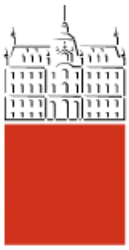
Univerza
v Ljubljani

Fakulteta za
gradbeništvo in
geodezijo

Jamova 2
1000 Ljubljana, Slovenija
telefon (01) 47 68 500
faks (01) 42 50 681
fgg@fgg.uni-lj.si



UNIVERZITETNI ŠTUDIJSKI PROGRAM VODARSTVO IN KOMUNALNO INŽENIRSTVO

Kandidat:

ŽIGA ŽIBERT

SODOBNI NAČINI PRIKAZOVANJA HIDROLOŠKIH MERITEV IN NAPOVEDI

Diplomska naloga št.: 293/VKI

CONTEMPORARY METHODS OF SERVING AND RENDERING HYDROLOGICAL MEASUREMENTS AND FORECAST

Graduation thesis No.: 293/VKI

Mentorica:
doc. dr. Mojca Šraj

Somentor:
doc. dr. Matevž Dolenc

Ljubljana, 20. 06. 2016

STRAN ZA POPRAVKE

Stran z napako	Vrstica z napako	Namesto	Naj bo

Spodaj podpisani študent Žiga Žibert, vpisna številka 26300138, avtor pisnega zaključnega dela študija z naslovom: »Sodobni načini prikazovanja hidroloških meritev in napovedi«

IZJAVLJAM

1. Obkrožite eno od variant a) ali b)

a) da je pisno zaključno delo študija rezultat mojega samostojnega dela;

b) da je pisno zaključno delo študija rezultat lastnega dela več kandidatov in izpolnjuje pogoje, ki jih Statut UL določa za skupna zaključna dela študija ter je v zahtevanem deležu rezultat mojega samostojnega dela;

2. da je tiskana oblika pisnega zaključnega dela študija istovetna elektronski obliki pisnega zaključnega dela študija;

3. da sem pridobil/-a vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v pisnem zaključnem delu študija in jih v pisnem zaključnem delu študija jasno označil/-a;

4. da sem pri pripravi pisnega zaključnega dela študija ravnal/-a v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil/-a soglasje etične komisije;

5. soglašam, da se elektronska oblika pisnega zaključnega dela študija uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;

6. da na UL neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja pisnega zaključnega dela študija na voljo javnosti na svetovnem spletu preko Repozitorija UL;

7. da dovoljujem objavo svojih osebnih podatkov, ki so navedeni v pisnem zaključnem delu študija in tej izjavi, skupaj z objavo pisnega zaključnega dela študija.

Ljubljana, 8.6.2016

Podpis študenta: _____

BIBLIOGRAFSKO – DOKUMENTACIJSKA STRAN IN IZVLEČEK

UDK: **164.08:556.18(497.4)(043.2)**

Avtor: **Žiga Žibert**

Mentorica: **doc. dr. Mojca Šraj**

Somentor: **doc. dr. Matevž Dolenc**

Somentor: **Nejc Pogačnik, univ. dipl. inž. vod. in kom. inž.**

Naslov: **Sodobni načini prikazovanja hidroloških meritev in napovedi**

Tip dokumenta: **Diplomska naloga – univerzitetni študij**

Obseg in oprema: **61 str., 4 preg., 25 sl.**

Ključne besede: **Odprti podatki javne uprave, hidrološki podatki, hidrološke napovedi, spletni servis, mobilna aplikacija**

Izvleček

V svetu, ki je postal popolnoma prepleten in povezan v enotno globalno telekomunikacijsko vas, je postala zelo pomembna dostopnost do podatkov vseh vrst. V diplomi je predstavljeno, kako je v Evropski uniji in Sloveniji urejen dostop do podatkov javnih ustanov in kakšen je njihov potencial za nadaljno uporabo. Na primeru hidrološke prognoze Agencije Republike Slovenije smo prikazali, katere podatke uporabljajo in proizvajajo pri svojem delu. Nadalje smo predstavili orodja in tehnologije, ki nam omogočajo izdelavo sistema za dostop do teh podatkov preko svetovnega spleta.

Predstavljeni hidrološki podatki se nahajajo v podatkovni zbirki, ki je znotraj lokalnega omrežja agencije in je dostopna samo napravam znotraj omrežja. Zato smo postavili podatkovno zbirko zunaj omrežja agencije, ki je dostopna preko svetovnega spleta. Nato smo razvili program, ki redno kopira hidrološke podatke iz lokalne podatkovne zbirko v zunanjo podatkovno zbirko. Dostop do vseh podatkov v zunanji podatkovni zbirki omogočimo s pomočjo spletnega servisa, ki smo ga postavili na spletnem strežniku, ki se nahaja zunaj omrežja agencije. Na podlagi spletnih zahtevkov spletni servis vrača hidrološke podatke v JSON obliki zapisa.

Na koncu naloge smo kot primer uporabe spletnega servisa izdelali Android mobilno aplikacijo. Aplikacija omogoča pregledovanje hidroloških meritev in napovedi preko preglednega seznama in zemljevida samodejnih hidroloških postaj površinskih voda. Za vsako postajo si lahko ogledamo graf pretoka in napovedi pretoka v odvisnosti od časa. Poleg tega aplikacija sprejema obvestila spletnega servisa, ko pretok na samodejnih hidroloških postajah površinskih voda preseže določen prag, in obvesti uporabnika o dogodku.

BIBLIOGRAPHIC DOCUMENTALISTIC INFORMATION AND ABSTRACT

UDC:	**164.08:556.18(497.4)(043.2)**
Author:	**Žiga Žibert**
Supervisor:	**Assist. Prof. Mojca Šraj, Ph.D.**
Cosupervisor:	**Assist. Prof. Matevž Dolenc, Ph.D.**
Cosupervisor:	**Nejc Pogačnik, Water Management and Communal Engineer**
Title:	**Contemporary methods of serving and rendering hydrological measurements and forecasts**
Document type:	**Graduation Thesis – University studies**
Scope and tools:	**61 p., 4 tab., 25 fig.**
Keywords:	**Open government data, hydrological measurements, hydrological forecasts, web services, mobile applicattion**

Abstract

Today's world is more and more connected with the modern technology so accessibility to all kinds of machine readable data has become very important. In this diploma we present what the European Union and the Slovenian Government have done to open the data, make it more accessible to the public and see what is the potential of the open government data. Then we show what kind of hydrological data is used and produced at the Hydrological Forecasting Department of the Slovenian Environmental Agency. We present the tools and technology used to build web services.

Presented hydrological data is located in the local database of the agency and is only accessible to devices on the local network. Because of this, we built database outside of local network that is accessible via world wide web and develop a program that regularly copies the data from the local database to the outer database. We expose all the data in our outer database with the web service that runs on a server that is located outside of the agency network. Based on the web requests web service returns different hydrological data in JSON format.

At the end we develop the Android mobile application that uses web service that we built, to show the data in user friendly style. Application enables a user to view hydrological measurements from stations in a list and a map view. We can also view chart of every station discharge and discharge forecast versus time. Besides that, the mobile application can also receive the notifications from the web service when certain stations discharge exceeds predefined threshold and automatically notifies user of the mobile application.

ZAHVALA

Mentorici doc. dr. Mojci Šraj sem hvaležen za vso spodbudo in pomoč pri izdelavi diplomske naloge. Brez njene nesebične podpore mi ne bi uspelo.

Zahvalil bi se tudi somentorju doc. dr. Matevžu Dolencu za vso tehnično pomoč in komentarje.

Agencija Republike Slovenije za okolje mi je omogočila dostop do vseh hidroloških podatkov, za kar se še posebej zahvaljujem sodelavcem v hidrološki prognozi.

Brez prijetnega vzdušja med študijem, za katerega so poskrbeli sošolci, bi mi bilo veliko težje. Še posebej se zahvaljujem Urški Lotrič in Marjanu Modercu, ki sta mi vedno stala ob strani.

Na koncu pa bi se rad zahvalil staršema in sestri za vso moralno in finančno podporo med študijem.

KAZALO

KAZALO SLIK

KAZALO PREGLEDNIC

KAZALO PROGRAMSKIH KOD

OKRAJŠAVE IN SIMBOLI

API	Application programming interface (vmesnik uporabniškega programa)
ARSO	Agencija Republike Slovenije za okolje
AWS	Amazon Web Services (skupek spletnih storitev v oblaku podjetja Amazon)
BOBER	Boljše opazovanje za boljše ekološke rešitve
EC2	Amazon Elastic Compute Cloud (platforma, ki je del AWS-a in omogoča najem virtualnih računalnikov)
GCM	Google cloud messaging (storitev sporočanja v oblaku)
GIS	Geographic information system (geografski informacijski sistem)
HTTP	Hypertext transfer protocol (protokol za prenos podatkov na spletu)
IDE	Integrated development environment (integrirano razvojno okolje)
INCA	Integrated nowcasting system for the Central European area (integrirani sistem za zelo kratkoročno napovedovanje vremena v srednji Evropi)
JSON	Javascript object notation (Javascript predmetni zapis)
MOP	Ministrstvo za okolje in prostor
MVC	Model-View-Controller (Model-Pogled-Nadzornik)
ORM	Object-relational mapper (tehnika za prevajanje podatkov med različnimi sistemi)
PIP	Pip install Python (programsko orodje, ki nam omogoča namestitev in upravljanje Python knjižnic)
PSI	Public sector information (informacije javnega značaja)
RDS	Relationship Database Service (relacijski podatkovni servis, ki je del AWS-a in omogoča najem različnih vrst podatkovnih baz v oblaku)
REST	Representational state transfer (arhitekturni stil za snovanje aplikacij na svetovnem spletu)
SHN	Sistem za hidrološko napovedovanje
SQL	Structured query language (strukturirani programski jezik za delo s podatkovnimi bazami)
SSH	Secure Shell (kriptografski omrežni protokol, ki omogoča varen oddaljen dostop preko nezavarovanega omrežja)
SSL	Secure socket layer (kriptografski protokol za varno komunikacijo preko omrežja – predhodnik TLS)
TLS	Transport layer security (kriptografski protokol za varno komunikacijo preko omrežja)
URL	Uniform Resource Locator (enolično določen naslov)

WMO World meteorological organization (Svetovna meteorološka organizacija)

XML Extensible Markup Language (razširljiv označevalni jezik)

1 UVOD

V 21. stoletju je prišlo do neverjetnega razvoja na področju informacijske tehnologije, ki je s seboj prinesel veliko število zelo zanimivih in na različnih področjih uporabnih tehnologij. Danes si na primer ne moremo predstavljati življenja brez pametnih telefonov, ki nam omogočajo, da smo na vsakem koraku povezani v svetovni splet in imamo na dosegu roke vse informacije, ki jih potrebujemo za nemoteno življenje. S tehnološkim razvojem je prišla še bolj v ospredje vrednost kakovostnih informacij in podatkov, do katerih lahko dostopajo različni uporabniki in podjetja in na podlagi teh podatkov ustvarjajo nove rešitve in produkte, ki izboljšujejo kvaliteto javnih in zasebnih storitev.

Na področju hidrologije je bilo v zadnjem obdobju v Sloveniji storjeno že veliko s širitvijo merilne mreže hidroloških površinskih postaj in nadgradnjo hidrološkega prognostičnega sistema v okviru projekta Bober (Petan, 2015). S tem je Agencija Republike Slovenije za okolje dobila sodobno in gosto mersko mrežo, ki je še dodatno podprta s sistemom za napovedovanje pretokov na vseh večjih rekah v Sloveniji. Iz izkušenj pa vemo, da je Slovenija geografsko zelo raznolika in da je popolna prostorska napovedljivost meteoroloških pojavov zelo težka, kar posledično pomeni, da imamo lahko hidrološko napoved, ki je količinsko pravilna, vendar prostorsko napačno umeščena. Pri tem je še toliko bolj pomembno delo hidrološkega prognostika, ki bdi nad razvojem dogodkov in obvešča pristojne službe o spremembah. Sodobne aplikacije, ki omogočajo prognostiku pregled nad zelo različnimi vrstami podatkov, zato potrebujejo podatkovni vir, ki izpostavi podatke v obliki, primerni za prikaz.

V nalogi smo prikazali, kako je urejeno področje podatkov javnega značaja na nivoju Evropske unije in Slovenije. Poskusili smo ugotoviti, ali je v interesu javnih organov, da je čim več podatkov dostopnih javnosti. Zanimalo nas je, na kakšen način naj bodo ti podatki dostopni javnosti. Predstavili smo tudi nekaj najboljših praks podatkovnih portalov oziroma spletnih servisov javnih podatkov v Sloveniji. Na primeru hidroloških podatkov Agencije Republike Slovenije za okolje smo izdelali spletni servis, primeren za uporabo v modernih spletnih in mobilnih aplikacijah. Poleg tega pa smo na primeru mobilne aplikacije prikazali uporabo izdelanega spletnega servisa.

2 ODPRTI PODATKI JAVNE UPRAVE

Odprtje javnih podatkov za posredovanje in ponovno uporabo ima lahko velikanske socialno-ekonomske učinke. Podatke, ki jih generira javni sektor, lahko v surovi obliki uporabijo različni subjekti pri izdelavi inovativnih storitev in produktov z dodano vrednostjo, ki spodbudijo ekonomsko rast, nastanek novih delovnih mest in investicije v podatkovni sektor gospodarstva. Poleg tega pa odprtje javnih podatkov in lažji dostop do njih pripomore k večji odgovornosti in preglednosti delovanja javnih ustanov (Evropska komisija, 2014).

Ena glavnih značilnosti držav Evropske unije v primerjavi z drugimi državam sveta je nadpovprečna vpletenost javnega sektorja v nacionalna gospodarstva, ki v povprečju zavzema med 25 in 50 % ekonomije določene države (Carrara, 2015). Posledično javne ustanove namenijo velik del sredstev ustvarjanju podatkov z velikim ekonomskim potencialom, ki močno presega korist javnega sektorja. Potreba in želja po odprtju podatkov javne uprave, ki bi lahko imeli pozitivne ekonomske učinke, ni nekaj novega, ne na nacionalni in ne na ravni Evropske unije. V preteklosti so določene članice evropske unije sprejele zakonodajo, ki je omogočala odprtje podatkov javne uprave in spodbujala njihovo ponovno rabo. Pri sprejemanju zakonodaj na tem področju je prišlo do velikih razlik med članicami, kar je pomenilo, da je prišlo do velikih odstopanj pri obsegu podatkov, ki so bili dostopni javnosti v posamezni državi (Evropska komisija, 2015). Fragmentacija na področju dostopa do podatkov javne uprave je imela škodljive učinke na razvoj notranjega trga Evropske unije, saj so se morala podjetja, ki so želela dostopati do podatkov javne uprave v različnih državah, posebej prilagajati zakonodaji vsake izmed članic.

2.1 Evropska unija

Zavedanje o vrednosti podatkov javne uprave in njihovi ponovni rabi je Evropska unija pokazala leta 2003, ko je v svoj pravni red sprejela direktivo o ponovni rabi informacij javnega značaja, katere glavni cilj je bil enakovredna obravnava vseh potencialnih uporabnikov podatkov javne uprave in poenotenje pravil, ki urejajo dostop do podatkov javne uprave v vseh državah članicah. Omenjena direktiva je leta 2013 dobila amandma, glavni poudarki le-tega so bili (Direktiva 2013/37/EU, 2013):

- večja podpora računalniško berljivim zapisom podatkov,
- v primeru, da se objava podatkov zaračuna, so nadomestila omejena na mejne stroške, ki nastanejo zaradi reprodukcije, zagotavljanja in širjenja podatkov,
- olajšanje dostopa do podatkov preko enotnih podatkovnih portalov, kjer lahko na enem mestu iščemo po različnih virih podatkov.

Poleg sprejemanja zakonodajnih ukrepov, ki odpirajo in spodbujajo ponovno rabo podatkov javnih ustanov, je Evropska unija sprejela tudi vrsto nezakonodajnih ukrepov. Eden izmed najpomembnejših je postavitev podatkovnega portala Evropske unije (Evropska komisija, 2016), kjer lahko dostopamo do vseh podatkov Evropska komisije in ostalih organizacij Evropske unije. Portal predstavlja enovito točko dostopa do rastočega spektra podatkov javnih ustanov Evropske unije, ki jih lahko uporablja kdorkoli v profitne in neprofitne namene. Portal poleg iskanja preko spletnega vmesnika omogoča iskanje in dostop do vseh podatkov portala preko spletnega servisa, ki omogoča razvoj aplikacij s strani tretjih oseb. Dostop do spletnega servisa je mogoč z večino programskih jezikov, ki podpirajo protokol HTTP (ang. *Hypertext transfer protocol*). Vsi zahtevki in odgovori spletnega servisa so v trenutno najbolj razširjeni obliki zapisa imenovani JSON (ang. *JavaScript Object Notation*), ki jo razumejo vse sodobne spletne in mobilne aplikacije. Spletni servis ima na podatkovnem portalu Evropske unije svojo podstran (Evropska unija, 2016), kjer lahko najdemo bogato dokumentacijo in primere kod dostopanja do podatkovnega servisa v različnih programskih jezikih.

Podatkovni portal Evropske unije je lep primer politike odprtih podatkov javne uprave, ki pa ni osamljen. Veliko držav evropske unije je postavilo svoje nacionalne podatkovne portale. Naj omenimo primer Španije, ki je na tem področju orala ledino, ko je leta 2009 kot prva članica Evropske unije postavila nacionalni podatkovni portal in je znana po svoji zelo liberalni politiki glede odprtosti podatkov javne uprave (Evropska komisija, 2015). V letih, ki so sledila, so skoraj vse ostale članice Evropske unije vzpostavile nacionalne podatkovne portale in tako ustvarile okolje, ki spodbuja ponovno rabo podatkov javne uprave.

Ena izmed trenutno največjih političnih prioritet (Evropska komisija, 2016) Evropske komisije je vzpostavitev enotnega digitalnega trga znotraj Evropske unije, kjer je zagotovljen prost pretok oseb, storitev in kapitala, kar pomeni, da lahko osebe in podjetja prosto dostopajo do spletnih storitev in informacij pod enakopravnimi pogoji. Omenili smo že, da je v zadnjih letih prišlo do razvoja podatkovnih portalov na nacionalnih ravneh. Da pa bi vse te portale povezali in maksimizirali njihovo uporabo ter poenostavili iskanje po podatkih različnih nacionalnih javnih uprav, je Evropska unija ustanovila projekt Evropski podatkovni portal, ki na enem mestu omogoča iskanje po več kot 240.000 podatkovnih sklopih iz 34 evropskih držav (Carrara, 2015). Podatki so razdeljeni v 13 različnih kategorij, ki zajemajo področja kmetijstva, prometa, znanosti in ostalih področij.

Vzporedno z vzpostavitvijo Evropskega podatkovnega portala je Evropska komisija naredila tudi korake na področju analiziranja učinkov politike odprtih podatkov javne uprave, zato da bi lahko izboljšala in ugotovila smotrnost aktivnosti na tem področju. Učinki politike odprtih podatkov javne uprave so neposredni in posredni. Pod neposredne sodijo dodana vrednost, realizirana v obliki prihodkov, število služb, udeleženih pri nudenju storitev in produktov, ter stroškovni prihranki.

Posredne učinke pa lahko še dodatno razdelimo na ekonomske, politične in socialne, ki jih največkrat občutijo uporabniki storitev in dobrin, ki jih ponujajo primarni porabniki podatkov javne uprave (preglednica 1) (Granickas, 2013).

Preglednica 1: Posredni učinki politike odprtih podatkov javne uprave (Granickas, 2013)

Posredni učinki politike odprtih podatkov javne uprave		
Ekonomski	Politični	Socialni
Nov zaposlitveni potencial	Transparentnost in odgovornost	Večja vključenost
Nove storitve in dobrine	Javna udeležba	Javna udeležba
Rast ekonomije znanja	Politična zavest/osveščenost	Dostop do informacij
Večja učinkovitost javnih storitev	Dostop do informacij	Podpora pri osebnih odločitvah
Rast		

V do sedaj najobsežnejši študiji učinkov odprtih podatkov javne uprave, ki jo je Evropska komisija izvedla v sodelovanju s svetovalnim podjetjem Capgemini (Evropska komisija, 2015), je bilo ugotovljeno, da bo potencialni trg storitev in dobrin, ustvarjenih na osnovi odprtih podatkov javne uprave v Evropski uniji leta 2016, znašal med 193 in 209 milijardami evrov. Projekcije tudi kažejo, da bo leta 2020 dosegel vrednost med 265 in 286 milijardami evrov. Ob tem moramo poudariti, da učinki politike odprtih podatkov javne uprave zarežejo v različne sektorje ekonomije in da bo največje pozitivne učinke čutil prav javni sektor, katerega delež naj bi v letu 2020 znašal 22 milijard evrov. Na področju rasti števila služb je bila narejena projekcija samo za privatni sektor, tako naj bi imel v letu 2016 potencial za 75.000 zaposlitev, v letu 2020 pa že 100.000, kar pomeni 33 % rast v obdobju petih let. Za javni sektor so tudi zelo zanimivi prihranki pri stroških, ki naj bi v letu 2020 znašali 1,7 milijarde evrov. Poleg tega so bile tudi narejene raziskave na področju povečane učinkovitosti zaradi uporabe odprtih podatkov javne uprave. Ocenjeno je bilo, da lahko rešijo 1.425 življenj na leto, kar pomeni, da zmanjšajo število žrtev prometnih nesreč v Evropski uniji za 5,5 %. Na področju prometa pa je bilo ocenjeno, da lahko odprti podatki javne uprave prihranijo 629 milijonov ur nepotrebnih čakalnih vrst v prometu po celotni Evropski uniji (Evropska komisija, 2015).

Sedaj, ko so države članice Evropske unije sprejele politiko odprtih podatkov javne uprave in se zavedamo njihovih pozitivnih učinkov na gospodarstvo in družbo, moramo sprejeti določene ukrepe, da lahko še naprej merimo učinke politike odprtih podatkov javne uprave in jo poizkušamo še izboljšati (Evropska komisija, 2015):

- stroške in koristi objave podatkov javne uprave naj se še bolj podrobno razišče,
- objava podatkov naj bo brezplačna oziroma naj maksimalno pokrije mejne stroške objave podatkov,
- podatkovni portali naj implementirajo analizo obiskovalcev spletne strani,
- na portalih je treba implementirati anketiranje uporabnikov in pridobiti njihovo mnenje o uporabniški izkušnji,
- javne ustanove naj izvedejo anketiranje na področju ponovne rabe podatkov javne uprave v privatnem sektorju,
- spodbuja naj se ponovno rabo podatkov javne uprave.

2.2 Slovenija

Področje odprtih podatkov javne uprave in njihovo ponovno rabo v Republiki Sloveniji ureja Zakon o dostopu do informacij javnega značaja (ZDIJZ, 2006) in Uredba o posredovanju in ponovni uporabi informacij javnega značaja (Uredba o posredovanju in ponovni uporabi informacij javnega značaja, 2005). Zakon ureja postopek, ki vsakomur omogoča prost dostop in uporabo informacij javnega značaja, s katerimi razpolagajo državni organi, organi lokalnih skupnosti, javne agencije, javni skladi in druge osebe javnega prava, nosilci javnih pooblastil in izvajalci javnih služb. Informacijo javnega značaja zakon definira kot informacijo, ki izvira iz delovnega področja organa, nahaja pa se v obliki dokumenta, zadeve, dosjeja, registra, evidence ali drugega dokumentarnega gradiva. Uredba natančneje določa posredovanje informacij javnega značaja v svetovni splet in prosilcem, zaračunljivost stroškov takega posredovanja, ponovno uporabo informacij javnega značaja in druge pogoje takšne uporabe ter poročanje o zagotavljanju dostopa do informacij javnega značaja.

Vsak državni organ v Republiki Sloveniji je zadolžen, da objavi katalog informacij javnega značaja, kjer lahko vsakdo razbere, katere informacije so na voljo na področju, ki ga pokriva določen organ (Kotnik Šumah, 2010). Poleg tega Ministrstvo za javno upravo predstavlja nekakšen centraliziran organ, ki se ukvarja z dostopom in ponovno rabo podatkov javne uprave in je zadolžen za objavljanje in osveževanje podatkov na spletu. Državni portal e-uprava predstavlja enotni spletni portal, preko katerega lahko javnost dostopa do velikega števila podatkov javne uprave. Poleg tega je treba omeniti tudi nekaj najbolj znanih portalov javnih podatkov (Ministrstvo za javno upravo, 2016):

- Prostorski portal, kjer lahko na enem mestu dostopamo do različnih prostorskih podatkov Geodetke uprave Republike Slovenije (Geodetska uprava Republike Slovenije, 2016).

- Geoportal Arso, ki omogoča iskanje po metapodatkih, pregledovanje različnih podatkovnih slojev v GIS prikazovalniku in prenos različnih okoljskih prostorskih podatkov v domeni Agencije Republike Slovenije za okolje (ARSO, 2016b).
- E-zemljiška knjiga, ki omogoča vpis in javno objavo podatkov o pravicah na nepremičninah in pravnih dejstvih v zvezi z nepremičninami (Sodstvo Republike Slovenije, 2010).
- Supervizor je spletna aplikacija Komisije za preprečevanje korupcije, ki omogoča vpogled v transakcije javnih institucij in družb v lasti države in občin (Komisija za preprečevanje korupcije, 2011).

Kot vidimo, je bilo na področju dostopa do podatkov javne uprave Republike Slovenije narejenih že kar nekaj korakov. Zelo zanimiva je rešitev spletne aplikacije Supervizor, ki poleg brskanja po podatkih preko spleta omogoča dostop do vseh podatkov preko spletnega, kar lahko tretje osebe oziroma razvijalci uporabijo za razvoj svojih aplikacij na osnovi javnih podatkov (Supervizor, 2014). V zadnjem času je bila zelo odmevna javna objava podatkov laserskega snemanja (lidar) za celotno območje Slovenije. Uporabniki lahko brezplačno prenesejo georeferenciran oblak točk ali obdelan klasificiran oblak točk za poljubno območje Republike Slovenije preko spletne aplikacije eVode (Ministrstvo za okolje in prostor, 2015a). Seveda pa pri objavi podatkov javne uprave ni vse rožnato. Tako zasledimo, da pri dostopanju do informacij javnega značaja na področju delovanja države in porabe proračunskih sredstev prihaja do težav predvsem v obliki visokih stroškov, ki velikokrat odvrnejo prosilce od zahtevka in preprečijo dostop do potencialnega dokaza o nepravilnosti ter zmanjšujejo transparentnost delovanja javnih organov, kar pa je eden od ciljev politike odprtih podatkov javne uprave (Albreht, 2015), tako da ima država na tem področju še kar nekaj dela.

3 HIDROLOŠKE MERITVE IN NAPOVEDI

Poplave so eden izmed naravnih pojavov, ki so z drugimi geološkimi procesi izoblikovale in še preoblikujejo zemeljsko površje. Poplavna območja so sestavni del vodotokov in kot del vodnega prostora predstavljajo pomemben vodni ekosistem in pomembno vplivajo na vodni režim, predvsem pri zmanjševanju konic poplavnih valov in bogatenju podtalnice. Pri analizi oziroma izvajanju različnih ukrepov varstva pred poplavami je treba upoštevati celovitost vodnega režima in celotno porečje obravnavati kot celoto (Brilly, 2012). Poplave se lahko pojavijo povsod po obilnem dežju, pri čemer so ranljive vse poplavne ravnice. Poleg tega lahko močne nevihte povzročijo hudourniške poplave, ki v Evropski uniji ogrožajo predvsem gorata območja, kjer lahko močne nevihte poleg hudourniških poplav povzročijo tudi zemeljske plazove in tako še dodatno ogrožajo človeška življenja in objekte.

Pojav poplav ni nekaj novega, tako lahko v različnih analizah preteklih dogodkov razberemo, da so bile poplave med letom 1991 in 2001 najpogostejša z vodo povezana naravna nesreča po svetu ter da povzročijo 15 % smrtnih žrtev, povezanih z naravnimi nesrečami (WMO, 2011). Samo v Evropi je bilo med letoma 1998 in 2009 213 večjih poplavnih dogodkov, ki so povzročili 1126 smrtnih žrtev, začasno preselitev približno pol milijona ljudi in za približno 52 milijard evrov izgub (Evropska okoljska agencija, 2015). V Sloveniji je po 2. svetovni vojni prišlo do izrazite koncentracije prebivalstva na dnu kotlin in širših dolin, ki predstavlja četrtino ozemlja Slovenije, na katerem živi kar 65,5 % slovenskega prebivalstva. Velik del teh prebivalcev je varnih pred poplavami, vendar pa obstajajo območja, ki so izpostavljena, tako v mestnih kot v podeželskih naseljih (Komac in sodelavci, 2008).

Poleg tega pa se pogosto pozabi, da je Slovenija prekrita z aktivnimi območji hudourniških poplav. Po grobih ocenah je v Sloveniji okoli 237.000 ha takšnih zemljišč, kar predstavlja 12 % slovenskega ozemlja (Mikoš, 1995). Hudourniške poplave so kratkotrajne in izjemno silovite, povzročajo pa jih razmeroma kratkotrajne in intenzivne padavine ob poletnih neurjih ali jesenskem deževju (Komac in sodelavci, 2008). O problematiki poplav in njihovih posledicah v Sloveniji še najbolj zgovorno pričajo podatki o smrtnih žrtvah (preglednica 2) in ocenjena neposredna škoda, ki za obdobje med letoma 1990 in 2014 znaša približno 1,8 milijarde evrov (Ministrstvo za okolje in prostor, 2015b).

Preglednica 2: Žrtve poplav v Sloveniji (Brilly, 2012)

Leto	Število žrtev
1926	10
1933	17
1954	22
1990	2
1998	2
2000	7
2004	1
2007	6
2010	5

Evropska unija je prepoznala potencialno nevarnost, ki jo predstavljajo poplave za človeška življenja, ekonomske dobrine in okolje, in leta sprejela Direktivo 2007/60/ES Evropskega parlamenta in sveta o oceni in obvladovanju poplavne ogroženosti (Poplavna direktiva), ki je državam članicam naložila sledeče programsko-načrtovalske naloge:

- priprava predhodne ocene poplavne ogroženosti v letu 2011,
- določitev območij pomembnega vpliva poplav na nivoju celotne države v letu 2012,
- izdelava kart poplavne nevarnosti in ogroženosti za ta območja v letu 2013,
- priprava načrtov za obvladovanje poplavne ogroženosti teh območij v letu 2015.

Z vidika hidrološke prognoze Agencije Republike Slovenije za okolje je predvsem zanimivo šesto poglavje Poplavne direktive, kjer lahko v 3. členu preberemo, da je eden od vidikov obvladovanja poplavne ogroženosti tudi sistem za napovedovanje poplav in zgodnje opozarjanje pred poplavami. Poleg tega je bila v okviru načrta zmanjševanja poplavne ogroženosti izvedena anketa med slovenskimi strokovnjaki s področja upravljanja z vodami, v kateri je bilo področje napovedovanja poplav identificirano kot eno zmed ključnih področij protipoplavnih ukrepov v Sloveniji (Ministrstvo za okolje in prostor, 2015b).

Zavedanje, da popoldne zaščite pred poplavami ni in da je treba dati poplavam prostor za razlivanje, je pripeljalo do potrebe po razvoju in nadgradnji nacionalnih operativnih hidroloških prognostičnih sistemov, katerih glavni cilj je pravočasno obvestilo o prihajajoči poplavi. To omogoči pristojnim službam in javnosti, da se pripravijo na prihajajoči dogodek. Sestavni deli takega sistema so (WMO, 2011):

- zbiranje podatkov o obsegu poplav v realnem času,
- priprava hidroloških napovedi in opozorilnih sporočil, ki na preprost način razložijo, kaj se dogaja in kaj se še lahko zgodi,
- komuniciranje in razširjanje hidroloških napovedi in opozorilnih sporočil,
- interpretacija hidroloških napovedi in meritev, ki ponudijo sveže informacije o možnih posledicah na terenu,
- odziv pristojnih služb na izdano opozorilno sporočilo,
- redni pregledi opozorilnega sistema po poplavnih dogodkih, ki pripomorejo k izboljšanju sistema.

Kakovosten in učinkovit hidrološki prognostični sistem naj bi zagotavljal vse informacije, ki jih za obrambo pred škodljivim delovanjem voda potrebujejo različni uporabniki. Razdelimo jih lahko na tri glavne skupine (Pogačnik, 2010):

- Uprava Republike Slovenije za zaščito in reševanje (URSZR),
- mediji,
- javnost.

Prvi in najbolj pomemben uporabnik informacij je zagotovo URSZR, ki ima vodilno vlogo pri sprožanju opozorilnih protokolov in vodenju protipoplavnih dejavnosti. Druga pomembna skupina so mediji, saj nastopajo v treh vlogah: kot uporabniki informacije za oblikovanje novic, kot partnerji za širjenje informacij, kar dodatno pripomore k hitrejši obveščenosti javnosti, in kot nadzorniki nad kriznim upravljanjem, ko poročajo s terena. Tretja ciljna skupina uporabnikov informacij so prebivalci na ogroženih območjih, ki so neposredno izpostavljeni visokim vodam. Zaradi velikih razlik v razumevanju opozorilnih informacij je treba skrbno oblikovati vsako opozorilo posebej in po samem dogodku analizirati proces opozarjanja in odziv javnosti med dogodkom ter po potrebi prilagoditi opozorilni sistem (Pogačnik, 2010).

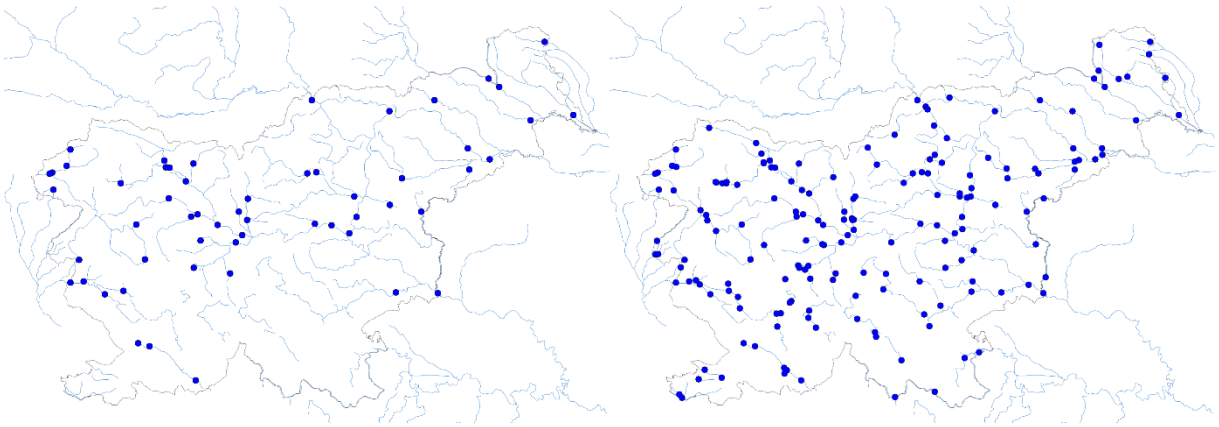
3.1 Agencija Republike Slovenije za okolje

Agencija Republike Slovenija za okolje je organ v sestavi Ministrstva za okolje in prostor. Osnovne naloge agencije so spremljanje, analiziranje in napovedovanje naravnih pojavov in procesov v okolju, tako da se zmanjša naravno ogroženost ljudi in njihovega premoženja. Agencija spremlja onesnaženost okolja in zagotavlja kakovostne javne okoljske podatke ter uresničuje zahteve varovanja okolja, ki izhajajo iz veljavnih predpisov ohranjanja naravnih virov, biotske raznovrstnosti in zagotavljanje trajnostnega razvoja države (Sluga, 2013).

3.1.1 Hidrološka merilna mreža

Z vodomernimi postajami na rekah in jezerih Agencija Republike Slovenije za okolje spremlja režim površinskih voda in hidrološke parametre, potrebne za spremljanje, napovedovanje in obveščanje o hidroloških razmerah, ugotavljanje količinskega stanja voda in hidroloških značilnosti vodnih teles ter za ocenjevanje kemijskega in ekološkega stanja voda. Hidrološki monitoring na rekah zajema meritve višin vodne gladine, hitrosti vode, pretokov, geometrijo merskih prerezov in meritve temperature vode ter vsebnosti suspendiranih materiala v vodi. Na jezerih pa obsega meritve vodostajev in temperaturo vode (ARSO, 2016a).

Agencija Republike Slovenije za okolje je v okviru projekta Nadgradnja sistema za spremljanje in analiziranje stanja vodnega okolja v Sloveniji (Bober), ki je potekal med letoma 2009 in 2015, posodobila merilno mrežo hidroloških površinskih postaj, saj je pred projektom potekal samodejni prenos podatkov na sedež agencije s samo 55 postaj. Podatki v realnem času pa so vitalnega pomena za spremljanje hidrološkega stanja in za pripravo hidroloških napovedi in opozoril. Po končanem projektu bober tako poteka samodejni prenos podatkov s kar 179 samodejnih postaj na površinskih vodah (Roškar, 2015) (slika 1).



Slika 1: Zemljevid samodejnih hidroloških površinskih postaj pred (levo) in po (desno) projektu Bober (ARSO, 2016a)

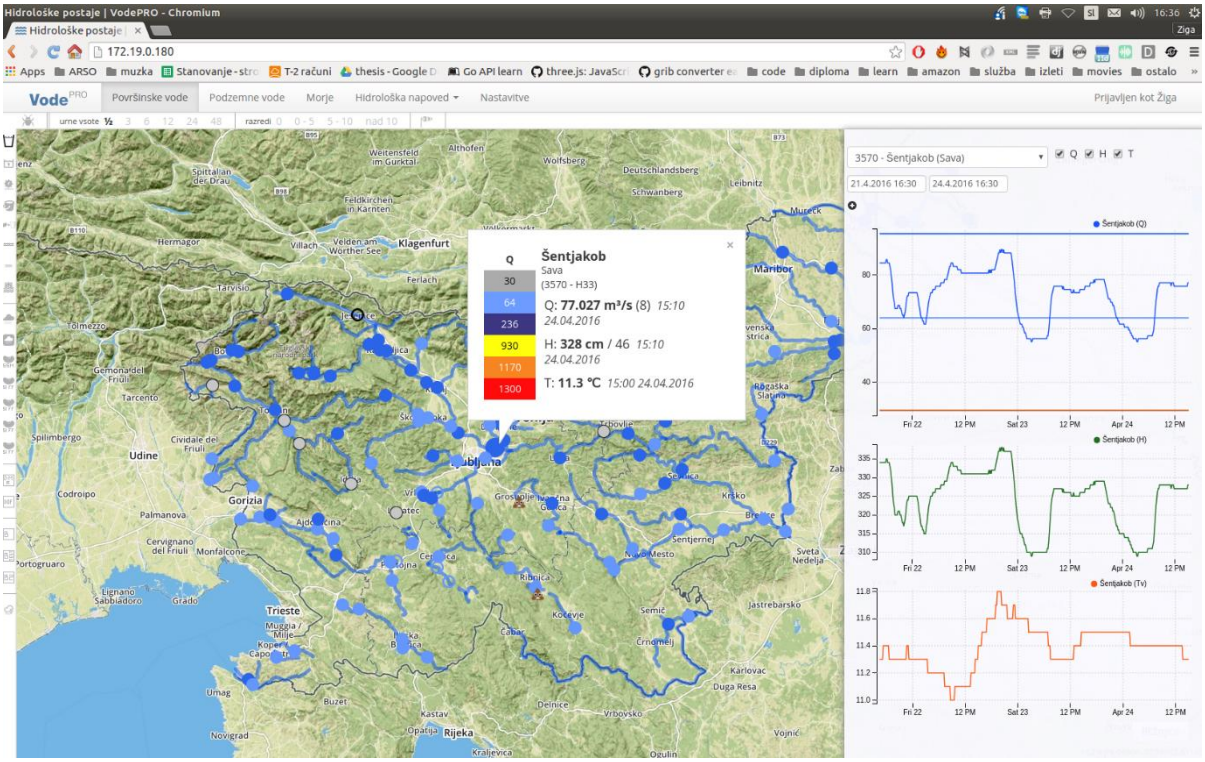
3.1.2 Sistem za hidrološko napovedovanje

Sistem za hidrološko napovedovanje (SHN) je skupek orodij, ki hidrološki prognozi Agencije Republike Slovenije za okolje omogoča napoved hidroloških spremenljivk za nekaj dni vnaprej. Hidrološki službi omogoča pravočasno in podrobnejše spremljanje stanja voda v slovenskih vodotokih, predvsem na področju napovedovanja poplavnih dogodkov na rekah. Razvoj sistema se je začel leta 2006, ko sta hidrološki službi avstrijske Štajerske in Slovenije v svoje delovanje vpeljali sistem za napovedovanje hidroloških razmer na reki Muri. Sistem je bil kmalu prepoznan kot zelo učinkovito orodje pri opravljanju vsakodnevnih nalog hidroloških služb, še posebej ob poplavnih dogodkih. Zato je Agencija za okolje v okviru projekta Nadgradnja sistema za spremljanje in

analiziranje stanja vodnega okolje v Sloveniji med letoma 2010 in 2015 izvedla širitev in nadgradnjo obstoječega sistema za hidrološko napovedovanje na porečje Save do meje s Hrvaško in Soče do izliva v Jadransko morje, pozneje pa tudi na preostala porečja Slovenije. SHN je nameščen na računalniškem sistemu Agencije Republike za okolje in deluje popolnoma avtomatsko. Sistem vsako uro zbere najnovejše meteorološke in hidrološke podatke iz mreže samodejnih merilnih postaj, aktualne napovedi različnih meteoroloških modelov in opravi simulacijo pretokov in vodostajev na različnih modelskih postavitvah za obdobje analize, kar praviloma pomeni za preteklih 72 ur in obdobje napovedi, kar praviloma pomeni za prihodnjih 144 ur. V sklepni fazi se rezultati sistema zapišejo v Oracle podatkovno bazo, kar omogoča aplikacijam, da dostopajo do rezultatov in jih prikažejo v svojih prikaznih rešitvah (Petan, 2015).

3.1.3 Aplikacija VodePro

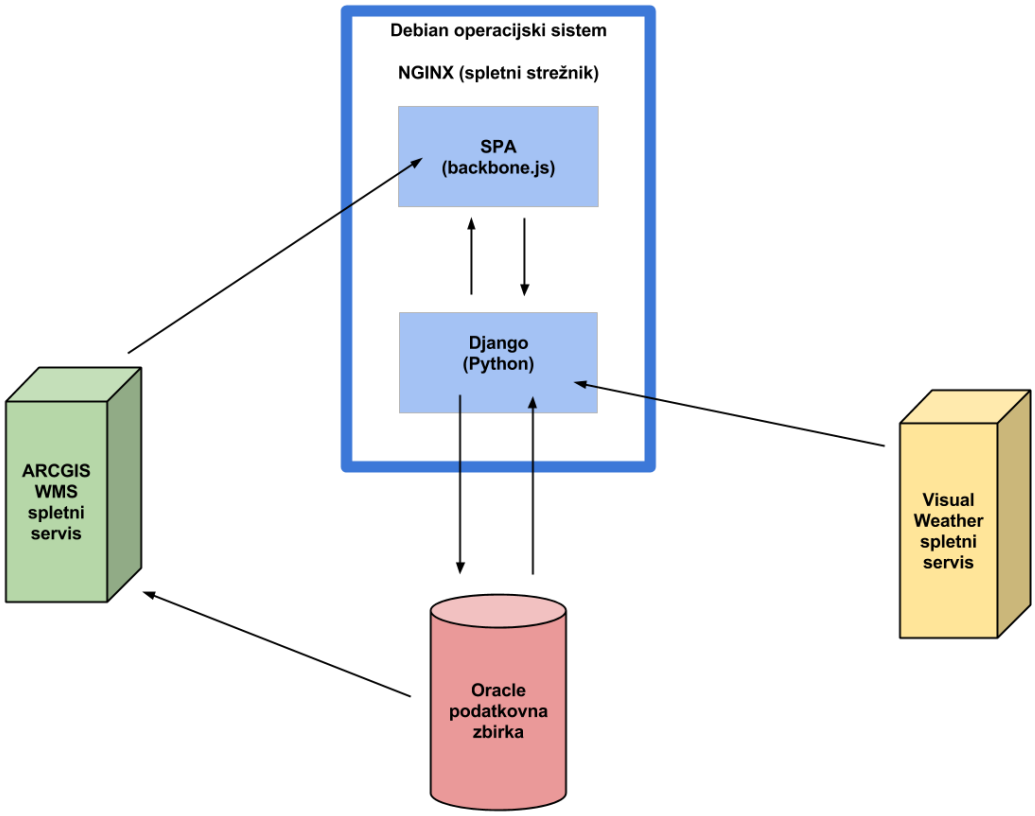
Je sodobna spletna aplikacija, ki jo v hidrološki prognozi Agencije Republike Slovenije za okolje uporabljajo pri vsakodnevnem delu. Aplikacija na enem mestu omogoča pregled vseh meteoroloških in hidroloških meritev in napovedi v realnem času, ter prognostiku na enem mestu zbere vse potrebne informacije za izdelavo hidrološke napovedi in po potrebi izdajo hidrološkega opozorila (slika 2). Aplikacija je bila razvita v sodelovanju s podjetjem XLAB in temelji na odprtokodnih programskih rešitvah, ki omogočajo njeno razširljivost in dodajanje novih funkcionalnosti na preprost način.



Slika 2: Spletna aplikacija VodePro

Vsa programska logika aplikacije je razvita v Javascript programskem jeziku. Do vseh podatkov aplikacija dostopa preko spletnega servisa, ki je spisan v Python programskem jeziku in omogoča, da pridobimo podatke iz Oracle podatkovne baze, jih po potrebi združujemo in filtriramo ter nato v Javascript predmetnem zapisu (JSON) posredujemo spletni aplikaciji VodePro (slika 3).

Potem ko hidrološki prognostik preko aplikacije VodePro izdela hidrološko napoved ali izda hidrološko opozorilo, se le ta zapiše v Oracle podatkovno bazo, od koder se jo nato z različnimi orodji posreduje ustreznim službam in javnosti.



Slika 3: Shema VodePro sistema

4 ORODJA IN TEHNOLOGIJA

V poglavju so opisane in predstavljene tehnologije ter orodja, potrebna za izdelavo sistema, ki bo omogočal serviranje podatkov preko svetovnega spleta in njihovo prikazovanje v mobilni aplikaciji Android.

4.1 Linux

Je prosto dostopen operacijski sistem, ki temelji na brezplačni in odprtokodni platformi razvoja in distribucije sistema. Glavni del Linux operacijskega sistema predstavlja Linux jedro, ki ga je prvič objavil Linus Torvalds 5.oktobra 1991. Od takrat pa do danes je Linux doživel nešteto posodobitev in različnih distribucij sistema, ki jih delimo na namizne različice in različice za strežnike. Med namiznimi različicami so najbolj razširjene (Linux, 2016b):

- Debian,
- Fedora,
- Mint,
- Ubuntu,
- Arch,
- Deepin,
- openSUSE.

Vsaka od namiznih distribucij ima svoj pristop pri grafični zasnovi sistema in programih, ki so že nameščeni na operacijskem sistemu. Za katero se odločimo je odvisno od nivoja znanja in osebnih želja uporabnika. Med strežniškimi distribucijami so najbolj razširjene (Linux, 2016b):

- Red Hat Enterprise,
- Ubuntu Server,
- CentOS,
- SUSE Enterprise.

Glavna značilnost strežniških distribucij je, da ne vsebujejo grafičnega vmesnika. Komunikacija med uporabnikom in sistemom tako poteka izključno preko terminalskega okna. Poleg tega imajo strežniške distribucije že nameščena določena orodja in programe, ki nam olajšajo postavitev in nadzor spletnih sistemov. Poleg prej omenjenih distribucij pa se Linux uporablja tudi na mnogih drugih področjih kot osnova za najrazličnejše sisteme. Trenutno najbolj razširjeni mobilni operacijski sistem Android temelji na osnovi Linux sistema (Linux, 2016a).

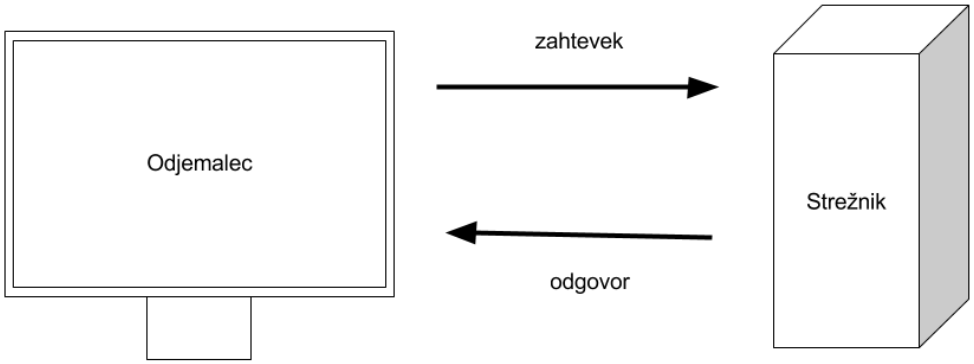
Razvoj in postavitev sistema ter mobilne aplikacije, predstavljene v diplomski nalogi, je v celoti potekal na operacijskem sistemi Linux Ubuntu. Vsa programska oprema in koda, predstavljena v

nadaljevanju, je preizkušena samo na tem sistemu. Vendar je pri tem potrebno poudariti, da smo pri razvoju uporabili odprtokodno programsko opremo, ki v večini primerov podpira vse tri najbolj razširjene operacijske sisteme Windows, Mac in Linux. Tako bi lahko razvoj potekal na kateremkoli od teh treh operacijskih sistemov. Tudi spletni strežnik, na katerem bo nameščena spletna aplikacija, predstavljena v nadaljevanju, bo nameščen na Linux Ubuntu Server virtualnem računalniku.

4.2 Spletni servis

Spletni servis je spletna aplikacija, ki omogoča komunikacijo med elektronskimi napravami na svetovnem spletu. Pri komunikaciji se tipično uporablja protokol HTTP in XML zapis podatkov, ki je strojno berljiv (W3C, 2014). Spletni servisi se uporabljajo kot vstopne točke v lokalne informacijske sisteme. Glavna motivacija za njihov razvoj je želja po izpostavitvi funkcionalnosti in podatkov internih sistemov preko svetovnega spleta na pregleden in kontroliran način (Alonso, 2004).

Tok podatkov med strežniki in odjemalci na svetovnem spletu temelji na protokolu HTTP, ki je najbolj razširjen protokol za izmenjavo podatkov. Protokol HTTP je protokol brez stanja povezave, kar pomeni, da strežnik vsako zahtevo odjemalca obravnava kot neodvisno od vseh prejšnjih zahtev. Tehnologija obstaja že vse od začetka devetdesetih let, verzijo HTTP/1.1, ki jo uporabljamo danes pri vsakodnevni uporabi naprav, ki so priključene v splet, pa poznamo že vse od leta od 1997 (HTTP, 2016).



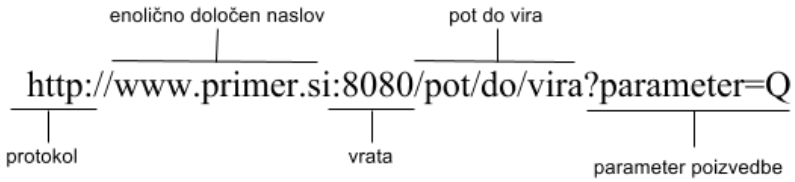
Slika 4: Prikaz komunikacije med odjemalcem in serverjem preko HTTP protokola (Podila, 2013)

V jedru vseh komunikacij preko protokola HTTP je dvojec zahtevek in odgovor (slika 4). Uporabnik preko zahtevka zaprosi oddaljen strežnik za določen vir, ki se nahaja na oddaljenem strežniku. Strežnik pa mu v primeru, da ta vir ima, vrne odgovor, ki vsebuje vir. V primeru, da je bil zahtevek uporabnika nepopoln ali pa da strežnik zahtevanega vira nima, uporabniku oddaljen strežnik odgovori z negativnim statusom. To pomeni, da vsak zahtevek uporabnika dobi odgovor v obliki statusne kode in sporočila odgovora. Vrsta odgovora oziroma statusna koda odgovora, ki jo oddaljen strežnik lahko

vrne uporabniku, je določena v specifikaciji protokola HTTP. Nekaj najbolj razširjenih statusnih kod (Podila, 2013):

- HTTP 200: zahtevek je bil uspešno obdelan,
- HTTP 301: zahtevan vir je dosegljiv na drugem URL-ju,
- HTTP 400: neuspešen/nepravilno oblikovan zahtevek,
- HTTP 500: prišlo je do napake na strežniku.

Vsak zahtevek uporabnika je poslan na spletni naslov URL (ang. *Uniform resource locator*), ki je enolično določen naslov in je sestavljen iz različnih elementov (slika 5). Prvi element določa protokol, preko katerega poteka komunikacija med uporabnikom in strežnikom, drugi element predstavlja naslov gostitelja, tretji element določa vrata, preko katerih poteka komunikacija, četrti element pove pot, kjer se nahaja vir na strežniku, in zadnji element, ki lahko določi enega ali več parametrov zahtevka, ki še natančneje določijo, kaj konkretno želimo od strežnika.



Slika 5: Struktura spletnega naslova (Podila, 2013)

Za izmenjavo podatkov se poleg protokola HTTP lahko za varno komunikacijo med strežnikom in odjemalcem uporabi protokol HTTPS, ki za komunikacijo uporabi dodaten enkripcijski SSL/TLS sloj, da zaščiti promet na spletu (Secure Shell, 2015). Privzeta vrata za komunikacijo preko protokola HTTP so vrata številka 80, vendar pa lahko številko vrat pri oblikovanju zahtevka točno določimo (slika 5). Pot do vira podatkov je vnaprej določena in nam omogoča, da dostopamo do točno določenih podatkov na strežniku, pri tem pa lahko določenemu viru podamo parametre, s katerimi lahko še bolj natančno določimo, kaj želimo, da nam vrne. Poleg spletnega naslova lahko uporabnik določi tudi tip zahtevka, ki ga pošlje na strežnik. S tipom zahtevka določi akcijo, ki se zgodi na strežniku. Najbolj pogosto uporabljeni tipi zahtevkov so (Podila, 2013):

- GET: pridobiti obstoječi vir iz strežnika,
- POST: poslati določen vir na strežnik,
- PUT: posodobiti obstoječi vir na strežniku,
- DELETE: izbrisati obstoječi vir na strežniku.

4.2.1 REST

V zadnjem času se za snovanje spletnih servisov uporablja vedno bolj popularen REST arhitekturni stil, kjer je vsak vir dostopen preko enolično določenega spletnega naslova. REST ni standardiziran protokol, ampak predstavlja skupek pravil, ki za dostopanje in spreminjanje virov uporabljajo standardne HTTP metode, kot so GET, PUT, POST in DELETE. REST spodbuja uporabo različnih HTTP metod, s pomočjo katerih izvedemo želeno operacijo na oddaljenem viru (preglednica 3).

Preglednica 3: Prikaz uporabe tipičnih HTTP metod v REST arhitekturnem stilu (Representational state transfer, 2016)

Enolično določen naslov (URL)	**HTTP metode**			
	GET	**PUT**	**POST**	**DELETE**
Seznam elementov http://www.primer.si/viri/	Pridobimo seznam elementov.	Zamenjamo obstoječi seznam elementov z novim seznamom.	Ustvarimo nov seznam elementov.	Izbrišemo celoten seznam elementov.
Element http.//www.primer.si/viri/23	Pridobimo vsebino elementa.	Nadomestimo obstoječi element z novim oziroma če element še en obstaja, ga ustvarimo.	Ustvarimo nov element v seznamu.	Izbrišemo element iz seznama.

REST ne predpisuje, na kakšen način naj strežnik in odjemalec komunicirata med seboj. Tako bodo nekatere storitve vračale tekstualno besedilo, druge pa binarne podatke. Najpogosteje se za komunikacijo uporablja JSON obliko zapisa podatkov, ki je neodvisna od programskega jezika, ki se uporablja za programiranje spletnega servisa in odjemalca ter v svetu spletnih in mobilnih aplikacij predstavlja podatkovno manj zahtevno alternativo XML obliki zapisa podatkov (Richardson, 2007).

4.3 Nginx

Nginx je odprtokodni spletni strežnik, ki ga lahko namestimo na vse najbolj razširjene operacijske sisteme današnjega časa. Razvit je bil kot alternativa Apache strežniku in daje močan poudarek hitrosti, nizki porabi sistemskih sredstev in možnostjo dela z veliko hkratnimi povezavami. Na operacijski sistem Linux Ubuntu ga namestimo z ukazom (DigitalOcean, 2015):

$ sudo apt-get install nginx

Po namestitvi se Nginx avtomatsko zažene in posluša promet na vratih 80. Uspešnost namestitve lahko preverimo tako, da odpremo spletni brskalnik in se pomaknemo na spletni naslov

http://localhost. Odpreti bi se nam morala pozdravna stran Nginx spletnega strežnika (slika 6), ki teče na našem lokalnem računalniku. Poleg tega lahko kadarkoli pridobimo status Nginx strežnika s pomočjo ukaza (Nginx, 2016):

$ sudo /etc/init.d/nginx status

Vse datoteke z nastavitvami Nginx strežnika se nahajajo v */etc/nginx/* dokumentu. Glavna konfiguracijska datoteka je */etc/nginx/nginx.conf*.

Slika 6: Pozdravno okno Nginx strežnika

4.4 Python

Python je odprtokodni, visoko nivojski, skriptni programski jezik, ki se lahko uporablja na različnih področjih, vse od spletnih aplikacij, numeričnih problemov, mikrokontrolerjev pa do programskih vmesnikov za različne vrste aplikacij. Glavna značilnost jezika je poudarek na berljivosti programske kode in možnosti, da v nekaj vrsticah spišemo programsko logiko, ki bi v ostalih programskih jezikih, kot sta Java in C++, zavzela veliko več vrstic. Python tolmači, ki omogočajo poganjanje Python programske kode, obstajajo za vse najbolj razširjene operacijske sisteme, tako da je mogoče poganjati Python programsko kodo skoraj povsod. Ena od prednosti Pythona je tudi obsežna standardna knjižnica, kar pomeni, da imamo na voljo že celo paleto orodij, ki nam olajšajo delo in pohitrijo reševanje problemov (Python, 2016).

Na preprostem primeru vidimo, kako hitro lahko spišemo Python skripto, ki se bo povezala na oddaljen spletni servis in pridobila trenutne vremenske razmere v Ljubljani ter jih izpisala v terminalsko okno (programska koda 1).

Programska koda 1: Python skripta, ki pokliče oddaljen servis in izpiše podatke v terminal

```
# coding=utf-8
import urllib2
import json

# naredimo HTTP GET klic na izbran URL naslov
url = urllib2.urlopen("http://api.openweathermap.org/data/2.5/weather?"
                      "q=Ljubljan,sl&"
                      "appid=<avtentikacijski_ključ>")

# preberemo odgovor spletnega servisa
vsebina = url.read()

# pretvorimo JSON odgovor v Python objekt
podatki = json.loads(vsebina)

# zapišemo vrednost trenutnega vremena v niz in ga izpišemo v terminal
print "Current weather in Ljubljana: %s." %
podatki['weather'][0]['description']
```

4.5 PostgreSQL

PostgreSQL je objektno-relacijski sistem za upravljanje podatkovnih baz, ki se uporablja kot podatkovni strežnik, katerega glavna funkcija je varno shranjevanje podatkov, upoštevanje vseh najbolj razširjenih SQL standardov in serviranje podatkov na zahtevo različnih zunanjih aplikacij. Lahko se uporablja za majhne aplikacijske rešitve, kot tudi za večje internetne aplikacije, ki imajo več hkratnih uporabnikov. Uporaba PostgreSQL programske opreme je prosto dostopna in odprtokodna rešitev, za razvoj katere skrbi mešanica podjetij in posameznikov. Namestimo ga lahko na vse najbolj razširjene operacijske sisteme, kot so Linux, Microsoft Windows in Mac OS X.

V nadaljevanju smo prikazali, kako se lahko s pomočjo programskega jezika Python povežemo na podatkovno bazo PostgreSQL in izvedemo poizvedbo za podatki ter jih izpišemo v terminal. Najprej moramo na računalnik namestiti knjižnico Psycopg2, ki nam omogoča povezavo med PostgreSQL podatkovno bazo in Python programskim jezikom. Namestimo jo s pomočjo programskega orodja pip (ang. Pip installs Python) z ukazom:
$ pip install psycopg2

Po uspešni namestitvi lahko odpremo naš najljubši urejevalnik kode in spišemo sledečo Python kodo, ki se poveže na izbrano PostgreSQL bazo, naredi poizvedbo in jo izpiše v terminal (programska koda 2) (Drake, 2005).

Programska koda 2: Python skripta, ki se poveže na PostgreSQL podatkovno zbirko in naredi poizvedbo

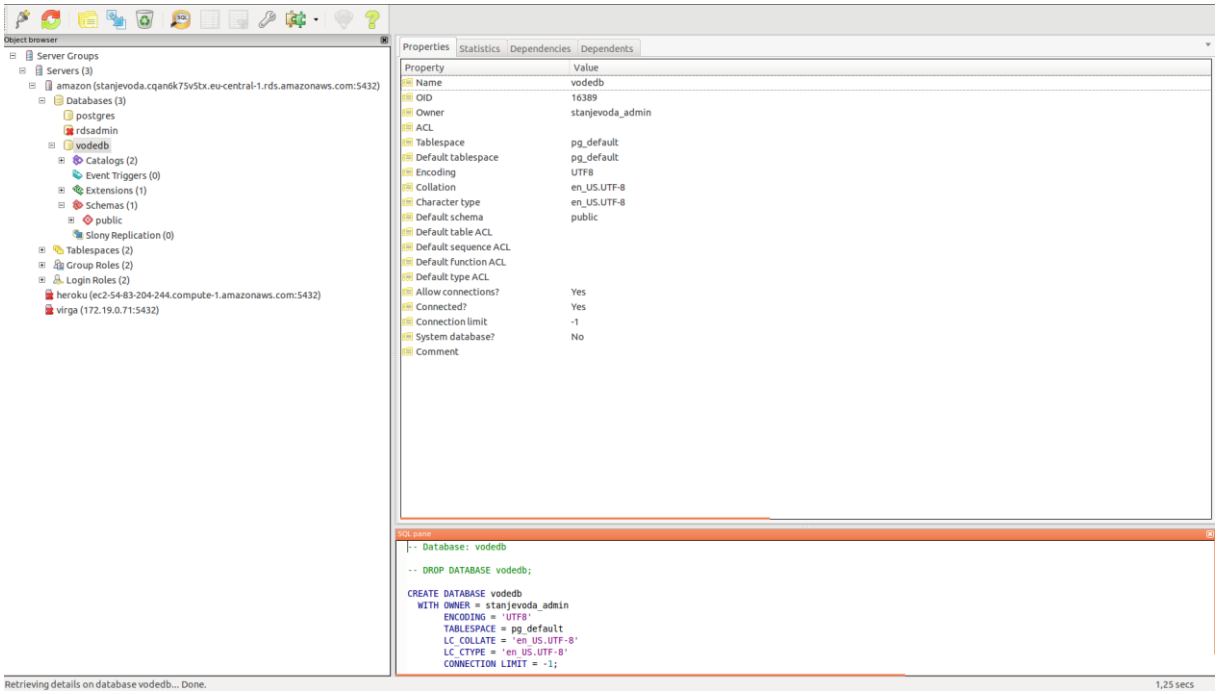
```
# coding=utf-8
import psycopg2

# podatki za vzpostavitev povezave
```

```
DATABASE = {
   'database': '<ime_podatkovne_baze>',
   'user': '<uporabnisko_ime>',
   'password': '<geslo>',
   'host': '<ime_gostitelja>',
   'port': '<stevilka_vrat>'
}

# vzpostavimo povezano z podatkovno bazo
connection = psycopg2.connect(
       database=DATABASE['database'],
       user=DATABASE['user'],
       password=DATABASE['password'],
       host=DATABASE['host'],
       port=DATABASE['port'])

# definiramo kazalec (ang. cursor)
cursor = connection.cursor()

# poženemo poizvedbo, ki pridobi podatke o vseh tabelah v podatkovni bazi
cursor.execute("SELECT * FROM information_schema.tables")

# zapišemo poizvedbo v Python spremenljivko
rows = cursor.fetchall()

# izpišemo vsako vrstico poizvedbe v terminal
for r in rows:
   print r

# zapremo cursor in povezavo na bazo
cursor.close()
connection.close()
```

Python skripto (programska koda 2) zaženemo tako, da odpremo terminal in se pomaknemo na mesto, kjer se skripta nahaja ter vpišemo ukaz:

$ python <ime_skripte>.py

Za lažje upravljanje PostgreSQL podatkovne baze je na operacijskem sistemu Ubuntu prednameščeno programsko orodje PgAdmin III (Slika 7), ki nam omogoča, da preko grafičnega vmesnika upravljamo z našo podatkovno bazo. Če želimo dodati novo podatkovno bazo v program PgAdmin III v orodni vrstici programa, izberemo File > Add server in v pogovorno okno vpišemo podatke naše podatkovne baze. Nato lahko znotraj programa brskamo po novo dodani podatkovni bazi in preko SQL konzole izvajamo operacije na podatkovni bazi (PgAdmin, 2016).



Slika 7: Grafično okno administratorskega okolja PgAdmin III

4.6 Oracle

Oracle je prav tako objektno-relacijski sistem za upravljanje podatkovnih baz, ki se uporablja kot podatkovni strežnik. Glavna razlika v primerjavi s PostgreSQL je, da je Oracle komercialna rešitev in prva tovrstna rešitev, ki je ugledala luč sveta že leta 1977. Na Uradu za hidrologijo in stanje okolja na Agenciji Republike Slovenije za okolje se uporablja za shranjevanje vseh podatkov, ki pridejo iz merilne mreže in aplikacij, ki jo uporabljajo kot podatkovni sloj (Bat, 2010).

Podatke, ki smo jih potrebovali v diplomi, smo iz podatkovne baze Oracle pridobili s pomočjo Python knjižnice cx_Oracle. Preden pa lahko začnemo z uporabo cx_Oracle knjižnice, moramo na lokalnem računalniku namestiti programske gonilnike in nastaviti določene parametre. Najprej moramo z uradne spletne strani podjetja Oracle (http://www.oracle.com/technetwork/database/features/instant-client/index.html) prenesti gonilnike za Oracle podatkovno bazo, tako osnovno kot razvojno verzijo gonilnikov. Oba gonilnika shranimo na lokalni računalnik in ju namestimo s pomočjo programske opreme Alien. Če na lokalnem računalniku še nimamo nameščene programske knjižnice Alien, to lahko storimo s terminalskim ukazom (Ubuntu, 2015):

$ sudo apt-get install alien

Po uspešni namestitvi Alien programske, se v terminalu pomaknemo tja, kamor smo shranili oba gonilnika in ju namestimo z ukazom:

$ sudo alien -i <basic_driver_name>.rpm

in

$ sudo alien -i <develop_driver_name>.rpm

Ko uspešno namestimo oba gonilnika, moramo dodati spremenljivko ORACLE_HOME v sistemsko pot in ji pripisati vrednost, ki bo kazala tja, kamor smo namestili Oracle knjižnico. To storimo tako, da odpremo .bashrc datoteko v domačem dokumentu trenutnega uporabnika in dodamo sledeče vrstice:

export ORACLE_HOME=/usr/lib/oracle/12.1/client64
export PATH=$PATH=$ORACLE_HOME/bin

Nato se pomaknemo v dokument */etc/ld.so.conf* in znotraj njega ustvarimo datoteko z imenom oracle.conf in ji dodamo vrstico, ki pove pot do Oracle knjižnice:

/usr/lib/oracle/12.1/client64

Zadnji korak pri namestitvi Oracle gonilnikov je, da naši novo nameščeni knjižnici dodamo datoteko, v kateri definiramo parametre Oracle podatkovne baze, na katero se želimo povezati. To storimo tako, da se pomaknemo v dokument /usr/lib/oracle/12.1/client64/network/admin/ in ustvarimo datoteko tnsnames.ora z sledečo vsebino:

```
ARSO_TAJFUN =
  (DESCRIPTION =
    (ADDRESS =
      (PROTOCOL = TCP)
      (HOST = <ime_gostitelja/IP naslov>)
      (PORT = <številka_vrat>)
      )
    (CONNECT_DATA =
        (SERVICE_NAME = <ime_servisa>)
    )
)
```

Po uspešno opravljenih korakih za namestitev Oracle gonilnikov lahko na lokalni računalnik namestimo cx_Oracle knjižnico z ukazom (Oracle, 2016):

$ pip install cx_Oracle

Sedaj imamo pripravljena vsa orodja, da se lahko povežemo na Oracle bazo, naredimo testno poizvedbo in izpišemo vrednosti v terminal (programska koda 3).

Programska koda 3: Python skripta, ki se poveže na Oracle podatkovno zbirko in naredi poizvedbo

```
# coding=utf-8
import cx_Oracle

# podatki za vzpostavitev povezave
DATABASE =  {
   'username': '<uporabniško_ime',
   'password': '<geslo>',
   'host': '<ime_gostitelja/IP_naslov>',
   'port': 'številka_vrat>',
   'service_name': '<ime_servisa>'
```

```
}

# vzpostavimo povezano z podatkovno bazo
connection = cx_Oracle.connect('%s/%s@%s:%s/%s'
                               % (DATABASE['username'],
                                  DATABASE['password'],
                                  DATABASE['host'],
                                  DATABASE['port'],
                                  DATABASE['service_name']))

# definiramo kazalec (ang. cursor)
cursor = connection.cursor()

# poženemo poizvedbo, ki pridobi seznam vseh postaj v tabeli merilna_mesta
cursor.execute("select * from merilna_mesta")

# zapišemo poizvedbo v Python spremenljivko
rows = cursor.fetchall()

# izpišemo vsako vrstico poizvedbe v terminal
for r in rows:
   print r
# zapremo cursor in povezavo na bazo
cursor.close()
connection.close()
```

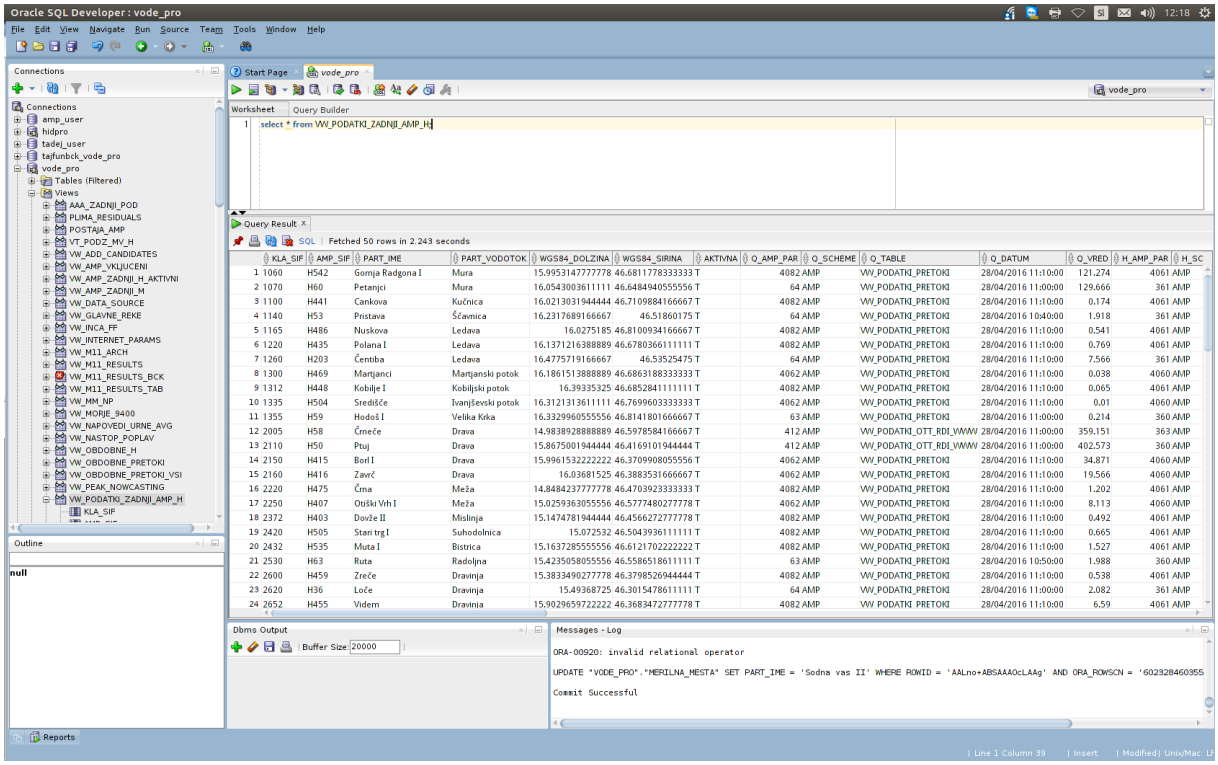
Tako kot pri PostgreSQL podatkovni bazi lahko tudi Oracle podatkovno bazo urejamo in pregledujemo s pomočjo grafičnega vmesnika imenovanega SQL developer (slika 8). Program lahko brezplačno prenesemo iz uradne spletne strani podjetja Oracle (http://www.oracle.com/technetwork/developer-tools/sql-developer/). Ko imamo program prenesen na lokalni računalnik, ga namestimo s pomočjo Alien programske opreme z ukazom:

$ sudo alien <sqldeveloper-X.X.X>.rpm

Po uspešni namestitvi lahko zaženemo SQL developer tako, da se v terminalu pomaknemo v dokument, kamor se je namestil sqldeveloper /opt/sqldeveloper/ in zaženemo ukaz:

$ sh sqldeveloper.sh

Novo povezavo na podatkovno bazo Oracle dodamo tako, da v podoknu Connections, znotraj SQL developerja, kliknemo gumb New connection in v pogovorno okno vnesemo podatke podatkovne baze ter kliknemo gumb Save. Sedaj lahko pričnemo z brskanjem po naši podatkovni bazi znotraj SQL developerja.



Slika 8: Grafični vmesnik administratorskega orodja SQLdeveloper

4.7 Django

Django je prosto dostopen in odprtokoden programski paket, ki nam omogoča razvoj spletnih aplikacij v Python programskem jeziku. Pri razvoju Django aplikacij sledimo paradigmi MVC (ang. *Model-View-Controller*), kar pomeni, da je aplikacija zgrajena iz treh med seboj povezanih, vendar samostojnih komponent. Nadzornik (ang. *Controller*) skrbi za to, da so modeli posodobljeni. Model hrani podatke, ki jih pridobi na podlagi zahtev Nadzornika. Pogled (ang. *View*) pa na podlagi podatkov, ki jih pridobi od Modela, prikaže spletno stran uporabniku (Django, 2015).

Django ima za vse tri zgoraj naštete komponente malo drugačno poimenovanje. Model oziroma podatkovni sloj je definiran s pomočjo objektno-relacijskega mapiranja, ki omogoča povezavo med podatkovnim slojem, definiranim v Python razredih in podatkovno bazo. Uradno Django podpira štiri najbolj razširjene podatkovne baze: PostgreSQL, MySQL, SQLite in Oracle. Nadzorni sloj je definiran, kot URL upravljalec, ki usmerja promet, ki prihaja proti aplikaciji. Na podlagi prometa, ki ga usmerja Nadzornik, tako imenovani Pogled procesira HTTP zahtevke in na podlagi HTML podlag in podatkov iz Modela prikaže spletno stran. Najlažji način namestitve Django programskega paketa je s pomočjo pip-a z ukazom (Django Software Foundation, 2016): *$ pip install Django*

Po uspešni namestitvi lahko avtomatsko ustvarimo Django projekt z ukazom:
$ django-admin startproject moj_projekt

Ukaz bo za nas ustvaril dokument *moj_projekt* v dokumentu, kjer se trenutno nahajamo. Znotraj dokumenta *moj_projekt* se bo nahajala naša nova Django aplikacija in že ustvarjene nekatere osnovne datoteke, ki jih potrebujemo za pričetek razvoja Django aplikacije.

Poglejmo si, kaj je za nas ustvaril ukaz django-admin:

moj_projekt/
    manage.py
    moj_projekt/
        __init__.py
        settings.py
        urls.py
        wsgi.py

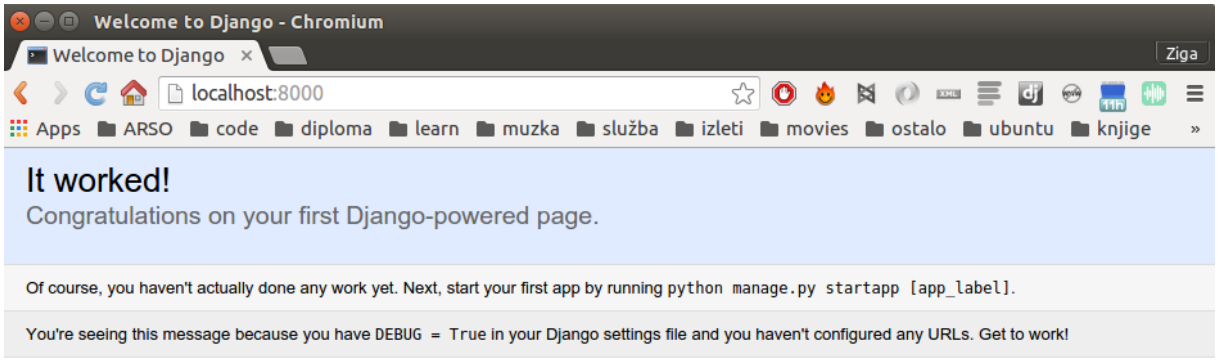
Datoteke so sledeče:

- zunanji dokument *moj_projekt* predstavlja ogrodje našega projekta,
- *manage.py* je terminalsko orodje, ki nam omogoča interakcijo z našim Django projektom,
- notranji dokument *moj_projekt* je Python paket za naš projekt,
- *moj_projekt/__init__.py* je prazna Python datoteka, ki pove Python programskemu jeziku, da je dokument, v katerem se nahaja Python paket,
- *moj_projekt/settings.py* vsebuje vse nastavitve našega Django projekta,
- *moj_projekt/urls.py* določa URL poti našega projekta,
- *moj_projekt/wsgi.py* je vstopna točka v naš projekt za vse WSGI združljive spletne serverje.

Za potrebe razvoja Django omogoča zagon aplikacije s pomočjo lokalnega razvojnega strežnika z ukazom:

$ python manage.py runserver

V brskalniku lahko preverimo uspešnost delovanja naše aplikacije tako, da vpišemo naslov http://localhost:8000 (slika 9)



Slika 9: Pozdravno okno uspešno ustvarjenega in zagnanega Django projekta

Sedaj ko smo postavili okolje za Django projekt, lahko s pomočjo terminalskega orodja manage.py ustvarimo aplikacijo znotraj našega projekta. Razlika med projektom in aplikacijo je v tem, da je vsaka aplikacija znotraj projekta zaokrožena celota, ki nekaj dela npr. blog, spletna stran, anketa. Pri tem je potrebno poudariti, da vsak projekt lahko vsebuje eno ali več aplikacij. Vsaka aplikacija pa je lahko v več projektih. Novo aplikacijo lahko ustvarimo z ukazom:
$ python manage.py startapp spletna_stran

Ukaz bo za nas znotraj projekta *moj_projekt* ustvaril dokument *spletna_stran*, ki bo sestavljen iz sledečih elementov:

spletna_stran/
    __init__.py
    admin.py
    migrations/
        __init__.py
    models.py
    views.py

Datoteke in dokumenti so sledeči:

- dokument *spletna_stran* vsebuje vse datoteke naše nove aplikacije,
- *admin.py* vsebuje django modele, ki jih želimo urejati preko administrativne konzole,
- dokument *migrations* vsebuje datoteke, ki vsebujejo migracije, ki smo jih izvedli na podatkovni bazi na podlagi vsebine v datoteki *models.py*,
- *models.py* vsebuje definicije podatkovnega sloja s pomočjo Python razredov,
- *views.py* grafični del aplikacije.

Na primeru bomo pokazali, kako prikažemo vsebino, ki jo proizvede naša aplikacija. V datoteki *spletna_stran/views.py* dodamo funckijo, ki bo vrnila vsebino na spletne strani (programska koda 4).

Programska koda 4: Funckija index, ki izriše vsebino naše spletne strani

```
# coding=utf-8
from django.http import HttpResponse

def index(request):
   return HttpResponse("Pozdravljen svet. Dobrodošli na moji spletni
strani")
```

Če želimo prikazati vsebino, ki smo jo definirali v funckiji index, jo moramo povezati z URL naslovom. To storimo s pomočjo URL skripte, v kateri povežemo funkcijo index z določenim URL naslovom. Najprej moramo v dokumentu *spletna_stran* ustvariti URL skripto z imenom urls.py in dodati nekaj vrstic Python kode (programska koda 5).

Programska koda 5: Skripta, ki poveže zahtevke, ki pridejo na našo aplikacijo, z ustreznimi funckijami, ki ustvarijo odgovor

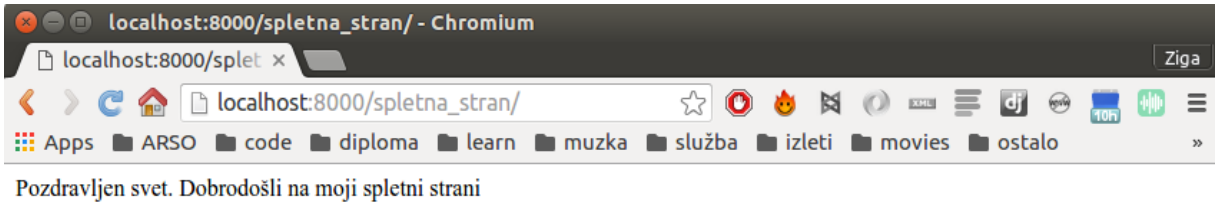
```
from django.conf.urls import url
from . import views

urlpatterns = [
   url(r'^$', views.index, name='index'),
]
```

Naslednji korak je, da v URL skripti našega projekta *moj_projekt/urls.py* usmerimo promet proti naši aplikaciji (programska koda 6)

Programska koda 6: Glavna skripta za usmerjanje prometa proti različnim aplikacijam našega projekta

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
   url(r'^spletna_stran/', include('spletna_stran.urls')),
   url(r'^admin/', admin.site.urls),
]
```

Naša aplikacija bo dosegljiva na naslovu (slika 10). Poleg tega smo omogočili dostop do administratorske konzole našega projekta, ki je dosegljiva na naslovu http://localhost:8000/admin/. Preden pa lahko oba naslova preizkusimo, zaženemo Django projekt s pomočjo ukaza:
$ python manage.py runserver



Slika 10: Prikaz uspešno postavljene spletne strani s pomočjo Django aplikacije

4.8 Bash

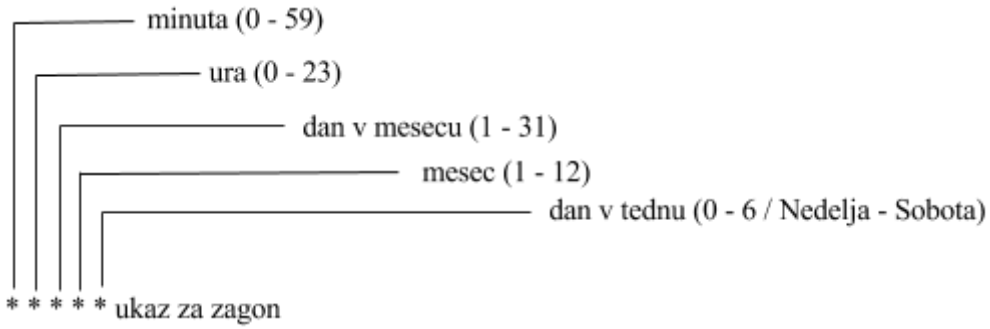
Bash je ukazni programski jezik za terminalsko okolje Unix, ki nam omogoča, da preko Bash ukazov izvajamo operacije na računalniku, kot je ustvarjanje novih datotečnih struktur, premikanje in kopiranje dokumentov ter ostalih operacij, ki nam omogočajo avtomatizacijo administrativnih procesov v terminalskem okolju (Bash, 2016). Na preprostem primeru Bash skripte (programska koda 7) lahko vidimo, kako ustvarimo nov dokument z imenom diploma in znotraj tega dokumenta ustvarimo novo tekstualno datoteko s poljubno vsebino.

Programska koda 7: Primer bash skripte

```
#!/bin/bash
# ustvarimo nov dokument
mkdir diploma
# pomaknemo se v nov dokument
cd diploma
# znotraj dokumenta ustvarimo novo tekstualno datoteka
touch pozdrav.txt
# zapišemo besedilo v tekstualno datoteko
echo Pozdravljen svet. Delam diplomo. Lep pozdrav, Žiga. > pozdrav.txt
```

4.9 Cron

Cron je orodje, ki nam omogoča periodično zaganjanje terminalskih ukazov na Unix operacijskih sistemih. Če želimo dodati terminalski ukaz v Cron, to storimo s pomočjo crontab datoteke, v katero dodamo novo vrstico, ki je sestavljena iz več delov, ločenih s presledki, zadnji del vrstice je terminalski ukaz (Cron, 2016) (slika 11):

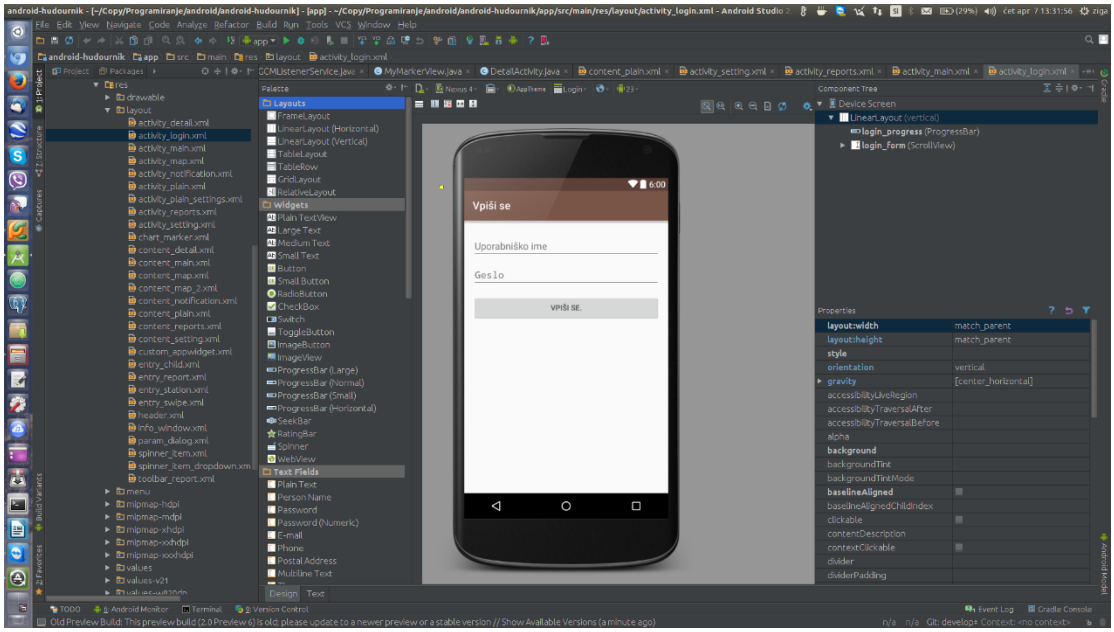


Slika 11: Struktura vrstice v crontab datoteki (Cron, 2016)

4.10 Android

Je mobilni operacijski sistem, ki je osnovan na prilagojeni verziji Linuxa. Prvotno je bil zasnovan v startup podjetju Android in nato leta 2005 prevzet s strani Googla z namenom vstopa na trg mobilnih naprav. Večino kode je Google izdal pod odprtokodno licenco Apache, kar pomeni, da lahko kdorkoli potegne celotno izvorno kodo z interneta in jo prilagodi svojim potrebam. Proizvajalci pametnih telefonov lahko prilagodijo in dodajo svoje vtičnike sistemu Android in tako diferencirajo svoje naprave od drugih. Glavna prednost sistema Android je celovit pristop k razvoju aplikacij, tako da je razvita aplikacija na voljo velikemu številu različnih naprav, ki jih poganja operacijski sistem Android.

Razvoj Android aplikacij poteka s pomočjo integriranega razvojnega okolja (*ang. Integrated development environment*), imenovanega Android Studio (slika 12), ki nam na enem mestu omogoča razvoj, testiranje, razhroščevanje in izvoz aplikacij v obliki, primerni za objavo na spletni trgovini Google Play.



Slika 12: Integrirano razvojno okolje Android Studio

Vsaka Android aplikacija je sestavljena iz ene ali več aktivnosti (*ang. activity*), ki omogočajo interakcijo z uporabnikom. Naenkrat je uporabniku lahko vidna le ena aktivnost. Od trenutka, ko se aktivnost prikaže uporabniku in dokler se ne skrije, gre skozi številne korake/dogodke, znane kot življenjski krog aktivnosti (*ang. activity's life cycle*). Poleg aktivnosti poznamo tudi delčke (*ang. fragments*), ki so bili uvedeni z različico Android operacijskega sistema 3.0. Delčki predstavljajo nekakšne miniaturne aktivnosti, ki jih lahko združujemo, da predstavljajo aktivnost. Poleg aktivnosti in delčkov pozna Android OS še en koncept, imenovan namen (*ang. intent*), ki predstavlja nekakšno lepilo, ki povezuje različne aktivnosti enakih ali drugih aplikacij. Aktivnost (*ang. Activity*) ustvarimo tako, da naš Java razred (*ang. Class*) GlavnaAktivnost podaljša (*ang. Extends*) osnovni razred Activity (programska koda 8).

Programska koda 8: Primer preproste Android Aktivnosti

```
package si.ziga.pozdravljensvet;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class GlavnaAktivnost extends Activity {
   private final static String TAG = GlavnaAktivnost.class.getSimpleName();
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.aktivnost_glavna_aktivnost);
       Log.d(TAG, "onCreate()");
   }
   public void onStart(){
    super.onStart();
```

```
    Log.d(TAG, "onStart()");
   }
   public void onRestart(){
    super.onRestart();
    Log.d(TAG, "onRestart()");
   }
   public void onResume(){
    super.onResume();
    Log.d(TAG, "onResume()");
   }
   public void onPause(){
    super.onPause();
    Log.d(TAG, "onPause()");
   }
   public void onStop(){
         super.onStop();
         Log.d(TAG, "onStop()");
   }
   public void onDestroy(){
         super.onDestroy();
         Log.d(TAG, "onDestroy()");
   }
}
```

Osnovni razred Aktivnost definira serijo dogodkov (programska koda 5), ki se zgodijo tekom življenjskega kroga aktivnosti (Lee, 2012):

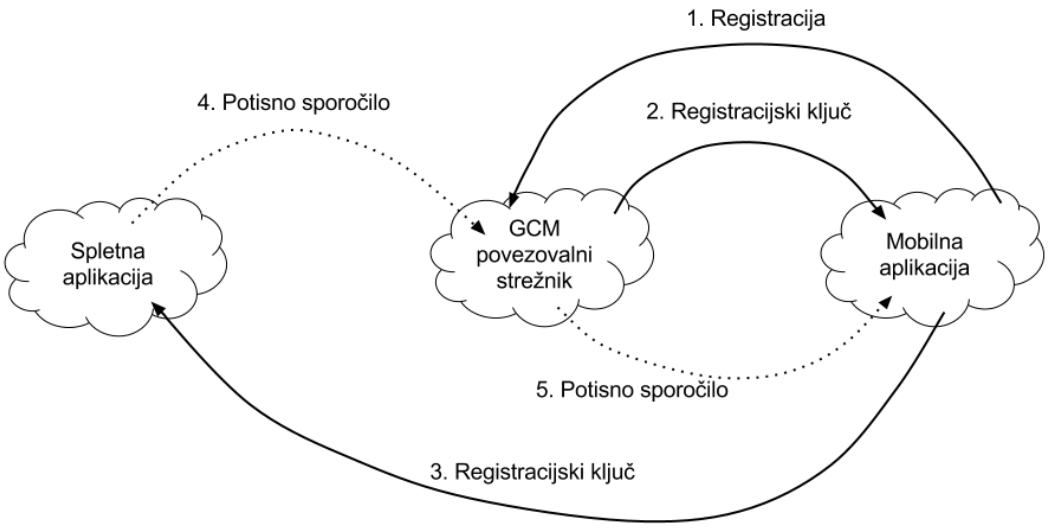
- onCreate() se sproži, ko je aktivnost prvič ustvarjena,
- onStart() se sproži, ko je aktivnost vidna uporabniku,
- onResume() se sproži, ko uporabnik začne aktivno sodelovati,
- onPause() se sproži, ko trenutna aktivnost počiva in v ospredje pride prejšnja aktivnost,
- onStop() se sproži, ko aktivnost ni več vidna uporabniku,
- onDestroy() se sproži, preden je aktivnost uničena s strani operacijskega sistema,
- onRestart() se sproži, ko se aktivnost ponovno prikaže, potem ko je bila ustvarjena (onStop).

4.11 Potisna sporočila

Tehnologija potisnih sporočil nam omogoča, da iz naše spletne aplikacije pošiljamo sporočila mobilnim napravam, ki so povezana v svetovni splet in v obratni smeri tudi sprejemamo sporočila. Podjetje Google ponuja storitev Google Cloud messaging (GCM), ki poskrbi za vse vidike pošiljanja sporočil in je popolnoma brezplačna. Mobilna aplikacija lahko sprejme sporočilo tudi takrat, ko ni v teku (Google, 2016a).

Sestavni deli sistema, ki omogoča pošiljanje potisnih sporočil, so sestavljeni iz (Grastovšek, 2014) (slika 13):

- GCM povezovalnega strežnika,
- spletne aplikacije in
- mobilne aplikacije.

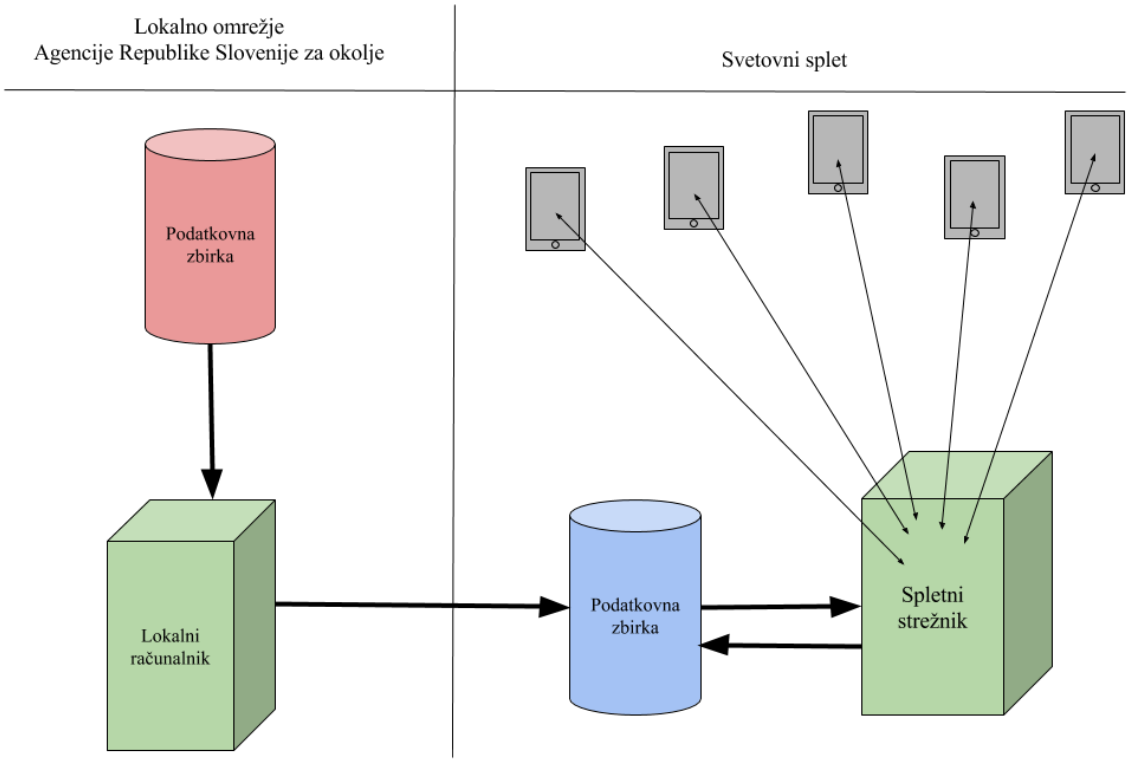


Slika 13: Diagram naročanja in prejemanja potisnih sporočil na platformi GCM (Grastovšek, 2014)

GCM povezovalni strežnik sprejme sporočila iz naše spletne aplikacije in jih pošlje mobilni aplikaciji. Poleg tega na zahtevo mobilne aplikacije, ki je implementirala GCM sistem za pošiljanje potisnih sporočil, izda registracijski ključ, ki predstavlja nekakšen unikaten identifikator mobilne naprave. Potem ko mobilna aplikacija prejme registracijski ključ, le-tega pošlje matični spletni aplikaciji, ta pa ga uporabi, ko pošilja sporočilo mobilni aplikaciji preko GCM povezovalnega strežnika (Google, 2016b; 2016c).

5 IZDELAVA SPLETNEGA SERVISA

Izdelavo spletnega servisa smo razdelili na dve logično zaključeni celoti, ki bosta delovali neodvisno druga od druge. Prva je sestavljena iz programa, ki skrbi za kopiranje podatkov iz podatkovne zbirke Agencije Republike Slovenije za okolje, ki se nahaja znotraj lokalnega omrežja in ni dostopna zunanjemu svetovnemu spletu, na zunanjo podatkovno zbirko, kjer so podatki dostopni spletnemu strežniku. Druga pa je sestavljena iz zunanje podatkovne zbirke in spletnega strežnika, na katerem je nameščena spletna aplikacija, s pomočjo katere serviramo podatke. Stična točka obeh sistemov je zunanja podatkovna zbirka (slika 14). Razlog za zunanjo podatkovno zbirko je varnostni, saj bomo tako popolnoma izolirali promet spletnega servisa samo na zunanjo podatkovno zbirko. Poleg tega pa bomo preprečili preobremenjenost notranje podatkovne zbirke in omogočili njeno nemoteno delovanje.

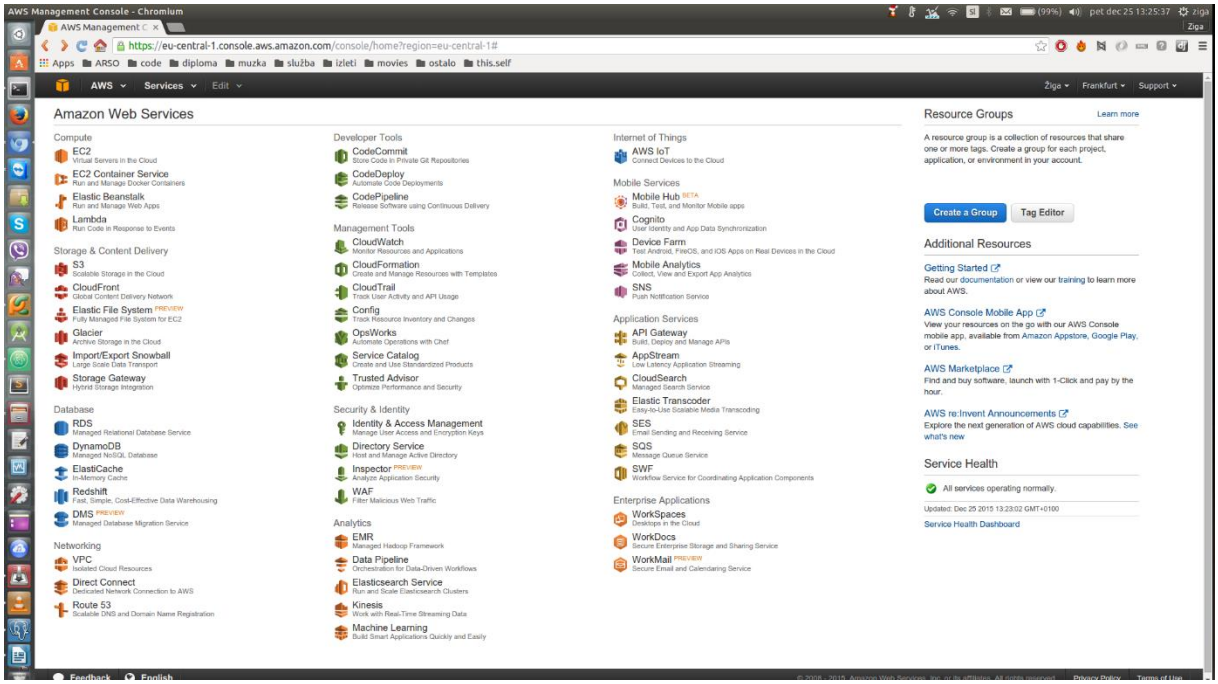


Slika 14: Shema celotnega sistema s prikazanim tokom podatkov

5.1 Amazon Web Services

Celoten zunanji del sistema, sestavljen iz podatkovne zbirke in virtualnega računalnika, je postavljen na spletni platformi AWS, ki omogoča, da na zelo preprost način preko spletnega vmesnika postavimo virtualne strežnike in podatkovne zbirke v oblaku. Prvo leto uporabe dobimo brezplačen dostop do

storitev v oblaku z določenimi omejitvami, kot je obseg prometa, velikost podatkovne zbirke in število instanc virtualnih računalnikov. Po uspešni registraciji na spletni strani AWS lahko preko spletnega administratorskega vmesnika ustvarjamo, administriramo in pregledujemo storitve AWS (Amazon, 2016) (slika 15).



Slika 15: Spletna stran AWS sistema

5.2 Virtualni računalnik

Preko AWS spletne strani smo ustvarili Amazon EC2 virtualni strežnik v oblaku z prednameščenim operacijskim sistemom Ubuntu 14.04. Med postopkom izdelave nove instance strežnika se nam ustvari certifikat, s pomočjo katerega se lahko preko SSH povežemo na oddaljeni računalnik z ukazom (Amazon, 2015a; 2015b; 2015c; Wikipedia, 2015):

$ ssh -i <pot_do_certifikata> uporabnik@enolično_določen_spletni_naslov_našega_strežnika

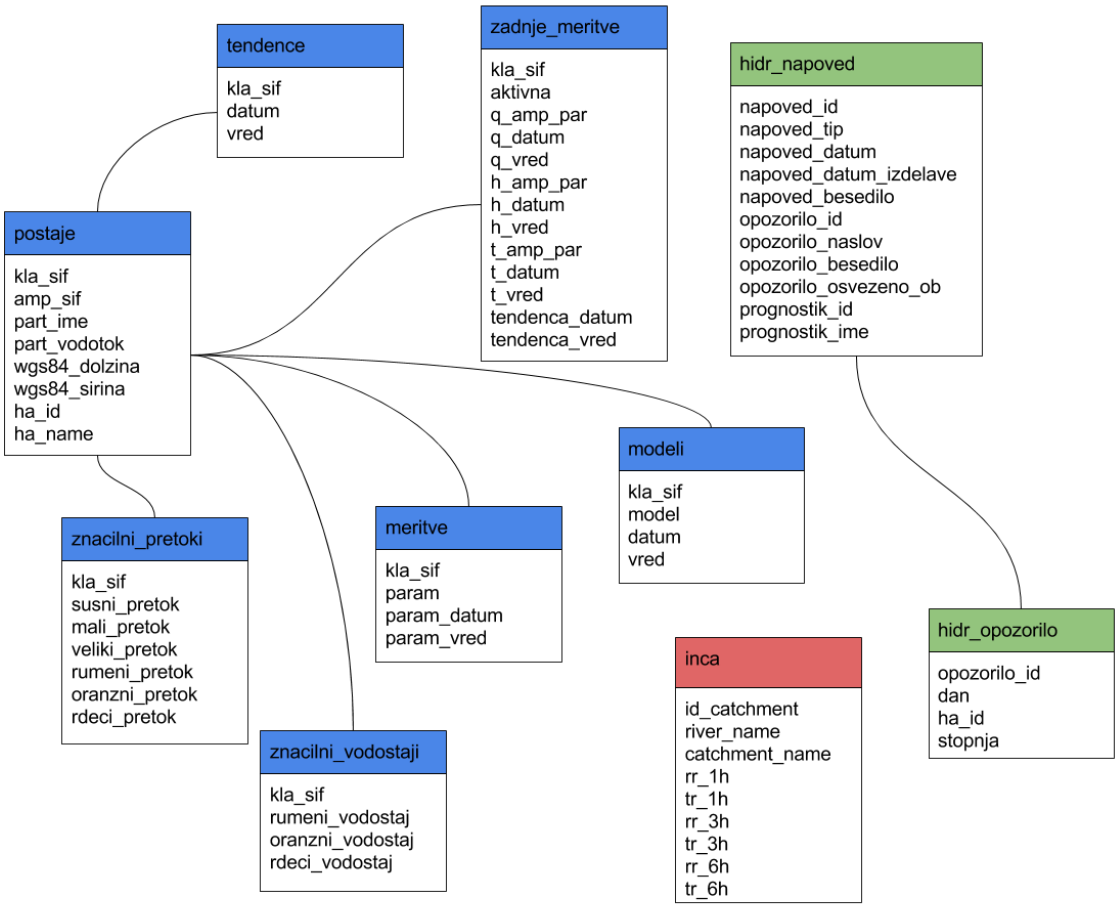
5.3 Podatkovna zbirka

Podatkovno zbirko PostgreSQL smo prav tako ustvarili preko Amazon AWS spletne konzole. Sistem za upravljanje podatkovnih zbirk se imenuje Amazon RDS in omogoča postavitev katerekoli od trenutno popularnih podatkovnih zbirk: MySQL, Oracle, PostgreSQL in ostalih. Preko preprostega čarovnika smo za potrebe naloge ustvarili PostgreSQL podatkovno zbirko, ki bo podatkovna osnova za celoten spletni servis, ki ga želimo izdelati v nadaljevanju. Uspešno delovanje naše nove podatkovne zbirke lahko preverimo s pomočjo brezplačnega administratorskega programa PgAdmin, ki je na operacijskem sistemu Ubuntu že prednameščen. Z nekaj preprostimi koraki dodamo podatke o

naši novi podatkovni zbirki v prej omenjeni program in že lahko administriramo oddaljeno podatkovno bazo (PgAdmin, 2016)

5.3.1 Podatkovna shema

Na zunanji PostgreSQL podatkovni zbirki smo ustvarili podatkovno shemo (slika 16), ki predstavlja podatkovni vir spletnega servisa in se bo osveževala s podatki, ki jih bo naš program kopiral iz notranje Oracle podatkovne zbirke.



Slika 16: Podatkovna shema zunanje podatkovne zbirke PostgreSQL

Tabela »postaje« je namenjena seznamu hidroloških površinskih postaj in pripadajočih podatkov, kot so ime postaje, ime vodotoka, na katerem je postaja, lokacija postaje in ime opozorilnega območja, na katerem je postaja. V tabeli »postaje« polje kla_sif predstavlja šifro, preko katere bomo lahko vse podatke v ostalih tabelah povezali z izbrano hidrološko površinsko postajo. Tabeli »znacilni_pretoki« in »znacilni_vodostaji« sta namenjeni podatkom o statističnih in opozorilnih vrednostih pretokov in vodostajev na posamezni hidrološki postaji. Tabela »zadnje_meritve« bo vsebovala najbolj sveže podatke meritev iz hidroloških površinskih postaj in trenutno aktivne senzorje na izbrani postaji.

Časovnemu preseku meritev vodostaja, pretoka in temperature vode za zadnjih 7 dni je namenjena tabela »meritve«. Časovne serije napovedi hidrološkega prognostičnega sistema bomo kopirali v tabelo »modeli«, kjer bodo za vsako lokacijo samodejne hidrološke površinske postaje na voljo rezultati različnih hidroloških prognostičnih modelov. V tabeli »tendence« bomo hranili podatke o tendenci pretoka na postajah hidroloških površinskih postaj za zadnjih 24 ur.

Poleg prej omenjenih tabel smo v podatkovno shemo vključili tudi tabeli »hidr_napoved« in »hidr_opozorilo«, ki sta namenjeni podatkom o hidrološki napovedi in opozorilu, ki jih hidrološki prognostik ročno izdela preko aplikacije VodePro, in sta zelo zanimivi z vidika vključevanja hidroloških produktov Agencije Republike Slovenije za okolje v ostale sisteme, predvsem v času izrednih razmer, ko preti nevarnost poplav.

Zadnja izmed tabel v podatkovni shemi je imenovana »inca« in vsebuje podatke o eno-, tri- in šest-urni akumulaciji padavin na posameznih podporečjih hidrološkega prognostičnega sistema. Podatki o akumulaciji so pridobljeni iz meteorološkega sistema, imenovanega Integrirani sistem za zelo kratkoročno napovedovanje vremena v srednji Evropi (Šajn Slak, 2012).

5.3.2 Osveževanje podatkovne zbirke

Za kopiranje podatkov iz notranje podatkovne zbirke Oracle v zunanjo podatkovno zbirko PostgreSQL smo izdelali program, ki je razdeljen na več manjših enot, od katerih je vsaka zadolžena, da za določen časovni interval osveži predpisano tabelo v zunanji podatkovni zbirki. Vsem enotam programa je skupno nekaj globalnih Python skript, ki olajšajo vzdrževanje in preglednost programa. Globalne Python skripte so dostopne vsem enotam programa in so sledeče:

- query.py vsebuje vse SQL poizvedbe na enem mestu,
- info.py vsebuje vse konstante programa (uporabniška imena, gesla, IP naslove, elektronske naslove),
- utils.py vsebuje metode, ki se uporabljajo zelo pogosto po celotnem projektu (povezovanje na bazo, pošiljanje elektronskih sporočil, poročanje o napakah in ostalo),
- set_up_remote.py vsebuje logiko, ki ustvari vse tabele (slika 16) na zunanji podatkovni zbirki PostgreSQL.

Poleg tega vsaka enota programa vsebuje:

- Python skripto,
- Bash skripto,
- Log dokument.

Vsaka Python skripta (programska koda 9) znotraj enote se poveže na notranjo podatkovno zbirko Oracle in naredi poizvedbo za izbranimi podatki. Nato se poveže na zunanjo podatkovno zbirko PostgreSQL in vstavi podatke, ki jih je pridobila v prejšnjem koraku, v predpisano tabelo.

Programska koda 9: Python skripta, ki kopira podatke podatke zadnjih meritev samodejnih hidroloških površinskih postaj

```
import sys
import utils
import query


try:
    # povežemo se na notranjo podatkovno zbirko Oracle
    connection_oracle = utils.get_connection('local')
    cursor_oracle = connection_oracle.cursor()

    # naredimo poizvedbo za zadnjimi meritvami
    data = query.get_latest_data(cursor_oracle)

    # odklopimo se od lokalne podatkovne zbirke Oracle
    cursor_oracle.close()
    connection_oracle.close()

    # povežemo se na zunanjo podatkovno zbirko PostgreSQL
    connection_postgre = utils.get_connection('remote')
    cursor_postgre = connection_postgre.cursor()

    # izbrišemo stare podatke iz tabele zadnje_meritve
    cursor_postgre.execute("delete from zadnje_meritve")

    # zapišemo podatke v csv datoteko
    utils.prepare_station_csv('temp.csv', data)

    # vstavimo podatke v tabelo zadnje_meritve
    query.insert_csv_data(cursor_postgre, 'temp.csv', 'zadnje_meritve',
null='NULL')

    # potrdimo spremembe na zunanji podatkovni zbirki PostgreSQL
    cursor_postgre.execute('commit')

    # odklopimo se od zunanje podatkovne zbirke PostgreSQL
    cursor_postgre.close()
    connection_postgre.close()
except Exception as e:
    # v primeru napake to javimo preko elektronskega sporočila
    utils.error_handler()
    sys.exit()
```

Bash skripta (programska koda 10) zažene Python skripto in pred vsakim zagonom poskrbi, da v ozadju ne teče ista verzija Python skripte in po potrebi nastavi sistemske spremenljivke.

Programska koda 10: Bash skripta, ki zažene Python skripto

```
#!/bin/bash
```

```
# preverimo, če v ozadju še vedno teče Python skripta latest.py in jo
ustavimo
ps axjf | grep 'latest.py' | awk {'print $2'} | xargs kill -9

# izvozimo sistemske spremenljivke, ki jih potrebuje Python skripta
latest.py za uspešno delovanje
export PYTHONPATH=/home/ziga/Documents/graduate-
thesis/server_side/remote_logic

# zaženemo Python skripto
python /home/ziga/Documents/graduate-
thesis/server_side/remote_logic/latest/latest.py
```

Za redno zaganjanje Bash skript v izbranih časovnih intervalih pa poskrbi orodje Cron, ki je že nameščen na vseh računalnikih Linux in se ga lahko upravlja preko datoteke crontab. Vsako izmed Bash skript zaženemo na izbrani časovni interval, tako da dodamo vrstico v datoteko crontab (programska koda 11). Vsak dogodek, ki se zgodi med poganjanjem skripte Bask, se zabeleži v dnevniško datoteko imenovano log.

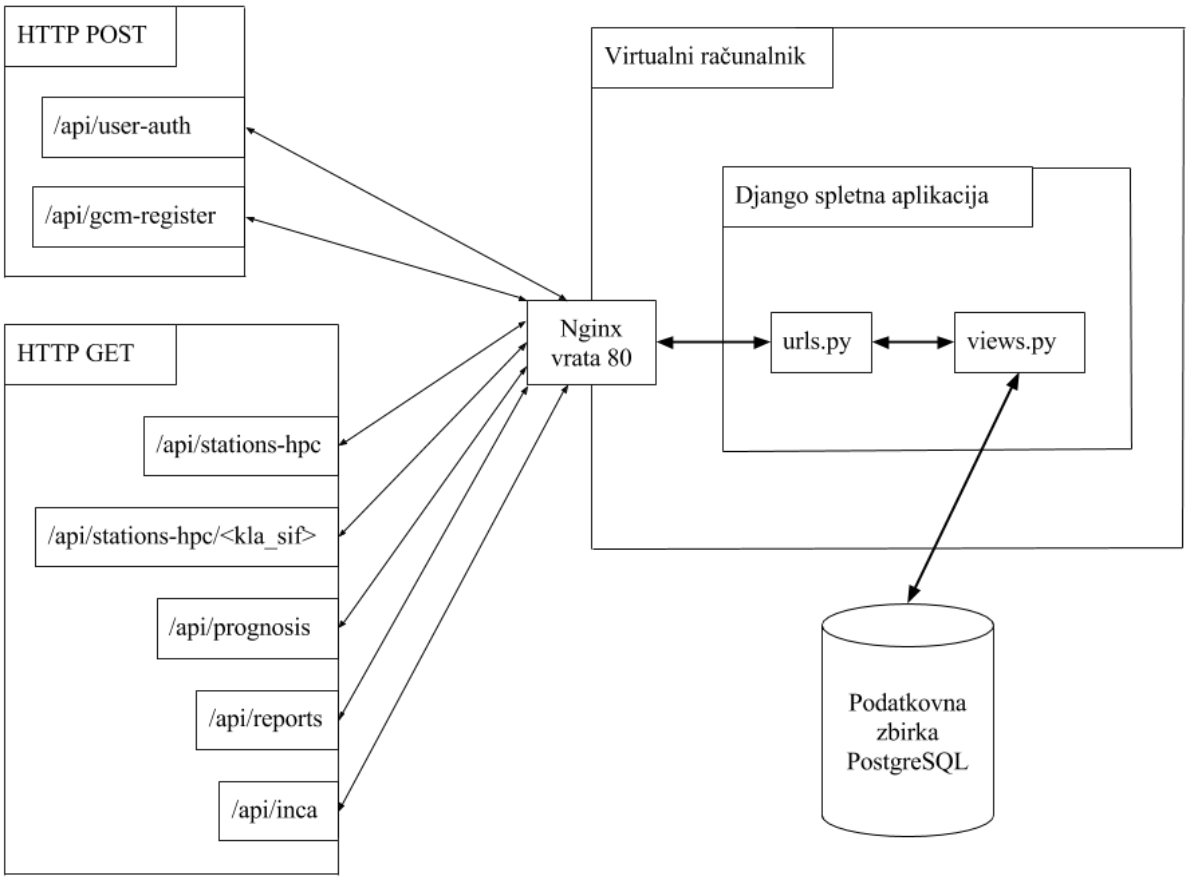
Programska koda 11: Primer crontab vrstice, s pomočjo katere periodično zaganjamo Bash skripto

```
6,16,26,36,46,56 * * * *  /home/ziga/Documents/graduate-
thesis/server_side/remote_logic/latest/latest.sh >
/home/ziga/Documents/graduate-
thesis/server_side/remote_logic/latest/cron.log 2>&1
```

Celoten program bo nameščen na lokalnem računalniku znotraj omrežja Agencije Republike Slovenije za okolje.

5.4 Spletna aplikacija

Spletna aplikacija je nameščena na zunanjem virtualnem računalniku in omogoča dostop do vseh podatkov, ki se nahajajo v zunanji podatkovni zbirki PostgreSQL, preko enolično določenih spletnih naslovov. Vsi podatki bodo na voljo v JSON obliki zapisa. Poleg dostopanja omogoča tudi shranjevanje registracijskih ključev, ki jih prejme od aplikacije Android, potem ko se ta uspešno registrira na Google Cloud Messaging storitev. Spletna aplikacija tako lahko pošilja sporočila Android napravam preko GCM povezovalnega strežnika.



Slika 17: Shema spletnega servisa

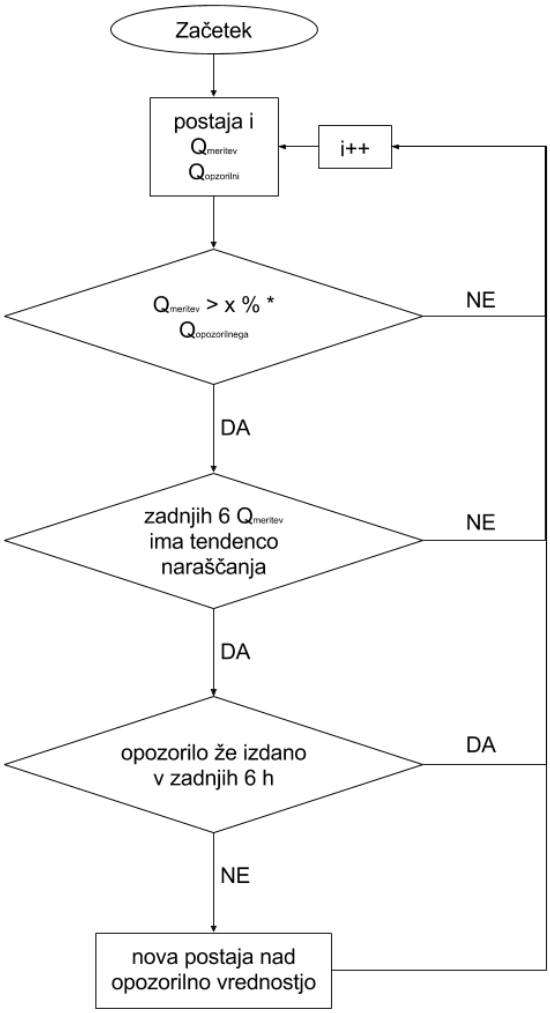
Vsi zahtevki pridejo na virtualni računalnik, ki ima statično določen IP naslov, preko vrat 80, na katerih Nginx spletni strežnik sprejema spletni promet in ga usmerja proti spletni aplikaciji Django (slika 17), ki na podlagi ustreznih zahtevkov vrača oziroma sprejema podatke (preglednica 4). Vsak zahtevek, ki ga prejme aplikacija Django, mora biti za uspešno izvedbo avtenticiran. To pomeni, da mora imeti vsak zahtevek v glavi zahtevka dodan avtentikacijski ključ, ki ga lahko pridobimo tako, da pošljemo uporabniško ime in geslo na spletni naslov */api/user-auth*. Če sta uporabniško ime in geslo veljavna, bo aplikacija Django vrnila avtentikacijski ključ, s pomočjo katerega lahko avtenticiramo vse ostale zahtevke.

Preglednica 4: Seznam podatkovnih virov spletnega servisa

url podnaslov	**HTTP metode**	**parameter**	**opis vira**
/api/user-auth	POST	uporabniško ime, geslo.	Pridobimo avtentikacijski ključ.
/api/gcm-register	POST	GCM registracijski ključ.	Shranimo GCM registracijski ključ.
/api/stations-hpc	GET	-	Seznam postaj z zadnjimi podatki o meritvah in napovedih.
/api/stations-hpc/<ID_postaje>	GET	ID postaje	Časovna serija meritev za zadnjih 72 ur in modelskih napovedi za naslednjih 144 ur.
/api/prognosis	GET	-	Besedilo hidrološke napovedi in opozorila ter seznam opozorilnih območij.
/api/reports	GET	-	Seznam opozoril avtomatskega sistema.
/api/inca	GET	-	Seznam podporečij HPS z vrednostmi eno, treh in šest urnih akumulacij padavin.

5.4.1 Opozarjanje

Dodatna funkcionalnost spletne aplikacije je, poleg serviranja podatkov, tudi redni avtomatski nadzor nad podatki o pretokih. V primeru, da na kateri od avtomatskih hidroloških postaj izmerjeni pretok preseže določen prag, spletna aplikacije preko GCM povezovalnega strežnika obvesti uporabnika mobilne aplikacije. Algoritem za nadzor (slika 18) je spisan v programskem jeziku Python in se zažene vsakih 10 min s pomočjo orodja Cron. Algoritem pridobi seznam vseh postaj s trenutnimi pretoki, za vsako postajo posebej pridobi tudi tendenco pretoka za zadnjih 6 meritev, in primerja izmerjene pretoke z opozorilnimi vrednostmi oziroma deležem opozorilne vrednosti. Če je izmerjeni pretok nad izbranim pragom in ima zadnjih 6 meritev pretoka tendenco naraščanja, potem se sproži proces pošiljanja potisnih sporočil mobilni aplikaciji (programska koda 12)(Google, 2016d). Poleg obveščanja preko mobilne aplikacije smo v proces avtomatskega opozarjanja dodali tudi obveščanje preko platforme Twitter in elektronske pošte.



Slika 18: Diagram algoritma za nadzor nad pretokom na samodejnih hidroloških površinskih postajah

Programska koda 12: Funkcija za pošiljanje potisnih sporočil

```
# coding=utf-8
import requests
import json


def send_noty(gcm_url, gcm_api_key, tokens, title, subtitle, content):
    """
    Funkcija za pošiljanje potisnih sporočil.
    :param gcm_url: spletni naslov GCM povezovalnega strežnika
    :param gcm_api_key:  avtentikacijski ključ za dostop do GCM
povezovalnega strežnika
    :param tokens: registracijski ključi Android aplikacij
    :param title: naslov potisnega spročočila
    :param subtitle: podnaslov potisnega sporočila
    :param content: vsebina potisnega sporočila
    """
    auth = 'key=%s' % gcm_api_key
    headers = {
        'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8',
```

```
        'Authorization': auth
    }
    data = {
        'title': title,
        'subtitle': subtitle,
        'body': content
    }
    payload = {
        'to': tokens,
        'data': json.dumps(data)
    }
    try:
        r = requests.post(gcm_url, data=payload, headers=headers)
        if r.status_code == 200:
            print "Potisna sporočila uspešno poslana"
    except Exception as e:
        print e
```

6 UPORABA SPLETNEGA SERVISA

Uporabo spletnega servisa bomo prikazali na primeru mobilne aplikacije, razvite za trenutno najbolj razširjenem operacijskem sistemu Android. Mobilna aplikacija preko svetovnega spleta dostopa do oddaljenih podatkov in jih na uporabniku prijazen način prikaže v mobilni aplikaciji.

6.1 Mobilna aplikacija Android

Aplikacija bo namenjana zaposlenim v hidrološki prognozi in bo omogočala, da si od kjerkoli na svojem pametnem telefonu ogledajo trenutne hidrološke podatke in so pravočasno avtomatsko obveščeni o spremembah, ki se dogajajo na terenu, o naraščajočih pretokih in hitro razvijajočih se nevihtnih celicah, ki lahko povzročijo hudourniške poplave.

6.1.1 Podatkovni sloj

Podatkovni sloj mobilne aplikacije lahko razdelimo na tri dele. Prvi del je namenjen avtentikaciji uporabnika aplikacije in pridobitvi avtentikacijskega ključa za dostop do spletnega servisa. Uporabnik mobilne aplikacije mora ob prvi uporabi aplikacije vnesti uporabniško ime in geslo. Oba parametra se s pomočjo metode HTTP POST pošljeta na oddaljeni spletni servis (programska koda 13), ki nam v primeru uspešne avtentikacije vrne avtentikacijski ključ, s pomočjo katerega se moramo v nadaljnjih zahtevkih predstaviti spletnemu servisu, da nam le-ta vrne podatke, ki jih zahtevamo.

Programska koda 13: Funckija, s pomočjo katere avtenticiramo uporabnika z uporabniškim imenom in geslom ter pridobimo avtentikacijski ključ

```
private User doLogin(String username, String password) {
    try {
        // HTTP POST zahtevku dodamo uporabniško ime in geslo
        RequestBody formBody = new FormBody.Builder()
        .add("username", username)
        .add("password", password)
        .build();

        // sestavimo HTTP POST zahtevek
        Request request = new Request.Builder()
        .url("http://stanjevoda.duckdns.org/api/user-auth")
        .post(formBody)
        .build();

        // s pomočjo OkHTTP knjižnice izvedemo HTTP POST klic
        OkHttpClient client = new OkHttpClient();
        Response response = client.newCall(request).execute();

        // če je zahtevek neuspešen, sprožimo napako
        if (!response.isSuccessful()) throw new
IOException(response.toString());

        // uspešen zahtevek vrne odgovor, v katerem se nahaja
avtentikacijski ključ
```

```
        JSONObject obj = new JSONObject(response.body().string());

        // ustvarimo nov Java objekt, imenovan User, v katerem se nahaja
uporabniško ime in avtentikacijski ključ
        User user = new User();
        user.setUsername(mUsername);
        user.setToken(obj.getString("token"));

        // vrnemo objekt User
        return user;
    }catch (Exception e){
        // v primeru napake vrnemo prazno vrednost
        return null;
    }
}
```

Drugi del je sestavljen iz registracije na GCM storitev. Mobilna aplikacija ob prvi uspešni avtentikaciji uporabnika zažene proceduro, ki zaprosi GCM povezovalni strežnik za registracijski ključ. Ko GCM povezovalni strežnik vrne registracijski ključ, ga aplikacija pošlje na spletni servis s pomočjo metode HTTP POST (programska koda 14). Isti registracijski ključ nato spletni servis uporabi za pošiljanje potisnih sporočil mobilni aplikaciji.

Programska koda 14: Funckija, s pomočjo katere shranimo GCM registracijski ključ na oddaljeni spletni servis

```
private boolean saveGCMtoken(String authTtoken, String GCMtoken){
    try{
        // sestavimo HTTP POST zahtevek, ki mu v glavo dodamo
avtentikacijski ključ
        // kot parameter zahtevka dodamo GCM registracijski ključ
        Request request=new Request.Builder()
        .url("http://stanjevoda.duckdns.org/api/gcm-register")
        .header("Authorization","Token "+authToken)
        .post(RequestBody.create(
        MediaType.parse("application/json; charset=utf-8"),
        new Gson().toJson(new GcmToken(GCMtoken))))
        .build();

        // pošljemo zahtevek na oddaljeni spletni servis
        OkHttpClient client=new OkHttpClient();
        Response response=client.newCall(request).execute();

        // v primeru napake sprožimo napako
        if(!response.isSuccessful())throw new IOException("Unexpected code
"+response);

        // če je oddaljeni spletni servis uspešno prejel in shranil GCM
registracijski ključ,
        // funckija vrne pozitiven odgovor
        return true;
    }catch(Exception){
        // če pride do napake, vrnemo negativno vrednost
        return false;
    }
}
```

Tretji in zadnji del je sestavljen iz klicev HTTP GET na spletni servis, s katerimi pridobimo podatke, ki ji prikazujemo v mobilni aplikaciji uporabnika (programska koda 15). Vsi zahtevki na spletni servis morajo v glavi zahtevka vsebovati avtentikacijski ključ, ki smo ga pridobili v prvem delu ob avtentikaciji uporabnika.

Programska koda 15: Funkcija, s pomočjo katere naredimo zahtevek HTTP GET zahtevek na oddaljeni spletni servis

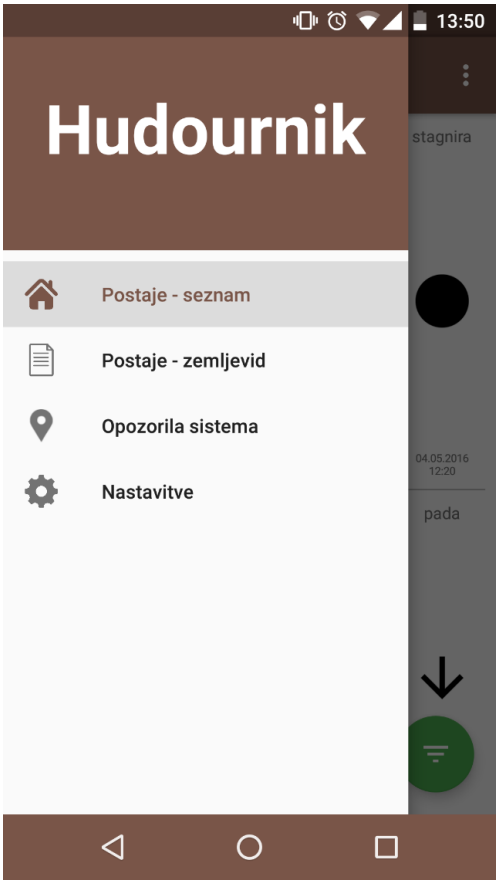
```
public void getData(String authToken) throws Exception {
    // sestavimo HTTP GET zahtevek, ki mu v glavo dodamo avtentikacijski
ključ
    Request request = new Request.Builder()
        .url("http://stanjevoda.duckdns.org/api/latest-hpc")
        .header("Authorization", "Token " + authToken)
        .build();

    // pošljemo zahtevek na oddaljeni spletni servis
    OkHttpClient client=new OkHttpClient();
    client.newCall(request).enqueue(new Callback() {

        Handler mainHandler = new Handler(getMainLooper());

        @Override
        public void onFailure(Call call, IOException e) {
            // pri klicanju oddaljenega spletnega serverja je prišlo do
napake
            mainHandler.post(new Runnable() {

                @Override
                public void run() {
                    // Aktivnosti, ki prikazuje podatke, sporočimo, da je
prišlo do napake
                    sendBroadcast(Preferences.EVENT_ERR_FETCH);
                }
            });
        }

        @Override
        public void onResponse(Call call, Response response) throws
IOException {
            // v primeru napake vržemo napako
            if (!response.isSuccessful()) throw new IOException("Unexpected
code " + response);

            // iz odgovora spletnega servisa preberemo vsebino
            final String resp = response.body().string();

            // shranimo vsebino na lokalni datotečni sistem
            writeData(resp, Preferences.PREFS_FILE_STATIONS);

            mainHandler.post(new Runnable() {

                @Override
                public void run() {
                    // Aktivnosti, ki prikazuje podatke, sporočimo, da so
le ti na voljo
                    sendBroadcast(Preferences.EVENT_NEW_OK);
                }
```

```
                });
            }
        });
}
```

6.1.2 Navigacijska struktura

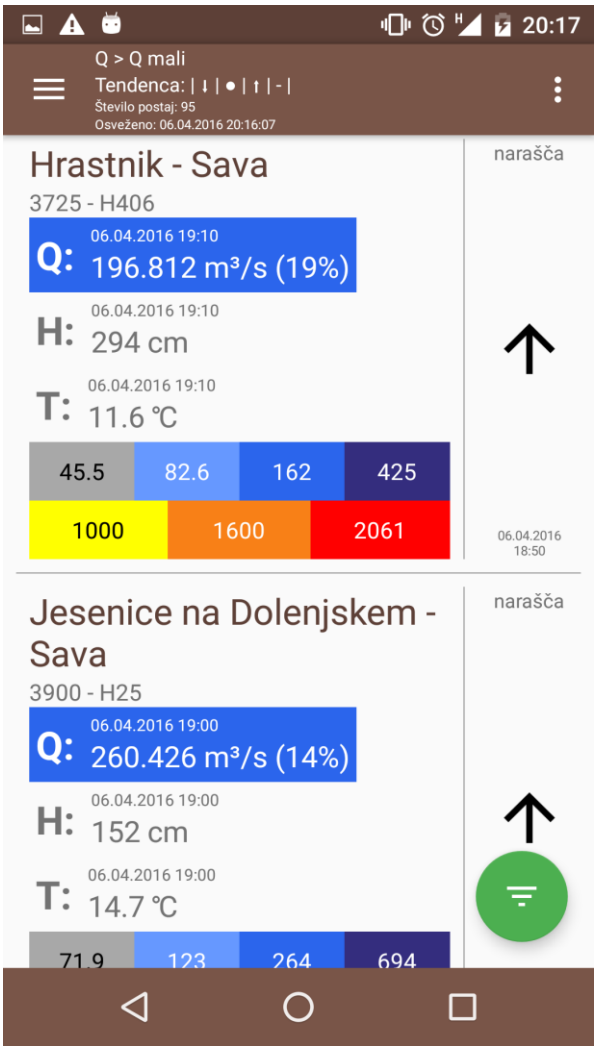
Glavna navigacijska točka aplikacije je navigacijski meni, do katerega lahko dostopamo preko gumba v zgornjem levem kotu aplikacije. Navigacijski meni vsebuje sledeče štiri gumbe (slika 19):

- Postaje - seznam,
- Postaje – zemljevid,
- Opozorila sistema,
- Nastavitve.



Slika 19: Navigacijski meni, ki vsebuje štiri gumbe, preko katerih se pomikamo po aplikaciji

S kliki na posamezen gumb navigacijskega menija se pomikamo med štirimi glavnimi grafičnimi vmesniki naše aplikacije. Klik na prvi gumb navigacijskega menija, imenovan Postaje – seznam, nam prikaže grafični vmesnik s seznamom samodejnih hidroloških postaj površinskih voda (slika 20).



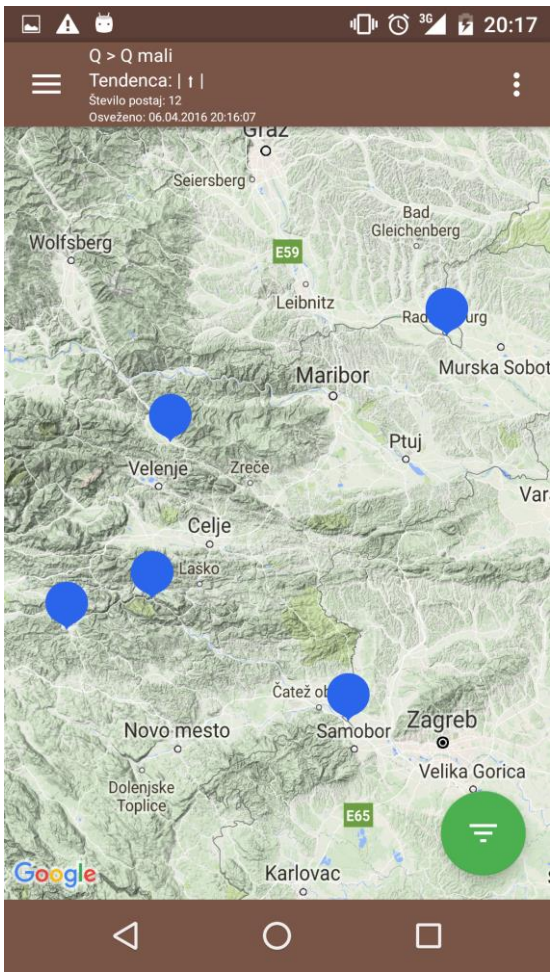
Slika 20: Grafični vmesnik, ki prikazuje seznam samodejnih hidroloških postaj površinskih voda

Vsak element seznama je sestavljen iz sledečih delov (slika 20):

- imena postaje in imena reke,
- klasične hidrološke šifre in šifre merilne mreže,
- trenutnega pretoka in procenta opozorilne vrednosti, ki jo dosega pretok (ozadje se obarva glede na to, v kateri razred pretoka pade trenutni pretok),
- trenutnega vodostaja,
- trenutne temperature vode,
- podatka o tendenci pretoka (označena s puščico),
- statističnih in opozorilnih vrednostih pretoka (tabela v spodnjem delu elementa).

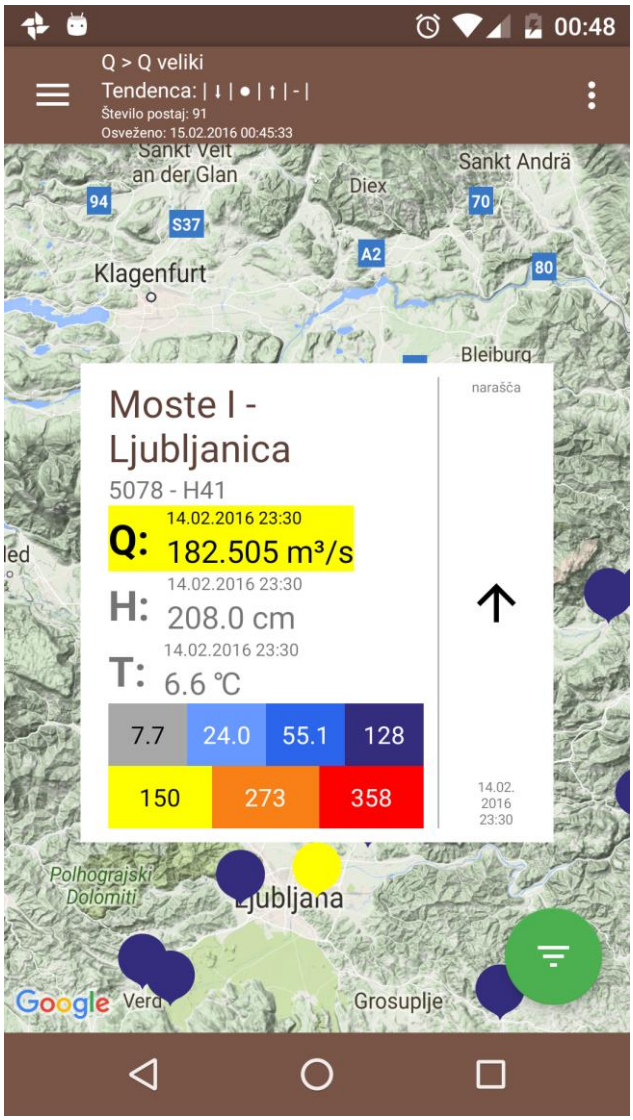
S klikom na poljuben element seznama se nam prikaže grafični vmesnik, ki nam izriše graf pretoka meritev za zadnjih 72 ur in napoved za naslednjih 144 ur v odvisnosti od časa za izbrano postajo.

Drugi gumb navigacijskega meniju se imenuje Postaje – zemljevid (slika 19). S klikom na ta gumb se nam prikaže grafični vmesnik, ki nam izriše zemljevid Slovenije z označenimi lokacijami samodejnih hidroloških postaj površinskih voda. Vsaka lokacija se obarva glede na to, v katerem razredu pretoka je trenutni pretok na izbrani lokaciji (slika 21).



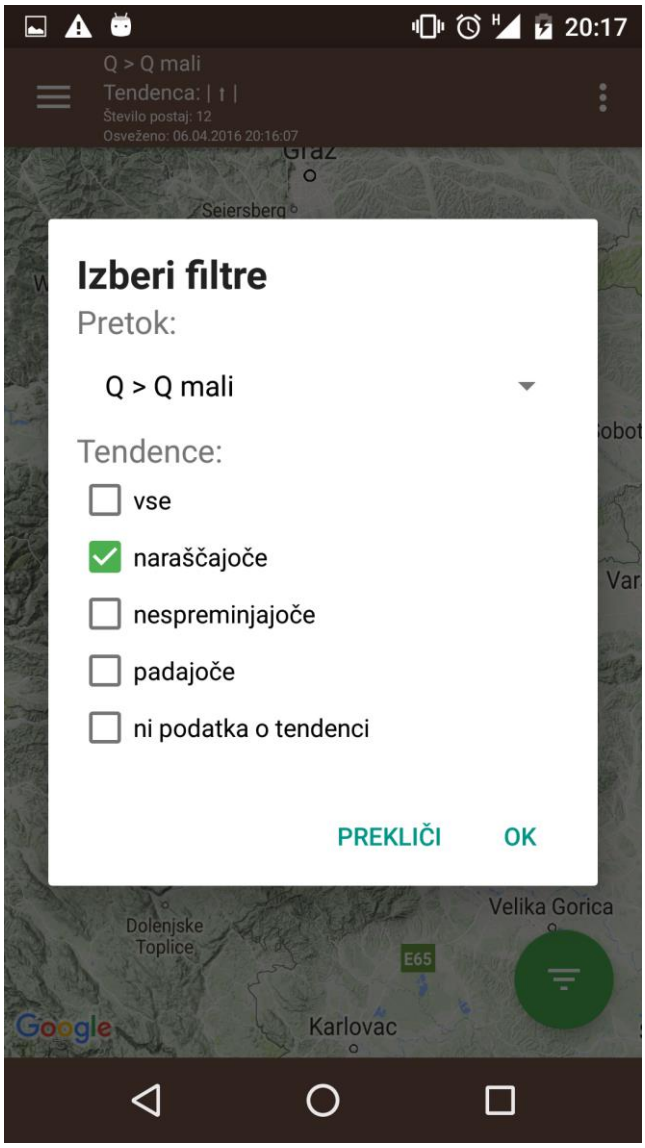
Slika 21: Grafični vmesnik zemljevida Slovenije, ki prikazuje lokacije samodejnih hidroloških postaj površinskih voda

S klikom na poljubno lokacijo postaje se nam prikaže informacijsko okno, ki nam izriše podrobne informacije o trenutnem pretoku, vodostaju, temperaturi vode, tendenci in statističnih ter opozorilnih vrednostih pretoka na izbrani postaji (slika 22).



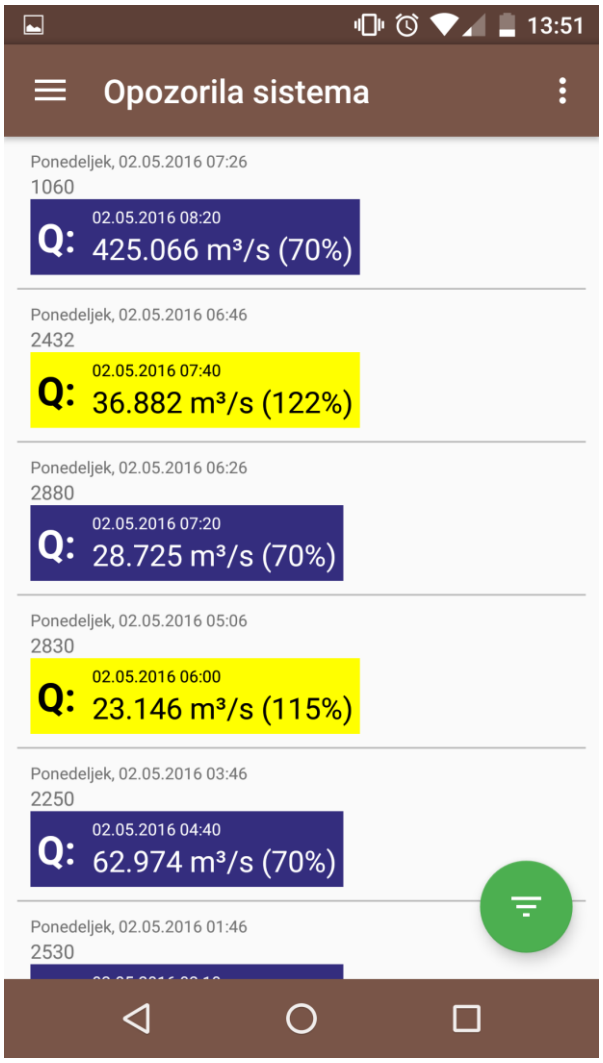
Slika 22: Grafični vmesnik, ki v informacijskem oknu prikazuje podrobne informacije na izbrani samodejni hidrološki površinski postaji

S klikom na informacijsko okno (slika 22) se nam, tako kot v prejšnjem primeru, prikaže grafični vmesnik, ki nam izriše graf pretoka meritev za zadnjih 72 ur in napoved za naslednjih 144 ur v odvisnosti od časa za izbrano postajo. Znotraj obeh grafičnih vmesnikov (slika 20, 21) je implementirano priročno orodje, ki nam omogoča filtriranje postaj po razredu pretoka, v katerem je postaja in tendenci pretoka. Tako lahko zelo hitro izluščimo postaje, ki nas zanimajo v dani situaciji (slika 23). Do orodja lahko dostopamo s klikom na gumb v spodnjem desnem kotu grafičnega vmesnika.



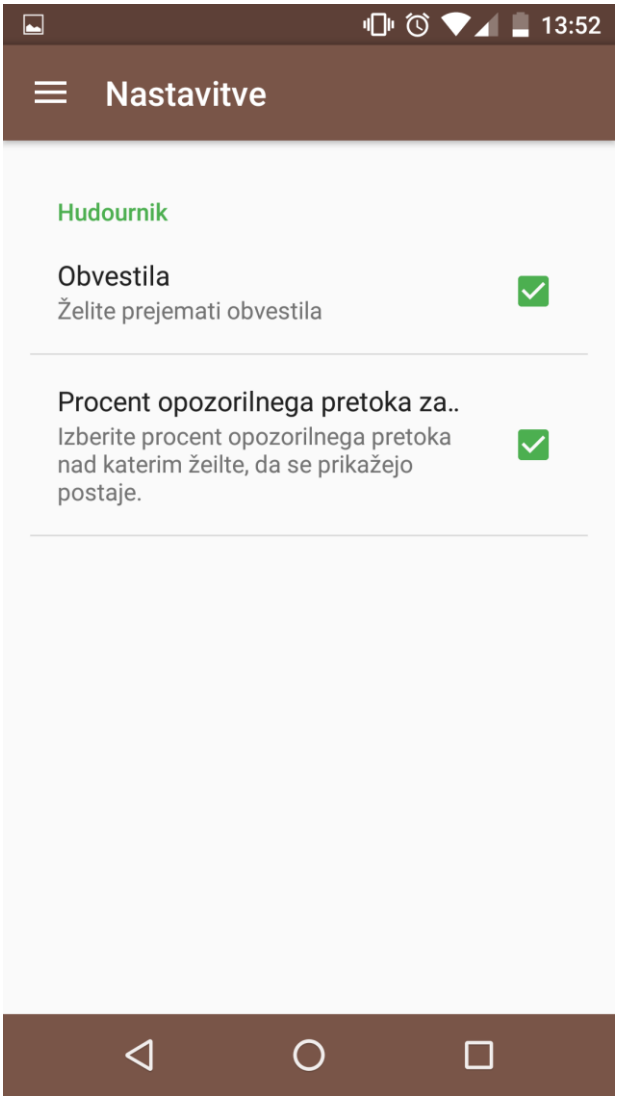
Slika 23: Grafični vmesnik za filtriranje seznama samodejnih hidroloških površinskih postaj

Tretji gumb navigacijskega menija je imenovan Opozorila sistema (slika 19). S klikom na ta gumb se nam prikaže grafični vmesnik, ki nam izriše seznam dogodkov, ki jih je zaznala spletna aplikacija pri svojem avtomatiziranem nadzoru nad podatki o pretokih (slika 24).



Slika 24: Grafični vmesnik, ki nam prikazuje seznam dogodkov, ki jih je zaznal avtomatski sistem za nadzor nad pretoki

Zadnji grafični vmesnik je namenjen nastavitvam aplikacije, preko katerih lahko uporabnik izključi prejemanje potisnih sporočil spletnega servisa in nastavi delež opozorilne vrednosti pretoka, ob katerem prejme opozorilo (slika 25).



Slika 25: Grafični vmesnik nastavitev

6.1.3 Potisna sporočila

Tehnologijo potisnih sporočil smo v mobilno aplikacijo dodali s pomočjo uradne knjižnice GCM, ki nam ponudi vrsto funkcij, s pomočjo katerih se vsaka aplikacija lahko registrira na storitev GCM in sprejema potisna sporočila (Google, 2016). Kot smo že omenili, se ob prvi uspešni avtentikaciji uporabnika zažene procedura, ki pridobi GCM registracijski ključ od GCM povezovalnega strežnika. Ta ključ shranimo na oddaljeni spletni servis, kot smo prikazali v prejšnjem poglavju. Poleg registracije pa je treba znotraj mobilne aplikacije implementirati GCM logiko, ki lahko sprejema sporočila od povezovalnega strežnika GCM in na podlagi teh sporočil opozarja uporabnika, kaj se dogaja. V okviru naloge mobilna aplikacija sprejema potisna sporočila v primeru, da pretok na katerikoli od samodejnih hidroloških površinskih postaj preseže v naprej določen prag. Potem ko mobilna aplikacija zazna, da je sprejela potisno sporočilo (programska koda 16), se sproži metoda, ki z

glasovnim in vizualnim opozorilom sporoči uporabniku naprave, da je prejel novo opozorilo avtomatskega opozorilnega sistema, izdelanega v poglavju 5.

Programska koda 16: Java razred, ki implementira funkcijo, s pomočjo katere sprejemamo potisna sporočila in obveščamo uporabnika aplikacije o novih opozorilih

```
public class AppGCMListenerService extends GcmListenerService {
    /**
     * Funkcija se sproži, ko naprava prejme novo potisno sporočilo
     *
     * @param from ID pošilatelja sporočila
     * @param data Podatki združeni v pare ključ - vrednost
     **/
    @Override
    public void onMessageReceived(String from, Bundle data) {
        try{
            // Preberemo vsebino potisnega sporočila in ga pretvorimo v
Java objekt
            Noty noty = new Gson().fromJson(data.getString("data"),
Noty.class);
            // poženemo metodo, ki obvesti uporabnika o novem sporočilu
            sendNotification(noty);
            }
        }catch (Exception e){
            Log.e(TAG, e.toString());
        }
    }
    /**
     * Funkcija, ki zvokovno in vizualno opozori uporabnika o novem
sporočilu
     *
     * @param noty Java objekt, ki vsebuje podatke sporočila
     */
    private void sendNotification(Noty noty) {
        Intent intent = new Intent(this, NotificationActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        intent.putExtra(Preferences.EXTRA_NOTY_ACTIVITY_ID, noty.getId());
        intent.putExtra(Preferences.EXTRA_NOTY_ACTIVITY_TITLE,
noty.getTitle());
        intent.putExtra(Preferences.EXTRA_NOTY_ACTIVITY_SUBTITLE,
noty.getSubtitle());
        intent.putExtra(Preferences.EXTRA_NOTY_ACTIVITY_BODY,
noty.getBody());
        PendingIntent pendingIntent = PendingIntent.getActivity(
                this,
                0 ,
                intent,
                PendingIntent.FLAG_UPDATE_CURRENT
        );
        Uri defaultSoundUri=
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM);
        NotificationCompat.Builder notificationBuilder = new
NotificationCompat.Builder(this)
                .setSmallIcon(R.mipmap.ic_noty)
                .setLargeIcon(BitmapFactory.decodeResource(getResources(),
R.mipmap.ic_launcher))
                .setContentTitle(noty.getTitle())
                .setContentText(noty.getSubtitle())
                .setAutoCancel(true)
                .setSound(defaultSoundUri)
```

```
                .setContentIntent(pendingIntent)
                .setColor(getColor(R.color.accent));
        NotificationManager notificationManager =
                (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

        notificationManager.notify(noty.getId(),
notificationBuilder.build());
    }
}
```

7 ZAKLJUČEK

V diplomski nalogi smo pregledali področje podatkov javne uprave. Ugotovili smo, da javni sektor proizvaja velike količine podatkov, ki imajo velik socialno-ekonomski potencial. To je zaznala tudi Evropska unija in sprejela vrsto zakonodajnih ukrepov, ki spodbujajo politiko odprtih podatkov javne uprave, ki so dostopni čim širšemu krogu uporabnikov. Na podlagi teh podatkov lahko zasebni uporabniki gradijo svoje rešitve in produkte. Pri tem je potrebno poudariti, da se vsem uporabnikom omogoči dostop do podatkov javne uprave pod enakimi pogoji. Poleg tega se pri obliki dostopanja do podatkov spodbuja uporabo sodobnih in v svetu razširjenih oblik zapisa podatkov, ki omogočajo računalniškim programom, da na preprost način berejo podatke.

V nadaljevanju smo na primeru Agencije Republike Slovenije za okolje pregledali, katere hidrološke podatke hrani in uporablja pri svojem delu. V okviru projekta Bober je agencija posodobila merilno mrežo samodejnih hidroloških površinskih postaj, tako jih ima sedaj 179, preko katerih lahko v realnem času spremlja razmere na slovenskih rekah. Poleg merilne mreže se je v okviru projekta posodobil hidrološki prognostični sistem, ki sedaj omogoča napovedovanje pretokov na vseh večjih rekah Slovenije za 144 ur vnaprej. Na primeru opisanih hidroloških podatkov smo želeli izdelati spletni servis, ki bi omogočal dostop do podatkov preko enovite dostopne točke in prikazal, kako lahko s sodobno tehnologijo izpostavimo logiko, ki jo imajo znotraj agencije. Tako smo v nadaljevanju prestavili tehnologije in orodja, ki jih potrebujemo za izdelavo spletnega servisa in izdelali delujoč spletni servis, ki omogoča dostop do hidroloških meritev in napovedi. Servis poleg serviranja podatkov omogoča tudi avtomatski nadzor nad meritvami pretokov in obveščanje uporabnikov preko različnih kanalov.

Med izdelavo in testiranjem spletnega servisa smo kot najšibkejši člen sistema prepoznali program, ki skrbi za kopiranje podatkov iz notranje podatkovne zbirke Oracle v zunanjo podatkovno zbirko PostgreSQL. Tako v obstoječi sistem dodajamo še en člen, ki ga je treba vzdrževati in administrirati. Po našem mnenju bi bila boljša rešitev, če bi se na nivoju podatkovne zbirke Oracle ustvarila zrcalna podatkovna baza, ki bi se redno osveževala s podatki iz osnovne podatkovne zbirke Oracle. Zrcalno podatkovno zbirko bi naredili dostopno za svetovni splet in bi tako lahko nadomestila zunanjo podatkovno zbirko PostgreSQL, ki smo jo postavili v diplomski nalogi.

Predstavljen spletni servis predstavlja dobro osnovo za nadaljnje korake v smeri izdelave javnega spletnega servisa Agencije Republike Slovenije za okolje, ki bi na enem mestu izpostavil vse javno dostopne podatke agencije. Prednosti takega sistema je več. Prva je zagotovo bolj pregleden in uporabnikom prijazen dostop do podatkov. Druga je zmanjšanje časa, potrebnega za vzdrževanje

sistema, saj bi se vsi javno dostopni podatki agencije izpostavili zunanjemu spletu preko enega sistema. Tretja je spodbuda uporabnikom, ki bi razvili aplikacije na osnovi podatkov agencije. Dejstvo pa je, da bi agencija lahko uporabila spletni servis kot podatkovni sloj v svojih aplikacijah in tako še dodatno zmanjšala stroške in čas, potreben za razvoj aplikacij. Na koncu naj poudarimo, da bi bilo pri postavitvi spletnega servisa treba posebno pozornost nameniti dokumentaciji, ki bi omogočila uporabnikom spletnega servisa, da bi imeli na enem mestu zbrane vse podatke o tem, kateri podatki so na voljo, na kakšen način lahko dostopamo do njih in kakšni so pogoji njihove uporabe.

Primer uporabe spletnega servisa v obliki mobilne aplikacije je lep primer aplikacije, ki lahko nastane na osnovi podatkov javne uprave, ki so dostopni preko spletnega servisa. Mobilna aplikacija omogoča, da lahko hidrološki prognostik v vsakem trenutku pregleda trenutne meritve in napovedi na lokacijah samodejnih hidroloških postaj površinskih voda in je avtomatsko obveščen v primeru, da meritve pretokov presegajo vnaprej določen prag.

VIRI

Agencija Republike Slovenije za okolje. 2016a. Program hidrološkega monitoringa površinskih voda za obdobje 2016-2020. http://www.arso.si/vode/poro%C4%8Dila%20in%20publikacije/Program%20hidrolo%C5%A1kega%20monitoringa%20povr%C5%A1inskih%20voda%202016-2020.pdf (Pridobljeno 10. 4. 2016)

Agencija Republike Slovenije za okolje. 2016b. Geoportal ARSO. http://gis.arso.gov.si/geoportal/catalog/main/home.page (Pridobljeno 16. 5. 2016.)

Albreht, M. 2015. Omejen dostop do informacij javnega značaja, objavljeno 23.6.2015. http://www.delo.si/novice/politika/omejen-dostop-do-informacij-javnega-znacaja.html (Pridobljeno 15. 4. 2016.)

Alonso, G., Casati, F., Kuno, H., Machiraju, V. 2004. Web Services - Concepts, Architecture and Applications. Springer: 123-149.

Amazon. 2015a. Connecting to Your Linux Instance Using SSH. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html (Pridobljeno 25. 12. 2015.)

Amazon. 2015b. Getting Started with Amazon EC2 Linux Instances. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html (Pridobljeno 25. 12. 2015.)

Amazon. 2015c. What Is Amazon EC2? http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (Pridobljeno 25. 12. 2015.)

Amazon. 2016. Amazon Web services. https://aws.amazon.com/ (Pridobljeno 29. 4. 2016.)

Bash. 2016. https://en.wikipedia.org/wiki/Bash_(Unix_shell) (Pridobljeno 30. 5. 2016.)

Bat, M., Miklavčič, J. 2010. Zborniki Mišičevih vodarskih dni 1989 – 2014: Kolomon in hidrolog – novosti v obdelavi podatkov državne hidrološke mreže. http://mvd20.com/LETO2010/R28.pdf (Pridobljeno 3. 1. 2016.)

Brilly, M. 2012. Ogroženost zaradi poplav v Republiki Sloveniji. I. kongres o vodah Slovenije 2012. http://ksh.fgg.uni-lj.si/kongresvoda/03_prispevki/01_vabljeniZnanstStrok/11_Brilly.pdf (Pridobljeno 23. 4. 2016.)

Carrara, W. 2015. Looking for Open Data from a different country? Try the European Data portal, objavljeno 16.11.2015. https://ec.europa.eu/digital-single-market/blog/looking-open-data-different-country-try-european-data-portal (Pridobljeno 12. 4. 2016.)

Cron. 2016. https://en.wikipedia.org/wiki/Cron (Pridobljeno 2. 5. 2016.)

DigitalOcean. 2015. Apache vs Nginx: Practical Considerations. https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations (Pridobljeno 2. 1. 2016.)

Direktiva 2003/98/ES Evropskega parlamenta in sveta z dne 17. novembra 2003 o ponovni rabi informacij javnega značaja. Uradni list Evropske unije, L 345/90.

Direktiva 2007/60/ES Evropskega parlamenta in sveta z dne 23. oktobra 2007 o oceni in obvladovanju poplavne ogroženosti. Uradni list Evropske unije, L 288/27.

Direktiva 2013/37/EU Evropskega parlamenta in sveta z dne 26. junija 2013 o spremembi Direktive 2003/98/ES o ponovni uporabi informacij javnega sektorja. Uradni list Evropske unije, L 175/1.

Django software foundation. 2016. Writing your first Django app, part 1. https://docs.djangoproject.com/en/1.9/intro/tutorial01/ (Pridobljeno 14. 1. 2016.)

Django. 2015. https://en.wikipedia.org/wiki/Django_(web_framework) (Pridobljeno 14. 1. 2015.)

Drake, J. 2005. Accessing PostgreSQL with Python and Psycopg, objavljeno 5. 8. 2005. http://www.devx.com/opensource/Article/29071 (Pridobljeno 2. 1. 2016.)

Evropska komisija. 2014. Guidelines on recommended standard licences, datasets and charging for the reuse of documents. Official journal of the European Union: str C 240/1 – C 240/10.

Evropska komisija. 2015. Creating value through open data.

https://www.capgemini-consulting.com/resource-file-access/resource/pdf/open_data_value.pdf (Pridobljeno 24. 3. 2016.)

Evropska komisija. 2016. Digital single market, objavljeno 25.02.2016. https://ec.europa.eu/digital-single-market/en/digital-single-market (Pridobljeno 13. 4. 2016.)

Evropska okoljska agencija. 2011. Disasters in Europe: more frequeent and causing more damage, objavljeno 12.1.2011. http://www.eea.europa.eu/highlights/natural-hazards-and-technological-accidents (Pridobljeno 23. 4. 2015.)

Evropska unija. 2016. European union open data portal. Developer's corner. https://open-data.europa.eu/en/developerscorner (Pridobljeno 13. 4. 2016.)

Geodetska uprava Republike Slovenije. 2016. Prostorski portal. http://www.e-prostor.gov.si/ (pridobljeno 16. 5. 2016.)

Google. 2016a. Cloud Messaging. https://developers.google.com/cloud-messaging/ (Pridobljeno 27. 4. 2016.)

Google. 2016b. Google API Console. https://console.developers.google.com/ (Pridobljeno 5. 5. 2016.)

Google. 2016c. Google Cloud Messaging: Overview. https://developers.google.com/cloud-messaging/gcm (Pridobljeno 27. 4. 2016.)

Google. 2016d. Simple Downstream messaging. https://developers.google.com/cloud-messaging/downstream#xmpp_message (Pridobljeno 27. 4. 2016.)

Granickas, K. 2013. Understanding the impact of releasing and re-using open government data. http://www.epsiplatform.eu/sites/default/files/2013-08-Open_Data_Impact.pdf (Pridobljeno 11. 4. 2016.)

Hrastovšek, Ž. Prototip sistema za obveščanje o izrednih dogodkih z uporabo tehnologije Google Cloud Messaging. Diplomska naloga. Ljubljana, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko (samozaložba Ž. Hrastovšek): 64 str.

Hypertext Transfer Protocol. 2016.
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol (Pridobljeno 3. 1. 2016.)

Komac, B., Natek, K., Zorn. M. 2008. Geografski vidiki poplav v Sloveniji. Založba ZRC: 180 str.

Komisija za preprečevanje korupcije. 2011. O supervizorju. http://supervizor.kpk-rs.si/. (Pridobljeno 15. 4. 2016.)

Kotnik Šumah, K. 2010. State of play: PSI Re-use in Slovenia.
https://www.europeandataportal.eu/sites/default/files/library/06_psi_re_use_in_slovenia.pdf
(Pridobljeno 15. 4. 2015.)

Lee, W. 2012. Beginning Android Application Development. Indianapolis, John Wiley & Sons: 503 str.

Linux. 2016a. https://en.wikipedia.org/wiki/Linux (Pridobljeno 23. 5. 2016.)

Linux. 2016b. What is Linux? https://www.linux.com/what-is-linux (Pridobljeno 23. 5. 2016.)

Mikoš, M. 1995. Soodvisnost erozijskih pojavov v prostoru. Gozdarski vestnik 53-2. Ljubljana: str 342 - 351.

Ministrstvo za javno upravo. 2016. Spletni dostop do javnih evidenc, ponovna uporaba in odprti podatki.
http://www.mju.gov.si/si/delovna_podrocja/transparentnost_in_dostop_do_informacij_javnega_znacaja/spletni_dostop_do_javnih_evidenc_ponovna_uporaba_in_odprti_podatki/ (Pridobljeno 15. 4. 2016.)

Ministrstvo za okolje in prostor. 2015a. Spletni brezplačni dostop do podatkov laserskega skeniranj (LIDAR), objavljeno 17. 4. 2015.
http://www.mop.gov.si/nc/si/medijsko_sredisce/novica/article/12029/5960/ (Pridobljeno 15. 4. 2016.)

Ministrstvo za okolje in prostor. 2015b. Načrt zmanjševanja poplavne ogroženosti.
http://www.mop.gov.si/fileadmin/mop.gov.si/pageuploads/podrocja/voda/nzpo/NZPO_SLO_2015_12_08.pdf (Pridobljeno 23. 4. 2016.)

Nginx. 2016. Beginner’s Guide.
http://nginx.org/en/docs/beginners_guide.html (Pridobljeno 13. 1. 2016.)

Oracle. Using Python With Oracle Database 11g.
http://www.oracle.com/technetwork/articles/dsl/python-091105.html (Pridobljeno 2. 1. 2016.)

Petan, S., Golob, A., Moderc, M. 2015. Sistem za hidrološko napovedovanje Agencije Republike Slovenije za okolje. Vetrnica, 0815: 14-20.

PgAdmin. 2016. pgAdmin 1.4 online documentation. http://www.pgadmin.org/docs/1.4/using.html (Pridobljeno 29. 4. 2016.)

Podila, P., 2013. HTTP: The Protocol Every Web Developer Must Know - Part 1, objavljeno 8.4.2013. http://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177 (Pridobljeno 5. 1. 2016.)

Pogačnik, N., Rotar, E., Rak, G. 2010. Opozarjanje javnosti pred škodljivim delovanjem voda: Začetek razvoja sistema hidroalarm v okviru sedanjega sistema za opozarjanje pred vremensko nevarnostjo - meteoalarm. Ujma, številka 24: 132-139.

Python. 2016.
https://en.wikipedia.org/wiki/Python_(programming_language) (pridobljeno 2. 5. 2016.)

Representational state transfer. 2016. https://en.wikipedia.org/wiki/Representational_state_transfer (Pridobljeno 28. 4. 2016.)

Richardson, L., Ruby, S. 2007. RESTful Web Services. O'Reilly Media: 419 str.

Roškar, J. BOBER Nadgradnja sistema za spremljanje in analiziranje stanja vodnega okolja v Sloveniji. 2015. Vetrnica, 0815: 38-53.

Secure Shell. 2015. https://en.wikipedia.org/wiki/Secure_Shell (Pridobljeno 25. 12. 2015.)

Sluga, G., Kočevar, H. 2013. Nadgradnja sistema za spremljanje in analiziranje stanja vodnega okolja v Sloveniji - Projekt Bober. 24. Mišičev vodarski dan 2013: 188-192.

Sodstvo Republike Slovenije. 2010. Zemljiška knjiga.
http://www.sodisce.si/javne_knjige/zemljiska_knjiga/ (Pridobljeno 16. 5. 2016.)

Supervizor. 2014. Programski vmesnik do spletne aplikacije Supervizor, objavljeno 22.5.2014. https://github.com/Supervizor/supervizor-api (Pridobljeno 15. 4. 2016.)

Šajn Slak, A., Kršmanc, R., Merše, J. 2012. INCA-CE – projekt, ki povezuje meteorološke službe osrednje Evrope s končnimi uporabniki. Vetrnica, 0412: 61-63.

Ubuntu. 2015.
https://help.ubuntu.com/community/Oracle%20Instant%20Client (Pridobljeno 2. 1. 2016.)

Uredba o posredovanju in ponovni uporabi informacij javnega značaja. Uradni list RS št. 76/2005.

World meteorological organization. 2011. Manual on flood forecasting and warning. http://www.wmo.int/pages/prog/hwrp/publications/flood_forecasting_warning/WMO%201072_en.pdf (Pridobljeno 23. 4. 2016.)

World Wide Web Constortium (W3C). 2014. Web services architecture.
https://www.w3.org/TR/ws-arch/#whatis (Pridobljeno 27. 4. 2016.)

Zakon o dostopu do informacij javnega značaja. Uradni list RS št. 51/2006.