("height",
scope.selected.iconDimension)
            }

            function redraw() {

                scaleX.domain([0, scope.selected.page.measureValue /
scope.selected.page.measure])
                        .range([0, scope.pdf.width]);

                xAxis.scale(scaleX)

                measureX.call(xAxis);

                yAxis.scale(scaleX)

                measureY.call(yAxis);

                layer.each(function (d, i) {
                    // Kvadrati
                    d3.select(this).selectAll("rect").remove()
                    squares =
d3.select(this).selectAll("rect").data(scope.selectionArea.filter(function (area) {
                        return (area.layerRef == d.id && area.pageRef == scope.pdf.page);
                    }), function (a, b) {
                        return a.id

                    }).enter()

                    squares.append("rect").style('opacity', null)
                        .style("stroke", function (d) {
                            color = scope.icons.find(function(icon) {
                                return d.iconRef == icon.id
                            }).color
                            return "#" + color})
                        .style("fill", "none")
                        .style("stroke-width", 2)
                        .attr("x", function (a, b) {
                            return a.coordinates[0] * scope.pdf.width
                        })
                        .attr("y", function (a, b) {
                            return a.coordinates[1] * scope.pdf.height
                        })
                        .attr("width", function (a, b) {
```

```
                            return a.coordinates[2] * scope.pdf.width
                        })
                        .attr("height", function (a, b) {
                            return a.coordinates[3] * scope.pdf.height
                        })

                    d3.select(this).selectAll(".area-icon").remove()

                    squaresIcons = d3.select(this).selectAll(".area-
icon").data(scope.selectionArea.filter(function (area) {
                        return (area.layerRef == d.id && area.pageRef == scope.pdf.page);
                    }), function (a, b) {
                        return a.id

                    }).enter()

                    squaresIcons.append("svg")
                        .attr('class', 'area-icon')
                        .html(
                            function(d, v){
                                icon = scope.icons.find(function (icon) {
                                    // console.log(icon.id, d.iconRef)
                                    return (icon.id == d.iconRef)
                                })
                                return icon.svg
                            })

                        .attr("y", function (d, i) {
                            icon = scope.icons.find(function (icon) {
                                    // console.log(icon.id, d.iconRef)
                                    return (icon.id == d.iconRef)
                            })
                            return d.coordinates[1] * scope.pdf.height - icon.offset[1] *
scope.selected.iconDimension
                        })
                        .attr("x", function (d, i) {
                            icon = scope.icons.find(function (icon) {
                                    // console.log(icon.id, d.iconRef)
                                    return (icon.id == d.iconRef)
                            })
                            return d.coordinates[0] * scope.pdf.width - icon.offset[0] *
scope.selected.iconDimension
                        })

                    // Ikone
                    d3.select(this).selectAll(".icon").remove()
                    icons =
d3.select(this).selectAll(".icon").data(scope.iconDefinition.filter(function (icon) {
                        return (icon.layerRef == d.id && icon.pageRef == scope.pdf.page);
                    }), function (a, b) {
                        return a.id
                    })
                        .enter()
                        .append("svg")
                        .attr('class', 'icon')
                        .html(
                            function(d, v){
```

```
                                icon = scope.icons.find(function (icon) {
                                    // console.log(icon.id, d.iconRef)
                                    return (icon.id == d.iconRef)
                                })
                                return icon.svg
                            })

                        .attr("y", function (d, i) {
                            icon = scope.icons.find(function (icon) {
                                    // console.log(icon.id, d.iconRef)
                                    return (icon.id == d.iconRef)
                            })
                            return d.coordinates[1] * scope.pdf.height - icon.offset[1] *
scope.selected.iconDimension
                        })
                        .attr("x", function (d, i) {
                            icon = scope.icons.find(function (icon) {
                                    // console.log(icon.id, d.iconRef)
                                    return (icon.id == d.iconRef)
                            })
                            return d.coordinates[0] * scope.pdf.width - icon.offset[0] *
scope.selected.iconDimension
                        })

                    resize();
                });
            }

            scope.$watch('selected.iconDimension', function (newValue, oldValue) {
                redraw();
            });

            scope.$watch('layerControl', function (newValue, oldValue) {
                if (newValue)

                //layerStack.selectAll("g").remove();
                    layer = layerStack.selectAll("g.layer")
                        .data(scope.layerControl, function (d) {
                            return d.id;
                        });

                layer.enter().append("g");

                layer.exit().remove();

                layer.attr("id", function (l) {
                    return "layer" + l.id;
                })
                    .attr("class", function (l) {
                        var css = l.name;
                        if (!l.visible) {
                            css = css + ' hidden hide' + ' layer';
                        }
                        return css + ' layer';
                    })
            }, true);
```

```
            d3.select('body').on('keydown', function () {

                if (d3.event.keyCode == 16) {
                    svgCanvas.on('.zoom', null)
                    layerStack.call(drag);
                }
            });

            var dragForMeasure = d3.behavior.drag()
                .on("dragstart", function () {
                    d3.event.sourceEvent.stopPropagation();
                    pinpoint = d3.mouse(this);
                    console.log(pinpoint)
                    layerStack.append("path")
                        .attr("class", "measure-line")
                        .attr("stroke", "blue")
                        .attr("stroke-width", 2)
                        .attr("fill", "none")
                })
                .on("drag", function() {

                    var line = d3.select('.measure-line')

                    lineData = [{"x": pinpoint[0], "y": pinpoint[1]},
                                {"x": d3.mouse(this)[0], "y": d3.mouse(this)[1]}
                                ]

                    var lineFunc = d3.svg.line()
                        .x(function(d) {return d.x})
                        .y(function(d) {return d.y})
                        .interpolate("linear")

                    line.attr('d', lineFunc(lineData))

                })
                .on("dragend", function () {
                        x = d3.mouse(this)[0];
                        y = d3.mouse(this)[1];

                        var length = Math.sqrt((pinpoint[0] - x)*(pinpoint[0] - x) +
(pinpoint[1] - y)*(pinpoint[1] - y))
                        console.log(pinpoint[0] - x, pinpoint[1] - y, length)

                        d3.select('.measure-line').remove()
                        var page = scope.pages.find(function(d){
                            return d.workspace_page == scope.pdf.page;
                        })
                        scope.selected.page.measure = length/scope.pdf.width
                        console.log(scope.selected.page.measure)

                        layerStack.on('.drag', null)
                        svgCanvas.call(zoom);
                    })


                scope.takeMeasure = function(){
                svgCanvas.on('.zoom', null);
```

```
                layerStack.call(dragForMeasure);
            }

            d3.select('body').on('keyup', function () {
                layerStack.on('.drag', null)
                svgCanvas.call(zoom);
            });

            layerStack.on('mousedown', function (d) {
                coordinates = d3.mouse(this);
                if ((d3.event.ctrlKey) && (scope.layerControl.find(function(d) {return d.id
== scope.selected.layerId}) != null )) {
                    svgCanvas.on('.zoom', null);
                    layerStack.call(drag);
                    d3.select("#layer" + scope.selected.layerId)
                        .append("svg")
                        .attr('class', 'icon')
                        .html(scope.selected.icon.svg)
                        .attr("y", (coordinates[1] - scope.selected.icon.offset[1] *
scope.selected.iconDimension) + "px")
                        .attr("x", (coordinates[0] - scope.selected.icon.offset[0] *
scope.selected.iconDimension) + "px")

                    resize();

                    // area defintion [x, y, width, height]
                    iconCoordinates = [coordinates[0] / scope.pdf.width, coordinates[1] /
scope.pdf.height];
                    var pos = scope.iconDefinition.length;

                    scope.$apply(function () {
                        scope.iconDefinition[pos] = {
                            id: pos + 1,
                            iconRef: scope.selected.icon.id,
                            layerRef: scope.selected.layerId,
                            pageRef: scope.pdf.page,
                            coordinates: iconCoordinates,
                        };
                    });
                }

            });
        }
    };
});  // kraj directive

app.directive("pagePreview", function ($window) {
    return {
        restrict: "EA",
        template: '<ul class="list-inline">' +
        '<li ng-repeat="page in [1,2,3,4,5,6,7,8]">' +
        '<img src="#"></img>{{page}}' +
        '</li>' +
        '</ul>',
        templateNamespace: 'html',
        replace: true,
        scope: {
```

```
        },
        link: function (scope, elem, attrs, ngModel) {
            scope.pages = [1,2,3,4,5,6,7,8];
        }
    }});


app.controller("picWindowCtrl", ["$scope","$window", "$http", function ($scope, $window,
$http) {

    var d3 = $window.d3

    $scope.nextPage = function () {
        $scope.pdf.page++;
        if ($scope.pdf.page <= 0) {
            $scope.pdf.page = 1;
        }
        if ($scope.pdf.page > $scope.pdf.numpages) {
            $scope.pdf.page = $scope.pdf.numpages;
        }
    };
    $scope.prevPage = function () {
        $scope.pdf.page--;
        if ($scope.pdf.page <= 0) {
            $scope.pdf.page = 1;
        }
        if ($scope.pdf.page > $scope.pdf.numpages) {
            $scope.pdf.page = $scope.pdf.numpages;
        }
    };
    $scope.nextPage10 = function () {
        $scope.pdf.page = $scope.pdf.page + 10;
        if ($scope.pdf.page <= 0) {
            $scope.pdf.page = 1;
        }
        if ($scope.pdf.page > $scope.pdf.numpages) {
            $scope.pdf.page = $scope.pdf.numpages;
        }
    };
    $scope.prevPage10 = function () {
        $scope.pdf.page = $scope.pdf.page - 10;
        if ($scope.pdf.page <= 0) {
            $scope.pdf.page = 1;
        }
        if ($scope.pdf.page > $scope.pdf.numpages) {
            $scope.pdf.page = $scope.pdf.numpages;
        }
    };

    $scope.selectionArea = [];
    $scope.iconDefinition = [];

    $scope.pdf = {scale: 1, page: 1, height: 100, width: 100, numpages: 1, zoom: 1};

    // Objekt - true ako je pdf, false za sliku
    $scope.page = function (type, source, source_num, zoom) {
        this.type = type;
```

```
        this.source = source;
        this.source_num = source_num;
        this.zoom = zoom;
        return this
    }

    $scope.getLegend = function(workspace_id) {
        $http.put("../get_legend", {workspace_id: workspace_id})
            .success(function(data, status) {
               $scope.icons = data
            })
            .error(function(data, status) {
                $scope.flag = false;
                      })
    }

    $scope.getLayers = function(workspace_id) {
        $http.put("../get_layers", {workspace_id: workspace_id})
            .success(function(data, status) {
                console.log(data.layers)
                $scope.layerControl = data.layers;
                $scope.iconDefinition = data.icons;
                $scope.selectionArea = data.areas;
            })
            .error(function(data, status) {
                $scope.flag = false;
                      })
    }

    $scope.getPages = function(workspace_id) {
        $scope.workspaceId = workspace_id
        $http.put("../get_pages", {workspace_id: workspace_id})
            .success(function(data, status) {
                $scope.pages = data;
                $scope.pdf.numpages = data[data.length - 1].workspace_page;
                $scope.getLegend(workspace_id)
                $scope.getLayers(workspace_id)
                //$scope.getIcons(workspace_id)

            })
            .error(function(data, status) {
                $scope.flag = false;
                      })
    }

    $scope.saveWorkspace = function() {
        var p = new Promise(function(resolve, reject) {
        $http.put("../save_workspace", {
                                    pages: $scope.pages,
                                    workspace_id: $scope.workspaceId,
                                    iconDefinition: $scope.iconDefinition,
                                    areaDefinition: $scope.selectionArea,
                                    layers: $scope.layerControl})
            .success(function(data, status) {
                resolve('Success!');
            })
            .error(function(data, status) {
```

```
                    })
        }).then(function() {console.log('spremljeno')})
    }


//    $scope.pages = [];
//    $scope.pages.push($scope.page(true,"link",1,1));

//    $scope.iconDefinition = []

    $scope.picture = {scale: 0.1};
    var layerTemplate = {name: "Layer1", visible: true, coordinates: []};

    $scope.action = {delete: false};

    $scope.layerControl = [{id: 1, name: "Layer1", visible: true}];

    $scope.appendLayer = function () {
        var idList = $scope.layerControl.map(function(layer)
        {
            return layer.id
        });
        lastId = Math.max.apply(null, idList)

        $scope.layerControl.push({id: lastId + 1, name: "Layer" + (lastId + 1), visible:
true});
    };

    $scope.appendSelection = function (areaCoordinates) {
        var pos = $scope.selectionArea.length;
        $scope.selectionArea[pos] = {id: "area" + pos + 1, coordinates: areaCoordinates,
pageRef: $scope.pdf.page};
    };

    $scope.deleteLayer = function (i) {
        $scope.layerControl.splice(i, 1);
    };

    $scope.selected = {layerId: 1, icon: 'cloud', iconDimension: 32};

    $scope.zoomIn = function () {
        $scope.pdf.zoom = $scope.pdf.zoom + 0.1
    };

    $scope.zoomOut = function () {
        $scope.pdf.zoom = $scope.pdf.zoom - 0.1
    };

    $scope.iconDimension = function(dim) {
        $scope.selected.iconDimension = dim
    }

    $scope.selectIcon = function(icon){
        $scope.selected.icon = icon
    }
}]);
```

```
app.config(function ($interpolateProvider, $locationProvider) {
    $interpolateProvider.startSymbol('{[{');
    $interpolateProvider.endSymbol('}]}');
    $locationProvider.html5Mode(true);
    $locationProvider.hashPrefix('!');
});
```